Thèse de doctorat

Contributions to Representation Learning with Graph Autoencoders and Applications to Music Recommendation

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°626 de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 8 mars 2022, par

GUILLAUME SALHA-GALVAN

Composition du Jury :

Thomas Bonald
Professeur, Télécom Paris   Président

Patrick Gallinari
Professeur, Sorbonne Université   Rapporteur

Yizhou Sun
Professeur, University of California, Los Angeles   Rapporteur

Markus Schedl
Professeur, Johannes Kepler Universität Linz   Examinateur

Jie Tang
Professeur, Tsinghua University   Examinateur

Michal Valko
Chercheur, INRIA & DeepMind   Examinateur

Michalis Vazirgiannis
Professeur, École polytechnique   Directeur de thèse

Romain Hennequin
Chercheur, Deezer   Co-directeur de thèse

Preface

This PhD thesis presents research conducted from October 2018 to September 2021. It is the outcome of a scientific collaboration between the music streaming service **Deezer**, where the PhD candidate is a permanent employee in the "Deezer Research" team in Paris, France, and the **Laboratoire d'informatique de l'École polytechnique (LIX)**, in Palaiseau, France. This project has been entirely funded by Deezer.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to **Romain Hennequin** from Deezer, who served as my primary supervisor over the past three years and devoted countless hours to reviewing this thesis, along with all my papers, code, posters, and presentations. In addition, Romain always supported me in this tumultuous but exciting journey that a PhD program inevitably constitutes. His scientific rigor and his intelligence, combined with his kindness and his availability, make him the perfect mentor. I learned a lot by his side and I sincerely hope, without really doubting it, that our personal ties will last long after this thesis.

I would also like to thank **Michalis Vazirgiannis**, my supervisor on the Polytechnique side, for his guidance and the confidence he has placed in me throughout this PhD. By welcoming me to his team at LIX, he made this thesis possible. His support allowed me to teach at Polytechnique and to collaborate with the talented LIX researchers mentioned in the next paragraphs. I am grateful not only for these opportunities but also for his trust, insightful advice, and perspective, which helped shape the direction of my research.

I am honored to have six world-class external researchers on my defense committee. In particular, **Patrick Gallinari** from Sorbonne Université and **Yizhou Sun** from UCLA both accepted to spend time and effort to review this thesis. I would like to thank them for their careful reports and valuable comments. Moreover, I am grateful to **Thomas Bonald** from Télécom Paris, who acted as President of this committee and shared his profound expertise on graph learning. I would also like to thank **Markus Schedl** from JKU, who traveled from Austria to attend the defense, and **Jie Tang** from Tsinghua University, who virtually joined us from China. They reinforced this jury by bringing their outstanding knowledge on recommender systems and graph mining, respectively. Last, but not least, I am grateful to **Michal Valko** from INRIA and DeepMind. His *"Graphs in Machine Learning"* course at ENS Paris-Saclay, back in 2015, initially triggered my fascination for graphs. His presence on my committee therefore means a lot to me.

Besides, I would like to thank **Deezer**, where I have been fortunate to work for the past five years, for having entirely funded my PhD project. This would never have been possible without the keen support of **Aurélien Hérault**, our Chief Data and Research Officer, who defended this project when few people believed in it. I am indebted for the opportunities he offered me.

In particular, when I started this PhD in 2018, I simultaneously joined our **Deezer Research** team as a researcher, after having worked as a data scientist at Deezer for nearly two years.

autoencoders and graph convolutional networks. Finally, like many researchers, I also owe a lot to **Alexandra Elbakyan** and her major contributions towards making science more accessible.

On a more personal level, I would like to thank **Damien Silès** for the invaluable pieces of life advice he has been providing me since we met, and for helping me to be proud of who I am. Moreover, my life in Paris would not be the same without the never-failing support of my close friends **Gautier Appert** and **Man Zhang**. It was inconceivable for me to write these acknowledgments without mentioning them. I am confident that our future will hold many more culinary discoveries and endless geek discussions. Of course, as this thesis deals with music, I also have a very special thought for my dear musician friends **Fabien Castillon**, **Mélissa Castillon** and **Kévin Hamard**.

Je passe désormais de l'anglais au français, afin d'adresser un remerciement tout particulier à l'ensemble de **ma famille**, et notamment à mon père, à ma mère, à ma sœur, à mon frère, ainsi qu'à mes quatre grands-parents. Je ne serai pas arrivé où j'en suis sans leur amour et leur soutien inconditionnels. J'espère que cette thèse les rendra fiers. Mes parents m'ont aussi accueilli, épaulé et supporté (je sais que ça n'est pas toujours facile) le temps de la rédaction de celle-ci dans mon Pays Basque natal. Grâce à eux, j'ai pu terminer ce long projet dans les meilleures conditions possibles. Je leur en suis reconnaissant.

I dedicate this last paragraph, perhaps the most important one, to my life partner **Jiaying Xu**. Without her encouragement and loving support, I would never have considered starting this thesis. She accompanied me and helped me in all the steps of this long project. In recognition of her patience, attention, and benevolence throughout these years, she should have received a diploma as well. I am extremely lucky to have her by my side, and I am confident that our future is bright and full of exciting new adventures.

Guillaume

Contributions à l'apprentissage de représentations à partir d'autoencodeurs de graphes, et applications à la recommandation musicale

MANDATORY ABSTRACT IN FRENCH / RÉSUMÉ OBLIGATOIRE EN FRANÇAIS

Les graphes sont des structures omniprésentes dans de nombreux domaines de recherche, allant de l'étude des réseaux sociaux à la recommandation automatique de contenu en passant par la biologie. Cette omniprésence est due à la prolifération de données représentant les relations ou les interactions entre des entités, aussi appelées des nœuds. Dans un graphe, ces relations et interactions sont résumées sous la forme de liens, aussi appelés des arêtes, qui relient ces nœuds entre eux.

Extraire de l'information pertinente au sujet des nœuds et des arêtes d'un graphe est essentiel pour espérer résoudre de nombreux problèmes d'apprentissage machine. Cela inclut notamment la prédiction de liens manquants entre des paires de nœuds, ainsi que la détection de communautés de nœuds, deux problèmes sur lesquels nous nous concentrons tout particulièrement au sein de cette thèse. Au cours des dernières décennies, diverses méthodes ont été proposées afin de résoudre de tels problèmes d'apprentissage. Alors que des approches traditionnelles proposaient de construire manuellement des indicateurs statistiques résumant les nœuds et leurs connexions, des améliorations significatives ont récemment été obtenues par des méthodes visant à directement apprendre, au travers d'un modèle, les meilleures représentations vectorielles des nœuds d'un graphe selon certains critères prédéfinis.

En particulier, les autoencodeurs de graphes (GAE) et les autoencodeurs variationnels de graphes (VGAE) se sont récemment imposés comme deux puissants groupes de méthodes permettant de construire de telles représentations vectorielles de manière non supervisée, avec des applications prometteuses à de divers problèmes d'apprentissage dont ceux cités précédemment. Au début de notre recherche, les GAE et VGAE souffraient néanmoins de plusieurs limitations majeures, que nous détaillons tout au long de cette thèse. Ces limitations entravaient l'utilisation des GAE et VGAE dans le cadre d'applications industrielles à grande échelle, par exemple liées à la recommandation musicale au sein du service de *streaming* musical Deezer, où ce doctorat a été réalisé.

Dans cette thèse, nous présentons plusieurs contributions permettant d'améliorer les GAE et VGAE et de faciliter de telles utilisations. Tout d'abord, nous proposons deux stratégies permettant de surmonter les problèmes de passage à l'échelle des GAE et VGAE, et d'entraîner ces

modèles sur des graphes ayant des millions de nœuds et d'arêtes tels que ceux disponibles chez Deezer. Ces stratégies exploitent respectivement des techniques de dégénérescence de graphes et de décodage stochastique de sous-graphes. Par ailleurs, nous présentons nos GAE et VGAE inspirés de la gravité (de l'anglais *Gravity-Inspired GAE and VGAE*). Il s'agit des premières extensions de ces modèles destinées aux graphes dirigés, qui sont omniprésents dans les applications industrielles. Ces extensions obtiennent de meilleurs résultats empiriques que plusieurs alternatives populaires pour différents problèmes de prédiction de liens manquants dirigés. Nous étudions également des extensions des GAE et VGAE destinées aux graphes dynamiques, dans lesquels les nœuds et les arêtes peuvent évoluer au cours du temps. En outre, nous démontrons que les GAE et VGAE existants, qui reposent en général sur des réseaux convolutifs de graphes (GCN) multi-couches, sont souvent inutilement complexes. Par conséquent, nous proposons de les simplifier, en ayant recours à des encodeurs linéaires. Enfin, nous présentons nos GAE et VGAE informés par la modularité (de l'anglais *Modularity-Aware GAE and VGAE*), qui permettent d'améliorer la détection de communautés de nœuds, tout en préservant de bonnes performances pour la prédiction de liens manquants.

Dans la dernière partie de cette thèse, nous évaluons nos méthodes sur plusieurs graphes extraits du service de *streaming* musical Deezer. Nous nous concentrons sur des problèmes de recommandation musicale à partir de graphes, souvent présentés sous la forme de problèmes de prédiction de liens manquants ou de détection de communautés de nœuds. En particulier, nous montrons que nos méthodes permettent d'améliorer la détection de communautés d'entités musicales à recommander aux mêmes utilisateurs, mais aussi de mieux classer des artistes similaires dans un contexte de démarrage à froid (de l'anglais *cold start*), et enfin de mieux modéliser la perception des genres musicaux à travers différentes cultures. Pour terminer, nous présentons également deux autres modèles, récemment déployés en production chez Deezer afin de recommander de la musique à des millions d'utilisateurs. Ces deux modèles permettent respectivement de personnaliser les carrousels de playlists et d'albums proposés aux utilisateurs sur la page d'accueil de Deezer et, dans un contexte de démarrage à froid, de recommander du contenu musical aux nouveaux utilisateurs du service, pour lesquels peu de données sont disponibles. Bien qu'étant moins directement liés aux GAE et VGAE, ces deux derniers projets fournissent un point de vue complémentaire sur des sujets de recommandation musicale connexes à ceux étudiés précédemment.

Contents

List of Figures

List of Tables

List of Acronyms and Symbols

Common Acronyms

- **AE**: Autoencoder;

- **ALS**: Alternating Least Squares;

- **AMI**: Adjusted Mutual Information;

- **AP**: Average Precision;

- **APP**: Asymmetric Proximity Preserving;

- **ARI**: Adjusted Rand Index;

- **AUC**: Area Under the ROC Curve;

- **CVAE**: Collaborative Variational Autoencoder;

- **DEAL**: Dual-Encoder Graph Embedding with Alignment;

- **DVGAE**: Dirichlet Variational Graph Autoencoder;

- **ELBO**: Evidence Lower Bound;

- **GAE**: Graph Autoencoder;

- **GCN**: Graph Convolutional Network;

- **GNN**: Graph Neural Network;

- **HOPE**: High Order Proximity Preserved Embedding;

- **KL**: Kullback-Leibler;

- **LINE**: Large-Scale Information Network Embedding;

- **MAP**: Mean Average Precision;

- **MeLU**: Meta-Learned User Preference Estimator;

- **MGAE**: Marginalized Graph Autoencoder;

- **NDCG**: Normalized Discounted Cumulative Gain;

- **NLP**: Natural Language Processing;

- **NN**: Neural Network;

- **ReLU**: Rectified Linear Unit;

- **ROC**: Receiver Operating Characteristic;

- **SGC**: Simple Graph Convolution;

- **SVD**: Singular Value Decomposition;

- **TS**: Thompson Sampling;

- **UCB**: Upper Confidence Bound;

- **VAE**: Variational Autoencoder;

- **VGAE**: Variational Graph Autoencoder;

- **VGAECD**: Variational Graph Autoencoder for Community Detection.

Common Linear Algebra Symbols and Operations For some dimension $p \in \mathbb{N}^*$, some vectors $a \in \mathbb{R}^p$ and $b \in \mathbb{R}^p$, and some square matrices $M \in \mathbb{R}^{p \times p}$ and $N \in \mathbb{R}^{p \times p}$:

- $\mathbf{1}_p$: the p-dimensional vector containing p entries all equal to 1;

- $a_i \in \mathbb{R}$: the i-th element of a, for some $i \in \{1, \ldots, p\}$;

- $M_{ij} \in \mathbb{R}$: the element at the i-th row and j-th column of M, for some $(i,j) \in \{1, \ldots, p\}^2$;

- $\|a\|_2 = \sqrt{\sum_{i=1}^{p} a_i^2} \in \mathbb{R}^+$: the Euclidean norm of a;

- $\|a - b\|_2 = \sqrt{\sum_{i=1}^{p} (a_i - b_i)^2} \in \mathbb{R}^+$: the Euclidean distance between a and b;

- $a^T b = \sum_{i=1}^{p} a_i b_i \in \mathbb{R}$: the inner product between a and b;

- $\text{diag}(a) \in \mathbb{R}^{p \times p}$: the $p \times p$ diagonal matrix verifying $\text{diag}(a)_{ii} = a_i$, $\forall i \in \{1, \ldots, p\}$ and $\text{diag}(a)_{ij} = 0$, $\forall (i,j) \in \{1, \ldots, p\}^2$ with $i \neq j$;

- $I_p = \text{diag}(\mathbf{1}_p)$: the $p \times p$ identity matrix;

- $MN \in \mathbb{R}^{p \times p}$: the matrix multiplication of M and N ($\neq NM$ in general). It extends to $M'N' \in \mathbb{R}^{p_1 \times p_3}$, for two matrices $M' \in \mathbb{R}^{p_1 \times p_2}$ and $N' \in \mathbb{R}^{p_2 \times p_3}$ with $p_1, p_2, p_3 \in \mathbb{N}^*$;

- $M^T \in \mathbb{R}^{p \times p}$: the transpose of A, verifying: $\forall (i,j) \in \{1, \ldots, p\}^2, M_{ij}^T = M_{ji}$;

- $M^{-1} \in \mathbb{R}^{p \times p}$: the inverse of M, assuming M is nonsingular, i.e., $MM^{-1} = M^{-1}M = I_p$;

- $\mathrm{Tr}(M) = \sum_{i=1}^{p} M_{ii} \in \mathbb{R}$: the trace of M;

- $M = U\Lambda U^{-1}$, with $U \in \mathbb{R}^{p \times p}$ and $\Lambda \in \mathbb{R}^{p \times p}$: the eigendecomposition of M, assuming M is diagonalizable. We denote by $u_i \in \mathbb{R}^p$ the eigenvector associated with an eigenvalue $\lambda_i \in \mathbb{R}$ of M, i.e., it verifies $Mu_i = \lambda_i u_i$. Here, Λ denotes the diagonal matrix containing the p eigenvalues $\lambda_1, \ldots, \lambda_p$ of M in the diagonal. $U = [u_1, \ldots, u_p]^T$ contains their corresponding eigenvectors $u_1, \ldots, u_p$ as rows. If U is orthogonal, we have: $M = U\Lambda U^T$;

- $\|M\|_F = \sqrt{\mathrm{Tr}(M^T M)} = \sqrt{\sum_{i=1}^{p} \sum_{j=1}^{p} M_{ij}^2} \in \mathbb{R}^+$: the Frobenius norm of M;

- $\|M - N\|_F = \sqrt{\sum_{i=1}^{p} \sum_{j=1}^{p} (M_{ij} - N_{ij})^2} \in \mathbb{R}^+$: the Frobenius distance between M and N.

Common Graph Representation Learning Symbols

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: a graph composed of a node set $\mathcal{V}$ and an edge set $\mathcal{E}$;

- $i \in \mathcal{V}$: a node i from $\mathcal{G}$;

- $(i, j) \in \mathcal{E}$: an edge connecting node i to node j in $\mathcal{G}$;

- $\mathcal{N}(i) \subseteq \mathcal{V}$, $\mathcal{N}_{\mathrm{in}}(i) \subseteq \mathcal{V}$, $\mathcal{N}_{\mathrm{out}}(i) \subseteq \mathcal{V}$: the neighborhood, incoming neighborhood and outcoming neighborhood of a node $i \in \mathcal{V}$, respectively;

- $w_{ij} \in [0, 1]$: the normalized weight associated with the edge (i, j);

- $n \in \mathbb{N}^*$: the number of nodes in $\mathcal{G}$, i.e., $n = |\mathcal{V}|$;

- $m \in \mathbb{N}$: the number of edges in $\mathcal{G}$, i.e., $m = |\mathcal{E}|$;

- $f \in \mathbb{N}^*$: the dimension of feature vectors describing each node;

- $x_i \in \mathbb{R}^f$: the feature vector describing node $i \in \mathcal{V}$;

- $X \in \mathbb{R}^{n \times f}$: the node feature matrix, stacking up all x_i vectors;

- $d \in \mathbb{N}^*$: the dimension of the node embedding space;

- $z_i \in \mathbb{R}^d$: the embedding vector of node $i \in \mathcal{V}$;

- $Z \in \mathbb{R}^{n \times d}$: the node embedding matrix, stacking up all z_i vectors;

- $A \in [0,1]^{n \times n}$: the adjacency matrix of $\mathcal{G}$;

- $\tilde{A} \in [0,1]^{n \times n}$: the symmetric normalization of A;

- $\tilde{A}_{\text{out}} \in [0,1]^{n \times n}$: the out-degree normalization of A;

- $\hat{A} \in [0,1]^{n \times n}$: the adjacency matrix reconstructed by a GAE or VGAE model;

- $D_{\text{in}} \in [0,n]^{n \times n}$: the in-degree matrix of $\mathcal{G}$;

- $D_{\text{out}} \in [0,n]^{n \times n}$: the out-degree matrix of $\mathcal{G}$;

- $D \in [0,n]^{n \times n}$: the degree matrix of $\mathcal{G}$, for an undirected graph;

- $L_{\text{un}} \in [-1,n]^{n \times n}$: the unnormalized Laplacian matrix of $\mathcal{G}$;

- $L_{\text{rw}} \in [-1,n]^{n \times n}$: the random-walk normalized Laplacian matrix of $\mathcal{G}$;

- $L_{\text{sym}} \in [-1,n]^{n \times n}$: the symmetrically normalized Laplacian matrix of $\mathcal{G}$;

- $K \in \{1, \ldots, n\}$: the number of communities in $\mathcal{G}$;

- $C_1 \subseteq \mathcal{V}, \ldots, C_K \subseteq \mathcal{V}$: a partition of $\mathcal{V}$ into K communities;

- $A_c \in \{0,1\}^{n \times K}$: the community membership matrix corresponding to the above partition;

- $A_s \in \{0,1\}^{n \times K}$: the s-regular sparsification of the above matrix A_c;

- $L \in \mathbb{N}^*$: the number of layers in a GNN, excluding the input layer;

- $d_l \in \mathbb{N}^*$: the dimension of layer $l \in \{0, \ldots, L\}$, with $l = 0$ denoting the input layer;

- $h_i^{(l)} \in \mathbb{R}^{d_l}$: the vectorial representation of node $i \in \mathcal{V}$ at layer $l \in \{0, \ldots, L\}$;

- $H^{(l)} \in \mathbb{R}^{n \times d_l}$: the matrix stacking up all $h_i^{(l)}$ vectors, for $l \in \{0, \ldots, L\}$;

- $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$: in a GCN, the weight matrix from layer l to $l+1$, for $l \in \{0, \ldots, L-1\}$;

- $\sigma : \mathbb{R} \to]0,1[$: the sigmoid activation function (the notation $]0,1[$ indicates that 0 and 1 are excluded from the set);

- $\text{ReLU} : \mathbb{R} \to \mathbb{R}^+$: the ReLU activation function;

- $\mathcal{L}_{\text{MODEL}} : \mathcal{I} \to \mathbb{R}$: the loss/objective function of a model, with input elements in a set $\mathcal{I}$;

- $\delta^*(\mathcal{G}) \in \{0, \ldots, n\}$: the degeneracy number of $\mathcal{G}$;

- $\mathcal{C}_{\delta^*(\mathcal{G})} \subseteq \mathcal{C}_{\delta^*(\mathcal{G})-1} \subseteq \ldots \subseteq \mathcal{C}_0 = \mathcal{V}$: the core decomposition of $\mathcal{G}$.

Part I

General Introduction

1

Context and Scope of this Thesis

This introductory chapter provides a general overview of this PhD thesis. Firstly, we present the context and objectives of this work, which is at the intersection of graph representation learning and music recommendation. Then, we detail the scientific contributions as well as the organization of the remainder of this thesis. We also list the publications that resulted from the research conducted during these three years.

1.1 Context and Objectives

Graph structures became ubiquitous in various fields ranging from web mining to biology, due to the proliferation of data representing entities, also known as (a.k.a.) *nodes* or *vertices*, connected by links a.k.a. *edges* summarizing their relations or their interactions. For instance, web graphs depict pages of the World Wide Web as nodes, and a node i will point to a node j via an edge if any hyperlink on page i refers to page j. Social networks such as Facebook and Twitter are graphs of users, connected through "friendship" or "following" relations. Citation graphs represent scientific articles connected through citation links. Protein-protein interaction graphs efficiently summarize biological proteins and their kinetic interactions [128].

At Deezer[1], where the research presented in this thesis was conducted, graphs also naturally emerge on numerous occasions. Deezer is a French music streaming service with, at the time of writing, more than 15 million active users from 180 countries. These users can "follow" each other on the service, hence creating a large social graph. They have access to a catalog of 73 million music tracks, and the musical description of this catalog also involves graph representations.

---

[1] https://www.deezer.com/
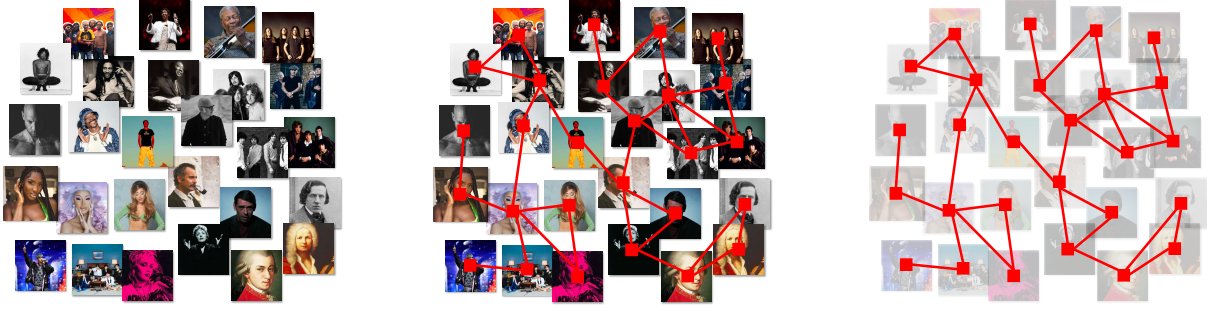
Figure 1.1: An example of a similar artists graph. While connections between artists are not naturally given (*left*), a graph of similar artists can be constructed, e.g., by connecting artists that are simultaneously listened to or liked by numerous users (*middle*). This graph can be used as an abstraction (*right*), e.g., for music recommendation.

Indeed, music tracks can be connected to artists, albums, music genres, or record labels, that can themselves be connected together through various semantic links (e.g., the song "*Kashmir*" is part of "*Physical Graffiti*", an album from the English band "*Led Zeppelin*"), generating a large knowledge graph [374] of musical entities. Previous studies emphasized the benefits of leveraging such representations for music information retrieval [297]. Besides, as we will develop in Chapter 10, Deezer possesses various graph ontologies [83, 321] that represent music genres. These are graphs of conceptually related music genres, connected through various relation-specific edges (e.g., "*rap west coast*" is a subgenre of "*hip hop*"; "*punk*" and "*electronic music*" are the origin of "*synthpunk*"). Lastly, Deezer also constructs similarity graphs from usage data, e.g., graphs of similar artists. Contrary to the aforementioned ones, these graphs are not naturally given. As illustrated in Figure 1.1, they are artificially built by connecting artists that are simultaneously listened to by numerous users on the service. They are subsequently processed in recommender systems and, as we will further develop in Chapters 8 and 9, they play a central role to help users discover new musical content on the service. The study of these similarity graphs, and of their application to industrial-level music recommendation problems, initially motivated the establishment of this PhD project.

Overall, extracting relevant information from the nodes and edges of a graph is crucial to tackle a wide range of machine learning problems [128, 348, 386, 398]. This includes the *link prediction* task [200, 225], which consists in inferring the presence of new or unobserved edges between some pairs of nodes, based on observed edges in the graph. This also includes *community detection* [31, 248], which consists in clustering nodes into similar subgroups according to a chosen similarity metric, as well as several other tasks mentioned throughout this thesis. As an illustration, Deezer often wants to predict new connections in the similar artists graph illustrated in Figure 1.1, corresponding to new artists pairs that users would enjoy listening to together, and which could be achieved by performing link prediction in the graph. Deezer would also like to learn clusters of similar artists, with the aim of providing usage-based recommendations (e.g., if users listen to several artists from a cluster, other unlisted artists from this same cluster could be recommended to them), which could be achieved through community detection.

Addressing such graph-based problems has been the objective of significant research efforts over the past decades [200, 213, 223, 225, 248]. Traditional approaches often focused on hand-engineered features. For instance, locating missing edges in graphs has been historically addressed via the construction of node similarity measures, such as the popular Adamic-Adar, Jaccard or Katz indices [225]. Nonetheless, as further detailed in Chapter 2, promising improvements were recently achieved by methods aiming to directly *learn* node representations [128, 129, 186, 188, 386] summarizing the graph under consideration. As illustrated in Figure 1.2, these *representation learning* methods compute vectorial representations of nodes

Figure 1.2: Visualization of node embedding representations for the Cora citation graph [323] of 2 708 scientific articles connected through citations, and obtained from the VGAE detailed in Chapter 7. Each point corresponds to an article, and colors denote their corresponding fields (not provided during training). This visualization was obtained by using the t-SNE method [363], permitting visualizing 32-dimensional VGAE embedding vectors in two dimensions.

in a *node embedding space* where node positions should reflect and summarize the initial graph structure. Then, usually, they assess the probability of a missing edge between two nodes, or their likelihood of belonging to the same community, by evaluating the proximity of these nodes in the space [55, 187, 370]. As we will explain throughout this thesis, these methods often learn such node embedding spaces by leveraging either random walk strategies [123, 287], matrix factorization techniques [43, 276] or graph neural networks (GNNs) [130, 188].

In particular, *graph autoencoders* (GAEs) and *variational graph autoencoders* (VGAEs) [187, 356, 370, 371] recently emerged as two powerful families of GNN-based node embedding methods. They both rely on an encoding-decoding strategy that, in a broad sense, consists of *encoding* nodes into an embedding space from which *decoding*, i.e., reconstructing the original graph should ideally be possible, by leveraging either a deterministic (for a GAE) or a probabilistic (for a VGAE) approach. The intuition behind this strategy is the following: if, starting from the embedding space, one can reconstruct a graph close to the true one, then one might conclude that embedding vectors preserve some important characteristics of the initial graph structure. Originally mainly designed for link prediction, at least in their modern formulation leveraging GNN encoders [187], the overall effectiveness of GAEs and VGAEs on this specific task as well as on several others has been widely experimentally confirmed over the past few years (see Chapter 2).

Besides their empirical performances, GAEs and VGAEs are of particular interest in the context of this PhD for two key reasons. First and foremost, they are suited for representation learning *in the absence of nodes labels*, contrary to many methods that are optimized in a supervised or semi-supervised fashion and require ground truth node labels during training [130, 188, 386]. This is desirable as, in most applications developed in this thesis, such labels will be unavailable. For

instance, we do not have access to ground truth groups of artists/nodes that should actually be recommended together. Secondly, contrary to some popular alternatives [123, 287, 368], GAEs and VGAEs can process *attributed graphs*, i.e., graphs in which each node is also described by its own feature vector. Once again, this is desirable as the graphs under consideration in our experiments will often be enriched by such descriptions. For instance, on a similar artists graph, artists could also be described by their own feature vectors providing some additional musical information, such as the music genres of each artist. GAEs and VGAEs could permit learning richer artist embedding representations by jointly capturing usage (through edges in the graph) and musical (through feature vectors) information on these artists. As we will emphasize in Chapter 8, leveraging feature vectors could also permit generalizing these methods to *inductive* settings [128], that require learning representations for new artists/nodes, unseen during training.

Nonetheless, at the beginning of this PhD, leveraging GAEs and VGAEs for industrial-level applications, e.g., for music recommendation at Deezer, was still a challenging task. Indeed, as we will explain in the next chapters, these models suffered from scalability issues [307, 308]. In 2018, in the scientific literature, experiments on GAEs and VGAEs were limited to graphs with at most a few thousand nodes and edges. The question of how to effectively scale them to larger graphs, such as those available at Deezer whose catalog includes millions of artists, albums, or music tracks, remained widely open. Besides, these models were originally designed for undirected and static graphs [187], while real-world graphs can also be directed (i.e., relations between nodes can be asymmetric) and/or dynamic (i.e., the graph structure can evolve over time). Moreover, some recent studies emphasized the relative limitations of standard GAEs and VGAEs for community detection [55, 56]. Lastly, while existing variants of these models often rely on complex neural encoders [309, 310], their actual benefit with respect to (w.r.t.) simpler strategies, that might be preferred in production environments, had never been thoroughly analyzed.

The objective of this PhD thesis is twofold. Firstly, we wish to address these limitations, with the general aim of improving GAEs and VGAEs, and of facilitating their application to real-world problems involving node embedding representations. Secondly, we aim to test and evaluate our proposed approaches on various industrial graphs extracted from the Deezer service. As detailed in Section 1.2, we will put the emphasis on graph-based music recommendation problems, often formulated as link prediction or community detection tasks.

1.2 Overview of Scientific Contributions

We now provide an outline of the remainder of this thesis, while simultaneously presenting our scientific contributions. First of all, in Chapter 2 from this Part I, we will review some key concepts related to machine learning on graphs. We will particularly focus on link prediction and community detection, which will be the two main evaluation tasks considered in experiments

throughout the next chapters. We will also provide an introduction to graph representation learning, focusing on *node* embedding methods, as well as an introduction to (variational) graph autoencoders. Then, in Part II, corresponding to Chapters 3 to 7, we will detail our contributions to improve representation learning with GAEs and VGAEs. Finally, in Part III, corresponding to Chapters 8 to 12, we will present several applications related to music recommendation at Deezer.

1.2.1 Representation Learning with Graph Autoencoders

The Part II of this thesis, focusing on contributions to node representation learning with graph autoencoders, will begin with two chapters fully dedicated to *scalability* concerns. Firstly, in Chapter 3, we will present the general framework proposed at the beginning of this PhD to scale GAEs and VGAEs to large graphs with millions of nodes and edges. This framework leverages *graph degeneracy* concepts [247] to train models only from a dense subset of nodes instead of using the entire graph. Together with a simple yet effective propagation mechanism, this approach improves scalability and training speed while, to some extent, preserving performance on downstream tasks such as link prediction and community detection. We will report an evaluation of the framework on several variants of existing GAEs and VGAEs. At the time of publication of the paper associated with this work, these experiments provided, to the best of our knowledge, the first application of these models to graphs with millions of nodes and edges.

In Chapter 4, we will subsequently introduce FastGAE, an alternative stochastic strategy to scale GAEs and VGAEs. FastGAE leverages graph mining-based sampling schemes and an effective subgraph decoding strategy to significantly lower the computational complexity of graph autoencoders, while preserving or even slightly improving their performances. We will propose an in-depth theoretical and experimental analysis of the proposed solution, showing that it behaves favorably when compared to the degeneracy framework from Chapter 3. FastGAE constitutes a faster and simpler improvement of our previous efforts.

Then, in Chapter 5, we will extend GAEs and VGAEs to *directed* graphs. As previously mentioned, their original versions were designed for undirected graphs [187]. They ignore the potential direction of the link when decoding edges from node embedding spaces. As we will argue, this is limiting for numerous real-life applications. In this chapter, we will present a method to effectively learn node embedding spaces from directed graphs using the GAE and VGAE paradigms. We will draw inspiration from physics to introduce a new *gravity-inspired decoder* scheme, able to reconstruct asymmetric relations from node embedding spaces. We will achieve competitive empirical results on three different directed link prediction tasks, for which standard GAEs and VGAEs perform poorly.

In Chapter 6, we will propose to *simplify* GAEs and VGAEs. While most existing variants of these models rely on multi-layer graph convolutional network (GCN) encoders to learn node embedding vectors, or on more sophisticated neural architectures [309, 310], we will show that

these encoders are actually unnecessarily complex for many applications. We will propose to replace them by simpler linear models w.r.t. the direct neighborhood (one-hop) adjacency matrix of the graph, involving fewer operations, fewer parameters, and no activation function. In various experiments, we will show that this simpler approach consistently reaches competitive performances w.r.t. GCN-based GAEs and VGAEs for numerous real-world graphs. This includes the Cora, Citeseer and Pubmed benchmark datasets [187, 323] commonly used to evaluate GAEs and VGAEs in the literature. Based on these results, we will question the relevance of repeatedly using these same datasets to compare complex graph autoencoders.

Lastly, in Chapter 7, we will focus on *community detection*. While GAEs and VGAEs emerged as powerful methods for link prediction, their performances were sometimes less impressive on community detection problems where, according to recent and concurring studies [55, 56, 307, 308], they can be outperformed by simpler alternatives such as the Louvain method [31]. At the beginning of this PhD project, it was still unclear to which extent one could improve community detection with a GAE or a VGAE, especially in the absence of node features. It was moreover uncertain whether one could do so while simultaneously preserving good performances on link prediction. In this chapter, we will show that jointly addressing these two tasks with high accuracy is possible. For this purpose, we will introduce and theoretically study a community-preserving message passing scheme, doping our GAE and VGAE encoders by considering both the initial graph structure and modularity-based prior communities when computing embedding spaces. We will also propose novel training and optimization strategies, including the introduction of a modularity-inspired regularizer complementing the existing reconstruction losses for joint link prediction and community detection. We will demonstrate the effectiveness of our approach, referred to as *Modularity-Aware* GAE and VGAE, through in-depth experiments.

1.2.2 Applications to Music Recommendation at Deezer

The Part III of this thesis will provide five additional chapters, focusing on applications to music recommendation. Firstly, in Chapter 8, we will adopt a graph-based approach to rank similar artists on Deezer. As illustrated in Figure 1.3, on an artist's profile page, music streaming services such as Deezer frequently recommend a ranked list of similar artists that fans also liked. However, implementing such a feature is challenging for new artists, for which usage data on the service (e.g., streams or likes) is not yet available. In this chapter, we will model this "*cold start*" *similar artists ranking* problem as a link prediction task in a directed and attributed graph, connecting artists to their most similar neighbors and incorporating side musical information as attribute vectors. Then, we will leverage the directed GAE/VGAE from Chapter 5 to learn node embedding representations from this graph, and to automatically rank the most similar neighbors of new artists using the gravity-inspired asymmetric decoder. We will empirically show the flexibility and effectiveness of this method on data extracted from Deezer's production system.
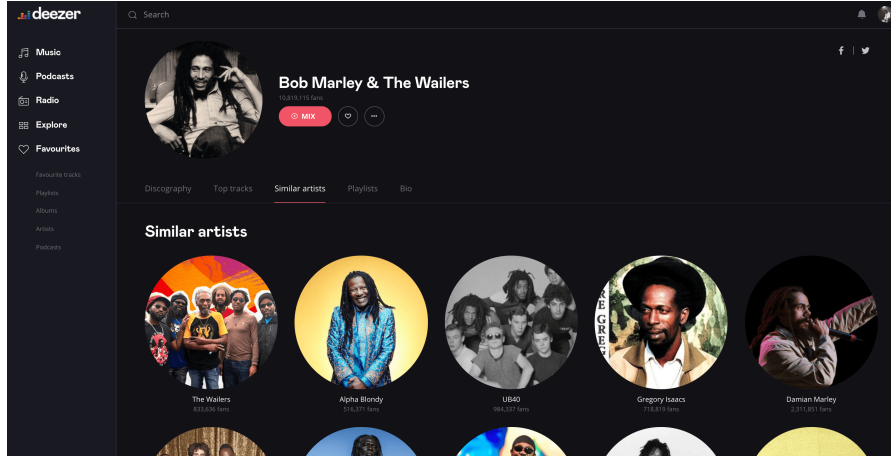
Figure 1.3: Some examples of "similar artists" recommended on the website version of Deezer.

In Chapter 9, we will give a broader overview of how Deezer has historically leveraged more general *similarity graphs* for music recommendation. While some technical details will be omitted for confidentiality reasons, we will explain how such graphs played a central role in production-facing algorithms such as Deezer's "*Flow*" feature, notably to detect communities of similar artists to recommend to users. In this direction, we will show the benefits of adopting GAEs and VGAEs, integrating advances from previous chapters, to improve community detection in large graphs of artists and albums with recommendation purposes. Moreover, we will mention future plans to extend the approach to other graphs such as graphs of music tracks, of users, or more general knowledge graphs, as well as to culture-specific graphs. Lastly, we will discuss experimental studies, done with a Master's intern at Deezer, to extend the proposed methods to *dynamic* graphs, incorporating new nodes and edges over time.

In Chapter 10, we will then focus on music genre *ontologies*. As explained in Section 1.1, these ontologies are graphs of conceptually related music genres, connected through various relation-specific edges [321]. In this chapter, we will leverage these graphs to model the music genre *perception* across (language-bound) cultures. The problem is the following: the music genre perception expressed through human annotations of artists or albums varies significantly across cultures [95]. These variations cannot be modeled as mere translations since we also need to account for cultural differences in the perception. This is an important issue for music streaming services such as Deezer, as these variations impact a wide range of tasks, ranging from localized playlist captioning to genre-driven music recommendation. In this work, we will study the feasibility of obtaining relevant cross-lingual, culture-specific music genre annotations based only on language-specific semantic representations, namely on 1) word embeddings of music genres and 2) graph ontologies [83]. Our study, focused on six languages, will show that unsupervised cross-lingual music genre annotation is feasible with high accuracy when combining both types of representations. This combination will concurrently be performed through a GAE/VGAE approach, and through an alternative simple yet effective method based on retrofitting [87].

Finally, during the three years of this PhD, several other projects less related to GAEs and VGAEs were also carried out jointly with colleagues from Polytechnique or Deezer, sometimes leading to scientific publications [26, 39, 82, 360]. For consistency reasons, we omit most of these additional projects in this thesis. We nonetheless chose to present two of them in the last Chapters 11 and 12, for two reasons. Foremost, these two works directly relate to music recommendation at Deezer, and consider some issues previously introduced in the thesis such as the cold start problem. Besides, these two projects led, not only to experiments on Deezer's data, but also to online A/B tests and subsequently to model deployment on the Deezer service. They will therefore complement the previous chapters by providing a larger overview of some of Deezer's strategies and production-facing algorithms to recommend music.

Specifically, in Chapter 11, we will present some research on *carousel* personalization. As illustrated in Figure 1.4, music streaming services frequently leverage swipeable carousels, i.e., ranked lists of items or cards (albums, artists, playlists...), to recommend personalized content to users on the homepage. However, selecting the most relevant items to display in these carousels is a challenging task, as items are numerous and users have different preferences. In this chapter, we will model carousel personalization as a contextual multi-armed bandit problem with multiple plays, cascade-based updates, and delayed batch feedback. We will empirically show the effectiveness of our framework at capturing characteristics of real-world carousels by addressing a large-scale playlist recommendation task on Deezer, through offline and online experiments.
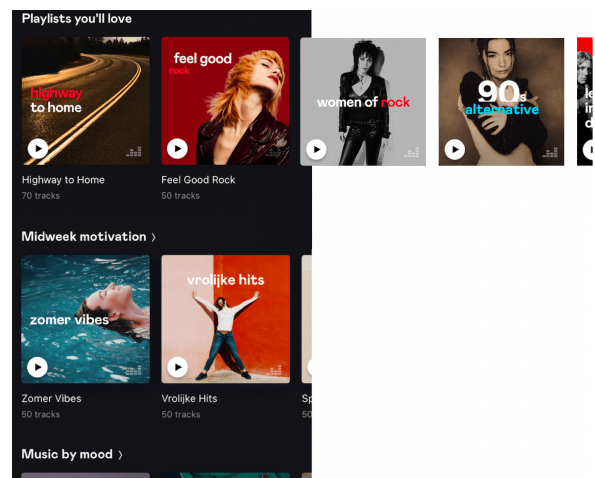
Figure 1.4: An example of a personalized swipeable carousel on the Deezer mobile app. These carousels, also referred to as sliders or shelves [252], consist of ranked lists of items or cards to recommend to users, e.g., playlists' cover images as in this figure. These playlists were created by professional curators from Deezer with the purpose of complying with a specific music genre, cultural area, or mood. A few playlists are initially displayed to each user, who can click on them or swipe on the screen to see some of the additional recommended playlists from the carousel.

Then, in Chapter 12, we will present the system recently deployed on Deezer to address the *user* cold start problem, and thus to recommend music to new users with few to no interactions with the catalog. The solution leverages a semi-personalized recommendation strategy, based on a deep neural network architecture and on a clustering of users from heterogeneous sources of information. We will show the practical impact of this system and its effectiveness at predicting the future musical preferences of cold start users on Deezer, through both offline and online experiments. These experiments will include tests on carousels from Chapter 11. We will also emphasize how this system enables us to provide more interpretable recommendations.

1.3 List of Publications

The research conducted during this PhD led to the publication of several scientific articles, that are listed thereafter. For the sake of simplicity, this list only mentions my current name G. Salha-Galvan. Nonetheless, I actually published under my old name G. Salha until 2020, and older publications are reported in their original form in the bibliography. In the remainder of this thesis, when a chapter presents some published results, the corresponding publication(s) will be specified at the beginning of the chapter.

Journals

- G. Salha-Galvan, R. Hennequin, J. B. Remy, M. Moussallam, M. Vazirgiannis, *FastGAE: Scalable Graph Autoencoders with Stochastic Subgraph Decoding.* **Neural Networks 142 (2021)**, 1 – 19, Elsevier (impact factor: 8.05).

- G. Salha-Galvan, J. F. Lutzeyer, G. Dasoulas, R. Hennequin, M. Vazirgiannis, *Modularity-Aware Graph Autoencoders for Joint Community Detection and Link Prediction.* This article is currently under review for publication in **Neural Networks**, Elsevier in 2022.

International Conferences

- G. Salha-Galvan, R. Hennequin, V. A. Tran, M. Vazirgiannis, *A Degeneracy Framework for Scalable Graph Autoencoders.* Proceedings of the 28th International Joint Conference on Artificial Intelligence (**IJCAI 2019**), 3353 – 3359.

- G. Salha-Galvan, S. Limnios, R. Hennequin, V. A. Tran, M. Vazirgiannis, *Gravity-Inspired Graph Autoencoders for Directed Link Prediction.* Proceedings of the 28th ACM International Conference on Information and Knowledge Management (**CIKM 2019**), 589 – 598.

- G. Salha-Galvan, R. Hennequin, M. Vazirgiannis, *Simple and Effective Graph Autoencoders with One-Hop Linear Models.* Proceedings of the 2020 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (**ECML - PKDD 2020**), 319 – 334.

- E. V. Epure, G. Salha-Galvan, M. Moussallam, R. Hennequin, *Modeling the Music Genre Perception across Language-Bound Cultures.* Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (**EMNLP 2020**), 4765 – 4779.

- E. V. Epure, G. Salha-Galvan, R. Hennequin *Multilingual Music Genre Embeddings for Effective Cross-Lingual Music Item Annotation.* Proceedings of the 21st International Society for Music Information Retrieval Conference (**ISMIR 2020**), 803 – 810.

- W. Bendada, <u>G. Salha-Galvan</u>, T. Bontempelli, *Carousel Personalization in Music Streaming Apps with Contextual Bandits.* Proceedings of the 14th ACM Conference on Recommender Systems (**RecSys 2020**), 420 – 425. Best short paper honorable mention.

- <u>G. Salha-Galvan</u>, R. Hennequin, B. Chapus, V. A. Tran, M. Vazirgiannis, *Cold Start Similar Artists Ranking with Gravity-Inspired Graph Autoencoders.* Proceedings of the 15th ACM Conference on Recommender Systems (**RecSys 2021**), 443 – 452. Best student paper honorable mention.

- V. A. Tran, <u>G. Salha-Galvan</u>, R. Hennequin, M. Moussallam, *Hierarchical Latent Relation Modeling for Collaborative Metric Learning.* Proceedings of the 15th ACM Conference on Recommender Systems (**RecSys 2021**), 302 – 309.

- L. Briand, <u>G. Salha-Galvan</u>, W. Bendada, M. Morlon, V. A. Tran, *A Semi-Personalized System for User Cold Start Recommendation on Music Streaming Apps.* Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (**KDD 2021**), 2601 – 2609.

Workshops and Demonstrations

- <u>G. Salha-Galvan</u>, R. Hennequin, M. Vazirgiannis, *Keep It Simple: Graph Autoencoders Without Graph Convolutional Networks.* Workshop on Graph Representation Learning, 33rd Conference on Neural Information Processing Systems (**NeurIPS 2019**).

- E. V. Epure, <u>G. Salha-Galvan</u>, F. Voituret, M. Baranes, R. Hennequin, *Muzeeglot : Annotation Multilingue et Multi-Sources d'Entités Musicales à partir de Représentations de Genres Musicaux.* Actes de la 27ème Conférence sur le Traitement Automatique des Langues Naturelles - Démonstrations (**TALN 2020**), 18 – 21.

Code and Data Releases

- Along with each of these articles, we publicly released our source code on a GitHub repository, for reproducibility and future usage of the proposed methods. Along with five of these articles, we also publicly released new datasets, that were either directly extracted from Deezer's private resources, or scraped and processed from the internet. More details are provided throughout this thesis, and on: https://github.com/deezer.

2

Background on Graph Representation Learning

This chapter formally introduces some key concepts related to graph structures and graph representation learning, mainly focusing on methods learning node embedding representations including graph neural networks and graph autoencoders. It does not aim to provide an exhaustive review of the fast-growing graph representation learning field (which could be the objective of an entire book [128]), but rather to present the important notions that will subsequently be used in the remainder of this PhD thesis.

2.1 Introduction

We begin this chapter with some general concepts and definitions related to graph structures.

2.1.1 Graph Structures: Generalities and Key Definitions

In its most general formulation, a *graph* is defined as follows.

Definition 2.1. A *graph* is an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprising:

- a set of *nodes* or *vertices* $\mathcal{V}$. For the sake of simplicity, we assume $\mathcal{V} = \{1, 2, \ldots, |\mathcal{V}|\}$;

- a set of *edges* or *links* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ connecting these nodes. In essence, they summarize the relations or interactions between nodes. We denote an edge going from a node $i \in \mathcal{V}$ to a node $j \in \mathcal{V}$ as $(i, j) \in \mathcal{E}$. Each edge is equipped with a positive *weight* w_{ij}, normalized to lie in the $[0, 1]$ set without loss of generality (w.l.o.g.), and indicating the intensity of the link from i to j. For *unweighted* graphs, we implicitly set $w_{ij} = 1$ for all $(i, j) \in \mathcal{E}$.

2.1. Introduction

In this thesis, we denote by n the number of the nodes in the graph, i.e., $n = |\mathcal{V}|$, and by m the number of edges in the graph, i.e., $m = |\mathcal{E}|$.

Remark: graph or network?   In this remark, we discuss our choice to refer to the mathematical object defined in Definition 2.1 as a "*graph*" and not as a "*network*", contrary to several references [31, 34, 240, 267, 323]. Some research studies explicitly present the two terms as synonyms [4, 49]. Hamilton [128] makes a distinction between their usage. In his book, "*graph*" refers to the abstract data structure on which we focus on in this thesis, while "*network*" describes real-world instances of this structure, e.g., social networks or computer networks. He also explains that "*network*" has been historically favored in the data mining and network science communities (studying such real-world data), while "*graph*" is prevalent in the graph theory (studying theoretical properties of the mathematical abstraction) and in the machine learning communities. We choose to employ "*graph*" as well in this thesis for consistency with recent advances in machine learning, but also to avoid terminological clashes with the term "*neural network*", which will be widely used in our work.

In this thesis, we will often represent a graph $\mathcal{G}$ by its *adjacency matrix*, defined as follows.

Definition 2.2. The *adjacency matrix* $A \in [0,1]^{n \times n}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as:

$$A_{ij} = \begin{cases} w_{ij}, & \text{if } (i,j) \in \mathcal{E}, \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

for all $(i,j) \in \{1, \ldots, n\}^2$.

This matrix can represent graphs with various properties, e.g., those illustrated in Figure 2.1:

- if A is a *binary* matrix, i.e., $A \in \{0,1\}^{n \times n}$, then $\mathcal{G}$ is an *unweighted* graph;

- if $A_{ii} > 0$ for any $i \in \{1, \ldots, n\}$, then $\mathcal{G}$ includes a *self-loop* connecting the node i to itself;

- if A is a *symmetric* matrix, i.e., $A_{ij} = A_{ji}$ for all $(i,j) \in \{1, \ldots, n\}^2$, then $\mathcal{G}$ is an *undirected* graph. It corresponds to cases where all edges are bidirectional. For instance, the Facebook social network is an undirected graph (if the user i is a "friend" of the user j, then j is also a "friend" of i) while the Twitter social network is *directed* (connections can be asymmetric, i.e., i can "follow" j while j does not "follow" i back);

- if $\mathcal{V}$ can be partitioned into two disjoint sets $\mathcal{V}_1$ and $\mathcal{V}_2$ with $A_{ij} > 0 \implies i \in \mathcal{V}_1$ and $j \in \mathcal{V}_2$, then $\mathcal{G}$ is a *bipartite* graph where any two nodes from the same partition set are unconnected. For instance, if $\mathcal{V}_1$ denotes a set of Deezer users, and $\mathcal{V}_2$ denotes a set of music tracks, then an edge (i,j) in such a graph could represent the fact that the user i likes the track j.

| (a) Undirected | (b) Directed | (c) Weighted | (d) Bipartite |
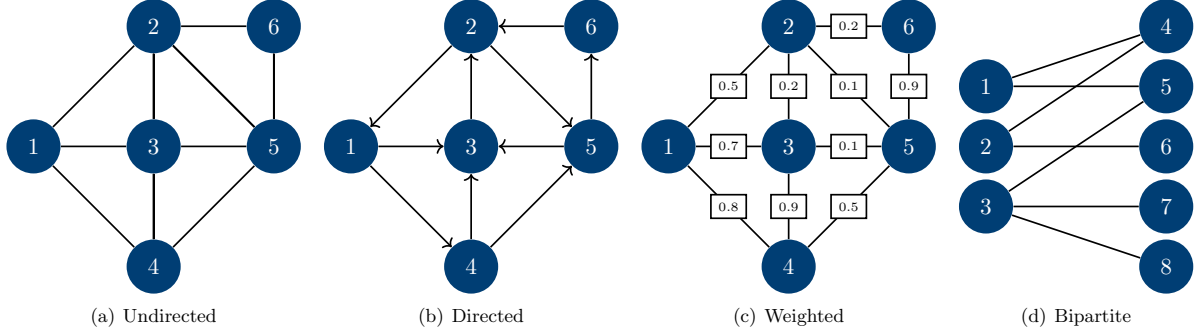
Figure 2.1: Examples of some different types of graphs.

Definitions 2.1 and 2.2 can also be straightforwardly extended to represent more complex structures. For instance, in Chapter 9, we will consider *dynamic graphs*, where nodes and edges can appear or disappear over time. They are often summarized by T discrete snapshots of standard graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$, $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, ..., $\mathcal{G}_T = (\mathcal{V}_T, \mathcal{E}_T)$ depicting the structural evolution over time [179], and therefore by T adjacency matrices A_1, A_2, ..., A_T. Moreover, in Chapter 10, we will consider structures of the form $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_p)$ where nodes can be connected through edges of p different *natures* [83]. They can be summarized as well by p adjacency matrices A_1, A_2, ..., A_p, each of them indicating edges of a particular nature.

We also use A to introduce the concept of neighboring nodes, which will be useful in our work.

Definition 2.3. The *incoming neighbors* $\mathcal{N}_{\text{in}}(i)$ and *outcoming neighbors* $\mathcal{N}_{\text{out}}(i)$ of a node $i \in \mathcal{V}$ from a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with adjacency matrix A, correspond to nodes "pointing towards i" and "towards which i points" through an edge, respectively. More formally:

$$\mathcal{N}_{\text{in}}(i) = \{j \in \mathcal{V}; A_{ji} > 0\} \text{ and } \mathcal{N}_{\text{out}}(i) = \{j \in \mathcal{V}; A_{ij} > 0\}. \tag{2.2}$$

For undirected graphs (where $A_{ij} = A_{ji}$), we simply denote by $\mathcal{N}(i)$ the *neighborhood* of the node i, defined as $\mathcal{N}(i) = \mathcal{N}_{\text{in}}(i) = \mathcal{N}_{\text{out}}(i)$. This same set will occasionally be referred to as the *direct neighborhood* or the *one-hop neighborhood* of the node i, in chapters where more general definitions of neighborhoods will concurrently be introduced.

In addition to the adjacency matrix A, we introduce several other matrices also frequently used for graph analysis.

Definition 2.4. The *in-degree matrix* D_{in} and *out-degree matrix* D_{out} of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with adjacency matrix A, are the $n \times n$ diagonal matrices defined as:

$$D_{\text{in}} = \text{diag}(A^T \mathbf{1}_n) \text{ and } D_{\text{out}} = \text{diag}(A \mathbf{1}_n), \tag{2.3}$$

where $\mathbf{1}_n$ denotes the n-dimensional vector containing n entries all equal to 1. For undirected graphs (where $A^T = A$) we simply denote by D the *degree matrix* defined as $D = D_{\text{in}} = D_{\text{out}}$.

In a nutshell, the i-th diagonal element of D_{in} (respectively, of D_{out}) corresponds to the sum of weights of all edges pointing towards node i (resp., going out of node i) in the graph under consideration. For unweighted graphs, this coincides with the number of incoming (resp., outcoming) neighbors of i. In the case of an undirected degree matrix D, diagonal elements equal the number of neighbors of i.

From Definition 2.4, we subsequently introduce the *symmetric normalization* of A.

Definition 2.5. The *symmetric normalization* of the adjacency matrix A of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with degree matrix D is:

$$\tilde{A} = (D + I_n)^{-\frac{1}{2}}(A + I_n)(D + I_n)^{-\frac{1}{2}}, \tag{2.4}$$

where I_n denotes the $n \times n$ identity matrix.

In $\tilde{A}$, each element (i, j) of the original adjacency matrix is re-weighted according to the degrees of nodes i and j. Specifically, we have $\tilde{A}_{ij} = \frac{A_{ij}}{\sqrt{(D_{ii}+1)(D_{jj}+1)}}$ for all $(i, j) \in \mathcal{V} \times \mathcal{V}$ with $i \neq j$, and $\tilde{A}_{ii} = \frac{A_{ii}+1}{D_{ii}+1}$ $(= \frac{1}{D_{ii}+1}$ in the absence of self-loop) for all $i \in \mathcal{V}$. $\tilde{A}$ will often be preferred to A in the *graph neural networks* we will introduce thereafter, for reasons related to information propagation that will be detailed along with the presentation of these methods (see Section 2.3 and 2.4). Other normalizations of A, such as variants involving D_{in} or D_{out} for directed graphs, will also occasionally be mentioned throughout this thesis (see, e.g., Chapter 5). Besides, other transformations of the adjacency matrix with useful properties have also been proposed to represent graphs. This includes *Laplacian matrices* [255, 368], defined as follows.

Definition 2.6. The *unnormalized Laplacian matrix* of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with adjacency matrix A and degree matrix D, is defined as:

$$L_{\text{un}} = D - A. \tag{2.5}$$

In addition, several matrices are sometimes referred to as normalized Laplacian matrices [368]. This includes the *random-walk normalized Laplacian matrix*, defined as:

$$L_{\text{rw}} = D^{-1}L_{\text{un}}, \tag{2.6}$$

as well as the *symmetrically normalized Laplacian matrix*, defined as:

$$L_{\text{sym}} = D^{-1/2}L_{\text{un}}D^{-1/2}. \tag{2.7}$$

Laplacian matrices are central to *spectral graph theory* [58] and to numerous related methods such as *Laplacian eigenmaps* and *spectral clustering* [270, 342, 368] that we will introduce

in Section 2.2 and consider in several experiments throughout this thesis. We refer to Von Luxburg [368] and Lutzeyer [240] for an in-depth presentation of the differences, the applications, and the mathematical properties of these Laplacian matrices, ranging from their symmetry to their positive semi-definiteness and their link to the number of connected components in $\mathcal{G}$.

Lastly, some graphs mentioned in our work will be *attributed graphs*. Each node will also be described by its own *feature* vector, a.k.a. *attribute* vector or *descriptive* vector [128]. We will encapsulate these descriptions in a *node feature matrix X*.

Definition 2.7. The *node feature matrix X* of an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which each node i is also described by its own vector $x_i \in \mathbb{R}^f$, is the $n \times f$ matrix stacking up all x_i vectors:

$$X = [x_1, \ldots, x_n]^T, \tag{2.8}$$

i.e., the i-th row of X corresponds to the feature vector x_i of node i.

2.1.2 Machine Learning on Graphs

A wide range of machine learning problems involve graphs, and require extracting relevant information from the nodes and edges of such structures [128, 129, 348, 386, 398]. In the following, we present some of these problems. We primarily focus on *link prediction* and on *community detection*, which will be the two evaluation tasks mainly considered in our experiments, as they are closely related to our applications at Deezer. We refer to several surveys [97, 128, 200, 248, 328, 386] for a broader overview of graph-based machine learning problems.

Link Prediction   The *link prediction* task [200, 225] consists in inferring the presence of new or unobserved edges between some pairs of nodes, based on the observed edges in the graph. Over the past decades, this problem initiated significant research efforts from the scientific community, with various real-world applications ranging from recommending new "friends" in social networks to predicting interactions between proteins [49, 187, 200, 225, 226, 328, 392, 402]. More formally, we define a link prediction model as follows.

Definition 2.8. Let us consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which some edges would be "missing", e.g., masked or yet to appear. A *link prediction* model for such a graph is a function:

$$p : \mathcal{V} \times \mathcal{V} \setminus \mathcal{E} \to [0, 1]. \tag{2.9}$$

For a given pair (i, j) of unconnected nodes, it returns an estimated probability $p(i, j)$ of a missing edge connecting i to j in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ under consideration.

Definition 2.8 is very general. For instance, p could correspond to a node similarity measure

directly computed from the graph, e.g., to the proportion of common edges between two nodes[1]. Alternatively, p could also correspond to a more sophisticated model with trainable parameters, e.g., to a neural network [200]. To assess the performance of such a model, researchers often adopt an evaluation methodology [187, 279, 402] consisting in:

- firstly, training this model on an (artifically) incomplete version of a graph, for which only a certain percentage of randomly sampled edges (for instance, 85% in [187]) are visible;

- then, constructing validation and test sets gathering:
    - node pairs corresponding to missing edges (5% and 10%, respectively, in [187]);
    - the same number of randomly picked unconnected node pairs in the graph;

- finally, evaluating the model's ability to distinguish edges from non-edges in these sets, using the model trained on the incomplete graph.

While all node pairs in such validation and test sets are observed to be unconnected, half of them actually correspond to missing edges from the original graph. Under this formulation, link prediction acts as a *binary classification* task, assessing to which extent the model correctly locates these missing edges despite their absence during training.

Researchers evaluate model performance on this link prediction task through binary classification metrics [148], such as the popular *Area Under the ROC Curve* (AUC) [88, 251] and *Average Precision* (AP) [301] scores, that we will adopt and discuss as well throughout our work[2].

Community Detection Another fundamental problem in graph-based machine learning, community detection consists in identifying $K < n$ clusters a.k.a. *communities* of nodes that, in some sense, are more similar to each other than to the other nodes [97, 314]. As link prediction, this problem initiated significant research efforts [77, 97, 232, 248, 280, 290, 314, 370]. Community detection has numerous real-world applications. This includes the segmentation of websites in a web graph according to thematic categories [154, 248], as well as the detection of densely connected subgroups of users (actual "*communities*") in online social networks [77, 280, 290]. More formally, we define a community detection model as follows.

Definition 2.9. Let us consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Denoting by $\mathcal{P}_K(\mathcal{V})$ the set of partitions of $\mathcal{V}$ of cardinality $K \leq n$, a *community detection* model is a function:

$$c : \mathcal{G} \rightarrow \mathcal{P}_K(\mathcal{V}), \tag{2.10}$$

---

[1]Such node similarity measures usually rely on homophily assumptions. This term describes the tendency of nodes to connect to "similar" nodes in the graph, which is observed in numerous real-world applications [200].

[2]We will use the implementations provided in the *scikit-learn* Python library for machine learning [284] and respectively described here: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html and here: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html.

Given the observed graph, it returns a partition of the node set $\mathcal{V}$ into K sets, denoted as follows:

$$C_1 \subseteq \mathcal{V}, \ldots, C_K \subseteq \mathcal{V}. \tag{2.11}$$

We denote $n_k = |C_k|$, for $k \in \{1, \ldots, K\}$.

In the scientific literature, the quality of such a partition is usually assessed through some predefined similarity metrics, e.g., unsupervised density-based metrics computed from the intra- and inter-cluster edge density [248], or scores such as the *Adjusted Mutual Information* (AMI) [366] and the *Adjusted Rand Index* (ARI) [159] that compare the partition to some ground truth node labels (when such labels are available) hidden during training. We will adopt and discuss these metrics as well throughout our work[3]. Figure 2.2 provides a illustrative example of a graph generated from a stochastic block model [2], where nodes are partitioned in $K = 4$ communities.
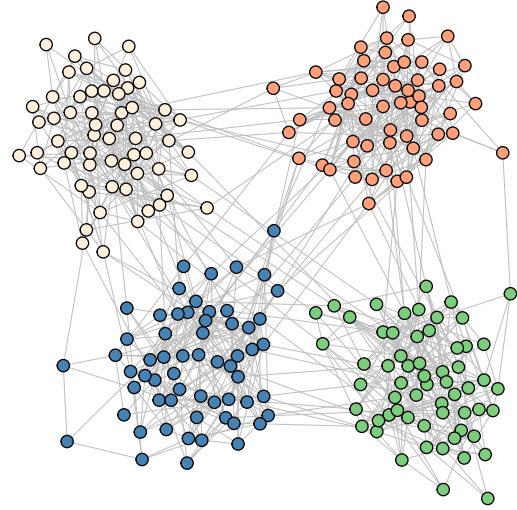
Figure 2.2: A graph with four ground truth communities, denoted by node colors. This synthetic graph was obtained through a stochastic block model [2], which generates community-based random graphs.

As is the case for link prediction, Definition 2.9 is voluntarily very general. Specific examples of community detection methods will be presented in the next sections and chapters. Also, while this definition implicitly assumes that K is known and fixed, we emphasize that some community detection methods instead involve an automatic selection of the appropriate number of communities to detect [31]. Besides, while the above partitioning implies that communities are *disjoint*, several overlapping community detection methods were also proposed in the scientific literature [145, 178, 405]. They are out of the scope of this thesis.

Other Tasks   Several other graph-based machine learning problems have also been introduced and widely studied over the last decades. While they will be less or not discussed in this thesis, we nevertheless mention several of them in this paragraph along with some indicative references. Examples of such other problems include: *influence maximization*, which consists in identifying the most "influent" nodes in a graph, e.g., the ones that will maximize the spread of a rumor in a social network according to some information diffusion process [44, 112, 180, 223, 312]; *graph-level clustering*, where the goal is to partition a set of several entire graphs, into groups of

---

[3]Again, we will use the implementations provided in *scikit-learn* [284] and respectively described here: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html   and here: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html.

structurally similar ones [65, 78, 129, 273]; *graph generation*, consisting in generating new graphs verifying some desirable properties, e.g., new biologically plausible molecules [169, 224, 233, 243, 336]; and several other problems ranging from supervised/semi-supervised *node classification* [66, 130, 188, 364, 383] to *edge classification* [7, 46, 207].

2.2 Learning Node Embedding Representations

To address these machine learning problems, significant efforts were recently devoted to the development of *representation learning* methods. In the remainder of this Chapter 2, we propose an introduction to this learning paradigm. For consistency with the content of this thesis, we focus on methods learning vectorial representations of *nodes* in an embedding space, and omit other existing methods, e.g., those designed for edge-level or graph-level representation learning [245, 265, 299, 365]. Moreover, at times, we partly draw inspiration from Hamilton's recent book on graph representation learning [128] to organize and present some key concepts. We refer to this book for more details and references on this topic.

Specifically, in this Section 2.2, we present the general objectives of representation learning on graphs. We also mention several popular methods, referred to as "*shallow*" by Hamilton [128], to learn node embedding spaces. In Section 2.3, we will consider representation learning with *graph neural networks* (GNNs). Finally, in Section 2.4, we will focus on the two GNN-based methods at the center of this thesis: *graph autoencoders* (GAEs) and *variational graph autoencoders* (VGAEs).

2.2.1 From Graph Structures to Vectorial Representations

Traditional approaches to tackle machine learning problems such as link prediction and community detection often directly operate on the graph structure under consideration. For instance, predicting the most likely locations of missing links has been historically addressed by the construction of hand-engineered *node similarity measures* [128, 200, 225] derived from the observed connections. Examples of such measures include the (previously cited) proportion of common neighbors between nodes in the graph, as well as several refined alternatives such as the popular Adamic-Adar, Jaccard, or Katz indices [225]. Besides, as of today, one of the most popular approaches for community detection remains the *Louvain* algorithm [31], which also directly operates on the graph. This greedy method clusters nodes by iteratively maximizing a graph-based measure referred to as the *modularity* [267, 314]. It will compare the observed density of connections in communities to the expected density in a configuration model graph, with equal degree distribution but allocating edges randomly without any specified community structure (we provide a more complete presentation of the Louvain method in Chapter 7).
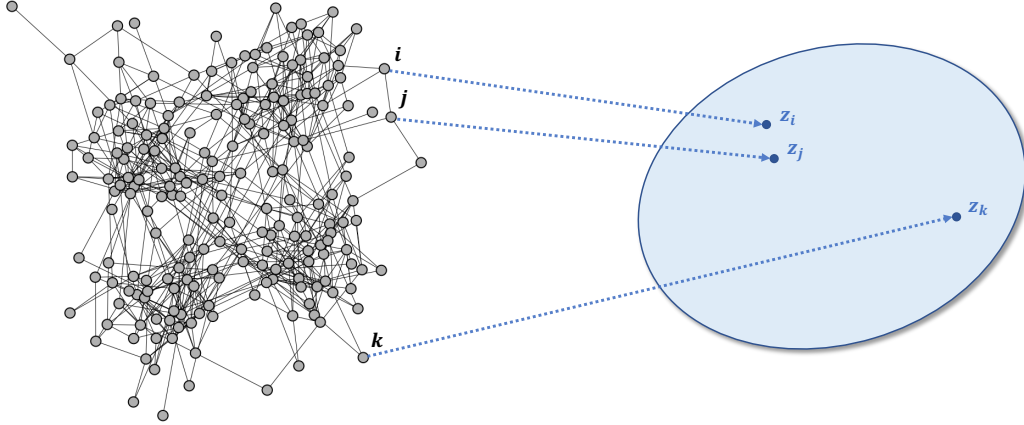
Figure 2.3: Illustration of the node embedding paradigm. The objective is to encode nodes from a graph (*left*) as vectors in a low-dimensional vector space (*right*), where nodes with "structural proximity" in the graph should be close. In essence, one should be able to subsequently leverage these vectors for downstream applications, such as link prediction (e.g., by assessing the likelihood of a link from i to j by computing $z_i^T z_j$ or $\|z_i - z_j\|_2$) or community detection (e.g., by running a k-means algorithm to cluster all z_i vectors).

Nonetheless, promising improvements on link prediction, community detection, and various other tasks were recently achieved by methods adopting another strategy. Instead of directly operating on the graph structure itself, these methods aim to *learn* node representations summarizing the graph under consideration [123, 128, 129, 186, 188, 287, 386]. As illustrated in Figure 2.3, the objective of these *representation learning* methods is to encode nodes as vectors in a low-dimensional vector space, called a *node embedding* space. In such a space, nodes with "structural proximity" in the graph (which could have various meanings [76]) should be close. More formally, each node $i \in \mathcal{V}$ from a graph $\mathcal{G}$ will be associated with an embedding vector $z_i \in \mathbb{R}^d$, where $d \ll n$ denotes the dimension of the node embedding space. In the following, we will often encapsulate these embedding vectors in the *node embedding matrix* Z defined as follows.

Definition 2.10. The *node embedding matrix* Z of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which each node $i \in \mathcal{V}$ is associated with an embedding vector $z_i \in \mathbb{R}^d$, is the $n \times d$ matrix stacking up all z_i vectors:

$$Z = [z_1, \ldots, z_n]^T, \tag{2.12}$$

i.e., the i-th row of Z corresponds to the embedding vector z_i of node i.

Over the last years, various methods, including those described thereafter in this chapter, have been proposed to learn such z_i vectors. Authors of these methods successfully optimized and leveraged these low-dimensional vectors, which are arguably easier to handle than a complex graph structure, for various downstream applications [128, 386]. For instance, some methods manage to directly assess the probability of a missing edge between two nodes by evaluating the proximity of these nodes in the embedding space, e.g., through pairwise Euclidean distances or inner products [55, 187, 370]. Similarly, in the presence of node embedding vectors, several studies reframed community detection as the standard problem consisting in clustering n vectors

in a d-dimensional vector space into K groups [232, 244]. Such a problem can then be solved by one of the numerous clustering methods such as the popular k-means algorithm [13, 244].

Based on the promising results of such a paradigm [128, 200, 232, 386, 398], the graph representation learning field has grown at a fast pace over the past decade [128], notably the sub-field involving research on *graph neural networks* (see Sections 2.3 and 2.4). In the last few years, international conferences on machine learning and data mining often received hundreds of submissions from graph representation learning articles [162, 163], and regularly organized dedicated workshops on related topics[4]. Some recent advances were also already successfully transposed to industrial-level applications ranging from product recommendation to malicious behavior detection, including on online services such as Alibaba [373], Pinterest [394], and Twitter [40].

2.2.2 Node Embeddings from Matrix Factorization

In the remainder of this chapter, we review several popular approaches aiming to learn node embedding vectors. Firstly, an important part of the existing literature involves *factorization* techniques.

Laplacian Eigenmaps and Spectral Clustering In particular, some seminal approaches make use of the *eigendecomposition* of Laplacian matrices introduced in Defintion 2.6 [25, 240, 270, 368]. One can show that these matrices have n non-negative eigenvalues $0 = \lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_1$, and that the geometric multiplicity of the 0 eigenvalue corresponds to the number of *connected components* in the graph. Furthermore, the corresponding *eigenvectors*[5] are indicators of nodes belonging to each of these components [368]. As a consequence, these eigenvectors can be used to cluster nodes according to their connected component membership. More interestingly, Laplacian eigenvectors are also useful to cluster nodes into K communities in a connected graph. In this direction, a fundamental approach referred to as *spectral clustering* [270, 330] consists in:

- computing the K eigenvectors corresponding to the K smallest eigenvalues of L_{un} (or L_{rw} or L_{sym}), and subsequently form the matrix $Z \in \mathbb{R}^{n \times K}$, in which the K columns correspond to these K n-dimensional eigenvectors;

- represent each node $i \in \mathcal{V}$ by the i-th row of Z, denoted z_i;

- run a k-means algorithm [244] to cluster the z_i vectors into communities.

---

[4]This includes the NeurIPS 2019 workshop on *Graph Representation Learning*: https://nips.cc/Conferences/2019/ScheduleMultitrack?event=13172; the ICML 2020 workshop on *Graph Representation Learning and Beyond*: https://icml.cc/Conferences/2020/ScheduleMultitrack?event=5715; and the KDD 2021 workshop on *Deep Learning on Graphs: Methods and Applications*: https://kdd.org/kdd2021/workshops.

[5]The eigenvector $u_i \in \mathbb{R}^n$ associated with an eigenvalue λ_i of L_{un} verifies: $L_{\text{un}} u_i = \lambda_i u_i$.
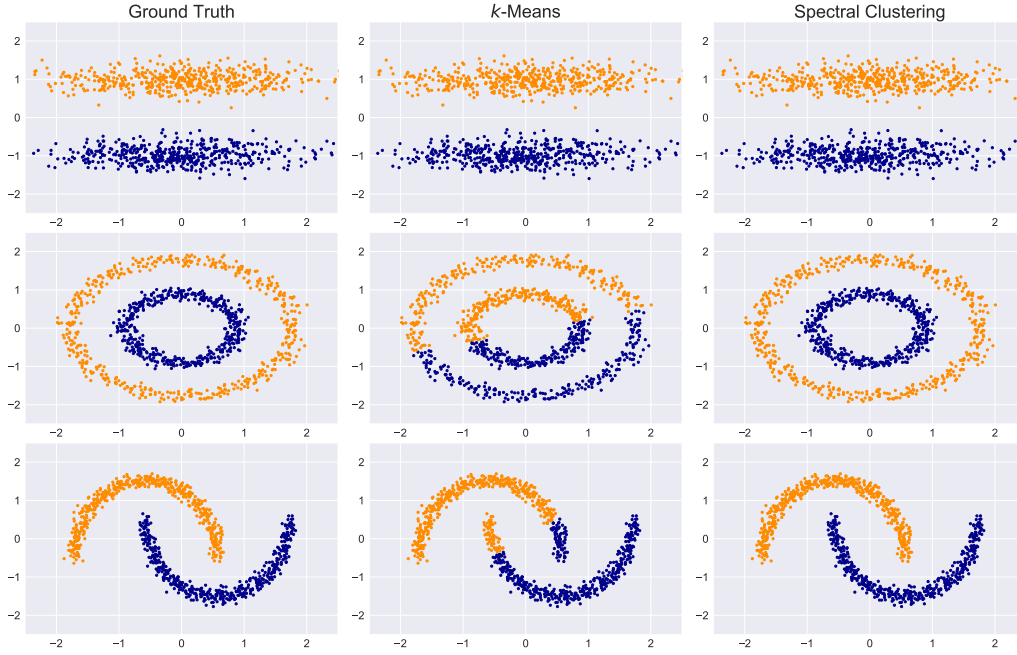
Figure 2.4: A comparison of k-means and spectral clustering. *Left*: three synthetic datasets of 1000 two-dimensional data points, clustered in two ground truth communities displayed in orange and in blue, respectively. *Middle*: the communities retrieved by running k-means algorithms in these two-dimensional spaces. *Right*: the communities retrieved by instead 1) creating a graph connecting data points (acting as nodes) to their ten nearest neighbors, and 2) applying spectral clustering on graphs. In 2/3 cases, a standard k-means fails to properly identify communities. This approach directly relies on Euclidean distances in the input space and extracts "compact" communities, which is relevant for dataset n°1 but not for datasets n°2 and n°3. On the contrary, the graph-based spectral clustering approach extracts and leverages the "connectivity" between data points which, while being more computationally expensive, permits identifying ground truth communities on all three datasets.

One can prove that communities identified via spectral clustering correspond to those obtained from the *relaxation* of NP-hard graph partitioning optimization problems, consisting in minimizing metrics that explicitly evaluate the quality of a node partition in a graph (specifically, a *RatioCut* or *NCut* metric depending on Laplacian matrices [368]). Spectral clustering can also interestingly be studied under the lens of perturbation theory [368], bringing additional theoretical justifications for such an approach. Overall, as illustrated in Figure 2.4, leveraging point/node-level connections permits identifying cluster structures where, in essence, "*connectivity*" is more relevant than "*compactness*" and where a standard k-means would therefore fail. Also, while a naive eigendecomposition of a Laplacian matrix incurs a cubic computational complexity $O(n^3)$ [128], several approximate methods have been studied to overcome such a cost [35, 68, 98]. Moreover, besides clustering, the above eigendecomposition procedure has also been interpreted as an effective way to project nodes in a low-dimensional node embedding space preserving connection information from the initial graph structure. The z_i vectors from the above matrix Z correspond to *embedding vectors* (hence the same notations as Definition 2.10) and such an approach is often referred to as *Laplacian eigenmaps* in the literature [25, 128].

Inner product methods In parallel, several other studies introduced factorization-based methods that rather leverage the *inner product* $z_i^T z_j$, between two node embedding vectors z_i and z_j, as an indicator of the structural proximity of nodes i and j in the graph. From a matrix perspective, these methods aim to learn an $n \times d$ node embedding matrix Z such as:

$$f(A) \approx ZZ^T, \tag{2.13}$$

for some pre-defined function $f : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$. Examples of such methods include the *distributed graph factorization* model [8] as well as the *GraRep* [43] and *HOPE* [276] methods. As explained by Hamilton [128], these methods primarily differ in how f is defined. While Ahmed et al. [8] directly leverage the adjacency matrix, i.e., $f(A) = A$, authors of GraRep and HOPE exploit more general matrices, based on powers of A [43] or on neighborhood overlap measures [276]. They aim to minimize loss functions corresponding or proportional to:

$$\mathcal{L} \approx \|f(A) - ZZ^T\|_2^2, \tag{2.14}$$

which can be achieved through matrix factorization algorithms such as *Singular Value Decomposition* (SVD) [113, 345]. We refer to Klema and Laub [191] for a detailed introduction to SVD and its applications.

2.2.3 Node Embeddings from Random Walks

We now turn to another family of embedding methods, relying on *stochastic* strategies. In such methods, nodes end up with similar embedding vectors when they frequently co-occur on short *random walks* computed over the graph [287]. As illustrated in Figure 2.5, a random walk is a list of nodes of length $T > 1$. The first node is selected from $\mathcal{V}$, and then the i-th node for $i \in \{2, ..., T\}$ is subsequently sampled from the neighbors of the $(i - 1)$-th node, either uniformly at random or following some pre-defined sampling technique [158].
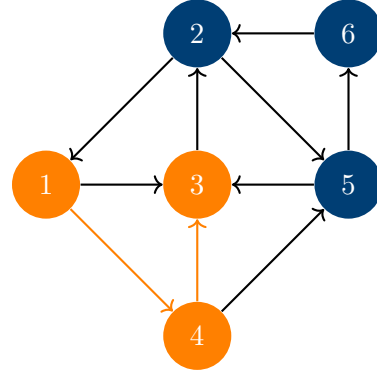
Figure 2.5: An example of a random walk of length $T = 3$, on the small directed graph from Figure 2.1(b). This random walk, colored in orange, starts at node n°1, then "visits" node n°4 and ends at node n°3.

Random walk-based methods often explicitly draw inspiration from *word2vec*-like models [111, 258] from natural language processing (NLP). These shallow neural network models learn word embedding representations, ensuring that words appearing in the same contexts, e.g., the same sentences, end up with close word embedding vectors. Here, nodes and random walks act as the counterparts of words and sentences, respectively.

More formally, and albeit under different formulations [123, 128, 158, 287], these node embedding methods usually encode nodes as z_i embedding vectors that should approximately verify:

$$p_T(j|i) \approx \frac{e^{z_i^T z_j}}{\sum\limits_{k \in \mathcal{V}} e^{z_i^T z_k}}, \tag{2.15}$$

where $p_T(j|i)$ denotes the probability that the node $j \in \mathcal{V}$ appears on a random walk of length T starting from node $i \in \mathcal{V}$. Examples of such methods include the popular *DeepWalk* [287] and *node2vec* [123] models. Precisely, DeepWalk and node2vec adopt different strategies to learn z_i vectors from random walks, either based on hierarchical softmax computations involving a fast binary-tree structure [287] or a noise contrastive learning approach with negative sampling [123] (we refer to Hamilton [128] as well as to original research articles on these methods [123, 287] for technical details, that we omit in this brief overview). Moreover, node2vec includes two additional hyperparameters, the "*return*" parameter p and the "*in-out*" parameter q, providing more flexibility when sampling random walks. They permit interpolating between random walks more akin to a breadth-first search or a depth-first search [62] over the graph. DeepWalk and node2vec have been explicitly presented as *scalable* methods in their original articles, backed by experimental evaluations on graphs with up to a few million nodes or edges [123, 287], observing linear increases in running times w.r.t. n in the case of node2vec [123].

Over the last few years, several other random walk-based methods have been proposed, e.g., with different architectures or sampling schemes. This includes *dynnode2vec* [246], *struct2vec* [300] and *Walklets* [288]. As explained by Hamilton [128], the popular *LINE* method [351] is also often presented among random walk approaches. While it does not explicitly draw random walks, LINE nonetheless shares strong conceptual motivations with DeepWalk and node2vec (see [128]). Moreover, related techniques have also been extended to multi-relational data and knowledge graphs [374]. More recently, Qiu et al. [294] proved that methods such as DeepWalk, LINE, and node2vec implicitly approximate and factorize some matrices, for which they derived closed forms. Therefore, to some extent, they unified random walk-based learning with the matrix factorization framework from Section 2.2.2.

2.2.4 On the Limitations of these Methods

The random walks and matrix factorization methods we presented in this Section 2.2 are referred to as "*shallow*" by Hamilton [128]. In his book, this term encompasses node embedding methods consisting in a simple embedding lookup based on a node's ID, and where an encoder directly optimizes a unique embedding vector for each node. While these methods led to promising applications over the past years [128, 129, 158], they nonetheless suffer from several limitations:

- firstly, the encoding function, which maps nodes into the embedding space, does not share

parameters between nodes. Hamilton [128] argues that this is inefficient from a statistical point of view, and that parameter sharing could permit learning representations capturing properties of the original graph structure more efficiently;

- besides, while attributed graphs are ubiquitous, these methods do not leverage node features/attributes. Such additional information, summarized in the matrix X from Definion 2.7, could however be relevant and useful in the learning process;

- lastly, these embedding methods are *transductive* [130]. By design, they only learn representations of nodes available during training. Without additional optimization steps, they usually can not provide representations for new nodes. This prevents applying such methods to *inductive* settings, that specifically require such generalization to new nodes after training. Such a generalization would be desirable for several applications mentioned throughout this thesis.

In the following sections, we review alternative node embedding methods that address these limitations. They rely on graph neural networks, which emerged as a popular formalism in the last years [386]. While random walk and matrix factorization methods will still be mentioned in the following chapters, as relevant baselines for various experimental analyses, graph neural networks will be at the center of our work, especially through the lens of graph autoencoders.

2.3 Representation Learning with Graph Neural Networks

We now turn to *graph neural networks* (GNNs), a general framework to extend deep neural networks [117, 205] to graph structures. As an introduction, it is worth noting that a straightforward use of standard *deep learning* models on graphs is usually a bad idea. For instance, by directly training a *convolutional neural network* (CNN) [101, 206] on an adjacency matrix A treated as a two-dimensional regular grid, one would aggregate information from a node i with information from nodes corresponding to rows/columns $i-1$ and $i+1$ in convolutional layers. While such an aggregation is relevant for neighboring pixels in an image, it is inappropriate for graphs as the node ordering in A is usually *arbitrary*. The node associated with the $(i-1)$-th row/column of A will not necessarily be connected or even "close" to the one associated with the i-th row/column. Alternatively, one could also want to learn node embedding vectors by directly processing a flattened version of A using a *multi-layer perceptron* (MLP) [303]. Again, in such an approach, the resulting node embedding representations would depend on the arbitrary ordering of nodes in A [128]. On the contrary, researchers will often favor neural network models that are *permutation equivariant*, meaning that a permutation of the node ordering would permute the output representations in a consistent way [181].

2.3.1 Neural Message Passing and Graph Neural Network (GNN) Models

Although numerous GNN models have been proposed and theoretically motivated over the last years, sometimes under different formalisms, most of them actually leverage a *neural message passing* process in which vector "*messages*" are iteratively exchanged between nodes and updated [107, 128, 274, 388]. Comprehensively, a GNN model usually takes as input:

- an adjacency matrix A, or any variant such as one of those presented in Section 2.1;

- some *input vectors* providing an initial representation for each node $i \in \mathcal{V}$. In numerous applications, graphs will be attributed and these vectors will correspond to *node features* $x_i \in \mathbb{R}^f$, summarized in the $n \times f$ matrix X from Definition 2.7. In the absence of such features, one can use node-level statistics [128]. Alternatively, another popular approach consists in simply setting $X = I_n$, i.e., the $n \times n$ identity matrix[6], thus mapping each node to a one-hot indicator [187, 188].

Then, a GNN usually consists in the succession of $L \geq 1$ *message passing iterations* a.k.a. *L layers* by analogy with standard neural networks. At each iteration $l \in \{1, \ldots, L\}$, we will obtain a *hidden embedding vector* $h_i^{(l)} \in \mathbb{R}^{d_l}$ for each node $i \in \mathcal{V}$ in the graph[7]. In particular, we set $d_L = d$ for the last layer. To compute such representations, we adopt a two-step process. Firstly, we *aggregate* vectors $h_j^{(l-1)}$ from nodes in the *neighborhood* $\mathcal{N}(i)$ of node i to derive the message $m_{\mathcal{N}(i)}^{(l-1)}$ to pass in layer l. This aggregation can correspond to a sum, an average, a concatenation or to some more complex operation [128]. Also, although $\mathcal{N}(i)$ will usually correspond to nodes directly connected to i, consistently with our notation from Definition 2.3, more general definitions of neighboring nodes have also been adopted in the scientific literature [130, 193]. To sum up, in this first step, we derive:

$$m_{\mathcal{N}(i)}^{(l-1)} = \text{AGGREGATE}^{(l)}\left(\{h_j^{(l-1)}; j \in \mathcal{N}(i)\})\right). \tag{2.16}$$

Secondly, we merge $m_{\mathcal{N}(i)}^{(l-1)}$ with node i's own previous representation $h_i^{(l-1)}$, to obtain $h_i^{(l)}$:

$$h_i^{(l)} = \text{UPDATE}^{(l)}\left(h_i^{(l-1)}, m_{\mathcal{N}(i)}^{(l-1)}\right). \tag{2.17}$$

Again, various strategies can be adopted for such an update step [128, 130]. In the above equations, $h_i^{(0)}$ corresponds to the input vector representation of node i, i.e., $h_i^{(0)} = x_i \in \mathbb{R}^f$ in the presence of node features. Also, the final $h_i^{(L)} \in \mathbb{R}^d$ will correspond to the actual node embedding vector eventually returned by this GNN model for node i, i.e.:

$$\forall i \in \mathcal{V}, z_i = h_i^{(L)}. \tag{2.18}$$

---

[6]Nonetheless, as we will emphasize in our work, such a choice will also make the GNN model *transductive*.

[7]We note that dimensions $d_1, \ldots, d_L$ of these vectors can differ across layers.

By design, such a process is *permutation equivariant* [128]. Intuitively, at each layer, a node i's representation will aggregate structural and feature information from its direct neighbors (and itself), assuming $\mathcal{N}(i)$ corresponds to the one-hop neighborhood of i. As, at layer n°2, these neighbors will have already aggregated information from their own neighbors, then $h_i^{(2)}$ will indirectly incorporate structural and feature information from the 2-hop neighborhood of i, i.e., nodes reachable from a path of length 2 starting from i in the graph. More generally, each $h_i^{(l)}$ will capture information up to the l-hop neighborhood of i and, in particular, the ultimate z_i vector will capture information up to the L-hop neighborhood.

2.3.2 A Popular GNN: Graph Convolutional Network (GCN)

Since the seminal models of Gori et al. [119], Merkwirth and Langauer [254] and Scarselli et al. [317], often praised as the first message-passing GNNs [128], several AGGREGATE($\cdot$) and UPDATE($\cdot$) functions have been proposed and compared in the scientific literature. We refer to relevant surveys [128, 386, 406] for an in-depth review. In this section, we focus on one of the most popular of these GNNs, that we will ourselves often implement in the experiments reported throughout this thesis: the *graph convolutional network* (GCN) from Kipf and Welling [188].

To introduce the GCN model consistently with its formulation from Kipf and Welling [188], we temporarily assume that $\mathcal{G}$ is *undirected*. Consistently with the notation from Definition 2.3, $\mathcal{N}(i)$ denotes the neighborhood of node $i \in \mathcal{V}$. For simplicity, we assume that $\mathcal{G}$ has no self-loop. From there, at each layer $l \in \{1, \dots, L\}$, Kipf and Welling [188] consider the following message passing operation:

$$
\begin{aligned}
h_i^{(l)} &= f^{(l)}\Big(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \tilde{A}_{ij} h_j^{(l-1)} \Big) \\
&= f^{(l)}\Big(\frac{1}{D_{ii}+1} h_i^{(l-1)} + \sum_{j \in \mathcal{N}(i)} \frac{A_{ij}}{\sqrt{(D_{ii}+1)(D_{jj}+1)}} h_j^{(l-1)} \Big).
\end{aligned}
\tag{2.19}
$$

In the above equation, $f^{(l)} : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ is a parameterized function combining:

- the multiplication[8] of hidden vectorial embedding representations by a trainable *weight matrix* $W^{(l-1)} \in \mathbb{R}^{d_{l-1} \times d_l}$. Several strategies, detailed thereafter, can be adopted to *tune* such a matrix, depending on the final application and on data availability;

- the use of a *non-linear activation function* on top of the above multiplication, analogously to standard neural networks [205]. Without loss of generality, Kipf and Welling [188] leverage a *Rectified Linear Unit* (ReLU) function: $\text{ReLU}(x) = \max(x, 0)$ [117]. This non-linear activation is often omitted for the last layer [187, 188].

---

[8]This operation also often involves the addition of a *bias* term [117], that we omit for the sake of clarity.

Summing up these operations using a matrix-level notation, we end up with the following defin-
ition of (multi-layer) GCN models.

Definition 2.11. A *multi-layer graph convolutional network* (GCN), with $L \geq 2$ layers, is a
function taking as input a symmetrically normalized adjacency matrix $\tilde{A} \in [0,1]^{n \times n}$ (as defined
in Definition 2.5), potentially equipped with a node feature matrix $X \in \mathbb{R}^{n \times f}$ (as defined in
Definition 2.7), and returning a node embedding matrix $Z \in \mathbb{R}^{n \times d}$ such as:

$$
\begin{cases}
H^{(0)} = X \text{ (or } I_n \text{ in the featureless case)} \\
H^{(l)} = \text{ReLU}(\tilde{A} H^{(l-1)} W^{(l-1)}), \text{ for } l \in \{1, ..., L-1\} \\
Z = H^{(L)} = \tilde{A} H^{(L-1)} W^{(L-1)},
\end{cases}
\tag{2.20}
$$

where each $H^{(l)}$ denotes the $n \times d_l$ matrix stacking all $h_i^{(l)}$ vectors, i.e., the i-th row of $H^{(l)}$
corresponds to the hidden embedding vector of node i at layer l. In particular, $d_0 = f$ (in
the presence of node features) or n (in the absence of node features), and $d_L = d$. Also,
$W^{(0)} \in \mathbb{R}^{d_0 \times d_1}, \ldots, W^{(L-1)} \in \mathbb{R}^{d_{L-1} \times d_L}$ are trainable weight matrices.

Unless explicitly stated otherwise[9], we will adopt this definition of the GCN in the remainder of
the thesis. In a nutshell, at each layer l, the GCN computes a vectorial representation for each
node $i \in \mathcal{V}$, by averaging the representations from layer $l-1$ of i's direct neighbors and of i itself.
This averaging operation is composed with a linear transformation via the weight matrices and
a ReLU activation. In their original work [188], Kipf and Welling tune $W^{(0)}, \ldots, W^{(L-1)}$ in
a (semi)-supervised fashion. Specifically, they iteratively minimize, by *gradient descent* [117],
a cross-entropy classification loss. This loss compares the z_i vectors transformed through a
softmax function [117] to d ground truth labels (article/node topics in citation networks in [188]).
Nonetheless, this strategy requires the availability of such labels. In Section 2.4, we will fully
detail how GCNs can alternatively be optimized in an *unsupervised* fashion, when acting as
encoders in the broader GAE and VGAE frameworks [186, 187].

Leveraging $\tilde{A}$ instead of A in a GCN has two main advantages related to *information propaga-
tion* in the graph. Firstly, using an unnormalized A would lead to a *summation* of vectorial
representations which, over layers, would change the scale of these vectors, especially for very
connected nodes. Moreover, in the absence of self-loops, multiplication by A would imply a
summation of vectors from all neighboring nodes *but not from the node itself*, which is nonethe-
less important in practice [188, 383]. Using $\tilde{A}$ addresses this problem, thanks to the addition of
the identity matrix I_n in the normalization (see Definition 2.5). In the last few years, several

---

[9]A notable counterexample is the one from Chapters 5, where we will adapt this message passing to *directed*
graphs. The symmetric normalization of A will be replaced by *out-degree normalization* $\tilde{A}_{\text{out}} = (D_{\text{out}} + I_n)^{-1}(A +
I_n)$, so that each node averages representations from neighbors *to which it points* through a directed edge, i.e.,
the outcoming neighbors. Another example is the one from Chapter 6 where we will consider the case $L = 1$,
leading to simpler linear models that are no longer *multi-layer* GCNs.

studies such as the ones from Klicpera et al. [192] and Wu et al. [383] empirically confirmed that using $\tilde{A}$ for message passing leads to better performance than some alternatives. Wu et al. [383] also proved that adding self-loops reduces the magnitude of the dominant eigenvalue of the corresponding Laplacian matrix: in essence, adding self-loops decreases the relative influence of distant nodes, which might help obtaining more stable representations and might be desirable for various applications. More recently, Dasoulas et al. [66] proposed a *parameterized* graph shift operator for GNN models such as GCNs. Specific parameter values of this operator result in the most commonly used graph matrices. Authors manage to incorporate these parameters in the training of GNNs, which permits learning the optimal message passing matrix for some given data- and task-specific application.

Since its formulation by Kipf and Welling [188] five years ago, the multi-layer GCN architecture emerged as one of the most popular ones in the graph representation learning community, with various successful applications to graph data from various domains ranging from biology to web mining and social networks [128, 320, 389, 406, 409]. As we will develop in Section 2.4, GCNs also play a central role in the GAE and VGAE models proposed by Kipf and Welling [186, 187] (see Section 2.4) and in numerous of their extensions [72, 124, 135, 279, 329, 361]. Besides these experimental successes, the popularity of GCN models can also be explained by their relative *simplicity* w.r.t. other GNNs, notably the ones directly based on *spectral graph convolutions* [41, 69] discussed in Section 2.3.3. Also, the evaluation of each GCN layer has a linear time complexity w.r.t. the number of edges m in the graph when leveraging sparse matrix-level operations [188], which is relatively low compared to several other models. Lastly, as we will further explain and empirically show in Chapters 8 and 9, GCN models can easily be leveraged in *inductive* settings when node features are available [313].

2.3.3 Theoretical Considerations

An active area of research in graph representation learning revolves around the study of potential theoretical guarantees for existing GNN models such as GCNs, as well as the development of new theoretically grounded models. As explained by Hamilton [128], GNNs independently emerged from three distinct theoretical motivations.

First and foremost, several GNN models were (and are still) explicitly conceived from the *graph signal processing* theory, generalizing the notion of convolution in a Euclidean space to graphs [41]. In parallel, other research studies motivated the GNN approach from its relation to Weisfeiler-Lehman *graph isomorphism* tests [130, 327, 380]. Lastly, analogies were also made between neural message passing and inference in *probabilistic graphical models* [65, 128].

In this section, we provide a brief overview of the first of these aspects, i.e., graph signal processing. We refer to Hamilton [128] for a review of the two other ones, that we omit in this thesis. While they are relevant and linked to active areas of research [250, 261, 264, 388], they

are less connected to the contributions presented in the next chapters.

GNN Models and Spectral Graph Convolutions Let us consider the orthogonal *spectral decomposition* of the symmetric message passing operator $\tilde{A}$ used in Equation (2.20),

$$\tilde{A} = U\Lambda U^T, \tag{2.21}$$

where Λ denotes the diagonal matrix containing the eigenvalues λ_i of $\tilde{A}$ (assumed to be ordered), and where $U = [u_1, \ldots, u_n]^T$ denotes the corresponding orthogonal matrix containing the eigenvectors u_i of $\tilde{A}$. Then, the computation performed in Equation (2.20) can be reformulated as:

$$\text{ReLU}(U\Lambda U^T H^{(l-1)}W^{(l-1)}) = \text{ReLU}\left(\sum_{i=1}^{n} \lambda_i u_i u_i^T H^{(l-1)}W^{(l-1)}\right). \tag{2.22}$$

Therefore, performing one message passing of hidden representations on the graph $\mathcal{G}$ defined by $\tilde{A}$, i.e., $\tilde{A}H^{(l-1)}$, can be interpreted as a *Fourier transform* of these hidden representations called *graph Fourier transform* [334]. The eigenvectors of $\tilde{A}$ act as a Fourier basis and the eigenvalues of $\tilde{A}$ define the Fourier coefficients. From such a setting, *spectral graph convolutions* are defined as element-wise product operations in this Fourier space.

When trying to perform a theoretical analysis of neural message passing steps in GNN models such as GCNs, it often turns out to be more insightful to instead consider the above decomposition [314], and therefore analyze eigenvalues and eigenvectors of the corresponding message passing operator (see, e.g., Chapter 7). Historically, the study of spectral graph theory [58, 342], and in particular the area of graph signal processing [275, 315], has yielded insights in the study of graphs. Therefore it is unsurprising that, in the study of the GNNs, the spectral analysis of these architectures appears as a promising avenue of research as well [21, 66, 102].

Such a spectral perspective has also given rise to a variety of architectures directly relying on graph signal processing, e.g., by proposing convolutional operations a.k.a. *filters* based on learnable functions applied to the diagonal terms of Λ. This includes the seminal model of Bruna et al. [41] and several more recent ones [69, 140, 214]. In particular, as this spectral approach is computationally costly[10], Defferrard et al. [69] proposed to approximate smooth filters in the spectral domain using Chebyshev polynomials [131]. Kipf and Welling [188] themselves mathematically derived their GCN architecture as a faster and localized first-order approximation of spectral graph convolutions, bringing further simplifications over Defferrard et al. [69]. Their analysis brings some theoretical foundation and motivation to the relatively intuitive message passing strategy described in Section 2.3.2.

---

[10]Filters are directly defined in the Fourier space, and computing the graph Fourier transform involves $O(n^2)$ operations. As already explained, computing the full eigendecomposition of an adjacency or a Laplacian matrix also itself suffers from an $O(n^3)$ complexity, when using a baseline implementation.

2.3.4   Recent Advances in Representation Learning with GNN Models

As mentioned in our introduction to node embedding methods in Section 2.2, the graph representation learning field has grown at a very fast pace over the past few years. This is especially true for the sub-field involving graph neural networks, to such an extent that it becomes increasingly difficult to keep track of all advances from the research community. Before introducing GAE and VGAE models in the next Section 2.4 and focusing on them in most of this thesis, this section will nonetheless try to provide a brief summary of some of the most notable other advances related to representation learning with GNNs.

Besides advances on provably powerful GNN models, e.g., based on the aforementioned spectral analyses or connections to Weisfeiler-Lehman tests [128, 250, 261, 264, 388], numerous research studies proposed novel message passing GNN architectures for various applications. Among the most impactful ones, GraphSAGE models from Hamilton et al. [130] incorporate generalized neighborhood aggregation schemes, suitable for inductive representation learning. Graph attention networks (GAT) from Veličković et al. [364] leverage *attention* mechanisms during message passing, permitting weighting the influence of each neighbor during information aggregation. Graph diffusion convolutions (GDC) from Klicpera et al. [193] resort to more general *diffusion* functions to identify which nodes one should aggregate information from, which is useful in the presence of graphs with noisy or arbitrarily defined edges. Jumping knowledge (JK) networks from Xu et al. [389] use representations from all hidden layers, and not only the last one, to compute embedding vectors. Graph Transformers from Dwivedi and Bresson [79] generalize transformer neural networks [382] to graphs. Several other models were additionally designed for specific structures such as knowledge graphs [320] or dynamic graphs [179], and recent works also introduced GNN models *without* message passing procedures [153, 383].

As large graphs with millions (or even billions) or nodes and edges are ubiquitous [307], several studies also aimed to scale GNN models, typically by using mini-batch sampling or other various approximate learning strategies. This includes the recent FastGCN [50], Cluster-GCN [53] and GraphSAINT [397] models, among others [51, 130, 394]. Moreover, while research on GNN models is often referred to as "*deep learning on graphs*" [406], we note that the most effective architectures actually leverage a few layers at most, contrary to other fields such as computer vision [205]. This is explained by the standard challenges related to training deep architectures, such as *vanishing gradients* in backpropagation [117], but also by the nature of real-world graph data (numerous graphs have a *small-world* structure [3], i.e., a few hops/layers already permit reaching any node from another one) and other graph-specific problems such as *over-smoothing* and *over-squashing* (see [11, 221]). While some works proposed techniques to train deeper GNNs, e.g., based on regularization [302, 407] or residual connections [115, 389], they often fail to outperform models with a few layers [326]. On the contrary, several other studies tend to show that *simpler* models can achieve competitive empirical performances. This includes the one of

Wu et al. [383] who introduced simple graph convolution (SGC), a variant of GCN removing nonlinearities and collapsing weight matrices. To finish, we point out that recent articles [150, 151, 326], including the one behind the Open Graph Benchmark initiative [151], also criticized the current evaluations of GNNs, pointing at discrepancies in experimental procedures and proposing new relevant tasks or datasets. These last three important aspects (scalability, simplification, better evaluation) will also be at the center of several of our studies on GAE and VGAE models.

2.4 Representation Learning with Graph Autoencoders

So far, we presented some general concepts and definitions related to graph representation learning. In this Section 2.4, we now leverage this background to formally introduce the GNN-based models that will be at the center of our research: *graph autoencoders*. In its most general formulation, this term can actually refer to *two* families of models, learning node embedding representation from graph data [186, 187, 310, 356, 371]:

- the *deterministic graph autoencoders*, often simply referred to as graph autoencoders (GAEs) in the following when there is no ambiguity, and presented in Section 2.4.1;

- the *variational graph autoencoders* (VGAEs), presented in Section 2.4.2.

They both rely on an *encoding-decoding strategy* that, in a broad sense, consists of *encoding* nodes into an embedding space from which *decoding*, i.e., reconstructing the original graph should ideally be possible, by leveraging either a deterministic (for a GAE) or a probabilistic (for a VGAE) approach. Intuitively, the ability to accurately reconstruct a graph from a node embedding space indicates that this space preserves some important information from the graph structure. As explained in the introduction, GAE and VGAE models are specifically suited for representation learning on graphs *in the absence of node labels*, i.e., in an unsupervised fashion. Simultaneously, GAE and VGAE models can process attributed graphs. As VGAEs emerged as effective alternatives to GAEs in several previous works described throughout this thesis, we see value in considering both variants of graph autoencoders in our work.

2.4.1 Deterministic Graph Autoencoder (GAE)

Albeit under various formulations, the encoding-decoding strategy has been widely adopted over the last years to learn embedding spaces in the absence of node labels [186, 187, 356, 370, 371]. In this thesis, we follow the formulation of Kipf and Welling [187]. While it is not the oldest article, their work is explicitly mentioned as the seminal reference in the majority of recent advances involving graph autoencoders, including [55, 56, 73, 124, 135, 156, 216, 217, 279, 329, 361].
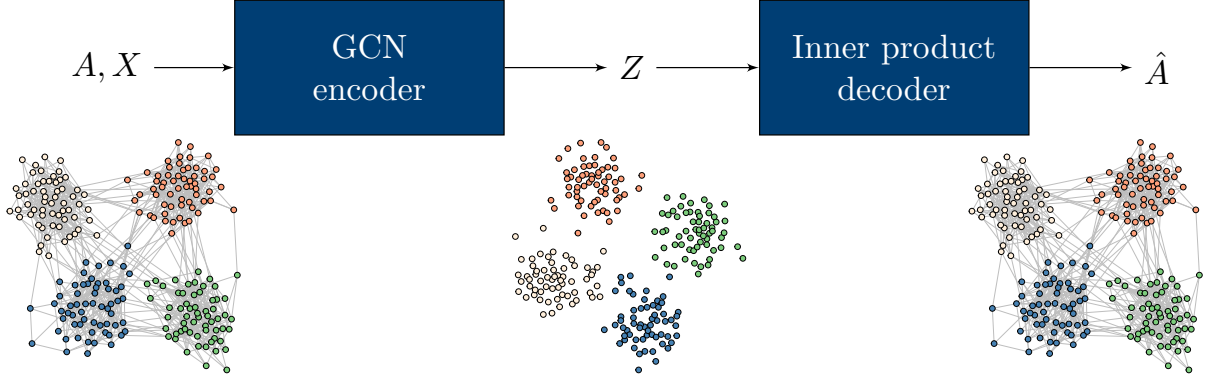
Figure 2.6: Schematic representation of a GAE model, as formulated by Kipf and Welling [187].

Graph autoencoders involve the combination of two building blocks: an *encoder* and a *decoder*. The above Figure 2.6 provides an illustration of a GAE model.

Encoder   In its most general formulation, the first of these two components, i.e., the *encoder*, is defined as follows.

Definition 2.12. An *encoder* is a function processing the adjacency matrix A and the node feature[11] matrix X of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and mapping each node $i \in \mathcal{V}$ from $\mathcal{G}$ to a low-dimensional embedding vector $z_i \in \mathbb{R}^d$ with $d \ll n$. Adopting the notation from Definition 2.10, we have:

$$Z = \text{Encoder}(A, X). \tag{2.23}$$

In practice, a GNN (with weights optimized using the procedure described thereafter) often acts as the encoder in GAE models. In particular, Kipf and Welling [187] leverage a multi-layer GCN [188], as defined in Definition 2.11, to encode nodes, i.e.:

$$Z = \text{GCN}(A, X). \tag{2.24}$$

To this day, multi-layer GCNs remain the most popular encoders in GAE extensions building upon Kipf and Welling [187], including [55, 124, 135, 156, 279, 307, 311, 329, 361], mainly thanks to the relative simplicity and reduced complexity of these models w.r.t. several GNN alternatives (see Section 2.3.2). Nonetheless, they can be replaced by various alternatives, including by faster [50, 53, 130], by more sophisticated [41, 69, 364] or, on the contrary, by simpler [56, 310] models.

---

[11]As in Section 2.3, we simply set $X = I_n$ when dealing with featureless graphs.

Decoder The second component of a GAE, i.e., the *decoder*, aims to reconstruct an $n \times n$ adjacency matrix $\hat{A}$, estimated from the learned embedding vectors. It is defined as follows.

Definition 2.13. Assuming an $n \times d$ node embedding matrix Z stacking up node embedding vectors $z_i \in \mathbb{R}^d$ for nodes of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a *decoder* is a function reconstructing an $n \times n$ estimated adjacency matrix $\hat{A}$ from Z:

$$\hat{A} = \text{Decoder}(Z). \tag{2.25}$$

While another neural network could act as a decoder [218, 282, 371], Kipf and Welling [187] and most of the aforementioned extensions rely on simpler *inner product* decoders:

$$\hat{A} = \sigma(ZZ^T), \tag{2.26}$$

where σ denotes the *sigmoid* function: $\sigma(x) = 1/(1 + e^{-x})$. Therefore, for all node pairs $(i, j) \in \mathcal{V} \times \mathcal{V}$, we have:

$$\hat{A}_{ij} = \sigma(z_i^T z_j) \in \,]0, 1[. \tag{2.27}$$

In such a setting, a large and positive inner product $z_i^T z_j$ in the node embedding space indicates the likely presence of an edge between nodes i and j in $\mathcal{G}$, according to the model. Again, the choice of inner product decoders is not restrictive, and recent efforts considered replacing them with alternatives verifying some desirable properties such as the ability to capture triads structures [329], to simultaneously reconstruct node features [347], or to reconstruct biologically plausible graphs in the case of autoencoders for molecular structures [233, 336].

Optimization We recall that GAE models aim to learn node embedding spaces from which one can accurately reconstruct graphs. The intuition behind this strategy is the following: if, starting from Z, one can reconstruct a graph close to the true one, i.e., $\hat{A} \approx A$, then these embedding vectors should manage to preserve some important characteristics of the initial graph structure, and should therefore be useful to perform downstream tasks such as link prediction.

Consequently, GAEs are trained to minimize *reconstruction losses*, which specifically evaluate the similarity between the decoded adjacency matrix $\hat{A}$ and the original one A. For instance, Kipf and Welling [187] tune weight matrices of their GCN encoders by iteratively minimizing, by gradient descent[12] [117], the following *cross-entropy* loss:

$$\mathcal{L}_{\text{GAE}} = \frac{-1}{n^2} \sum_{(i,j) \in \mathcal{V} \times \mathcal{V}} \left[A_{ij} \log(\hat{A}_{ij}) + (1 - A_{ij}) \log(1 - \hat{A}_{ij}) \right]. \tag{2.28}$$

In the case of sparse graphs where unconnected node pairs significantly outnumber the connected

---

[12]Specifically, Kipf and Welling [187] use the *Adam* algorithm, a prevalent method for first-order gradient-based optimization in neural networks architectures, fully described in the original work of Kingma and Ba [184].

ones, i.e., the graph's edges, it is common to reweight the "positive terms" in Equation (2.28) by a factor $w_{\text{pos}} > 1$ [187, 307, 371]. We note that an exact evaluation of $\mathcal{L}_{\text{GAE}}$ requires the reconstruction of the entire matrix $\hat{A}$, which suffers from a quadratic $O(dn^2)$ time complexity. Such a decoding approach is therefore unsuitable for graphs with more than a few thousand nodes. Scalability concerns will be thoroughly discussed and addressed in Chapters 3 and 4. Lastly, we note that this minimization scheme based on reconstruction losses permits optimizing a GCN, or any encoder, in an unsupervised fashion, i.e., without node-level ground truth labels (as in Section 2.3.2).

2.4.2 Variational Graph Autoencoder (VGAE)

Kipf and Welling [187] also considered a probabilistic variant of GAE, extending the *variational autoencoder* (VAE) from Kingma and Welling [185]. Besides constituting generative models with promising applications to graph generation [169, 233, 336] (see the next paragraphs), variants of *variational graph autoencoder* (VGAE) models also turned out to be effective alternatives to GAE on several downstream applications such as link prediction or community detection tasks [56, 135, 187, 311]. In the following, we briefly review key concepts related to VAE, and subsequently present the VGAE model.

Latent Models and VAE   Variational autoencoders are *latent variable* models. In essence, these are probabilistic models explaining some observed variable $x \in \mathcal{X}$ through some unobserved latent variable $z \in \mathcal{Z}$. Denoting by $p_\theta(x)$ the distribution of x, parameterized by some $\theta \in \Theta$, we have:

$$p_\theta(x) = \int_{\mathcal{Z}} p_\theta(x|z)p(z)dz. \tag{2.29}$$

The right-hand side of Equation (2.29) introduces the conditional distribution $p_\theta(x|z)$, to explicitize the unobserved underlying role that z plays in the generation of x. In such latent variable models, the objective is usually to estimate the parameters of $p_\theta(x|z)$, referred to as the *generative* model, which maximize $p_\theta(x)$ for some observed data and some known *prior distribution $p(z)$* on z. As explained by Kipf [186], in many applications, one would like to use neural networks as generative models, e.g., in combination with a Gaussian distribution:

$$p_\theta(x|z) = \mathcal{N}(x; \text{NN}(z), \Sigma), \tag{2.30}$$

where this notation refers to the density of a Gaussian distribution, with a mean vector $\text{NN}(z)$ determined by a neural network (θ corresponds to its weights), and with a fixed variance matrix Σ. Unfortunately, in a large number of applications, this formulation of $p_\theta(x)$ will be *intractable*[13].

---

[13]We note that this is not specific to neural networks. For instance, we refer to Blei et al. [30] for a presentation of the *Gaussian Mixture Model* (GMM), a famous example of latent variable model. In a GMM, $p_\theta(x)$ is intractable due to a sum over the latent variables in the integrand of $p_\theta(x)$.

It will be impossible to optimize θ analytically, e.g., via maximum likelihood estimation [74].

To address this issue, variational autoencoders (VAE) from Kingma and Welling [185] leverage concepts from *variational inference* [74], and consider the optimization of a *tractable* lower bound of $p_\theta(x)$, referred to as the *evidence lower bound* (ELBO). It involves an approximate *posterior distribution* $q_\phi(z|x)$ with parameters $\phi \in \Phi$. Formally, using Jensen's inequality [166], we have:

$$\log p_\theta(x) = \log \int_{\mathcal{Z}} \frac{q_\phi(z|x)}{q_\phi(z|x)} p_\theta(x|z) p(z) dz \geq \int_{\mathcal{Z}} q_\phi(z|x) \log \frac{p_\theta(x|z) p(z)}{q_\phi(z|x)} dz. \tag{2.31}$$

The right-hand side of Equation (2.31) corresponds to the ELBO, that we reformulate as follows:

$$\begin{aligned}
\text{ELBO} &= \int_{\mathcal{Z}} q_\phi(z|x) \log \frac{p_\theta(x|z) p(z)}{q_\phi(z|x)} dz \\
&= \int_{\mathcal{Z}} q_\phi(z|x) \log p_\theta(x|z) dz + \int_{\mathcal{Z}} q_\phi(z|x) \log \frac{p(z)}{q_\phi(z|x)} dz \\
&= \int_{\mathcal{Z}} q_\phi(z|x) \log p_\theta(x|z) dz - \int_{\mathcal{Z}} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z)} dz \\
&= \mathbb{E}_{q_\phi(z|x)} \Big[\log p_\theta(x|z) \Big] - \mathcal{D}_{\text{KL}} \Big(q_\phi(z|x) || p(z) \Big),
\end{aligned} \tag{2.32}$$

where $\mathbb{E}_q$ denotes the expectation under the distribution q and $\mathcal{D}_{\text{KL}}$ denotes the Kullback-Leibler divergence [199]. In the context of VAE models, neural networks often characterize $q_\phi(z|x)$, which is then referred to as the *inference* model [185]. For instance, assuming two neural networks $\text{NN}^{(\mu)}$ and $\text{NN}^{(\Sigma)}$ providing a mean vector and a variance matrix, one can set:

$$q_\phi(z|x) = \mathcal{N}(z; \text{NN}^{(\mu)}(x), \text{NN}^{(\Sigma)}(x)). \tag{2.33}$$

Here, ϕ would correspond to neural weights. Parameters θ and ϕ are jointly optimized by maximizing the ELBO by gradient ascent, using some available data samples x from a dataset. We underline that the choices of $q_\phi(z|x)$ and $p(z)$ are often driven by computational constraints. In this thesis, we will often assume that they are Gaussian distributions with different parameters. The Kullback-Leibler divergence between two Gaussian distributions has a closed form (see Doersch [74]) which facilitates gradient computation. Also, gradients of the above expectation are usually estimated through Monte Carlo approximations [74, 185], and using a *reparameterization trick* [185]. This trick consists in positioning all sampling steps from the VAE model in the input layer, to avoid backpropagating errors through a layer that samples z vectors from $q_\phi(z|x)$, which is a non-continuous operation and has no gradient. For instance, let us assume the inference model from Equation (2.33). Instead of sampling $z \sim \mathcal{N}(\text{NN}^{(\mu)}(x), \text{NN}^{(\Sigma)}(x))$, one can sample $\varepsilon \sim \mathcal{N}(0, I)$ in the input layer, and subsequently compute:

$$z = \text{NN}^{(\mu)}(x) + \text{NN}^{(\Sigma)}(x)^{1/2} \times \varepsilon. \tag{2.34}$$

2.4. Representation Learning with Graph Autoencoders
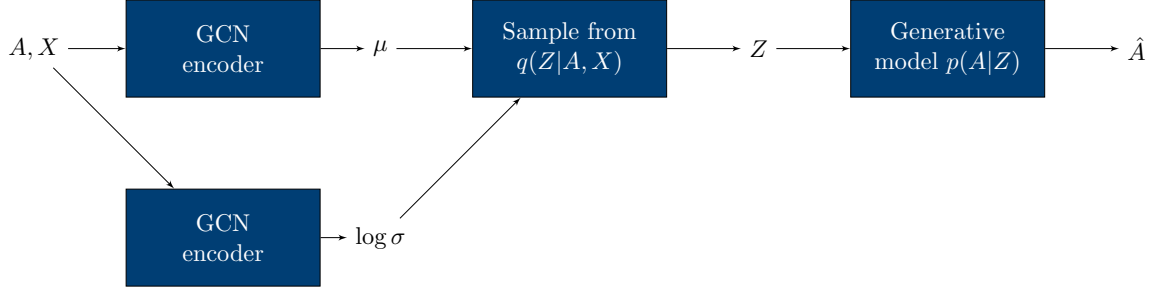
Figure 2.7: Schematic representation of a VGAE model, as formulated by Kipf and Welling [187].

Once parameters of the VAE are optimized, the inference model can infer latent variables z of new data samples x. Analogously to previous sections, these latent variables will often act as *embedding vectors* summarizing input data. Simultaneously, the generative model can be used to generate data from a latent variable z, e.g., new unseen data samples drawn from the prior distribution $p(z)$. For instance, VAE models have been successfully transposed to image generation problems [74, 277]. We refer to the comprehensive tutorial of Doersch [74] for a broader introduction to VAE models and a discussion of their foundations and their applications.

From VAE to VGAE   As explained at the beginning of Section 2.4.2, Kipf and Welling [187] extended the VAE framework to graph structures. Their *variational graph autoencoder* (VGAE) provides an alternative strategy to learn node embedding vectors in an unsupervised fashion, assuming that these vectors are drawn from specific distributions. As standard VAE models, a VGAE incorporates an inference model and a generative model. In the following, by analogy with the GAE approach, we will also often refer to these components as the VGAE's encoder and decoder, respectively. The above Figure 2.7 provides an illustration of a VGAE model.

Encoder   In their work, Kipf and Welling [187] assume that each embedding vector $z_i \in \mathbb{R}^d$ corresponds to the *latent* vector of a node $i \in \mathcal{V}$, following the VAE paradigm. This vector is a sample drawn from a d-dimensional Gaussian distribution, with mean vector $\mu_i \in \mathbb{R}^d$ and variance matrix $\Sigma_i = \text{diag}(\sigma_i^2) \in \mathbb{R}^{d \times d}$ (with $\sigma_i \in \mathbb{R}^d$). They rely on *two encoders* to learn these parameters. Denoting the $n \times d$ matrices stacking up the d-dimensional mean and (log)-variance vectors for each node by μ and by $\log \sigma$[14], respectively, they set:

$$\mu = \text{Encoder}_\mu(A, X) \text{ and } \log \sigma = \text{Encoder}_\sigma(A, X). \tag{2.35}$$

As is the case for GAEs, multi-layer GCNs often act as encoders:

$$\mu = \text{GCN}_\mu(A, X) \text{ and } \log \sigma = \text{GCN}_\sigma(A, X). \tag{2.36}$$

---

[14]We allow a slight clash of notation here (as σ also denotes the sigmoid activation function), for consistency with the commonly used notation from the literature (see, e.g., [187]), and due to very low risks of confusion.

Then, they adopt a *mean-field inference model* for Z [187]. With $\mathcal{N}(\cdot|\mu_i, \mathrm{diag}(\sigma_i^2))$ denoting the density of a Gaussian distribution with mean vector μ_i and variance matrix $\mathrm{diag}(\sigma_i^2)$, we have:

$$q(Z|A, X) = \prod_{i=1}^{n} q(z_i|A, X), \text{ where } q(z_i|A, X) = \mathcal{N}(z_i|\mu_i, \mathrm{diag}(\sigma_i^2)), \tag{2.37}$$

Decoder   In the VGAE setting, the actual embedding vectors z_i are sampled from the above normal distributions. From such embedding representations, VGAE models then require a *generative* model $p(A|Z, X)$, to act as a graph *decoder*. As for GAE, Kipf and Welling [187] rely on simple inner products together with sigmoid activation functions to reconstruct edges:

$$\hat{A}_{ij} = p(A_{ij} = 1|z_i, z_j) = \sigma(z_i^T z_j), \tag{2.38}$$

where the embedding vectors z_i, z_j are sampled from the distribution in Equation (2.37). Then, the authors assume the following generative model which factorizes over the edges, and is conditionally independent of X for simplicity [186]:

$$p(A|Z, X) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(A_{ij}|z_i, z_j). \tag{2.39}$$

Optimization   During training, and similarly to VAE models [185], Kipf and Welling [187] iteratively maximize the tractable *evidence lower bound* (ELBO) [185] of the model's likelihood, written as follows in the context of a VGAE:

$$\mathcal{L}_{\mathrm{VGAE}} = \mathbb{E}_{q(Z|A,X)}\Big[\log p(A|Z, X)\Big] - \mathcal{D}_{KL}\Big(q(Z|A, X)||p(Z)\Big). \tag{2.40}$$

This ELBO is iteratively maximized w.r.t. weights of the two GCN encoders, by gradient ascent. In the above Equation (2.40), $p(Z)$ corresponds to a unit Gaussian prior (i.e., $\mathcal{N}(0, I_d)$) on the distribution of the latent vectors, that can also be interpreted as a regularization term on the magnitude of the embedding vectors [124, 314]. As is the case for $\mathcal{L}_{\mathrm{GAE}}$ from Equation (2.28), an exact evaluation of the ELBO suffers from an $O(dn^2)$ time complexity.

2.4.3   Applications and Limitations

The question of how to properly determine the quality of node embedding representations learned from GAE and VGAE models is crucial. While one could directly report reconstruction losses [371], recent research articles instead strove to apply the GAE and VGAE models to downstream evaluation tasks, which permits reporting more insightful metrics [187, 356, 371]. In particular, Kipf and Welling [187] originally evaluated their GAE and VGAE models on *link prediction* problems in three popular citation networks (Cora, Citeseer, and Pubmed) [323],

adopting the train/validation/test methodology described in Section 2.1.2. In their experiments, GAEs and VGAEs reached competitive link prediction scores w.r.t. DeepWalk [287] and Laplacian eigenmaps [368], that we presented in Section 2.2. They also recalled an additional benefit of the GAE and VGAE models over these baselines, which is the ability to leverage both the graph structure and node features when learning embedding spaces.

Over the last five few years, the overall effectiveness of the GAE and VGAE paradigms at addressing link prediction has been widely confirmed experimentally [28, 73, 124, 133, 135, 156, 279, 299, 307, 310, 311, 329, 359, 361]. Numerous research efforts proposed and evaluated variants of GAEs and VGAEs designed for this specific task, improving their performances by considering more refined encoders [133, 135, 217, 308, 385], decoders [124, 135, 311, 329, 347] or regularization techniques [156, 279, 361]. Most of these extensions were actually published during the time of this PhD thesis. The exact technical contribution of several of them will be further detailed in this thesis, when used as baselines for our own proposed methods.

Other research studies on GAE and VGAE models successfully addressed different downstream tasks that are closely related to link prediction, such as edge classification [299] or graph-based recommendation [28, 133, 400]. GAE and VGAE models have also be applied to (semi-supervised) node classification [135, 359], canonical correlation analysis [174] and to community detection [55, 56, 126, 216, 262, 370]. Last, but not least, researchers were also interested in leveraging VGAEs as generative models, especially to generate molecular graph data. We refer for instance to [169, 233, 243, 336] for recent advances in graph generation with VGAE models. This aspect will *not* be at the center of our work in this thesis; we will instead mainly focus on the *inference* component of VGAEs. As already explained, VGAE models emerged as effective alternatives to deterministic GAEs on several downstream applications in some of the above references, which we will further analyze in the next chapters. Consequently, in this thesis, we saw value in considering both GAE and VGAE models to learn node embedding spaces.

Despite these promising advances, at the beginning of this PhD, i.e., in 2018, transposing these recent advances to industrial-level applications, e.g., at Deezer, was still a challenging task. As mentioned in this section, the standard GAE and VGAE methods suffer from scalability issues. We argue in the next chapters that standard techniques from deep learning, such as mini-batch sampling, often fail to provide satisfying solutions to this problem [307, 308]. As a consequence, in 2018, experiments on GAEs and VGAEs were limited to graphs with at most a few thousand nodes and edges. Among other challenges, standard GAE and VGAE models were designed for undirected and static graphs [187], and often neglected simple but effective encoding schemes [310]. Recent studies [55, 56] also emphasized their relative limitations on community detection applications. The next part of this PhD thesis will present our contributions to improve GAEs and VGAEs and facilitate their application to real-world problems.

Part II

Contributions to Representation Learning with Graph Autoencoders

3

A Degeneracy Framework for Scalable Graph Autoencoders

This chapter presents research conducted with Romain Hennequin, Viet-Anh Tran, and Michalis Vazirgiannis, and published in the proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019) [308].

3.1 Introduction

We begin this Part II of the thesis, which presents our contributions to representation learning with GAE and VGAE models, with two chapters fully dedicated to *scalability* concerns. Indeed, while these models emerged as powerful node embedding methods, they also suffer from scalability issues, preventing them to be applied to large graphs with millions of nodes and edges such as those available at Deezer. More precisely, we identify two sources of complexity:

- firstly, as explained in Section 2.4.1, GNN models often act as GAE/VGAE *encoders*. The encoding step can therefore be computationally costly if the GNN models under consideration themselves involve complex operations. Nonetheless, at the time of this work, several scalable GNNs had already been proposed in the scientific literature. To this day, GCNs remain the most popular encoders for GAE and VGAE models. As explained in Section 2.3.2, the cost of evaluating each layer of a GCN evolves linearly w.r.t. the number of edges m [188]. This can be improved by instead encoding nodes with a FastGCN [50], with a Cluster-GCN [53], or by using simpler graph convolutions [383] or other recently proposed stochastic strategies [51, 130, 394, 397] for improved scalability;

- moreover, and more importantly, the *decoding* step of standard GAE and VGAE models usually suffers from a high computational complexity. In particular, Kipf and Welling [187] and numerous extensions of their work leverage inner product decoders $\hat{A} = \sigma(ZZ^T)$. They require the multiplication of the dense matrices Z and Z^T, to compute reconstruction losses at each training iteration. As explained in Section 2.4.1, this decoding scheme has an $O(dn^2)$ complexity, as most of the alternative decoders from the literature, which also require computing inner products or Euclidean distances [124, 311, 329]. Storing $n \times n$ dense matrices $\hat{A}$ can also potentially lead to memory issues for a large n. Some existing decoders are even more complex: for instance, the VGAE model of Simonovsky and Komodakis [336] includes a graph matching step with an $O(n^4)$ complexity[1]. As a consequence, the aforementioned efforts to scale GNNs (that were achieved in a supervised setting, and out of the wider GAE and VGAE unsupervised frameworks where GNN encoders are only a building block) are *not* sufficient to scale GAEs and VGAEs. These models will usually still suffer from (at least) a quadratic complexity due to their costly decoding operations.

As a result, at the time of this work, GAE and VGAE models had only been applied to relatively small graphs with, at most, a few thousand nodes and edges. While the majority of works were still aiming to reconstruct *entire graphs* at each GAE/VGAE training iteration, we acknowledge that Kipf and Welling [187] and Grover et al. [124] briefly mentioned *node sampling* and *edge sampling* ideas as possible extensions. We will later observe that a direct uniform sampling of nodes or edges in reconstruction losses is often suboptimal (and, in Chapter 4, we will ourselves propose some refined stochastic sampling strategies for loss approximation). We note that Shi et al. [329] also incorporated mini-batch sampling ideas in their GAE/VGAE models, but rather to reconstruct *triads* of nodes and improve performances than to provide scalable models. They did not report running times nor experiments on large graphs. To sum up, the question of how to effectively scale GAE and VGAE models remained unsatisfactorily addressed.

In this Chapter 3, we present the solution proposed at the beginning of this PhD to address this issue. More specifically, our contribution is threefold:

- we introduce a general framework to scale GAE and VGAE models to large graphs with millions of nodes and edges, by optimizing the reconstruction loss (for GAE) or the ELBO objective (for VGAE) only from a dense and representative subset of nodes, and then by propagating embedding representations in the entire graph. These nodes are selected using *graph degeneracy* [247] concepts. Such an approach considerably improves scalability while, to some extent, preserving performances on downstream evaluation tasks;

- we apply this framework to five real-world graph datasets and discuss empirical results on several variants of GAE and VGAE models for two learning tasks: link prediction and

---

[1]This is nonetheless acceptable in their work, as they focus on small molecular graphs with fewer than 40 nodes [336].

community detection. Our implementation of this framework is publicly available [308]. At the time of publication of this work (early 2019), these experiments provided, to the best of our knowledge, the first application of GAE and VGAE models to graphs with millions of nodes and edges;

- we show that our models achieve competitive results w.r.t. several popular scalable unsupervised node embedding methods, such as node2vec and DeepWalk from Section 2.2.3. It emphasizes the relevance of pursuing further research towards scalable graph autoencoders.

This chapter is organized as follows. In Section 3.2, we present our proposed degeneracy framework for scalable graph autoencoders, and we explain how to learn node embedding spaces using a GAE or a VGAE model trained on a subset of nodes. We report and discuss our experimental evaluation in Section 3.3, and we conclude in Section 3.4. In Section 3.5, we provide additional tables and proofs, placed out of the "main" chapter for the sake of brevity and readability.

3.2 Scaling GAEs and VGAEs with Graph Degeneracy

Throughout this paper, we adopt the notation from Chapter 2 and further assume that $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an *undirected* graph, without self-loops. Additionally, and for the sake of simplicity, we assume in this section that nodes are *featureless*, i.e., $X = I_n$. Therefore, GAE/VGAE models only learn node embedding vectors from A. Node features will be re-introduced in Section 3.3.

3.2.1 Overview of the Degeneracy Framework

To deal with large graphs, we propose to optimize the reconstruction loss (for GAE models) or the ELBO variational lower bound (for VAE models) only from a wisely selected subset of nodes, instead of using the entire graph $\mathcal{G}$ which would be intractable. More precisely, we proceed as follows:

- **Step 1:** firstly, we identify the nodes on which the GAE/VGAE model should be trained, by computing a *k-core decomposition* [247] of the graph. The selected subgraph is the so-called *k-degenerate* version of the original one. We justify this choice in Section 3.2.2, and explain how we choose the value of k;

- **Step 2:** secondly, we train a graph autoencoder (a GAE or a VGAE, following the architecture of Kipf and Welling [187] or any variant/extension) on this k-degenerate subgraph. Hence, we derive node embedding vectors for the nodes included in this subgraph;

- **Step 3:** regarding the nodes of $\mathcal{G}$ that are not in this subgraph, we infer their embedding representations using a simple and fast propagation heuristic, presented in Section 3.2.3.
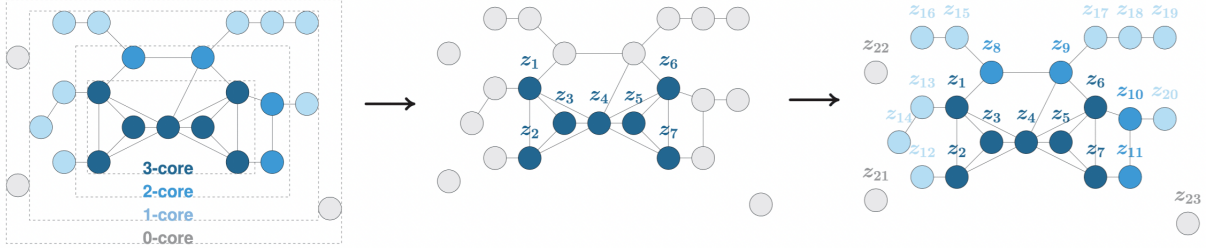
Figure 3.1: Schematic overview of our degeneracy framework for scalable graph autoencoders. After extracting the smaller k-core a.k.a. k-degenerate subgraph of $\mathcal{G}$, using the $O(\max(m, n))$ algorithm described in Section 3.2.2 (*left*), we train a GAE or a VGAE model on this subgraph only (*middle*). In this figure, the model is trained on the 3-core subgraph. While this training procedure still suffers from (at least) a quadratic complexity, the input subgraph is significantly smaller, which makes the training tractable. However, we obtain embedding vectors z_i for nodes belonging to this subgraph, but not for others. We subsequently infer embedding vectors of the remaining nodes (*right*), by leveraging the $O(m)$ algorithm described in Section 3.2.3.

In the above Figure 3.1, we illustrate the three steps of this "*degeneracy framework*" for scalable graph autoencoders. In a nutshell, the training step (Step 2) still has a potentially high complexity (e.g., a quadratic complexity, when using the GAE or VGAE from Kipf and Welling [187]). However, the subgraph processed by the autoencoder will be significantly smaller than $\mathcal{G}$, making the training tractable. Moreover, we will show in the next sections that Steps 1 and 3 have linear running times w.r.t. the number of edges m in the graph. Therefore, this framework significantly improves speed and scalability and, as we will experimentally confirm in Section 3.3, can effectively process large graphs with up to millions of nodes and edges.

3.2.2 Graph Degeneracy and k-Core Decomposition

In this section, we detail our Step 1, i.e., the identification of a representative subgraph on which the GAE or VGAE model should be trained. We resort to the *k-core decomposition* [24, 247], a powerful tool to analyze the structure of a graph. Formally, the *k-core*, or *k-degenerate* version of graph $\mathcal{G}$, is defined as follows.

Definition 3.1. The *k-core* or *k-degenerate* version of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the largest subgraph in which every node has a degree at least equal to k *within the subgraph*. Therefore, in a k-core, each node is connected to at least k nodes, that are themselves connected to at least k nodes. We denote by $\mathcal{C}_k \subseteq \mathcal{V}$ the set of nodes belonging to the k-core of $\mathcal{G}$.

Definition 3.2. The *degeneracy* number $\delta^*(\mathcal{G}) \in \{0, \ldots, n\}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the maximum k for which the k-core is not empty.

As illustrated in Figure 3.2, nodes from each core $\mathcal{C}_0, \ldots, \mathcal{C}_{\delta^*(\mathcal{G})}$ form a nested chain, i.e.:

$$\mathcal{C}_{\delta^*(\mathcal{G})} \subseteq \mathcal{C}_{\delta^*(\mathcal{G})-1} \subseteq \ldots \subseteq \mathcal{C}_0 = \mathcal{V}. \tag{3.1}$$

3.2. Scaling GAEs and VGAEs with Graph Degeneracy

In Step 2, we train a GAE or a VGAE model, either only on the $\delta^*(\mathcal{G})$-core version of $\mathcal{G}$, or on a larger k-core subgraph, i.e., for a $k < \delta^*(\mathcal{G})$. Why do we choose to resort to a core decomposition in the context of GAE and VGAE models? Our justification for this strategy is two-fold. The first reason is *computational*: the k-core decomposition can be computed in a linear running time for an undirected graph [24]. Algorithm 1 describes[2] the procedure we adopt in this work. Specifically, to construct a k-core, the strategy is to recursively remove all nodes with degree lower than k and their edges from $\mathcal{G}$ until no node can be removed. It involves sorting nodes by degrees, which can be achieved in $O(n)$ time using a variant of bin-sort, and going through all nodes and edges once (see [24] for details). The time complexity of Algorithm 1 is $O(\max(m, n))$, with $\max(m, n) = m$ in many real-world graphs, and with the same space complexity with sparse matrices, as explained by Batagelj and Zaversnik [24].

Figure 3.2: A graph of degeneracy 3 and its cores. Some nodes are labeled for the purpose of Section 3.2.3.

Algorithm 1 k-core Decomposition
___
Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: Set of k-cores $\mathcal{C} = \{\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_{\delta^*(\mathcal{G})}\}$
1: Initialize $\mathcal{C} = \{\mathcal{V}\}$ and $k = \min_{v \in \mathcal{V}} D_{vv}$
2: **for** $i = 1$ **to** n **do**
3:     v = node with smallest degree in $\mathcal{G}$
4:     **if** $D_{vv} > k$ **then**
5:         Append $\mathcal{V}$ to $\mathcal{C}$
6:         $k = D_{vv}$
7:     **end if**
8:     $\mathcal{V} = \mathcal{V} \setminus \{v\}$ and remove edges linked to v
9: **end for**
___

Our second reason to rely on the k-core decomposition is that, despite being relatively simple, it has been proven to be a very *useful tool to extract representative subgraphs* over the past years, with numerous applications ranging from community detection [105] to keyword extraction in graph-of-words [357] and core-based kernels for graph similarity [272]. Our work, therefore, builds upon these successes. We refer to the recent survey of Malliaros et al. [247] for a broader overview of the history, theory, extensions, and applications of core decomposition.

On the Selection of k   To select k, one must often face an inherent *performance/speed trade-off*, that we will illustrate in our experiments. Intuitively, reconstructing very small subgraphs during the GAE/VGAE training will speed up computations but, as we will later verify, this might also deteriorate performances. Besides, on large graphs, training a GAE or a VGAE will usually be impossible on the lowest cores (i.e., on the largest subgraphs) due to overly large memory requirements. In our experiments from Section 3.3, we will adopt a simple strategy when dealing with large graphs, consisting in training models on the lowest computationally tractable cores, i.e., on the largest possible subgraphs. In practice, these subgraphs are significantly smaller

___

[2]In Algorithm 1, D_{vv} denotes the *degree* of node v, consistently with the notation of Definition 2.4.

than the original ones (at least 95% of nodes are removed, in the datasets under consideration). Moreover, when running experiments on medium-size graphs where all cores would be tractable, we will plainly avoid choosing $k < 2$ (since $\mathcal{V} = \mathcal{C}_0 = \mathcal{C}_1$, or $\mathcal{C}_0 \approx \mathcal{C}_1$, in all our graphs). As we will show, setting $k = 2$, i.e., removing *leaves* from the graph, will often empirically appear as a good option, preserving performances w.r.t. models trained on $\mathcal{G}$ while significantly reducing running times by pruning up to 50% of nodes in our graphs.

3.2.3 Propagation of Embedding Vectors

From Steps 1 and 2, we obtain embedding vectors $z_i \in \mathbb{R}^d$ for each node $i \in \mathcal{C}_k$ of a pre-selected k-core. Step 3 consists in the inference of such representations for the remaining nodes of $\mathcal{G}$, in a scalable way. We recall that we consider featureless nodes in this section.

Our strategy starts by assigning embedding representations to nodes directly connected to the k-core. We average the values of their embedded neighbors *and* of the nodes being embedded at the same step of the process. For instance, in the graph of Figure 3.2, to compute z_D and z_E we would solve the system:

$$\begin{cases} z_D = \frac{1}{2}(z_A + z_E) \\ z_E = \frac{1}{3}(z_B + z_C + z_D) \end{cases} \tag{3.2}$$

or a weighted mean, if edges are weighted. Then, we would repeat this process on the neighbors of these newly embedded nodes, and so on until no new node would be reachable. Taking into account the fact that nodes D and E are themselves connected is important. Indeed, node A from the maximal core is also a second-order neighbor of E. Exploiting such a proximity when computing z_E empirically improves performances, as it also strongly impacts all the following nodes whose latent vectors will then be derived from z_E, i.e., nodes F, G and H in Figure 3.2.

More generally, let $\mathcal{V}_1$ denote the set of nodes whose embedding vectors are already computed, $\mathcal{V}_2$ the set of nodes connected to $\mathcal{V}_1$ and without embedding vectors, A_1 the $|\mathcal{V}_1| \times |\mathcal{V}_2|$ adjacency matrix linking nodes from $\mathcal{V}_1$ to nodes from $\mathcal{V}_2$, and A_2 the $|\mathcal{V}_2| \times |\mathcal{V}_2|$ adjacency matrix of nodes in $\mathcal{V}_2$. We normalize A_1 and A_2 by the total degree in $\mathcal{V}_1 \cup \mathcal{V}_2$, i.e., we divide rows by row sums of the $(A_1^T | A_2)$ matrix row-concatenating A_1^T and A_2. We denote by $\tilde{A}_1$ and $\tilde{A}_2$ these normalized versions. We already learned the $|\mathcal{V}_1| \times d$ embedding matrix Z_1 for nodes in $\mathcal{V}_1$. To implement our strategy, we want to derive a $|\mathcal{V}_2| \times d$ embedding matrix Z_2 for nodes in $\mathcal{V}_2$, verifying:

$$Z_2 = \tilde{A}_1 Z_1 + \tilde{A}_2 Z_2. \tag{3.3}$$

The solution of this system is:

$$Z^* = (I - \tilde{A}_2)^{-1} \tilde{A}_1 Z_1, \tag{3.4}$$

which exists since $(I - \tilde{A}_2)$ is *strictly diagonally dominant* and thus *invertible* from the Levy-

3.3. Scaling GAEs and VGAEs with Graph Degeneracy

Algorithm 2 Propagation of Embedding Vectors

Input: Graph $\mathcal{G}$, list of embedded nodes $\mathcal{V}_1$, node embedding matrix $Z_1 \in \mathbb{R}^{|\mathcal{V}_1| \times d}$ (already learned), number of iterations t

Output: A d-dimensional node embedding vector for each node in $\mathcal{G}$

1: $\mathcal{V}_2 =$ set of non-embedded nodes reachable from $\mathcal{V}_1$
2: **while** $|\mathcal{V}_2| > 0$ **do**
3:     $A_1 = |\mathcal{V}_1| \times |\mathcal{V}_2|$ adjacency matrix connecting nodes from $\mathcal{V}_1$ to nodes from $\mathcal{V}_2$
4:     $A_2 = |\mathcal{V}_2| \times |\mathcal{V}_2|$ adjacency matrix connecting nodes from $\mathcal{V}_2$ together
5:     $\tilde{A}_1, \tilde{A}_2 =$ normalized versions of A_1, A_2, with rows divided by row sums of $(A_1^T | A_2)$
6:     Randomly initialize $Z_2 \in \mathbb{R}^{|\mathcal{V}_2| \times d}$ *(rows of Z_2 will be embedding vectors of nodes in $\mathcal{V}_2$)*
7:     **for** $i = 1$ **to** t **do**
8:         $Z_2 = \tilde{A}_1 Z_1 + \tilde{A}_2 Z_2$
9:     **end for**
10:    $\mathcal{V}_1 = \mathcal{V}_2$
11:    $\mathcal{V}_2 =$ set of non-embedded nodes reachable from $\mathcal{V}_1$
12: **end while**
13: Assign random vectors to remaining unreachable nodes

Desplanques theorem [353]. Unfortunately, the exact computation of Z^* has a cubic complexity. We approximate it by initializing Z_2 with random values in $[-1, 1]$ and iterating:

$$Z_2 \leftarrow \tilde{A}_1 Z_1 + \tilde{A}_2 Z_2 \tag{3.5}$$

until convergence to a *fixed point*, guaranteed to happen exponentially fast, as stated below.

Proposition 3.3. Let us denote by $Z^{(t)}$ the $|\mathcal{V}_2| \times f$ matrix obtained by iterating $Z^{(t)} = \tilde{A}_1 Z_1 + \tilde{A}_2 Z^{(t-1)}$ t times starting from some random initial matrix $Z^{(0)}$. Let $\| \cdot \|_F$ be the Frobenius matrix norm. Then, exponentially fast,

$$\|Z^{(t)} - Z^*\|_F \xrightarrow[t \to +\infty]{} 0, \tag{3.6}$$

where Z^* is the optimal solution from Equation (3.4).

We prove Proposition 3.3 in Section 3.5. Our propagation process is summarized in Algorithm 2. If some nodes are unreachable by such a process because $\mathcal{G}$ is not connected, then we assign them random vectors. Using sparse representations for $\tilde{A}_1$ and $\tilde{A}_2$, the memory requirement for Algorithm 2 is $O(m + nf)$, and the computational complexity of each evaluation of line 7 also increases linearly w.r.t. the number of edges m in the graph. In practice, t is small: we set $t = 10$ in our experiments (we illustrate the impact of t in Section 3.5). The number of iterations in the while loop of line 2 corresponds to the size of the longest shortest-path connecting a node to the k-core, a number bounded above by the diameter of the graph which increases at a $O(\log(n))$ speed in numerous real-world graphs [47]. In the next section, we will empirically confirm that both Steps 1 and 3 run in linear time, and therefore scale to large graphs.

3.3   Experimental Analysis

In this section, we empirically evaluate our degeneracy framework for scalable graph autoencoders. Although all main results and conclusions are presented and discussed here, we report additional and more complete tables in the "supplementary" Section 3.5 for the sake of brevity. An implementation of our framework is publicly available on GitHub[3].

3.3.1   Experimental Setting

Datasets   In this chapter, we provide experimental results on five graphs of increasing sizes. First and foremost, for comparison purposes, we study the three medium-size citation networks[4] used by Kipf and Welling [187]: Cora ($n = 2\ 708$ and $m = 5\ 429$), Citeseer ($n = 3\ 327$ and $m = 4\ 732$), and Pubmed ($n = 19\ 717$ and $m = 44\ 338$) [323]. In these graphs, nodes are documents and edges are citation links. As Kipf and Welling [187], we ignored edges' directions in experiments, i.e., we considered undirected versions of these graphs (we refer to our Chapter 5 for an extension of GAE and VGAE models to directed graphs). Documents/nodes have sparse bag-of-words feature vectors, of sizes 3 703, 1 433, and 500 respectively. Each document also has a class label corresponding to its topic: in Cora (respectively in Citeseer, in Pubmed), nodes are clustered in six classes (resp. in seven classes, in three classes) that we use as ground truth communities in the following. Classes are roughly balanced. These datasets are common benchmarks for evaluating GAEs and VGAEs [124, 135, 187, 279, 329, 359, 361, 370] (we discuss the relevance and the limitations of these benchmarks in Chapter 6). For these three medium-size graphs, we can directly compare the performance of our framework to standard GAEs and VGAEs, as training these standard models is still computationally affordable.

Then, we consider two larger graphs from Stanford's SNAP project [212]. The first one is the Google web graph[5] ($n = 875\ 713$ and $m = 4\ 322\ 051$). Nodes are web pages and edges represent hyperlinks between these pages. Data do not include ground truth communities. The second one is the US Patent citation network[6] ($n = 2\ 745\ 762$ and $m = 13\ 965\ 410$), originally released by the National Bureau of Economic Research (NBER) and representing citations between patents. Nodes belong to one out of six patent categories, that act as ground truth communities; we removed nodes without categories from the original Patent graph. For both graphs, we once again ignored edges' directions.

For these five graphs, we illustrate in Figure 3.3 the number of nodes in each k-core subgraph. We note that, for Google (resp. for Patent), we were unable to train autoencoders on the 0 to 15-cores (resp. on the 0 to 13-cores) in our machines due to memory errors. Overall, using

---

[3]https://github.com/deezer/linear_graph_autoencoders
[4]https://linqs.soe.ucsc.edu/data
[5]https://snap.stanford.edu/data/web-Google.html
[6]https://snap.stanford.edu/data/cit-Patents.html
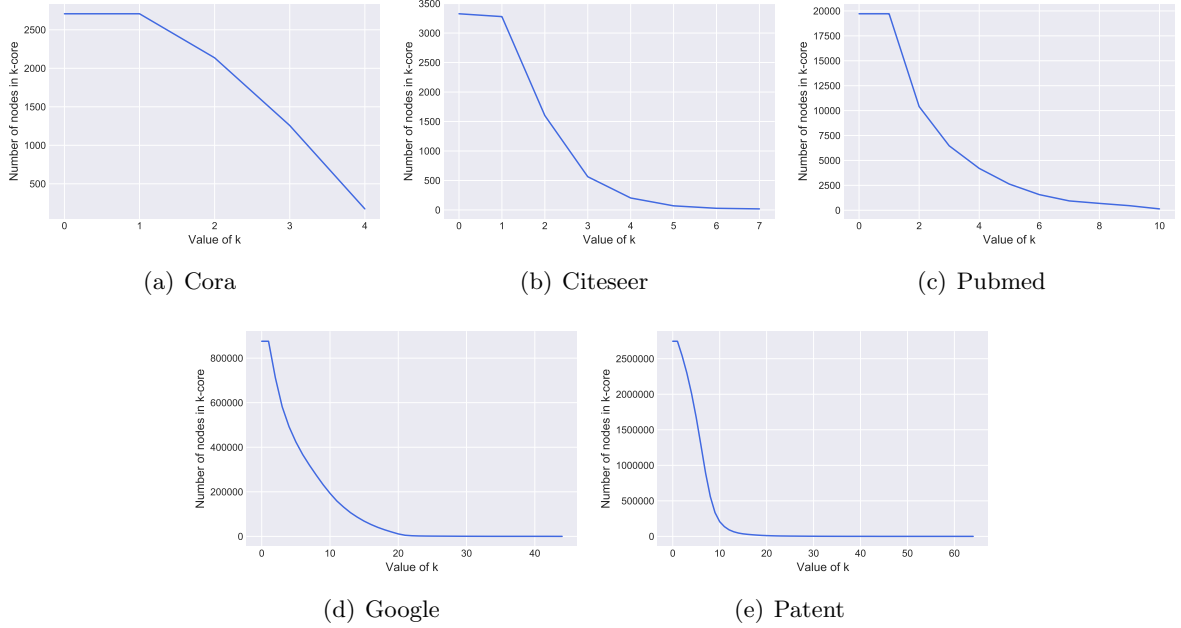
Figure 3.3: Core decomposition of the Cora, Citeseer, Pubmed, Google, and Patent graphs.

our NVIDIA GTX 1080 GPU, we encountered memory errors during the GAE/VGAE training when we were trying to decode graphs with more than 35 000 nodes.

Tasks We consider two downstream tasks for evaluation. Firstly, we consider *link prediction*, following the methodology described in Section 2.1.2. We train all models described thereafter on masked graphs for which 15% of edges were randomly removed for medium-size graphs (resp. 5% for large graphs). Then, we create validation and test sets from the removed edges (resp. from 5% and 10% of edges for medium-size graphs, and resp. 2% and 3% for large graphs) and from the same number of randomly sampled unconnected node pairs. Using the decoder's predictions $\hat{A}_{ij}$, we evaluate the model's ability to classify edges from non-edges, using the mean *Area Under the ROC Curve* (AUC) and *Average Precision* (AP) scores (see Section 2.1.2) on test sets. As explained in Section 2.4.3, link prediction is the most common task to evaluate GAE and VGAE models in the literature. We, therefore, found it essential to consider it as well in our work.

We also consider *community detection* experiments, on datasets with ground truth communities. For this task, after training models on complete versions of the graphs, we run k-means algorithms [14] in node embedding spaces to cluster the z_i vectors. We compare these clusters to the ground truth using the mean *Adjusted Mutual Information* (AMI) score (see Section 2.1.2).

Models We apply our framework to ten graph autoencoders: the seminal GAE and VGAE models from Kipf and Welling [187] with 2-layer GCN encoders and inner product decoder; two deeper variants of GAE and VGAE with 3-layer GCN encoders, denoted DeepGAE and

DeepVGAE; the Graphite and Variational Graphite extended models from Grover et al. [124] incorporating a reverse message passing scheme; Pan et al.'s adversarially regularized variants of GAE and VGAE (denoted ARGA and ARVGA) [279]; and two other models, denoted ChebAE and ChebVAE, and consisting in variants of GAE and VGAE with ChebNets [69] of order 3 instead of GCN encoders. All models were trained on 200 epochs to return 16-dimensional embedding vectors, except for Patent (32-dimensional embedding vectors). We included a 32-dimensional hidden layer in GCN encoders (two for the DeepGAE and DeepVGAE models), used the Adam optimizer [184], training models without dropout [343] and with a learning rate of 0.01. We performed full-batch gradient descent[7] and used the reparameterization trick from Kingma and Welling [185], described in Section 2.4.2. We used the public implementations of these models (see their respective references).

We also compare our results to the DeepWalk [287], LINE [351] and node2vec [123] methods mentioned in Section 2.2.3. We focus on these methods because they directly claim scalability. For each model, hyperparameters were tuned on AUC scores using validation sets. For DeepWalk [287], we trained models from 10 random walks of length 80 per node with a window size of 5, on a single epoch for each graph. We used similar hyperparameters for node2vec [123], setting $p = q = 1$, and LINE [351] enforcing second-order proximity. We used the public implementations provided by the authors. Due to underperforming results on some graphs with 16-dimensional embeddings, we had to increase dimensions, up to 64, to compete with autoencoders. We also implemented a Laplacian eigenmaps/spectral clustering baseline when this approach was tractable (embedding axes are the first 64 eigenvectors of $\mathcal{G}$'s Laplacian matrix) as well as the Louvain method [31] for community detection.

For all models, we used Python[8] and especially the TensorFlow library [1], training models on an NVIDIA GTX 1080 GPU and running other operations on a double Intel Xeon Gold 6134 CPU.

3.3.2 Results and Discussion

Medium-Size Graphs For Cora, Citeseer, and Pubmed, we apply our framework to all possible subgraphs from the 2-core to the $\delta^*(\mathcal{G})$-core and on entire graphs, which is still computationally tractable. Table 3.1 reports mean AUC and AP scores and their standard errors on 100 runs (masked edges are different for each run[9]) along with mean running times, for the link prediction task on Pubmed with the VGAE model from Kipf and Welling [187]. Sizes of k-cores vary over runs due to the edge masking process in link prediction; this phenomenon does not

---

[7]In these experiments, we omitted two simple heuristics to approximate losses, consisting in 1) reconstructing "mini-batches" of nodes/edges randomly (uniformly) sampled, and 2) incorporating negative sampling techniques [187]. These two methods will be presented in detail in Chapter 4, where they will be considered as baselines (including on the same graphs as experiments from this chapter). We will show that they usually fail to effectively scale GAE/VGAE models while preserving performances, for reasons that will be discussed in Chapter 4.

[8]Our code notably builds upon Thomas Kipf's implementation of GAE models: https://github.com/tkipf/gae

[9]Excluding the validation set, which is only extracted once. Validation edges never appear in test sets.

3.3. Experimental Analysis

Table 3.1: Link prediction on the Pubmed graph ($n = 19\,717$, $m = 44\,338$), using the VGAE model from Kipf and Welling [187] with 2-layer GCN encoders and inner product decoder, its k-core variants using our degeneracy framework, and other baselines. All VGAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 100 runs. **Bold** numbers correspond to the best scores and best running time. Scores *in italic* are within one standard deviation range from the best score.

| Model | Size of input k-core | Mean Perf. on Test Set | | Mean Running Times (in sec.) | | | | |
		AUC (in %)	AP (in %)	k-core dec.	Model train	Propagation	Total	Speed gain
VGAE on $\mathcal{G}$	-	83.02 ± 0.13	$\mathbf{87.55 \pm 0.18}$	-	710.54	-	710.54	-
on 2-core	$9,277 \pm 25$	$\mathbf{83.97 \pm 0.39}$	85.80 ± 0.49	1.35	159.15	0.31	160.81	$\times 4.42$
on 3-core	$5,551 \pm 19$	*83.92 ± 0.44*	85.49 ± 0.71	1.35	60.12	0.34	61.81	$\times 11.50$
on 4-core	$3,269 \pm 30$	82.40 ± 0.66	83.39 ± 0.75	1.35	22.14	0.36	23.85	$\times 29.79$
on 5-core	$1,843 \pm 25$	78.31 ± 1.48	79.21 ± 1.64	1.35	7.71	0.36	9.42	$\times 75.43$
...	...	...	...	...	...	...	...	...
on 8-core	414 ± 89	67.27 ± 1.65	67.65 ± 2.00	1.35	1.55	0.38	3.28	$\times 216.63$
on 9-core	149 ± 93	61.92 ± 2.88	63.97 ± 2.86	1.35	1.14	0.38	$\mathbf{2.87}$	$\times \mathbf{247.57}$
DeepWalk	-	81.04 ± 0.45	84.04 ± 0.51	-	342.25	-	342.25	-
LINE	-	81.21 ± 0.31	84.60 ± 0.37	-	63.52	-	63.52	-
node2vec	-	81.25 ± 0.26	85.55 ± 0.26	-	48.91	-	48.91	-
Laplacian	-	83.14 ± 0.42	86.55 ± 0.41	-	31.71	-	31.71	-

occur for community detection. We report similar tables for other datasets/tasks in Section 3.5.

We observe that our framework significantly improves running times w.r.t. training a VGAE on the entire graph $\mathcal{G}$. Running times decrease when k increases (up to $\times 247.57$ speed gain in Table 3.1), which was expected since the k-core becomes smaller. Overall, we observe this improvement on all other datasets, on both tasks, and for other GAE/VGAE variants (see Section 3.5). In particular, we confirm that Steps 1 and 3 of our framework, i.e., the core decomposition and the propagation of embedding representations, are computationally efficient. Simultaneously, for low cores, and especially for 2-cores, performances are consistently competitive w.r.t. models trained on entire graphs, and sometimes even slightly better (e.g., +0.95 point in AUC for link prediction on Pubmed in Table 3.1). This highlights the relevance of our propagation process, and the fact that training models on smaller graphs is easier. Training a model on a 2-core actually consists in removing "leaves" in the graph, which might appear as a simple but effective way to reduce noise during the training phase. Training models on higher cores leads to even faster results, but at the price of a loss in performance. This corresponds to what we referred to as the *performance/speed trade-off* in Section 3.2.2.

Large graphs We now consider the two larger graphs. Table 3.2 reports mean AMI scores and standard errors over 10 runs, for community detection on Patent with the VGAE model from Kipf and Welling [187]. Moreover, the two Tables 3.3 and 3.4 provide more summarized results to compare results obtained with the GAE/VGAE variants. Specifically, in Table 3.3, we report link prediction results on Google, reporting performances from all variants trained on the 17-core. In Table 3.4 we report community detection results on Patent, reporting performances from all variants trained on the 15-core. We report more complete tables in Section 3.5.

Table 3.2: Community detection on the Patent graph ($n = 2\,745\,762$, $m = 13\,965\,410$), using the VGAE model from Kipf and Welling [187] with 2-layer GCN encoders and inner product decoder, trained on the 14-core to 18-core subgraphs using our degeneracy framework, and other baselines. All VGAE models learn embedding vectors of dimension $d = 32$. Scores are averaged over 10 runs. We omit two baselines: DeepWalk due to too long running times on our machine, and spectral clustering due to memory errors. **Bold** numbers correspond to the best scores and best running time. Scores *in italic* are within one standard deviation range from the best score.

Model	Size of input	Mean Performance	Mean Running Times (in sec.)			
	k-**core**	**AMI (in %)**	k-core dec.	Model train	Propagation	**Total**
VGAE on 14-core	46 685	**25.22 ± 1.51**	507.08	6 390.37	120.80	7 018.25 (1h57)
on 15-core	35 432	*24.53 ± 1.62*	507.08	2 589.95	123.95	3 220.98 (54min)
on 16-core	28 153	*24.16 ± 1.96*	507.08	1 569.78	123.14	2 200.00 (37min)
on 17-core	22 455	*24.14 ± 2.01*	507.08	898.27	124.02	1 529.37 (25min)
on 18-core	17 799	22.54 ± 1.98	507.08	**551.83**	126.67	**1 185.58 (20min)**
LINE	-	23.19 ± 1.82	-	33 063.80	-	33 063.80 (9h11)
Louvain	-	11.99 ± 1.79	-	13 634.16	-	13 634.16 (3h47)
node2vec	-	*24.10 ± 1.64*	-	26 126.01	-	26 126.01 (7h15)

Overall, we reach similar conclusions w.r.t. medium-size graphs, both in terms of good performance and of scalability. However, comparison with standard models trained on $\mathcal{G}$, i.e., without using our framework, is impossible on these graphs due to overly large memory requirements. We therefore only compare performances obtained from several computationally tractable cores, illustrating once again the inherent performance/speed trade-off when choosing k and validating previous insights: increasing k accelerates training times, but tends to decrease performances.

Comparison to Baselines   While it is impossible to compare our results to GAE and VGAE models trained on $\mathcal{G}$, we observe that our framework provides competitive results w.r.t. other popular scalable baselines. Our framework is faster on large graphs while achieving comparable or outperforming performances in most experiments (e.g., +1.12 AMI point for our VGAE on 14-core in Table 3.2 w.r.t. node2vec, which is also slower). These competitive performances on large graphs emphasize the relevance of pursuing research towards more scalable graph autoencoders. On the other hand, we note that several of these baselines, notably Louvain and node2vec, are better to cluster nodes in Cora and Pubmed (+10 points in AMI for Louvain on Cora in Section 3.5) which questions the global ability of existing GAE or VGAE models to identify communities in a robust way. In Chapter 6, we will provide an in-depth investigation of the relative limitations of GAE and VGAE models on community detection.

GAE/VGAE variants   For both tasks, we observe that the adversarial training strategy adopted by ARGA/ARGVA, as well as Graphite's variant decoding schemes, and ChebNet-based encoders tend to slightly improve predictions w.r.t. standard GAE and VGAE models. However, results are often very close to those obtained from the standard 2-layer GCN-based GAE and VGAE models. This questions the relevance of overcomplexifying these graph autoencoders. In Chapter 5, we will instead consider *simpler* yet effective variants of GAEs and VGAEs.

Table 3.3: Link prediction on the Google graph ($n = 875\ 713$, $m = 4\ 322\ 051$) using all GAE/VGAE variants trained on the 17-core ($|\mathcal{C}_{17}| = 23\ 787 \pm 208$) using our degeneracy framework, and the best baseline. Scores are averaged over 10 runs. **Bold** numbers correspond to the best scores and best running time. Scores *in italic* are within one standard deviation range from the best score.

Model	Perf. on Test Set		Total
(using framework, k=17)	AUC (in %)	AP (in %)	run. time
GAE	94.02 ± 0.20	94.31 ± 0.21	23min
VGAE	93.22 ± 0.40	93.20 ± 0.45	**22 min**
DeepGAE	93.74 ± 0.17	92.94 ± 0.33	24min
DeepVGAE	93.12 ± 0.29	92.71 ± 0.29	24min
Graphite	93.29 ± 0.33	93.11 ± 0.42	23min
Var-Graphite	93.13 ± 0.35	92.90 ± 0.39	**22 min**
ARGA	93.82 ± 0.17	94.17 ± 0.18	23min
ARVGA	93.00 ± 0.17	93.38 ± 0.19	23min
ChebGAE	$\mathbf{95.24 \pm 0.26}$	$\mathbf{96.94 \pm 0.27}$	41min
ChebVGAE	*95.03 ± 0.25*	*96.58 ± 0.21*	40min
node2vec on $\mathcal{G}$	*94.89 ± 0.63*	*96.82 ± 0.72*	4h06
(best baseline)			

Table 3.4: Community detection on the Patent graph ($n = 2\ 745\ 762$, $m = 13\ 965\ 410$) using all GAE/VGAE variants trained on the 15-core ($|\mathcal{C}_{15}| = 35\ 432 \pm 208$) using our degeneracy framework, and the best baseline. Scores are averaged over 10 runs. **Bold** numbers correspond to the best score and best running time. Scores *in italic* are within one standard deviation range from the best score.

Model	Performance	Total
(using framework, k=15)	AMI (in %)	run. time
GAE	*23.76 ± 2.25*	56min
VGAE	*24.53 ± 1.51*	**54min**
DeepGAE	*24.27 ± 1.10*	1h01
DeepVGAE	*24.54 ± 1.23*	58min
Graphite	*24.22 ± 1.45*	59min
Var-Graphite	*24.25 ± 1.51*	58min
ARGA	*24.26 ± 1.18*	1h01
ARVGA	*24.76 ± 1.32*	58min
ChebGAE	*25.23 ± 1.21*	1h41
ChebVGAE	$\mathbf{25.30 \pm 1.22}$	1h38
node2vec on $\mathcal{G}$	*24.10 ± 1.64*	7h15
(best baseline)		

Furthermore, in this study, we did not observe any clear empirical distinction between deterministic and variational autoencoders. For instance, while various deterministic GAEs outperform their variational counterparts in Table 3.3 (e.g., 94.02% AUC for a standard GAE vs 93.22% for a VGAE), variational models reach better results in Table 3.4 (e.g., 24.53% AMI for a standard VGAE vs 23.76% for a GAE).

Besides, while we mainly focused on featureless graphs, our framework easily extends to *attributed graphs*, by adding features from the k-core as input of GAE/VGAE models. In this direction, we report experiments on GAE and VGAE models *with node features* (when available, i.e., for Cora, Citeseer, and Pubmed) for both tasks in Section 3.5. The addition of node features improves scores (e.g., from 85.24% to 88.10% AUC for a GAE trained of the 2-core of Cora).

Limitations   Despite these promising results, our framework still suffers from several limitations. For instance, in the experiments mentioned in the previous paragraph, features were not included in Step 3's propagation. Future work might study more efficient strategies to integrate node features. Moreover, we noticed that our performances tend to deteriorate when our framework is applied to the highest cores, i.e., the smallest subgraphs. In addition, our work implicitly assumes the existence of a tractable core subgraph, which might not always exist in practice. For instance, if a graph $\mathcal{G}$ has a k-core subgraph too large for GAE/VGAE training, and a $(k + 1)$-core too small (or even empty), our degeneracy framework will fail to provide relevant node embedding representations. Experiments from Chapter 4 will consider this case.

3.4 Conclusion

In this chapter, we presented a general framework to scale graph autoencoders and variational graph autoencoders. This framework leverages graph degeneracy concepts to train models only from a dense subset of nodes instead of using the entire graph. Together with a simple yet effective propagation mechanism, our approach improves scalability and training speed while, to some extent, preserving performances on link prediction and community detection.

We evaluated our framework on ten variants of existing GAE and VGAE, providing the first application of these models to large graphs with up to millions of nodes and edges. Last, but not least, we achieved empirically competitive results w.r.t. several popular scalable node embedding methods such as node2vec and DeepWalk, in a majority of our experiments. This emphasizes the relevance of pursuing further research towards more scalable GAE and VGAE models.

Simultaneously, in our experiments, we also identified several limitations of our GAE and VGAE models. Some of them are directly related to our degeneracy framework, such as the potentially suboptimal use of node features and the assumption, sometimes unverified, that the graph under consideration includes tractable core subgraphs. In the next Chapter 4, we will introduce FastGAE, an alternative strategy to scale GAEs and VGAEs, that addresses these limitations.

Some other limitations are not directly related to our degeneracy framework, but rather to the GAE and VGAE models themselves. This includes the inability of these models to reconstruct directed graphs, as well as their relatively lower performance in some community detection experiments. These important aspects will be further discussed and addressed in this thesis, respectively in Chapters 5 and 7.

3.5 Appendices

This supplementary section provides additional tables related to our experiments. We also prove our Proposition 3.3, stated in Section 3.2. They were placed out of the main content of Chapter 3 for the sake of brevity and readability.

Proof of Proposition 3.3

We have:

$$
\begin{aligned}
Z^{(t)} - Z^* &= [\tilde{A}_1 Z_1 + \tilde{A}_2 Z^{(t-1)}] - [\tilde{A}_2 Z^* + (I - \tilde{A}_2)Z^*] \\
&= \tilde{A}_1 Z_1 + \tilde{A}_2 Z^{(t-1)} - \tilde{A}_2 Z^* - (I - \tilde{A}_2)(I - \tilde{A}_2)^{-1}\tilde{A}_1 Z_1 \\
&= \tilde{A}_2(Z^{(t-1)} - Z^*).
\end{aligned}
\tag{3.7}
$$

So, $Z^{(t)} - Z^* = \tilde{A}_2^t(Z^{(0)} - Z^*)$. Then, as a consequence of the Cauchy-Schwarz inequality [344]:

$$\|Z^{(t)} - Z^*\|_F = \|\tilde{A}_2^t(Z^{(0)} - Z^*)\|_F \leq \|\tilde{A}_2^t\|_F\|Z^{(0)} - Z^*\|_F. \tag{3.8}$$

Furthermore, $\tilde{A}_2^t = PD^tP^{-1}$, with $\tilde{A}_2 = PDP^{-1}$ denoting the eigendecomposition of the symmetric matrix $\tilde{A}_2$. For the diagonal matrix D^t we have:

$$\|D^t\|_F = \sqrt{\sum_{i=1}^{|\mathcal{V}_2|}|\lambda_i^t|^2} \leq \sqrt{|\mathcal{V}_2|}(\max_i |\lambda_i|)^t \tag{3.9}$$

with λ_i denoting the i-th eigenvalue of $\tilde{A}_2$. Since $\tilde{A}_2$ has non-negative entries, we derive from the Perron–Frobenius theorem [238] that:

- the maximum absolute value among eigenvalues of $\tilde{A}_2$ is reached by a nonnegative real eigenvalue;

- $\max_i \lambda_i$ is bounded above by the maximum degree in $\tilde{A}_2$'s graph.

By definition, each node in $\mathcal{V}_2$ has at least one connection to $\mathcal{V}_1$. Moreover, rows of $\tilde{A}_2$ are normalized by row sums of $(A_1^T|A_2)$, so the maximum degree in $\tilde{A}_2$'s graph is strictly lower than 1. We conclude, with the above two bullet points, that $0 \leq |\lambda_i| < 1$ for all $i \in \{1, ..., |\mathcal{V}_2|\}$, so $0 \leq \max_i |\lambda_i| < 1$. This result implies that:

$$\|D^t\|_F \to_t 0 \tag{3.10}$$

exponentially fast, and so does $\|\tilde{A}_2^t\|_F \leq \|P\|_F\|D^t\|_F\|P^{-1}\|_F$, and then $\|Z^{(t)} - Z^*\|_F$.

Additional Tables

Link Prediction   Tables 3.5 to 3.9 provide more complete results for the link prediction task. For medium-size graphs, we apply our framework to all possible subgraphs from the 2-core to the $\delta^*(\mathcal{G})$-core and on entire graphs for comparison, for the VGAE model with 2-layer GCN encoders and inner product decoder [187]. Sizes of k-cores vary over runs due to the edge masking process. We obtained comparable performance/speed trade-offs for its GAE counterpart and for other GAE/VGAE variants: for the sake of brevity, we therefore only report results on 2-core for these models. Also, we only report the best baseline for the sake of brevity, and refer to the IJCAI paper [308] for exhaustive results. For the Google and Patent graphs, comparison with full models on $\mathcal{G}$ is impossible due to overly large memory requirements. As a consequence, we apply our framework to the five largest k-cores (in terms of number of nodes) that were tractable using our machines. We report mean AUC and AP and their standard errors on 10 runs (train incomplete graphs and masked edges are different for each run) along with mean

(a) Pubmed: VGAE model trained on 8-core

(b) Google: VGAE model trained on 18-core
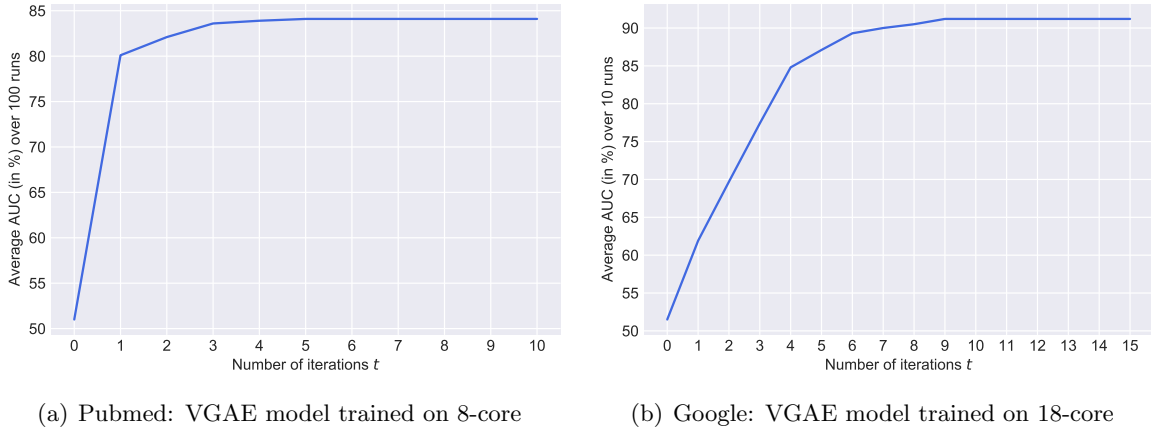
Figure 3.4: Impact of the number of iterations t during progagation on mean AUC scores.

running times, for the standard VGAE model [187]. For other models, we only report results on the second largest cores for the sake of brevity. We chose the second largest cores (17-core for Google, 14-core for Patent), instead of the largest cores to lower running times.

Community Detection Tables 3.10 to 3.12 provide more complete results for the community detection task. As explained in Section 3.4, we run k-means over embedding vectors and report AMI scores w.r.t. ground truth communities. We used scikit-learn's implementation [284] with k-means++ initialization [17]. We do not report results for the Google graph, due to the lack of ground truth communities. Also, we obtained very low scores on Citeseer, which suggests that node features are more useful than the graph structure to explain labels. As a consequence, we also omit this graph here (community detection on Citesser will nonetheless be considered later in this thesis, in Chapters 4 and 7) and focus on Cora, Pubmed, and Patent in Tables 3.10 to 3.12. Tables are constructed in a similar fashion w.r.t. the previous ones for link prediction. Graph AE/VAE models and baselines were trained with identical hyperparameters w.r.t. *link prediction* task (which we will question in Chapter 7). As link prediction, we only report the best baseline in tables for the sake of brevity, and refer to the IJCAI paper [308] for exhaustive results.

Impact of the number of iterations t To finish, we illustrate the impact of the number of iterations t during propagation on performances in Figure 3.4. We display the evolution of mean AUC on two different graphs and cores w.r.t. the value of t. Overall, all scores stabilize by setting $t > 5$ (resp. $t > 10$) in all medium-size graphs (resp. all large graphs). We specify that the number of iterations has a negligible impact on running time. In our experiments, we set $t = 10$ for all models leveraging our degeneracy framework.

---

[10]In this Table 3.5 and in the following ones, we chose not to report numbers in *italic*, contrary to Section 3.2. This is due to the fact that several "best" scores are now presented (one for each table subsection), making the *italic* notation from Section 3.2 potentially ambiguous in these tables.

3.5. Appendices

Table 3.5: Link prediction on the Cora graph ($n = 2\,708$, $m = 5\,429$), using the standard VGAE [187] model on all cores, and GAE/VGAE variants on the 2-core. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 100 runs. **Bold** numbers correspond to the best scores, in each table subsection[10].

Model	Size of input k-**core**	Mean Perf. on Test Set		Mean Running Times (in sec.)			
		AUC (in %)	AP (in %)	k-core dec.	Model train	Propagation	Total
VGAE on $\mathcal{G}$	-	84.07 ± 1.22	$\mathbf{87.83 \pm 0.95}$	-	15.34	-	15.34
on 2-core	$1\,890 \pm 16$	$\mathbf{85.24 \pm 1.12}$	87.37 ± 1.13	0.16	8.00	0.10	8.26
on 3-core	862 ± 26	84.53 ± 1.33	85.04 ± 1.87	0.16	2.82	0.11	3.09
on 4-core	45 ± 13	72.33 ± 4.67	71.98 ± 4.97	0.16	$\mathbf{0.98}$	0.12	$\mathbf{1.26}$
GAE on 2-core	$1\,890 \pm 16$	85.17 ± 1.02	87.26 ± 1.12	0.16	8.05	0.10	8.31
DeepGAE on 2-core	$1\,890 \pm 16$	86.25 ± 0.81	87.92 ± 0.78	0.16	8.24	0.10	8.50
DeepVGAE on 2-core	$1\,890 \pm 16$	86.16 ± 0.95	87.71 ± 0.98	0.16	8.20	0.10	8.46
Graphite on 2-core	$1\,890 \pm 16$	86.35 ± 0.82	88.18 ± 0.84	0.16	9.41	0.10	9.67
Var-Graphite on 2-core	$1\,890 \pm 16$	$\mathbf{86.39 \pm 0.84}$	88.05 ± 0.80	0.16	9.35	0.10	9.61
ARGA on 2-core	$1\,890 \pm 16$	85.82 ± 0.88	88.22 ± 0.70	0.16	7.99	0.10	8.25
ARVGA on 2-core	$1\,890 \pm 16$	85.74 ± 0.74	88.14 ± 0.74	0.16	7.98	0.10	8.24
ChebGAE on 2-core	$1\,890 \pm 16$	86.15 ± 0.54	88.01 ± 0.39	0.16	15.78	0.10	16.04
ChebVGAE on 2-core	$1\,890 \pm 16$	86.30 ± 0.49	$\mathbf{88.29 \pm 0.50}$	0.16	15.65	0.10	15.91
GAE with node features on 2-core	$1\,890 \pm 16$	$\mathbf{88.10 \pm 0.87}$	89.36 ± 0.88	0.16	8.66	0.10	8.92
VGAE with node features on 2-core	$1\,890 \pm 16$	87.97 ± 0.99	$\mathbf{89.53 \pm 0.96}$	0.16	8.60	0.10	8.86
Laplacian eigenmaps (*best baseline*)	-	$\mathbf{86.53 \pm 1.02}$	$\mathbf{87.41 \pm 1.12}$	-	2.78	-	2.78

Table 3.6: Link prediction on the Citeseer graph ($n = 3\,327$, $m = 4\,732$), using the standard VGAE [187] model on all cores*, and GAE/VGAE variants on the 2-core. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 100 runs. **Bold** numbers correspond to the best scores, in each table subsection. * 6-core and 7-core are not reported due to their frequent vanishing after edge masking.

Model	Size of input k-**core**	Mean Perf. on Test Set		Mean Running Times (in sec.)			
		AUC (in %)	AP (in %)	k-core dec.	Model train	Propagation	Total
VGAE on $\mathcal{G}$	-	$\mathbf{78.10 \pm 1.52}$	$\mathbf{83.12 \pm 1.03}$	-	22.40	-	22.40
on 2-core	$1\,306 \pm 19$	77.50 ± 1.59	81.92 ± 1.41	0.15	4.72	0.11	4.98
on 3-core	340 ± 13	76.40 ± 1.72	80.22 ± 1.42	0.15	1.75	0.14	2.04
on 4-core	139 ± 13	73.34 ± 2.43	75.49 ± 2.39	0.15	1.16	0.16	1.47
on 5-core	46 ± 10	65.47 ± 3.16	68.50 ± 2.77	0.15	$\mathbf{0.99}$	0.16	$\mathbf{1.30}$
GAE on 2-core	$1\,306 \pm 19$	78.35 ± 1.51	82.44 ± 1.32	0.15	4.78	0.11	5.04
DeepGAE on 2-core	$1\,306 \pm 19$	$\mathbf{79.32 \pm 1.39}$	82.80 ± 1.33	0.15	4.99	0.11	5.25
DeepVGAE on 2-core	$1\,306 \pm 19$	78.52 ± 1.02	82.43 ± 0.97	0.15	4.95	0.11	5.21
Graphite on 2-core	$1\,306 \pm 19$	78.61 ± 1.58	82.81 ± 1.24	0.15	5.88	0.11	6.14
Var-Graphite on 2-core	$1\,306 \pm 19$	78.51 ± 1.62	82.72 ± 1.25	0.15	5.86	0.11	6.12
ARGA on 2-core	$1\,306 \pm 19$	78.89 ± 1.33	82.89 ± 1.03	0.15	4.54	0.11	4.80
ARVGA on 2-core	$1\,306 \pm 19$	77.98 ± 1.39	82.39 ± 1.09	0.15	4.40	0.11	4.66
ChebGAE on 2-core	$1\,306 \pm 19$	78.62 ± 0.95	83.22 ± 0.89	0.15	8.87	0.11	9.13
ChebVGAE on 2-core	$1\,306 \pm 19$	78.75 ± 1.03	$\mathbf{83.23 \pm 0.76}$	0.15	8.75	0.11	9.01
GAE with node features on 2-core	$1\,306 \pm 19$	81.21 ± 1.86	$\mathbf{83.99 \pm 1.52}$	0.15	5.51	0.11	5.77
VGAE with node features on 2-core	$1\,306 \pm 19$	$\mathbf{81.88 \pm 2.23}$	83.83 ± 1.85	0.15	5.70	0.11	5.96
Laplacian eigenmaps (*best baseline*)	-	$\mathbf{80.56 \pm 1.41}$	$\mathbf{83.98 \pm 1.08}$	-	3.77	-	3.77

Table 3.7: Link prediction on the Pubmed graph ($n = 19\,717$, $m = 44\,338$), using the standard VGAE [187] model on all cores*, and GAE/VGAE variants on the 2-core. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 100 runs. **Bold** numbers correspond to the best scores, in each table subsection. * 10-core is not reported due to its frequent vanishing after edge masking.

Model	Size of input k-core	Mean Perf. on Test Set		Mean Running Times (in sec.)			
		AUC (in %)	AP (in %)	k-core dec.	Model train	Propagation	**Total**
VGAE on $\mathcal{G}$	-	83.02 ± 0.13	$\mathbf{87.55 \pm 0.18}$	-	710.54	-	710.54
on 2-core	$9\,277 \pm 25$	$\mathbf{83.97 \pm 0.39}$	85.80 ± 0.49	1.35	159.15	0.31	160.81
on 3-core	$5\,551 \pm 19$	83.92 ± 0.44	85.49 ± 0.71	1.35	60.12	0.34	61.81
on 4-core	$3\,269 \pm 30$	82.40 ± 0.66	83.39 ± 0.75	1.35	22.14	0.36	23.85
on 5-core	$1\,843 \pm 25$	78.31 ± 1.48	79.21 ± 1.64	1.35	7.71	0.36	9.42
...	...	...	...	...	...	...	...
on 8-core	414 ± 89	67.27 ± 1.65	67.65 ± 2.00	1.35	1.55	0.38	3.28
on 9-core	149 ± 93	61.92 ± 2.88	63.97 ± 2.86	1.35	$\mathbf{1.14}$	0.38	$\mathbf{2.87}$
GAE on 2-core	$9\,277 \pm 25$	84.30 ± 0.27	$\mathbf{86.11 \pm 0.43}$	1.35	167.25	0.31	168.91
DeepGAE on 2-core	$9\,277 \pm 25$	84.61 ± 0.54	85.18 ± 0.57	1.35	166.38	0.31	168.04
DeepVGAE on 2-core	$9\,277 \pm 25$	84.46 ± 0.46	85.31 ± 0.45	1.35	157.43	0.31	159.09
Graphite on 2-core	$9\,277 \pm 25$	84.51 ± 0.58	85.65 ± 0.58	1.35	167.88	0.31	169.54
Var-Graphite on 2-core	$9\,277 \pm 25$	84.30 ± 0.57	85.57 ± 0.58	1.35	158.16	0.31	159.82
ARGA on 2-core	$9\,277 \pm 25$	84.37 ± 0.54	86.07 ± 0.45	1.35	164.06	0.31	165.72
ARVGA on 2-core	$9\,277 \pm 25$	84.10 ± 0.53	85.88 ± 0.41	1.35	155.83	0.31	157.49
ChebGAE on 2-core	$9\,277 \pm 25$	$\mathbf{84.63 \pm 0.42}$	86.05 ± 0.70	1.35	330.37	0.31	332.03
ChebVGAE on 2-core	$9\,277 \pm 25$	84.54 ± 0.48	86.00 ± 0.63	1.35	320.01	0.31	321.67
GAE with node features on 2-core	$9\,277 \pm 25$	84.94 ± 0.54	85.83 ± 0.58	1.35	168.62	0.31	170.28
VGAE with node features on 2-core	$9\,277 \pm 25$	$\mathbf{85.81 \pm 0.68}$	$\mathbf{88.01 \pm 0.53}$	1.35	164.10	0.31	165.76
Laplacian eigenmaps (*best baseline*)	-	$\mathbf{83.14 \pm 0.42}$	$\mathbf{86.55 \pm 0.41}$	-	31.71	-	31.71

Table 3.8: Link prediction on the Google graph ($n = 875\,713$, $m = 4\,322\,051$), using the standard VGAE [187] model trained on the 16 to 20-cores, and GAE/VGAE variants on the 17-core. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 10 runs. **Bold** numbers correspond to the best scores, in each table subsection.

Model	Size of input k-core	Mean Perf. on Test Set		Mean Running Times (in sec.)			
		AUC (in %)	AP (in %)	k-core dec.	Model train	Propagation	Total
VGAE on 16-core	$36\,854 \pm 132$	$\mathbf{93.56 \pm 0.38}$	$\mathbf{93.34 \pm 0.31}$	301.16	2 695.42	25.54	3 022.12 (50min)
on 17-core	$23\,787 \pm 208$	93.22 ± 0.40	93.20 ± 0.45	301.16	1 001.64	28.16	1 330.86 (22min)
on 18-core	$13\,579 \pm 75$	91.24 ± 0.40	92.34 ± 0.51	301.16	326.76	28.20	656.12 (11min)
on 19-core	$6\,613 \pm 127$	87.79 ± 0.31	89.13 ± 0.29	301.16	82.19	28.59	411.94 (7min)
on 20-core	$3\,589 \pm 106$	81.74 ± 1.17	83.51 ± 1.22	301.16	$\mathbf{25.59}$	28.50	$\mathbf{355.55}$ (6 min)
GAE on 17-core	$23\,787 \pm 208$	94.02 ± 0.20	94.31 ± 0.21	301.16	1 073.18	28.16	1 402.50 (23min)
DeepGAE on 17-core	$23\,787 \pm 208$	93.74 ± 0.17	92.94 ± 0.33	301.16	1 137.24	28.16	1 466.56 (24min)
DeepVGAE on 17-core	$23\,787 \pm 208$	93.12 ± 0.29	92.71 ± 0.29	301.16	1 088.41	28.16	1 417.73 (24min)
Graphite on 17-core	$23\,787 \pm 208$	93.29 ± 0.33	93.11 ± 0.42	301.16	1 033.21	28.16	1 362.53 (23min)
Var-Graphite on 17-core	$23\,787 \pm 208$	93.13 ± 0.35	92.90 ± 0.39	301.16	989.90	28.16	1 319.22 (22min)
ARGA on 17-core	$23\,787 \pm 208$	93.82 ± 0.17	94.17 ± 0.18	301.16	1 053.95	28.16	1 383.27 (23min)
ARVGA on 17-core	$23\,787 \pm 208$	93.00 ± 0.17	93.38 ± 0.19	301.16	1 027.52	28.16	1 356.84 (23min)
ChebGAE on 17-core	$23\,787 \pm 208$	$\mathbf{95.24 \pm 0.26}$	$\mathbf{96.94 \pm 0.27}$	301.16	2 120.66	28.16	2 449.98 (41min)
ChebVGAE on 17-core	$23\,787 \pm 208$	95.03 ± 0.25	96.82 ± 0.72	301.16	2 086.07	28.16	2 415.39 (40min)
node2vec (*best baseline*)	-	$\mathbf{94.89 \pm 0.63}$	$\mathbf{96.82 \pm 0.72}$	-	14 762.78	-	14 762.78 (4h06)

3.5. Appendices

Table 3.9: Link prediction on the Patent graph ($n = 2\,745\,762$, $m = 13\,965\,410$), using the standard VGAE [187] model trained on the 14 to 18-cores, and GAE/VGAE variants on the 15-core. All GAE/VGAE models learn embedding vectors of dimension $d = 32$. Scores are averaged over 10 runs. **Bold** numbers correspond to the best scores, in each table subsection.

Model	Size of input k-core	Mean Perf. on Test Set		Mean Running Times (in sec.)			
		AUC (in %)	AP (in %)	k-core dec.	Model train	Propagation	Total
VGAE on 14-core	$38\,408 \pm 147$	$\mathbf{88.48 \pm 0.35}$	$\mathbf{88.81 \pm 0.32}$	507.08	3 024.31	122.29	3 653.68 (1h01)
on 15-core	$29\,191 \pm 243$	88.16 ± 0.50	88.37 ± 0.57	507.08	1 656.46	123.47	2 287.01 (38min)
on 16-core	$23\,132 \pm 48$	87.85 ± 0.47	88.02 ± 0.48	507.08	948.09	124.26	1 579.43 (26min)
on 17-core	$18\,066 \pm 143$	87.34 ± 0.56	87.64 ± 0.47	507.08	574.25	126.55	1 207.88 (20min)
on 18-core	$13\,972 \pm 86$	87.27 ± 0.55	87.78 ± 0.51	507.08	$\mathbf{351.73}$	127.01	$\mathbf{985.82}$ **(16min)**
GAE on 15-core	$29\,191 \pm 243$	87.59 ± 0.29	87.30 ± 0.28	507.08	1 880.11	123.47	2 510.66 (42min)
DeepGAE on 15-core	$29\,191 \pm 243$	87.71 ± 0.31	87.64 ± 0.19	507.08	2 032.15	123.47	2 662.70 (44min)
DeepVGAE on 15-core	$29\,191 \pm 243$	87.03 ± 0.54	87.20 ± 0.44	507.08	1 927.33	123.47	2 557.88 (43min)
Graphite on 15-core	$29\,191 \pm 243$	85.19 ± 0.38	86.01 ± 0.31	507.08	1 989.72	123.47	2 620.27 (44min)
Var-Graphite on 15-core	$29\,191 \pm 243$	85.37 ± 0.30	86.07 ± 0.24	507.08	1 916.79	123.47	2 547.34 (42min)
ARGA on 15-core	$29\,191 \pm 243$	$\mathbf{89.22 \pm 0.10}$	$\mathbf{89.40 \pm 0.11}$	507.08	2 028.46	123.47	2 659.01 (44min)
ARVGA on 15-core	$29\,191 \pm 243$	87.18 ± 0.17	87.39 ± 0.33	507.08	1 915.53	123.47	2 546.08 (42min)
ChebGAE on 15-core	$29\,191 \pm 243$	88.53 ± 0.20	88.91 ± 0.20	507.08	3 391.01	123.47	4 021.56 (1h07)
ChebVGAE on 15-core	$29\,191 \pm 243$	88.75 ± 0.19	89.07 ± 0.24	507.08	3 230.52	123.47	3 861.07 (1h04)
node2vec (*best baseline*)	-	$\mathbf{95.04 \pm 0.25}$	$\mathbf{96.01 \pm 0.19}$	-	26 126.01	-	26 126.01 (7h15)

Table 3.10: Community detection on the Cora graph ($n = 2\,708$, $m = 5\,429$), using the standard VGAE [187] model trained on all cores, and GAE/VGAE variants on the 2-core. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 100 runs. **Bold** numbers correspond to the best scores, in each table subsection.

Model	Size of input k-core	Mean Performance AMI (in %)	Mean Running Times (in sec.)			
			k-core dec.	Model train	Propagation	Total
VGAE on $\mathcal{G}$	-	29.52 ± 2.61	-	15.34	-	15.34
on 2-core	2 136	34.08 ± 2.55	0.16	9.94	0.10	10.20
on 3-core	1 257	$\mathbf{36.29 \pm 2.52}$	0.16	4.43	0.11	4.70
on 4-core	174	35.93 ± 1.88	0.16	$\mathbf{1.16}$	0.12	$\mathbf{1.44}$
VGAE with node features on $\mathcal{G}$	-	$\mathbf{47.25 \pm 1.80}$	-	15.89	-	15.89
on 2-core	2 136	45.09 ± 1.91	0.16	10.42	0.10	10.68
on 3-core	1 257	40.96 ± 2.06	0.16	4.75	0.11	5.02
on 4-core	174	38.11 ± 1.23	0.16	1.22	0.12	1.50
GAE on 2-core	2 136	34.91 ± 2.51	0.16	10.02	0.10	10.28
DeepGAE on 2-core	2 136	35.30 ± 2.52	0.16	10.12	0.10	10.38
DeepVGAE on 2-core	2 136	34.49 ± 2.85	0.16	10.09	0.10	10.35
Graphite on 2-core	2 136	33.91 ± 2.17	0.16	10.97	0.10	11.23
Var-Graphite on 2-core	2 136	33.89 ± 2.13	0.16	10.91	0.10	11.17
ARGA on 2-core	2 136	34.73 ± 2.84	0.16	9.99	0.10	10.25
ARVGA on 2-core	2 136	33.36 ± 2.53	0.16	9.97	0.10	10.23
ChebGAE on 2-core	2 136	$\mathbf{36.52 \pm 2.05}$	0.16	19.22	0.10	19.48
ChebVGAE on 2-core	2 136	37.83 ± 2.11	0.16	20.13	0.10	20.39
Louvain (*best baseline*)	-	$\mathbf{46.76 \pm 0.82}$	-	1.83	-	1.83

Table 3.11: Community detection on the Pubmed graph ($n = 19\,717$, $m = 44\,338$), using the standard VGAE [187] model trained on all cores, and GAE/VGAE variants on the 2-core. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 100 runs. **Bold** numbers correspond to the best scores, in each table subsection.

Model	Size of input k-core	Mean Performance AMI (in %)	k-core dec.	Model train	Propagation	Total
VGAE on $\mathcal{G}$	-	22.36 ± 0.25	-	707.77	-	707.77
on 2-core	10 404	23.71 ± 1.83	1.35	199.07	0.30	200.72
on 3-core	6 468	$\mathbf{25.19 \pm 1.59}$	1.35	79.26	0.34	80.95
on 4-core	4 201	24.67 ± 3.87	1.35	34.66	0.35	36.36
on 5-core	2 630	17.90 ± 3.76	1.35	14.55	0.36	16.26
...	...	...	...	...	...	...
on 10-core	137	10.79 ± 1.16	1.35	1.15	0.38	2.88
VGAE with node features on $\mathcal{G}$	-	$\mathbf{26.05 \pm 1.40}$	-	708.59	-	708.59
on 2-core	10 404	24.25 ± 1.92	1.35	202.37	0.30	204.02
on 3-core	6 468	23.26 ± 3.42	1.35	82.89	0.34	84.58
on 4-core	4 201	20.17 ± 1.73	1.35	36.89	0.35	38.59
on 5-core	2 630	18.15 ± 2.04	1.35	16.08	0.36	17.79
...	...	...	...	...	...	...
on 10-core	137	11.67 ± 0.71	1.35	$\mathbf{0.97}$	0.38	$\mathbf{2.70}$
GAE on 2-core	10 404	22.76 ± 2.25	1.35	203.56	0.30	205.21
DeepGAE on 2-core	10 404	24.53 ± 3.30	1.35	205.11	0.30	206.76
DeepVGAE on 2-core	10 404	25.63 ± 3.51	1.35	200.73	0.30	202.38
Graphite on 2-core	10 404	26.55 ± 2.17	1.35	209.12	0.30	210.77
Var-Graphite on 2-core	10 404	$\mathbf{26.69 \pm 2.21}$	1.35	200.86	0.30	202.51
ARGA on 2-core	10 404	23.68 ± 3.18	1.35	207.50	0.30	209.15
ARVGA on 2-core	10 404	25.98 ± 1.93	1.35	199.94	0.30	201.59
ChebGAE on 2-core	10 404	25.88 ± 1.66	1.35	410.81	0.30	412.46
ChebVGAE on 2-core	10 404	26.50 ± 1.49	1.35	399.96	0.30	401.61
node2vec (*best baseline*)	-	$\mathbf{29.57 \pm 0.22}$	-	48.91	-	48.91

Table 3.12: Community detection on the Patent graph ($n = 2\,745\,762$, $m = 13\,965\,410$), using the standard VGAE [187] model trained on the 14 to 18 cores, and GAE/VGAE variants on 15-core. All GAE/VGAE models learn embedding vectors of dimension $d = 32$. Scores are averaged over 10 runs. **Bold** numbers correspond to the best scores, in each table subsection.

Model	Size of input k-core	Mean Performance AMI (in %)	k-core dec.	Model train	Propagation	Total
VGAE on 14-core	46 685	$\mathbf{25.22 \pm 1.51}$	507.08	6 390.37	120.80	7 018.25 (1h57)
on 15-core	35 432	24.53 ± 1.62	507.08	2 589.95	123.95	3 220.98 (54min)
on 16-core	28 153	24.16 ± 1.96	507.08	1 569.78	123.14	2 200.00 (37min)
on 17-core	22 455	24.14 ± 2.01	507.08	898.27	124.02	1 529.37 (25min)
on 18-core	17 799	22.54 ± 1.98	507.08	$\mathbf{551.83}$	126.67	$\mathbf{1\,185.58}$ **(20min)**
GAE on 15-core	35 432	23.76 ± 2.25	507.08	2 750.09	123.95	3 381.13 (56min)
DeepGAE on 15-core	35 432	24.27 ± 1.10	507.08	3 007.31	123.95	3 638.34 (1h01)
DeepVGAE on 15-core	35 432	24.54 ± 1.23	507.08	2 844.16	123.95	3 475.19 (58min)
Graphite on 15-core	35 432	24.22 ± 1.45	507.08	2 899.87	123.95	3 530.90 (59min)
Var-Graphite on 15-core	35 432	24.25 ± 1.51	507.08	2 869.92	123.95	3 500.95 (58min)
ARGA on 15-core	35 432	24.26 ± 1.18	507.08	3 013.28	123.95	3 644.31 (1h01)
ARVGA on 15-core	35 432	24.76 ± 1.32	507.08	2 862.54	123.95	3 493.57 (58min)
ChebGAE on 15-core	35 432	25.23 ± 1.21	507.08	5 412.12	123.95	6 043.15 (1h41)
ChebVGAE on 15-core	35 432	$\mathbf{25.30 \pm 1.22}$	507.08	5 289.91	123.95	5 920.94 (1h38)
node2vec (*best baseline*)	-	$\mathbf{24.10 \pm 1.64}$	-	26 126.01	-	26 126.01 (7h15)

4

Scalable Graph Autoencoders with Stochastic Subgraph Decoding

This chapter presents research conducted with Romain Hennequin, Jean-Baptiste Remy, Manuel Moussallam, and Michalis Vazirgiannis, and published in Elsevier's Neural Networks journal (impact factor: 8.05) in 2021 [307].

4.1 Introduction

In this chapter, we introduce an alternative stochastic method to scale GAE and VGAE models. This method, referred to as FastGAE in the following, was developed in 2020, i.e., a year after the research presented in Chapter 3. FastGAE constitutes an improvement of our previous efforts towards more scalable graph autoencoders.

More specifically, we propose to leverage graph mining-based sampling schemes and an effective subgraph decoding strategy to significantly lower the computational complexity of graph autoencoders, while preserving or even slightly improving their performances. We previously argued that a random (uniform) node sampling is suboptimal in the context of GAE and VGAE models, which we will experimentally confirm in Section 4.3. In this chapter, we however explain that, by leveraging graph mining techniques, one can also derive more effective sampling schemes that, in essence, aim to reconstruct "wisely selected" random subparts of an original graph during training. We provide an in-depth theoretical and experimental analysis of the proposed solution, showing that it behaves favorably when compared to the degeneracy framework from Chapter 3. We also show that FastGAE addresses some of the limitations of this degeneracy framework.

This chapter is organized as follows. In Section 4.2, we present and analyze FastGAE, our method for scalable graph autoencoders with stochastic subgraph decoding. We report our experimental evaluation of this method and a discussion of our results in Section 4.3, and we conclude in Section 4.4. In Section 4.5, we provide some proofs as well as an additional figure, placed out of the "main" chapter for the sake of brevity and readability.

4.2 FastGAE: Scaling GAE and VGAE with Stochastic Subgraph Decoding

In this section, we introduce our stochastic method to scale GAE and VGAE models. We refer to it as FastGAE when applied to GAEs, and as variational FastGAE when applied to VGAEs.

4.2.1 A Stochastic Subgraph Decoding Strategy

Encoding the Entire Graph... As detailed in Section 3.1 from the previous chapter, the encoding step of GAE and VGAE models can be computationally costly, if the GNN encoders under consideration themselves involve complex operations. Nonetheless, several scalable GNNs have been proposed in the scientific literature. We explain in this same section that GCN models [188] and their scalable extensions [50, 53, 383, 397] can effectively process large graphs.

Throughout this chapter, we therefore rely on these models to *encode all the nodes* from a graph $\mathcal{G}$ into a node embedding space. More precisely, in the following experiments, we implement standard GCN encoders [188] for the sake of simplicity and an easier comparison to the literature. This choice is made without loss of generality. The method described in this section would remain valid for any other encoder producing a node embedding matrix Z, and notably for faster and/or less complex variants of GCNs [50, 53] in the case of very large graphs (e.g., with hundreds of millions or with billions of nodes) for which the $O(n)$ complexity of a forward GCN pass would become unaffordable.

...But Decoding Stochastic Subgraphs However, while computing node embedding vectors through a forward GCN pass is relatively fast, *tuning the weights* of this encoder in the GAE and VGAE settings requires the reconstruction of the entire matrix $\hat{A}$ at each training iteration which, as explained in the previous chapters, suffers from a quadratic complexity and is intractable for large graphs with more than a few thousand nodes and edges.

To overcome this issue, we propose to *approximate* reconstruction losses and ELBO objectives, by computing their values only from wisely selected random subparts of the original graph. More precisely, at each training iteration, we aim to decode a different sampled subgraph of $\mathcal{G}$ with $n_{(S)}$ nodes, with $n_{(S)} < n$ being a fixed parameter. Let $\mathcal{G}_{(S)} = (\mathcal{V}_{(S)}, \mathcal{E}_{(S)})$ be such a sampled

subgraph, with $\mathcal{V}_{(S)} \subset \mathcal{V}$, $|\mathcal{V}_{(S)}| = n_{(S)}$, and with $\mathcal{E}_{(S)}$ denoting the subset of edges connecting the nodes in $\mathcal{V}_{(S)}$. Instead of reconstructing the $n \times n$ matrix $\hat{A}$, we propose to reconstruct the smaller $n_{(S)} \times n_{(S)}$ matrix $\hat{A}_{(S)}$ with:

$$\hat{A}_{(S)ij} = \sigma(z_i^T z_j), \ \ \forall (i,j) \in \mathcal{V}_{(S)}^2, \tag{4.1}$$

and to only consider the quality of $\hat{A}_{(S)}$ w.r.t. its ground truth counterpart $A_{(S)}$, as measured by a cross entropy loss for GAEs, and by an ELBO objective for VGAEs. We propose to use the resulting approximate loss for gradients computations and for GCN weights updates by gradient descent. We draw *a different subgraph $\mathcal{G}_{(S)}$ at each training iteration*, using the sampling methods detailed in the next section.

4.2.2   Node Sampling with Graph Mining

(Naive) Uniform Node Sampling A very simple way to obtain such subgraphs would consist in *uniformly sampling $n_{(S)}$* nodes from the set $\mathcal{V}$ at each training iteration. However, in such a strategy, there is no guarantee that the most important links (or absence of links) from the original graph structure will be preserved in the drawn subgraphs to reconstruct during the training phase. As we will confirm in Section 4.3, this usually significantly impacts the quality of the final node embedding, leading to underperforming performances on downstream evaluation tasks. As a consequence, in the following sections, we propose and study more refined strategies, aiming to *leverage the graph structure* to obtain a more effective sampling.

Node Sampling with Graph Mining We propose to consider alternative sampling methods, that increase the probability of including some particular nodes in the drawn subgraph w.r.t. some others. Let $f : \mathcal{V} \to \mathbb{R}^+$ denote some measure of the relative *importance* of nodes in the graph, obtained through graph mining methods. Assuming such a function is available, we draw inspiration from word sampling in natural language processing [111, 258] and propose to set the probability to pick each node $i \in \mathcal{V}$ as the first element of $\mathcal{V}_{(S)}$ as:

$$p_i = \frac{f(i)^\alpha}{\sum\limits_{j \in \mathcal{V}} (f(j)^\alpha)}, \tag{4.2}$$

with $\alpha \in \mathbb{R}^+$. Then, assuming we sample $n_{(S)}$ *distinct* nodes *without replacement*, each remaining node $i \in \mathcal{V} \setminus \mathcal{V}_{(S)}$ has a probability $p_i / \sum_{j \notin \mathcal{V}_{(S)}} p_j$ to be picked as the second element of $\mathcal{V}_{(S)}$, and so on until $|\mathcal{V}_{(S)}| = n_{(S)}$. The previous division is a simple normalization to ensure that $\sum_{j \notin \mathcal{V}_{(S)}} p_j = 1$ at each sampling step. Alternatively, one could also sample $n_{(S)}$ nodes *with replacement*: it simplifies computations, as sampling probabilities are then independent of previous draws and remain fixed to p_i, but a node could then be drawn several times. We stress

out that, in our implementation, both variants return similar results. We adopt the former.

In a nutshell, important nodes according to f are *more likely to be selected for decoding*, and the hyperparameter α helps sharpening (for $\alpha > 1$) or smoothing (for $\alpha < 1$) the distribution. Setting $\alpha = 0$ leads to the aforementioned uniform node sampling. In our experiments, we will evaluate two importance measures f from graph mining:

- the *degree* of each node, i.e., $f(i) = D_{ii} = \sum_{j \in \mathcal{V}} A_{ij}$ following the notation of Definition 2.4;

- the *core number* of each node, i.e., $f(i) = c(i)$. As presented in Definition 3.1, the k-core version of a graph is its largest subgraph for which every node has a degree higher or equal to k within this subgraph. Here, the core number $c(i)$ of a node i corresponds to the largest value of k for which i is in the k-core. This choice constitutes a more *global* importance measure than the *local* node degree.

Besides their popularity and their complementarity, we also choose to focus on these two metrics for *computational efficiency*. Indeed, contrary to other potential importance metrics based on influence maximization [180], random walks [211] or centrality measures [266], both can be evaluated in a linear $O(m)$ running time [24]. As we will empirically check in Section 4.3, this permits fast and scalable computations of probability distributions, which is crucial for our FastGAE method whose primary objective is scalability. We refer the interested reader to the work of Leskovec and Faloutsos [211] and of Chiericetti et al. [54] for a broader overview of other existing graph sampling methods.

4.2.3 Theoretical Considerations

We now briefly present some theoretical considerations related to FastGAE that, for the sake of readability, will be further developed and proved in the "supplementary" Section 4.5.

On Approximate Losses In the case of degree and core-based sampling strategies, some node pairs from the graph are more likely to appear in subgraphs than others. The probability to draw a node i, or an edge incident to i, increases with p_i and with $f(i)$ for $\alpha > 0$. As a consequence, at each gradient descent iteration, the approximate loss (say $\mathcal{L}^{\text{FastGAE}}$) is *biased* w.r.t. the standard GAE or VGAE loss that would have been computed on $\mathcal{G}$ (say $\mathcal{L}$), i.e., $\mathbb{E}(\mathcal{L}^{\text{FastGAE}}) \neq \mathcal{L}$ in general. For completeness, in Propositions 4.1, 4.2 and 4.3 of Section 4.5, *we provide a theoretical analysis, in which we fully explicit the expected loss* $\mathbb{E}(\mathcal{L}^{\text{FastGAE}})$ *that we actually stochastically optimize in FastGAE, as well as the formal probabilities to sample a given node or node pair at each training iteration*. Moreover, we will show in Section 4.4 that, despite such a bias, optimizing this alternative loss does not deteriorate the quality of node

embeddings. On the contrary, we will provide insights exhibiting the fact that re-weighting node pairs from high degree/core nodes can actually be beneficial.

On the Selection of $n_{(S)}$ When selecting $n_{(S)}$, one faces a performance/speed trade-off, as for our degeneracy framework. Reconstructing very small subgraphs speeds up the training but, as we later verify, this might also deteriorate performances. While we claimed in the previous paragraph that stochastically minimizing $\mathbb{E}(\mathcal{L}^{\text{FastGAE}})$ instead of $\mathcal{L}$ might be beneficial, we also acknowledge that, for small values of $n_{(S)}$, the actual loss $\mathcal{L}^{\text{FastGAE}}$ computed at a given training iteration can significantly deviate from its expectation. In this chapter, we propose to use these deviations as a criterion to select a relevant subgraph size. In Propositions 4.5 and 4.6 of the "supplementary" Section 4.5, we leverage concentration inequalities to derive a theoretically-grounded *threshold size*, denoted $n_{(S)}^*$ in the following, for which, at each training iteration, the deviation between the evaluation of $\mathcal{L}^{\text{FastGAE}}$ for each node and its expectation is proven to be bounded with a high probability. This proposed subgraph size is of the form:

$$n_{(S)}^* = C\sqrt{n} \tag{4.3}$$

where the constant $C > 0$ depends on the deviation magnitude and probability, and is explicitly presented in Section 4.5. Our experiments will confirm the relevance of this choice.

4.2.4 On Complexity and Links to Related Work

Before diving into experiments, we discuss the complexity of FastGAE and its links and differences w.r.t. some other scalable methods, including our degeneracy from Chapter 3.

Complexity of FastGAE As previously detailed, both the GCN encoder and the sampling step of FastGAE have a linear time complexity w.r.t. the number of edges m in the graph. Moreover, our decoder runs in $O(n_{(S)}^2)$ time, with $n_{(S)}$ being significantly smaller than n in practice. In particular, setting $n_{(S)} = n_{(S)}^*$ ensures a $O(n)$ time complexity for decoding (as $n_{(S)}^{*2} = (C\sqrt{n})^2 = C^2 n$) and an overall $O(m+n)$ linear time complexity for a complete FastGAE training iteration. Faster bounds can also be achieved by lowering $n_{(S)}$ or by replacing GCNs with another encoder. Therefore, as we will empirically verify in Section 4.3, our framework is significantly faster and more scalable than standard GAE and VGAE models.

Differences with Related Work At first glance, one might want to compare FastGAE with the methods aiming to scale the GNN/GCN models mentioned in Section 2.3.4. We would like to emphasize that FastGAE is *not* directly comparable to these methods, e.g., to FastGCN [50] that also samples nodes. FastGCN is a GCN-like model, optimized to classify

node labels in a (semi) supervised fashion. It samples the *neighborhood* of each node when averaging vector representations in forward passes. On the contrary, in this chapter, after *full* GCN forward passes, we instead sample *subgraphs to reconstruct*, in order to approximate the *reconstruction* loss/objective of two unsupervised models, in which GCNs are only a building part (the encoder) of a larger framework (the GAE or the VGAE). Both settings therefore address different problems. As explained in Section 4.2.1, methods such as FastGCN could actually be used *in conjunction with* FastGAE, as alternative encoders replacing GCNs.

Furthermore, FastGAE is also more elaborated than data cleaning methods that simply consist in removing some nodes from a graph, e.g., the low-degree ones, to reduce its size. Indeed, in the case of FastGAE with degree sampling, low-degree nodes are still 1) *fully* used in the GCN encoder, and 2) might also appear in *some* subgraphs that we decode (but less often than high-degree nodes). As we leverage new different subgraphs at each iteration, we explore different parts of the *entire* graph during training.

Lastly, we note that *effective subset selection* for faster learning has already provided promising results in the machine learning community [116, 358]. Contrary to these works, we focus on an unsupervised graph-based problem, and our sampling methods remain fixed throughout learning as we rely on graph mining to select $\mathcal{G}_{(S)}$.

Differences with Chapter 3   Overall, FastGAE is more flexible than the degeneracy framework from Chapter 3, and addresses some of its limitations. For instance, as the degeneracy framework reconstructs one of the cores $\mathcal{C}_0, \ldots, \mathcal{C}_{\delta^*(\mathcal{G})}$ during training, it assumes the existence of at least one tractable core subgraph in $\mathcal{G}$. If none of these cores has an appropriate size (because they are all either too large or too small), then it will fail to learn relevant embedding representations. On the contrary, FastGAE does not rely on such an assumption. This method permits reconstructing subgraphs *of any size* $n_{(S)} < n$, where $n_{(S)}$ is a selectable parameter.

In addition, in Chapter 3, we criticized the suboptimal use of node features during the propagation step. On the contrary, in FastGAE, node features are processed exactly as in standard GCN-based GAE and VGAE models. Lastly, FastGAE is conceptually simpler (in essence, the method consists in a "smarter" mini-batch sampling strategy for graph decoding), which we consider being another advantage, as simple solutions often have the most impact.

4.3   Experimental Analysis

In this section, we present an in-depth experimental evaluation of our proposed method to scale GAE and VGAE models. We publicly released the code of FastGAE on GitHub[1].

---

[1] https://github.com/deezer/fastgae

4.3.1 Experimental Setting

Datasets We provide experiments on seven graphs of increasing sizes. As in Chapter 3, we study the Cora, Citeseer, Pubmed, Google, and Patent graphs, presented in Section 3.3.1. In the paper associated with this work [307], we also considered two other large graphs:

- the Youtube social network of users (edges are friendship connections), available on Konect[2], and with $n = 3\ 223\ 589$ nodes and $m = 9\ 375\ 374$ edges;

- a synthetic graph, denoted SBM, generated from a *stochastic block model* which is a generative model for random graphs [2]. In this last graph, by design, nodes are clustered in 100 groups of 1000 nodes, acting as ground truth communities. Two nodes from the same community (resp. from different communities) are connected by an edge with probability 2×10^{-2} (resp. 2×10^{-4}). The SBM graph has $n = 100\ 000$ nodes and $m = 1\ 498\ 844$ edges.

Our evaluation therefore includes graphs with various characteristics, sizes, and from four different families (citation networks, social networks, web graphs, and stochastic block model graphs). As in Chapter 3, we consider undirected versions of these graphs. We refer to Chapter 5 for an extension of GAE and VGAE models to directed graphs.

Tasks We consider the *link prediction* and *community detection* tasks (for nodes with ground truth communities), already described in Section 3.3.1, with a completely similar setting and with the same evaluation metrics.

Models: Standard and FastGAE-based GAE/VGAE In the upcoming experiments, for the seven graphs and the two evaluation tasks, we compare standard GAE and VGAE models (when they are tractable) to FastGAE-based versions of these models. All GAE and VGAE models, with and without FastGAE, were optimized for the *link prediction* task. More specifically, we selected the best sets of hyperparameters in terms of mean AUC scores on validation sets. Instructions to easily run a similar validation are provided in our source code.

We trained models for 200 iterations (resp. 300) for graphs with $n < 100\ 000$ (resp. $n \geq 100\ 000$), and thoroughly checked the convergence of all models for these values (in terms of loss stability in the validation set). Other hyperparameters for these models are described thereafter.

Our encoders are 2-layer GCNs[3] (we tested models with 1 to 3 layers). They include 32-dimensional hidden layers, and 16-dimensional output layer, which means that the dimension

---

[2]https://konect.cc/networks/
[3]Experiments for Chapter 3 considered several different encoders but reported few to no empirical difference w.r.t. 2-layer GCNs. For the sake of brevity and clarity, experiments from this Chapter 4 will only report results obtained from 2-layer GCN encoders.

of embedding vectors is equal to $d = 16$. We emphasize that we also tested models with $d \in \{32, 64, 128\}$, reaching similar conclusions w.r.t. $d = 16$ (the impact of d is further discussed in Section 4.3.4).

Besides, for all models, we used the Adam optimizer [184], without dropout (we tested models with dropout values in $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$). Regarding learning rates for such an optimizer, we tested values from the grid $\{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2\}$. We eventually picked a learning rate of 0.1 for Patent with uniform sampling, and of 0.01 otherwise as, once again, these values returned the best mean AUC scores on validation sets. Once again, we mainly used TensorFlow [1], training models on an NVIDIA GTX 1080 GPU, and running other operations on a double Intel Xeon Gold 6134 CPU.

Models: Other Baselines For completeness, we also compare standard GAE/VGAE and FastGAE-based models to the few other existing methods to scale GAE/VGAE:

- we consider a simple *negative sampling* strategy, briefly mentioned by Kipf and Welling [187]. We reconstruct all edges but only $|\mathcal{E}|$ randomly picked unconnected node pairs to compute losses. We leveraged methods made available in PyTorch Geometric [96] to estimate losses, with consistent dropout values, learning rates, and architectures w.r.t. the above models;

- we also compare to the degeneracy framework from Chapter 3, denoted as *Core-GAE* in the next tables. We used our own implementation [308] with optimal values (regarding mean AUC scores on validation sets) for the hyperparameter k detailed in next tables, and with consistent dropout values, learning rates, and architectures w.r.t. the above models;

- besides, while some other sampling ideas were briefly mentioned (as possible extensions) in the recent literature on graph autoencoders [124, 310], they actually consist in particular cases of FastGAE, namely with *uniform* sampling.

Lastly, in addition to an extensive comparison between the different GAE/VGAE models, we also report results obtained with three non GAE/VGAE-based baselines: the Louvain method (for community detection) [31], node2vec[4] [123] and Laplacian eigenmaps [284, 368]. We adopt similar hyperparameters w.r.t. experiments from the previous chapter, with the notable exception that we also set $d = 16$ for Laplacian eigenmaps and node2vec (again, the impact of d is further discussed in Section 4.3.4).

---

[4]We omit comparison to other random walk-based methods DeepWalk [287] and LINE [351] in this chapter, due to quite similar performances w.r.t. node2vec on some of our preliminary tests.

4.3.2 Preliminary Results on High Degree/Core Nodes

Before studying FastGAE we report important insights from preliminary experiments on standard GAE and VGAE models. They motivated the design of our framework and emphasize the relevance of sampling high-degree/core nodes. On the medium-size Cora, Citeseer, and Pubmed graphs, we trained standard GAE and VGAE models, but *tried to mask k nodes and their edges* from the computation of reconstruction losses, for different values of k. Such a masking procedure is expected to lower performances, as the model leverages less information about the quality of the reconstruction for learning.

Figure 4.1 shows that, when these k removed nodes are the *top-k highest degrees/cores nodes*, performances on the link prediction task tumble down. On the contrary, removing the *k nodes with minimal degrees or core numbers* from the loss leads to almost no drop, and even slightly better results on Pubmed, which suggests that removing non-informative nodes might even be beneficial for learning. In Figure 4.2, we report similar results for community detection. These ablation studies suggest that, when implementing stochastic subgraph decoding strategies for scalability, sampling high-degree/core nodes is indeed crucial to learn meaningful embeddings. FastGAE, which explicitly exploits these structural node properties, and optimizes a reconstruction loss that re-weights high degrees/cores node pairs, behaves consistently w.r.t. such insights.

4.3.3 Results on Medium-Size Graphs

We now evaluate FastGAE and its variational FastGAE variant. Firstly, we focus on *medium-size graphs*. For Cora, Citeseer, and Pubmed, we can compare FastGAE to standard graph autoencoders. The next Table 4.1 details mean AUC and AP scores and standard errors over 100 runs with different train/test splits for link prediction on the (featureless) Pubmed graph with GAE models. For the sake of brevity, we report more summarized results for other medium-size graphs, for VGAE and for community detection, in Table 4.3 and Figure 4.3 (for link prediction) as well as in Table 4.5 (for community detection).

FastGAE vs Standard GAE/VGAE   In Table 4.1, we observe that, for sample sizes roughly 20 times smaller than n, FastGAE models with *degree* and *core* sampling both achieve competitive or even outperforming[5] results w.r.t. standard GAE on Pubmed (e.g., +2.31 AUC points for FastGAE with degree sampling and $n_{(S)} = 5\,000$). Furthermore, FastGAE models are also significantly *faster*: in Table 4.1 for instance, our approach with degree sampling is up to $\times\,252.78$ faster without performance degradation. The additional operation required by our framework,

---

[5]At first glance, the fact that FastGAE sometimes even slightly *outperforms* standard GAE or VGAE models might be surprising. This improvement is actually consistent with recent research on the benefits of mini-batch-based GNNs [151, 302]. It comes from the relevance of the two core and degree-based sampling schemes that we consider and from the stochastic nature of the training, which might tend to avoid local minima more easily [190].

(a) Cora - Degree masking   (b) Citeseer - Degree masking   (c) Pubmed - Degree masking

(d) Cora - Core masking   (e) Citeseer - Core masking   (f) Pubmed - Core masking
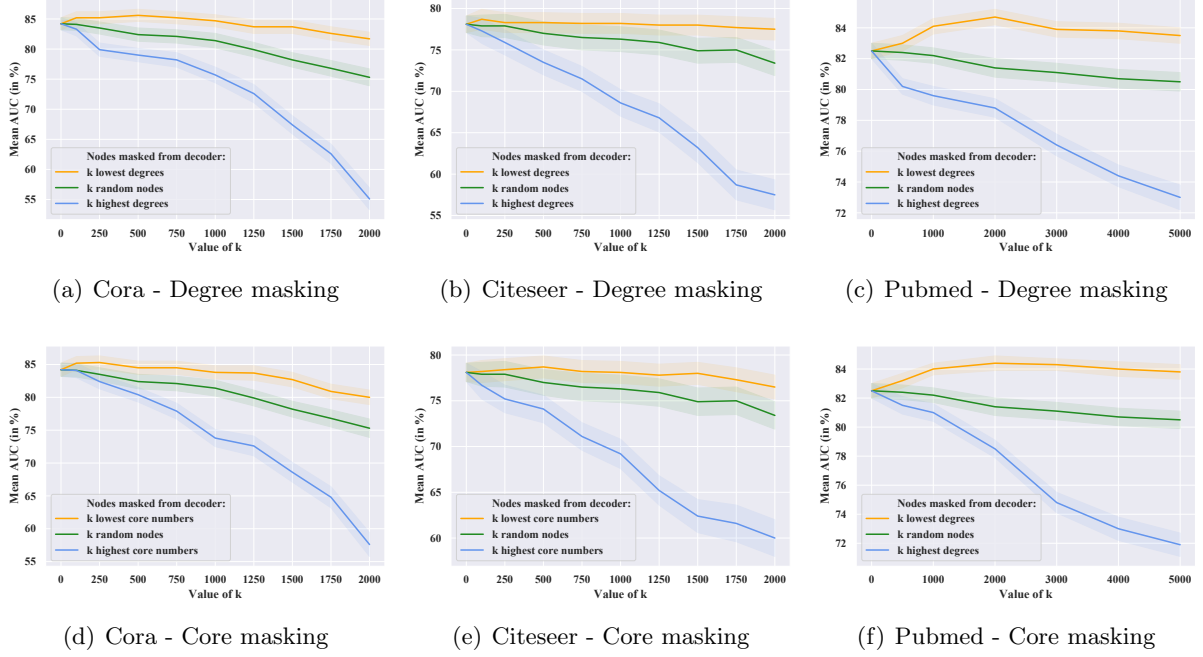
Figure 4.1: Link prediction on the featureless Cora, Citesser and Pubmed graph using standard VGAE models, but trained while masking k nodes and their connections from the decoder/reconstruction loss. AUC scores are averaged over 100 runs with random train/test splits.

(a) Cora - Degree masking   (b) Citeseer - Degree masking   (c) Pubmed - Degree masking

(d) Cora - Core masking   (e) Citeseer - Core masking   (f) Pubmed - Core masking
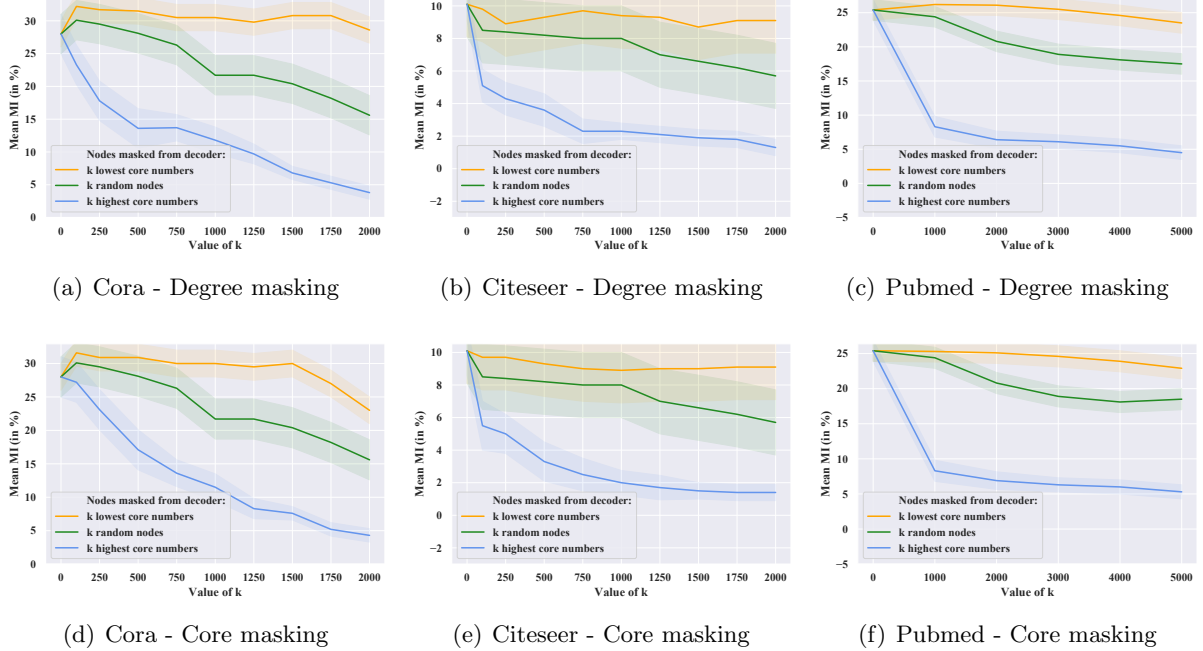
Figure 4.2: Community detection on the featureless Cora, Citesser and Pubmed graphs using standard VGAE models, but trained while masking k nodes and their connections from the decoder/reconstruction loss. AMI scores are averaged over 100 runs with random train/test splits.

4.3. Experimental Analysis

Table 4.1: Link prediction on the featureless Pubmed graph ($n = 19\,717$, $m = 44\,338$) using a standard GAE, FastGAE with degree, core and uniform sampling, and other baselines. For degree and core sampling, values of the hyperparameter α (as defined in Equation (4.2)) were tuned as illustrated in Figure 4.4. All GAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 100 runs. **Bold** numbers correspond to the best scores and best running time. Scores *in italic* are within one standard deviation range from the best ones. Subgraphs sizes annotated with * correspond to the $n_{(S)}^*$ threshold, as introduced in Equation (4.3).

Model	Subgraphs size $n_{(S)}$	Average Perf. on Test Set		Average Running Times (in seconds)			
		AUC (in %)	AP (in %)	Compute p_i	Train model	**Total**	Speed gain w.r.t. GAE
Standard GAE	-	82.51 ± 0.64	87.42 ± 0.38	-	811.43	811.43	-
FastGAE with **degree** sampling ($\alpha = 1$)	5 000	$\mathbf{84.82 \pm 0.32}$	$\mathbf{88.19 \pm 0.23}$	0.01	14.41	14.42	$\times\,56.27$
	2 500	84.12 ± 0.40	87.56 ± 0.30	0.01	5.72	5.73	$\times\,141.61$
	1 187*	83.67 ± 0.42	87.01 ± 0.31	0.01	3.20	3.21	$\times\,252.78$
	500	82.68 ± 0.51	85.89 ± 0.47	0.01	2.98	2.99	$\times\,271.38$
	250	80.77 ± 0.55	84.05 ± 0.51	0.01	2.83	2.84	$\times\,285.71$
FastGAE with **core** sampling ($\alpha = 2$)	5 000	*84.62 ± 0.24*	*88.09 ± 0.16*	1.75	15.98	17.73	$\times\,45.77$
	2 500	83.69 ± 0.34	87.28 ± 0.31	1.75	7.51	9.26	$\times\,87.63$
	1 187*	82.53 ± 0.46	86.28 ± 0.37	1.75	4.81	6.56	$\times\,123.69$
	500	80.96 ± 0.52	84.86 ± 0.46	1.75	4.57	6.32	$\times\,128.39$
	250	79.53 ± 0.53	83.10 ± 0.50	1.75	4.44	6.19	$\times\,131.08$
FastGAE with **uniform** sampling	5 000	81.08 ± 0.48	85.90 ± 0.60	-	13.90	13.90	$\times\,58.37$
	2 500	78.72 ± 0.74	83.50 ± 0.75	-	5.48	5.48	$\times\,148.07$
	1 187*	77.28 ± 0.89	81.89 ± 0.91	-	3.10	3.10	$\times\,261.75$
	500	75.09 ± 2.05	78.53 ± 2.04	-	2.98	2.98	$\times\,271.29$
	250	74.12 ± 2.07	77.72 ± 1.22	-	2.82	$\mathbf{2.82}$	$\mathbf{\times\,287.74}$
Core-GAE, $k = 2$ (best choice)	-	84.30 ± 0.27	86.11 ± 0.43	-	168.91	168.91	$\times\,4.80$
Core-GAE, $k = 9$ (fastest choice)	-	61.65 ± 0.94	64.82 ± 0.72	-	2.92	2.92	$\times\,277.89$
Negative sampling GAE	-	81.19 ± 0.68	83.21 ± 0.40	-	111.79	111.79	$\times\,7.28$
node2vec	-	81.25 ± 0.26	85.55 ± 0.26	-	48.91	48.91	$\times\,16.59$
Laplacian eigenmaps	-	83.14 ± 0.42	86.55 ± 0.41	-	31.71	31.71	$\times\,25.59$

i.e., computing the p_i distribution, is efficient in practice, especially for degree sampling. By further reducing the subgraph size $n_{(S)}$, one can achieve even faster results, while only losing a few AUC/AP points in performance.

In Table 4.3, Table 4.5 and Figure 4.3, we consolidate our results by reaching similar conclusions on VGAE, on other medium-size graphs (with and without features) and on community detection. In Figure 4.3, we also illustrate that, even for relatively low $n_{(S)}/n$ proportions, our proposed method achieves comparable performances w.r.t. baselines.

Comparison of Uniform, Core-based, and Degree-based FastGAE In all our experiments, we observe that FastGAE with core and degree sampling both outperform FastGAE (and variational FastGAE) with uniform sampling. Furthermore, core and degree sampling also return more stable scores, i.e., with lower standard errors, especially when the number of samples $n_{(S)}$ is relatively small. Such results confirm the empirical superiority of strategies that leverage the graph structure w.r.t. pure random strategies.

| (a) Pubmed | (b) Google | (c) Youtube |
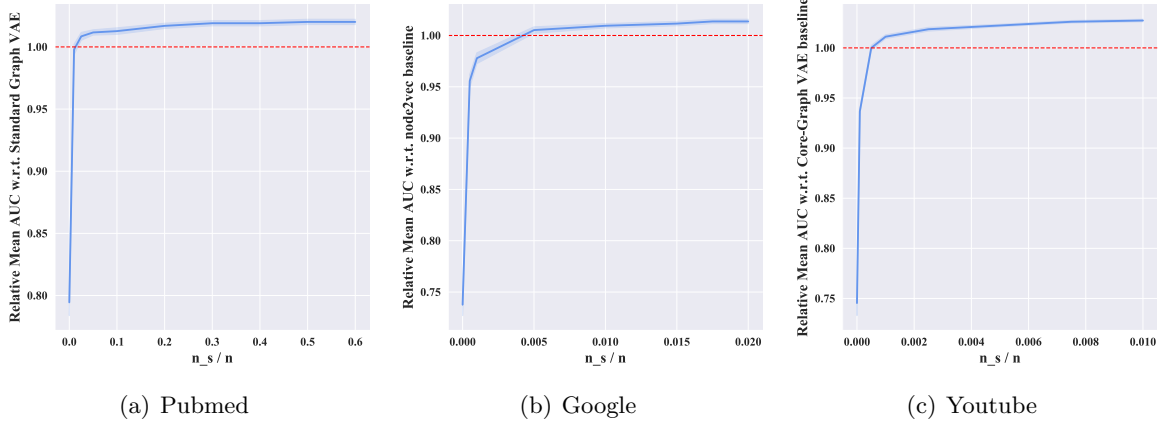
Figure 4.3: Summarized results for link prediction on the featureless Pubmed, Google and Youtube graphs: relative mean AUC scores of degree-based Variational FastGAE models w.r.t. standard VGAE (for Pubmed) or w.r.t. the best scalable baseline (for Google and Youtube) depending on the proportion of sampled nodes $n_{(S)}/n$ in decoders. Even for relatively low $n_{(S)}/n$ proportions, Variational FastGAE achieves comparable or slightly better performances w.r.t. a standard VGAE or to the best scalable baseline (results above the red line). The FastGAE paper [307] provides similar figures for all other graphs, reaching comparable conclusions.

FastGAE vs Baselines In Table 4.1, Table 4.3 and Table 4.5, these models also outperform the other few existing methods to scale GAE and VGAE, usually by a wide margin. For instance, in Table 4.1, we show that, to achieve (almost) comparable link prediction performances w.r.t. FastGAE on Pubmed, our degeneracy framework *Core-GAE* [308] requires longer running times (see *Core-GAE* with $k = 2$), and that faster variants significantly underperform (almost -20 AUC points for *Core-GAE* with $k = 9$ w.r.t. FastGAE with degree sampling). As previously explained, FastGAE is also conceptually simpler than Core-GAE, which we consider to be another advantage of our approach.

Besides, FastGAE-based models are faster and more effective than the ones leveraging negative sampling [96] (e.g., +3.63 AUC points for FastGAE with degree sampling and $n_{(S)} = 20\,000$ w.r.t. *Negative sampling GAE* in Table 4.1). This performance gain might be explained by the more systematic inclusion of *unconnected pairs of important nodes*[6] in the losses of FastGAE-based models. Last, but not least, our proposed framework is also competitive w.r.t. the popular non GAE/VGAE-based baselines in most cases. The only exception concerns the community detection experiments on Cora and Citeseer (see Table 4.5) where the Louvain baseline [31] outperforms GAE/VGAE models, which we discuss thereafter.

---

[6]Indeed, when performing negative sampling for GAE, we only reconstruct a few random unconnected node pairs, and ignore the others. However, reconstructing some of these neglected pairs might actually be crucial. Let us consider two nodes with high core number or centrality: knowing that these two important nodes are *not* connected is critical to learn meaningful embeddings. The FastGAE sampling scheme ensures a more systematic inclusion of these important "negative pairs" in the decoding step than negative sampling.

4.3. Experimental Analysis

Table 4.2: Link prediction on the Patent graph ($n = 2\,745\,762$, $m = 13\,965\,410$), using FastGAE with degree, core and uniform sampling, and other baselines. The standard GAE is intractable for this graph. For degree and core sampling, values of the hyperparameter α (as defined in Equation (4.2)) were tuned as illustrated in Figure 4.4. All GAE models learn embedding vectors of dimension $d = 16$. Scores are averaged over 10 runs. **Bold** numbers correspond to the best scores and best running time. Scores *in italic* are within one standard deviation range from the best one. Subgraphs sizes annotated with * correspond to the $n^*_{(S)}$ threshold, as introduced in Equation (4.3).

Model	Subgraphs size $n_{(S)}$	Average Perf. on Test Set		Average Running Times (in seconds)		
		AUC (in %)	AP (in %)	Compute p_i	Train model	Total
Standard GAE	-	*(intractable)*		*(intractable)*		
FastGAE with **degree** sampling ($\alpha = 2$)	20 000	*92.91 ± 0.22*	*93.35 ± 0.21*	0.30	4 401.67	4 401.97 (1h13)
	16 425*	**93.02 ± 0.23**	**93.39 ± 0.23**	0.30	3 693.32	3 693.62 (1h02)
	10 000	91.76 ± 0.23	91.74 ± 0.21	0.30	1 164.22	1 164.52 (19 min)
	2 500	87.53 ± 0.50	87.42 ± 0.51	0.30	537.99	538.29 (9 min)
	1 000	85.55 ± 0.62	85.96 ± 0.55	0.30	500.12	500.42 (8 min)
FastGAE with **core** sampling ($\alpha = 2$)	20 000	90.71 ± 0.21	91.70 ± 0.19	668.05	4 800.58	5 468.63 (1h31)
	16 425*	90.48 ± 0.21	90.85 ± 0.23	668.05	4 027.90	4 695.95 (1h18)
	10 000	89.08 ± 0.25	88.65 ± 0.24	668.05	1 232.03	1 900.08 (32 min)
	2 500	82.50 ± 0.51	81.42 ± 0.60	668.05	544.64	1 222.69 (20 min)
	1 000	73.99 ± 0.70	75.24 ± 0.74	668.05	503.88	1 171.93 (19 min)
FastGAE with **uniform** sampling	20 000	85.97 ± 0.26	87.71 ± 0.25	-	4 397.89	4 387.89 (1h13)
	16 425*	84.40 ± 0.25	86.11 ± 0.25	-	3 602.66	3 602.66 (1h00)
	10 000	83.77 ± 0.28	83.37 ± 0.26	-	1 106.01	1 106.01 (18 min)
	2 500	70.66 ± 0.35	71.16 ± 0.38	-	485.03	485.03 (8 min)
	1 000	59.34 ± 0.83	58.83 ± 1.30	-	438.02	**438.02 (7 min)**
Core-GAE, $k = 14$ (best choice)	-	88.06 ± 0.27	88.94 ± 0.23	-	4 805.11	4 805.11 (1h20)
Core-GAE, $k = 21$ (fastest choice)	-	86.94 ± 0.69	87.23 ± 0.71	-	619.01	619.01 (10 min)
Negative sampling GAE	-	86.11 ± 0.48	86.70 ± 0.49	-	2 392.96	2 392.96 (40 min)
node2vec	-	*92.96 ± 0.23*	*93.36 ± 0.20*	-	25 851.39	25 851.39 (7h11)
Laplacian eigenmaps	-	*(intractable)*		*(intractable)*		

On the hyperparameter α In the "supplementary" Section 4.5, we report optimal values of α for all graphs. We recall that $\alpha \in \mathbb{R}^+$ is the hyperparameter introduced in Equation (4.2), which helps balance important and "less important" nodes during sampling. Setting $\alpha = 0$ leads to the uniform sampling setting where all nodes are sampled with an equal probability. On the contrary, by setting $\alpha \to \infty$ we would always sample the most important nodes. Experiments from Figure 4.4 in Section 4.5 show that these two extreme cases are usually suboptimal, and that a careful tuning of α (e.g., $\alpha = 2$ for core sampling in Table 4.1) improves performances.

On the threshold $n^*_{(S)}$ In Section 4.2.3, we introduced a theoretically-grounded threshold $n^*_{(S)} = C\sqrt{n}$ to select the subgraph size. Overall, in all our tables, selecting the proposed $n^*_{(S)}$ provided interesting performance/speed trade-offs, leading to fairly competitive results w.r.t. standard GAE/VGAE models and the best baselines, while being significantly faster.

4.3.4 Results on Large Graphs

We now report the evaluation of FastGAE and variational FastGAE on the four large graphs from our experiments: SBM, Google, Youtube, and Patent. The above Table 4.2 details mean

AUC and AP scores and standard errors over 10 runs with different train/test splits for link prediction on the Patent graph with FastGAE. We also report more summarized results (for the sake of brevity) for link prediction on SBM, Google, Youtube, and Patent in Table 4.4, and summarized results for community detection on SBM in Table 4.5. As in Table 4.2, all scores are averaged over 10 runs with different train/test splits for link prediction.

FastGAE vs Scalable GAE/VGAE Baselines On large graphs, a direct comparison with standard GAEs and VGAEs is unfortunately impossible. However, our FastGAE and variational FastGAE models almost always outperform the other existing approaches to scale GAE and VGAE models, usually by a wide margin. For instance, for link prediction on Patent (Table 4.2), degree-based and core-based FastGAE models with $n_{(S)} = 20\ 000$, 16 425 and 10 000 all outperform the best *Core-GAE* by up to roughly 5 AUC points (for degree-based FastGAE with $n_{(S)} = 20\ 000$) and with comparable or better running times. Regarding the *Core-GAE* baseline [308], we also point out that, in one of our large graphs, namely on the SBM one, this method was even *intractable* due to the lack of *size decreasing* core structure on this graph. Indeed, the 21-core of SBM includes 95 200 nodes, which is too large to train a graph AE or VAE on our machines, and the 22-core is empty. Requiring a size decreasing core structure is a drawback of *Core-GAE* w.r.t. the more flexible FastGAE approach, which we already mentioned in Section 4.2.4.

Moreover, as for medium-size graphs, we also observe that core-based and degree-based Fast-GAE tend to significantly outperform negative sampling (e.g., up to +6.8 AUC points for link prediction on Patent in Table 4.2. Also, *Negative sampling GAE* never appears as the best baseline in Table 4.5 nor in Table 4.4), consolidating our previous conclusions. Besides, as before, the proposed $n_{(S)}^*$ provides quite effective performance/speed trade-offs and will constitute an interesting heuristic to help future FastGAE users select subgraph sizes.

Comparison of Uniform, Core-based and Degree-based FastGAE As for medium-size graphs, core-based sampling and degree-based sampling are empirically more effective than uniform sampling (e.g., in Table 4.2, +6.94 AUC points for FastGAE with degree sampling on Patent, with $n_{(S)} = 20\ 000$), and associated with lower standard errors. We observe that computing the p_i probabilities through core-based sampling is longer on large graphs, but brings no empirical benefit w.r.t. degree-based sampling: we therefore recommend using degree-based sampling for large graphs.

FastGAE vs Non-GAE/VGAE Baselines, and the Case of Community Detection For the link prediction task, the best FastGAE models usually reach competitive results w.r.t. node2vec while being significantly faster. However, regarding community detection, we observe in Table 4.5 that the *Louvain* baseline outperforms GAE and VGAE models on SBM, a phe-

nomenon that we also noted on the Cora and Citeseer graphs in the previous section. We conjecture that current graph autoencoders models might be suboptimal to effectively reconstruct communities in graph data. This claim is consistent with some of our experiments from Chapter 3 and elsewhere in the scientific literature [55]. As our objective, in the paper associated with FastGAE [307], was to scale existing GAE and VGAE models, but not to ensure nor to claim their superiority over all other methods for community detection, we did not further investigate this limit in this paper. Nonetheless, in Chapter 7, we will propose a method, referred to as Modularity-Aware Graph Autoencoders [314], to improve the reconstruction of communities from GAE/VGAE-based node embedding spaces.

On the embedding dimension d Our tables present results for $d = 16$ for all methods and all graphs. Nonetheless, we reached similar conclusions for $d = 32$, 64 and 128: although performances sometimes slightly improved by increasing d, the *ranking* of the different models remained unchanged. We also considered re-optimizing d individually for each model as in Chapter 3 (to cover potential cases where the impact of d on the performance of each model would have been different) but, again, it did not modify the ranking of these models in terms of AUC, AP and AMI scores.

On the number of training iterations   As detailed in Section 4.3.1, all GAE and VGAE models, with or without our FastGAE framework, were trained for 200 iterations (resp. 300) for graphs with $n < 100\,000$ (resp. $n \geq 100\,000$). We thoroughly checked the convergence of all models, by assessing the stabilization of performances in terms of AUC scores on validation sets. Using a fixed number of iterations is common in recent research on GAE and VGAE [28, 135, 187]. We nonetheless think that *early-stopping* [109] would also be a relevant alternative strategy, that could lead to additional speed-ups, and might deserve further investigations in future works. Besides, we observed that, for very small values of $n_{(S)}$, increasing the number of training iterations did not significantly improve our results. To improve scores on such settings, increasing the sampling size $n_{(S)}$ was overall more effective than increasing the number of training iterations.

4.4   Conclusion

In this chapter, we introduced FastGAE, a stochastic method to scale GAE and VGAE models. We publicly released our Python/TensorFlow implementation of this method along with the paper associated with this work [307]. We demonstrated its effectiveness on several large graphs with up to millions of nodes and edges, both in terms of speed, scalability, and performance. We outperformed the few existing approaches to scale GAE and VGAE models, including our degeneracy framework from Chapter 3. We also showed that FastGAE addresses some of the

limitations of this framework. As a consequence, we consider that FastGAE constitutes an improvement of our previous efforts and, in the remainder of this thesis, we will rather resort to FastGAE when dealing with large graphs.

Besides its empirical superiority over the degeneracy framework from Chapter 3 in a majority of our experiments, FastGAE is also conceptually simpler. We believe that simple solutions have the most impact. At the time of writing, the FastGAE method has already been explicitly mentioned and used in other research experiments, e.g., in [157, 298, 387]. Xiang Sheng (`xiangsheng1325` on GitHub) also recently developed a PyTorch implementation of FastGAE[7].

In addition, we emphasize that our method easily extends to GAE and VGAE models with alternative GNN encoders. In our experiments, the GCN encoders of standard GAE and VGAE models [187] and of FastGAE-based models could easily be replaced by any alternative architecture learning the embedding matrix Z in another way, e.g., by a FastGCN [50], a Cluster-GCN [53] or a GraphSAGE [130] encoder. Besides, FastGAE easily extends to alternative *decoders*. For instance, one could replace the symmetric inner product decoder from our experiments with some more elaborated decoders [124, 329], including the gravity-inspired decoder that we will ourselves propose in Chapter 5 to reconstruct *directed* graphs.

Simultaneously, we identify several future research directions for improvements. Apart from the aforestated limit (see Section 4.3.3) of current GAE and VGAE models on community detection, which we will investigate in Chapter 7, we underline that the proposed FastGAE method could underperform on very sparse graphs. Indeed, in such a scenario, the subgraphs to reconstruct might include a large proportion of isolated nodes, which would negatively impact the learning process. Moreover, in the case of large graphs with a lot of sparsely connected components, we recommend applying FastGAE separately on each component. Lastly, in this chapter, we assumed that the graph was fixed. This aspect might appear as a limit, that could motivate future interesting studies on extensions of FastGAE for scalable *dynamic* graph embeddings, potentially with a dynamic selection of $n_{(S)}$.

---

[7] https://github.com/xiangsheng1325/fastgae_pytorch

4.4. Conclusion

Table 4.3: Link prediction on all medium-size graphs. For each graph, for brevity, we only report the **best** GAE **or** VGAE model in terms of AUC and AP scores, a few representative degree-based FastGAE versions of this model, and the best baseline (among Core-GAE/VGAE, Negative sampling GAE/VGAE, node2vec and Laplacian eigenmaps). Scores are averaged over 100 runs. For degree sampling, values of the hyperparameter α (as defined in Equation (4.2)) were tuned as illustrated in Figure 4.4. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. **Bold** numbers correspond to the best scores and best running time. Scores *in italic* are within one standard deviation range from the best score.

Dataset	Model	Average Perf. on Test Set		Avg. Run. Times (in sec.)			Speed Gain
		AUC (in %)	AP (in %)	Comp. p_i	Train model	Total	
Cora	Standard GAE	84.79 ± 1.10	$\mathbf{88.45 \pm 0.82}$	-	3.87	3.87	-
	FastGAE (degree, $\alpha = 2$)						
	- with $n_{(S)} = 250$	84.13 ± 1.20	86.65 ± 1.23	0.002	1.46	**1.462**	$\times\,\mathbf{2.65}$
	- with $n_{(S)} = n^*_{(S)} = 440$	84.74 ± 0.81	87.42 ± 0.75	0.002	1.56	1.562	$\times\,2.48$
	- with $n_{(S)} = 1\,000$	84.75 ± 0.84	*87.77 ± 0.81*	0.002	1.65	1.652	$\times\,2.34$
	Best baseline						
	Laplacian eigenmaps	$\mathbf{86.49 \pm 0.98}$	*87.42 ± 1.04*	-	2.49	2.49	$\times\,1.55$
Cora with features	Standard VGAE	*91.64 ± 0.92*	$\mathbf{92.66 \pm 0.91}$	-	4.25	4.25	-
	Var. FastGAE (degree, $\alpha = 2$)						
	- with $n_{(S)} = 250$	90.50 ± 1.10	91.10 ± 1.08	0.002	2.30	**2.302**	$\times\,\mathbf{1.85}$
	- with $n_{(S)} = n^*_{(S)} = 440$	90.82 ± 1.07	91.44 ± 1.13	0.002	2.52	2.522	$\times\,1.69$
	- with $n_{(S)} = 1\,000$	$\mathbf{91.72 \pm 0.98}$	*92.36 ± 1.11*	0.002	2.87	2.872	$\times\,1.48$
	Best baseline						
	Core-VGAE, $k = 2$	87.94 ± 1.12	89.00 ± 1.11	-	3.09	3.09	$\times\,1.38$
Citeseer	Standard GAE	78.25 ± 1.69	$\mathbf{83.79 \pm 1.24}$	-	5.25	5.25	-
	FastGAE (degree, $\alpha = 1$)						
	- with $n_{(S)} = 250$	77.28 ± 1.11	81.29 ± 0.92	0.002	1.47	**1.472**	$\times\,\mathbf{3.57}$
	- with $n_{(S)} = n^*_{(S)} = 488$	78.30 ± 1.30	*82.42 ± 1.09*	0.002	1.58	1.582	$\times\,3.32$
	- with $n_{(S)} = 1\,000$	78.31 ± 1.25	*82.40 ± 0.99*	0.002	1.61	1.612	$\times\,3.26$
	Best baseline						
	Laplacian eigenmaps	$\mathbf{80.42 \pm 1.38}$	*83.75 ± 1.12*	-	3.50	3.50	$\times\,1.50$
Citeseer with features	Standard VGAE	$\mathbf{90.72 \pm 1.01}$	$\mathbf{92.05 \pm 0.97}$	-	6.28	6.28	-
	Var. FastGAE (degree, $\alpha = 1$)						
	- with $n_{(S)} = 250$	89.37 ± 1.69	89.63 ± 1.83	0.002	2.32	**2.322**	$\times\,\mathbf{2.70}$
	- with $n_{(S)} = n^*_{(S)} = 488$	*90.10 ± 1.33*	90.15 ± 1.50	0.002	2.62	2.622	$\times\,2.40$
	- with $n_{(S)} = 1\,000$	*90.22 ± 1.14*	90.16 ± 1.20	0.002	2.89	2.892	$\times\,2.17$
	Best baseline						
	Core-VGAE, $k = 2$	81.85 ± 1.72	83.65 ± 1.64	-	2.55	2.55	$\times\,2.46$
Pubmed	Standard GAE	82.51 ± 0.64	87.42 ± 0.38	-	811.43	811.43	-
	FastGAE (degree, $\alpha = 1$)						
	- with $n_{(S)} = 500$	82.68 ± 0.51	85.89 ± 0.47	0.01	2.98	**2.99**	$\times\,\mathbf{271.38}$
	- with $n_{(S)} = n^*_{(S)} = 1\,187$	83.67 ± 0.42	87.01 ± 0.31	0.01	3.20	3.21	$\times\,252.78$
	- with $n_{(S)} = 5\,000$	$\mathbf{84.82 \pm 0.32}$	$\mathbf{88.19 \pm 0.23}$	0.01	14.41	14.42	$\times\,56.27$
	Best baseline						
	Core-GAE, $k = 2$	84.30 ± 0.27	86.11 ± 0.43	-	168.91	168.91	$\times\,4.80$
Pubmed with features	Standard GAE	$\mathbf{96.28 \pm 0.36}$	*96.29 ± 0.25*	-	952.63	952.63	-
	FastGAE (degree, $\alpha = 1$)						
	- with $n_{(S)} = 500$	95.08 ± 0.45	95.24 ± 0.46	0.01	3.53	**3.54**	$\times\,\mathbf{269.10}$
	- with $n_{(S)} = n^*_{(S)} = 1\,187$	95.45 ± 0.26	95.70 ± 0.30	0.01	4.01	4.02	$\times\,237.56$
	- with $n_{(S)} = 5\,000$	*96.12 ± 0.20*	$\mathbf{96.35 \pm 0.19}$	0.01	19.74	19.75	$\times\,48.23$
	Best baseline						
	Core-GAE, $k = 2$	85.34 ± 0.33	86.06 ± 0.24	-	40.22	40.22	$\times\,23.69$

Table 4.4: Link prediction on all large graphs. For each graph, for brevity, we only report the **best** GAE **or** VGAE model in terms of AUC and AP scores, a few representative degree-based FastGAE versions of this model, and the best baseline (among Core-GAE/VGAE, Negative sampling GAE/VGAE and node2vec). Scores are averaged over 10 runs. For degree sampling, values of the hyperparameter α (as defined in Equation (4.2)) were tuned as illustrated in Figure 4.4. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. **Bold** numbers correspond to the best scores and best running times. Scores *in italic* are within one standard deviation range from the best score.

Dataset	Model	Average Perf. on Test Set		Average Running Times (in sec.)		
		AUC (in %)	AP (in %)	Compute p_i	Train model	Total
SBM	Standard VGAE	*(intractable)*		*(intractable)*		
	Var. FastGAE (degree, $\alpha = 2$)					
	- with $n_{(S)} = 2\ 000$	79.37 ± 0.52	80.68 ± 0.84	0.03	27.36	**27.39**
	- with $n_{(S)} = n_{(S)}^* = 2\ 673$	80.96 ± 0.35	83.69 ± 0.60	0.03	30.66	30.69
	- with $n_{(S)} = 5\ 000$	$\mathbf{81.45 \pm 0.39}$	$\mathbf{84.30 \pm 0.82}$	0.03	43.86	43.89
	Best baseline					
	node2vec	80.89 ± 0.32	83.51 ± 0.29	-	1 328.82	1 328.82 (22 min)
Google	Standard GAE	*(intractable)*		*(intractable)*		
	FastGAE (degree, $\alpha = 1$)					
	- with $n_{(S)} = 2\ 500$	94.52 ± 0.26	95.50 ± 0.11	0.14	122.53	**122.67**
	- with $n_{(S)} = n_{(S)}^* = 7\ 911$	*95.75 ± 0.24*	*96.62 ± 0.09*	0.14	158.63	158.77
	- with $n_{(S)} = 10\ 000$	$\mathbf{95.91 \pm 0.19}$	*96.64 ± 0.12*	0.14	168.10	168.24
	Best baseline					
	node2vec	94.89 ± 0.63	$\mathbf{96.82 \pm 0.72}$	-	14 762.78	14 762.78 (4h06)
Youtube	Standard VGAE	*(intractable)*		*(intractable)*		
	Var. FastGAE (degree, $\alpha = 5$)					
	- with $n_{(S)} = 3\ 000$	81.14 ± 0.19	86.61 ± 0.16	0.28	453.22	**453.50 (8min)**
	- with $n_{(S)} = n_{(S)}^* = 15\ 179$	81.83 ± 0.15	*87.21 ± 0.15*	0.28	2 964.51	2 964.79 (49min)
	- with $n_{(S)} = 20\ 000$	$\mathbf{82.31 \pm 0.18}$	$\mathbf{87.36 \pm 0.15}$	0.28	3 596.03	3 596.31 (1h00)
	Best baseline					
	Core-VGAE, $k = 40$	80.53 ± 0.23	82.45 ± 0.20	-	12 433.51	12 433.51 (3h27)
Patent	Standard GAE	*(intractable)*		*(intractable)*		
	FastGAE with (degree, $\alpha = 2$)					
	- with $n_{(S)} = 5\ 000$	90.66 ± 0.25	90.76 ± 0.22	0.30	605.75	**606.05 (10min)**
	- with $n_{(S)} = n_{(S)}^* = 16\ 425$	$\mathbf{93.02 \pm 0.23}$	$\mathbf{93.39 \pm 0.23}$	0.30	3 693.32	3 693.62 (1h02)
	- with $n_{(S)} = 20\ 000$	*92.91 ± 0.22*	*93.35 ± 0.21*	0.30	4 401.67	4 401.67 (1h13)
	Best baseline					
	node2vec	*92.96 ± 0.23*	*93.36 ± 0.20*	-	25 851.39	25 851.39 (7h11)

4.4. Conclusion

Table 4.5: Community detection on all graphs with communities. For each graph, for brevity, we only report the **best** GAE **or** VGAE model in terms of AMI, a few representative degree-based FastGAE versions of this model, and the best baseline. Scores are averaged over 100 runs (except SBM: 10 runs). For degree sampling, values of the hyperparameter α (as defined in Equation (4.2)) were tuned as illustrated in Figure 4.4. All GAE/VGAE models learn embedding vectors of dimension $d = 16$. **Bold** numbers correspond to the best scores and best running times. Scores *in italic* are within one standard deviation range from the best ones.

Dataset	Model	Average Performance AMI (in %)	Average Running Times (in sec.)			Speed Gain
			Compute p_i	Train model	Total	
Cora	Standard GAE	30.88 ± 2.56	-	3.90	3.90	-
	FastGAE (degree, $\alpha = 2$)					
	- with $n_{(S)} = 250$	33.32 ± 2.61	0.002	1.51	**1.512**	$\times$ **2.58**
	- with $n_{(S)} = n^*_{(S)} = 440$	34.64 ± 2.45	0.002	1.59	1.592	$\times$ 2.45
	- with $n_{(S)} = 1\ 000$	35.56 ± 2.80	0.002	1.67	1.672	$\times$ 2.33
	Best baseline					
	Louvain	**46.72 $\pm$ 0.85**	-	1.79	1.79	$\times$ 2.18
Cora with features	Standard VGAE	*44.84 $\pm$ 2.63*	-	4.32	4.32	-
	Var. FastGAE (degree, $\alpha = 2$)					
	- with $n_{(S)} = 250$	41.35 ± 3.49	0.002	2.40	2.402	$\times$ 1.80
	- with $n_{(S)} = n^*_{(S)} = 440$	42.89 ± 2.72	0.002	2.67	2.672	$\times$ 1.62
	- with $n_{(S)} = 1\ 000$	*45.02 $\pm$ 2.81*	0.002	2.92	2.922	$\times$ 1.48
	Best baseline					
	Louvain	**46.72 $\pm$ 0.85**	-	1.79	**1.79**	$\times$ **2.41**
Citeseer	Standard VGAE	9.85 ± 1.24	-	5.44	5.44	-
	Var. FastGAE (degree, $\alpha = 1$)					
	- with $n_{(S)} = 250$	9.34 ± 1.48	0.002	1.77	**1.772**	$\times$ **3.07**
	- with $n_{(S)} = n^*_{(S)} = 488$	10.02 ± 1.42	0.002	2.02	2.022	$\times$ 2.69
	- with $n_{(S)} = 1\ 000$	10.16 ± 1.41	0.002	2.19	2.192	$\times$ 2.48
	Best baseline					
	Louvain	**16.39 $\pm$ 1.45**	-	2.41	2.41	$\times$ 2.26
Citeseer with features	Standard VGAE	*20.17 $\pm$ 3.07*	-	6.45	6.45	-
	Var. FastGAE (degree, $\alpha = 1$)					
	- with $n_{(S)} = 250$	*20.49 $\pm$ 3.74*	0.002	2.80	2.802	$\times$ 2.30
	- with $n_{(S)} = n^*_{(S)} = 488$	*20.53 $\pm$ 3.45*	0.002	2.88	2.882	$\times$ 2.24
	- with $n_{(S)} = 1\ 000$	**20.94 $\pm$ 3.21**	0.002	3.11	3.112	$\times$ 2.07
	Best baseline					
	Cora-Graph VAE, $k = 2$	16.53 ± 1.95	-	2.76	**2.76**	$\times$ **2.33**
Pubmed	Standard VGAE	20.52 ± 2.97	-	856.05	856.05	-
	Var. FastGAE (degree, $\alpha = 1$)					
	- with $n_{(S)} = 500$	16.86 ± 4.84	0.01	3.17	**3.18**	$\times$ **269.20**
	- with $n_{(S)} = n^*_{(S)} = 1\ 187$	18.84 ± 4.78	0.01	3.61	3.62	$\times$ 236.49
	- with $n_{(S)} = 5\ 000$	*22.81 $\pm$ 4.80*	0.01	14.95	14.96	$\times$ 57.22
	Best baseline					
	Core-VGAE, $k = 2$	**23.56 $\pm$ 3.12**	-	50.11	50.11	$\times$ 17.08
Pubmed with features	Standard VGAE	25.43 ± 1.47	-	970.67	970.67	-
	Var. FastGAE (degree, $\alpha = 1$)					
	- with $n_{(S)} = 500$	29.04 ± 4.17	0.01	4.03	**4.04**	$\times$ **240.26**
	- with $n_{(S)} = n^*_{(S)} = 1\ 187$	**31.11 $\pm$ 3.27**	0.01	4.65	4.66	$\times$ 208.30
	- with $n_{(S)} = 5\ 000$	*30.89 $\pm$ 3.01*	0.01	20.01	20.02	$\times$ 48.49
	Best baseline					
	Core-VGAE, $k = 2$	24.35 ± 1.55	-	57.09	57.09	$\times$ 17.00
SBM	Standard VGAE	*(intractable)*		*(intractable)*		-
	Var. FastGAE (degree, $\alpha = 2$)					
	- with $n_{(S)} = 2\ 500$	30.77 ± 0.32	0.03	52.01	**52.04**	-
	- with $n_{(S)} = n^*_{(S)} = 2\ 673$	30.89 ± 0.30	0.03	53.98	54.01	-
	-with $n_{(S)} = 5\ 000$	32.28 ± 0.26	0.03	61.96	61.69	-
	Best baseline					
	Louvain	**35.90 $\pm$ 0.14**	-	464.11	464.11	-

4.5   Appendices

This supplementary section provides the theoretical analyses announced in Section 4.2.3 as well as an additional figure presenting the optimal values of the hyperparameter α in our experiments. They were placed out of the main content of Chapter 4 for the sake of brevity and readability.

On Approximate Losses

Let us recall that, in our FastGAE framework, at each training iteration we run a full GCN forward pass and sample a subgraph $\mathcal{G}_{(S)} = (\mathcal{V}_{(S)}, \mathcal{E}_{(S)})$. Then, we evaluate reconstruction losses only on this subgraph, which involves fewer operations w.r.t. standard decoders, and we use the resulting approximate loss for GCN weights updates via gradient descent. More precisely, in standard implementations of GAE/VGAE and assuming an unweighted graph, the cross entropy loss and the negative of the ELBO's expectation part are empirically derived by computing the following node pairs average at each training iteration:

$$\mathcal{L} = \frac{1}{n^2} \sum_{(i,j)\in\mathcal{V}^2} \mathcal{L}_{ij}(A_{ij}, \hat{A}_{ij}), \tag{4.4}$$

with[8]: $\mathcal{L}_{ij}(A_{ij}, \hat{A}_{ij}) = -[A_{ij}\log(\hat{A}_{ij}) + (1 - A_{ij})\log(1 - \hat{A}_{ij})]$. In the FastGAE framework, we instead compute:

$$\mathcal{L}_{\text{FastGAE}} = \frac{1}{n^2_{(S)}} \sum_{(i,j)\in\mathcal{V}^2} \mathbb{1}_{((i,j)\in\mathcal{V}^2_{(S)})} \mathcal{L}_{ij}(A_{ij}, \hat{A}_{ij}), \tag{4.5}$$

with $\mathbb{1}_{((i,j)\in\mathcal{V}^2_{(S)})} = 1$ if $(i,j) \in \mathcal{V}^2_{(S)}$ and 0 otherwise. We recall that, for variational FastGAE, we need to substract the Kullback-Leibler (KL) divergence [199], as in the ELBO of standard VGAE, to obtain our actual objective function. At this stage, two options are possible:

- computing the KL term only on the nodes in the subgraph;

- or, computing the KL term on all nodes.

We consider that the two options are valid. The first one ensures that the resulting loss is a proper lower bound of the likelihood computed on this subgraph. The second one, despite violating this property, can nonetheless be empirically convenient and interpreted as the addition of a *regularization term* on all node embedding vectors (penalizing large deviations w.r.t. a $\mathcal{N}(0, I_d)$ prior distribution on these vectors) to the *performance* term $\mathcal{L}_{\text{FastGAE}}$. In our experiments, both

---

[8]In most implementations, as explained in Section 2.4.1, the terms with $A_{ij} = 1$ are re-weighted in the loss, in case of sparse A. They are multiplied by $w_{\text{pos}} \geq 1$, a positive links re-weighting scalar parameter which is usually inversely proportional to the graph sparsity. In our analyses, to clarify notations, we omit this scalar multiplication, which is equivalent to implicitly assuming that $w_{\text{pos}} = 1$. This simplification is made without loss of generality and all results remain valid for any $w_{\text{pos}} > 1$.

options returned similar results. In the following propositions, we assume that the KL term is computed on *all* nodes for simplicity, and we therefore only approximate the performance term $\mathcal{L}$, both in the GAE and in the VGAE settings.

Propositions 4.1 and 4.2 detail the formal probabilities to sample a given node or a given node pair at each training iteration. We consider both sampling variants *with* and *without* replacement (see Section 3.2) for this analysis, as the former significantly simplifies results w.r.t. the latter.

Proposition 4.1. Let $\mathcal{G}_{(S)} = (\mathcal{V}_{(S)}, \mathcal{E}_{(S)})$ be a subgraph of $\mathcal{G}$ obtained from sampling $n_{(S)}$ nodes **with** replacement using the node sampling strategy of FastGAE. Let i and j denote two distinct nodes from the original graph $\mathcal{G}$: $(i, j) \in \mathcal{V}^2$. Then:

$$\mathbb{P}\Big(i \in \mathcal{V}_{(S)}\Big) = 1 - (1 - p_i)^{n_{(S)}}. \tag{4.6}$$

Also:

$$\mathbb{P}\Big((i,j) \in \mathcal{V}_{(S)}^2\Big) = 1 - \Big[(1 - p_i)^{n_{(S)}} + (1 - p_j)^{n_{(S)}} - (1 - p_i - p_j)^{n_{(S)}}\Big]. \tag{4.7}$$

Proof. In this setting, sampling probabilities are independent of previous sampling steps, and remain fixed to p_i. Therefore, for node $i \in \mathcal{V}$, we have:

$$\mathbb{P}\Big(i \notin \mathcal{V}_{(S)}\Big) = (1 - p_i)^{n_{(S)}}.$$

Indeed, for i *not* to belong to $\mathcal{V}_{(S)}$, it must not be selected at any of the $n_{(S)}$ draws, which happens with probability $1 - p_i$ for each draw. Therefore:

$$\mathbb{P}\Big(i \in \mathcal{V}_{(S)}\Big) = 1 - (1 - p_i)^{n_{(S)}}.$$

Moreover, let i and j denote two distinct nodes from the original graph $\mathcal{G}$: $(i, j) \in \mathcal{V}^2$. We have:

$$\mathbb{P}\Big((i,j) \notin \mathcal{V}_{(S)}^2\Big) = \mathbb{P}\Big(i \notin \mathcal{V}_{(S)} \text{ or } j \notin \mathcal{V}_{(S)}\Big)$$
$$= \mathbb{P}\Big(i \notin \mathcal{V}_{(S)}\Big) + \mathbb{P}\Big(j \notin \mathcal{V}_{(S)}\Big) - \mathbb{P}\Big(i \notin \mathcal{V}_{(S)}, j \notin \mathcal{V}_{(S)}\Big)$$

with, using the previous result, $\mathbb{P}(i \notin \mathcal{V}_{(S)}) = (1 - p_i)^{n_{(S)}}$ and $\mathbb{P}(j \notin \mathcal{V}_{(S)}) = (1 - p_j)^{n_{(S)}}$. Using a similar argument, we also obtain:

$$\mathbb{P}\Big(i \notin \mathcal{V}_{(S)}, j \notin \mathcal{V}_{(S)}\Big) = \Big(1 - (p_i + p_j)\Big)^{n_{(S)}}.$$

Therefore:

$$\mathbb{P}\Big((i,j) \notin \mathcal{V}_{(S)}^2\Big) = \Big[(1 - p_i)^{n_{(S)}} + (1 - p_j)^{n_{(S)}} - (1 - p_i - p_j)^{n_{(S)}}\Big].$$

And:

$$\mathbb{P}\Big((i,j) \in \mathcal{V}^2_{(S)}\Big) = 1 - \mathbb{P}\Big((i,j) \notin \mathcal{V}^2_{(S)}\Big) = 1 - \Big[(1-p_i)^{n_{(S)}} + (1-p_j)^{n_{(S)}} - (1-p_i-p_j)^{n_{(S)}}\Big].$$

Lastly, for self-loops:

$$\mathbb{P}\Big((i,i) \in \mathcal{V}^2_{(S)}\Big) = \mathbb{P}\Big(i \in \mathcal{V}_{(S)}\Big) = 1 - (1-p_i)^{n_{(S)}}.$$

$\square$

Proposition 4.2. Let $\mathcal{G}_{(S)} = (\mathcal{V}_{(S)}, \mathcal{E}_{(S)})$ be a subgraph of $\mathcal{G}$ obtained from sampling $n_{(S)}$ nodes **without** replacement using the node sampling strategy of FastGAE. Let i and j denote two distinct nodes from $\mathcal{G}$: $(i,j) \in \mathcal{V}^2$. Then:

$$\mathbb{P}\Big(i \in \mathcal{V}_{(S)}\Big) = \sum_{\mathcal{U} \in \mathbf{U}(i)} p_{u_1} \prod_{k=2}^{n_{(S)}} \frac{p_{u_k}}{1 - \sum_{k'=1}^{k-1} p_{u_{k'}}}, \tag{4.8}$$

where $\mathbf{U}(i) = \{\mathcal{U} \subset \mathcal{V}, |\mathcal{U}| = n_{(S)} \text{ and } i \in \mathcal{U}\}$ is the set of all ordered subsets of $n_{(S)}$ distinct nodes including node i. For a given set $\mathcal{U} \in \mathbf{U}(i)$, we denote by $(u_1, u_2, ..., u_{n_{(S)}})$ its ordered elements. Also,

$$\mathbb{P}\Big((i,j) \in \mathcal{V}^2_{(S)}\Big) = \sum_{\mathcal{U} \in \mathbf{U}(i) \cap \mathbf{U}(j)} p_{u_1} \prod_{k=2}^{n_{(S)}} \frac{p_{u_k}}{1 - \sum_{k'=1}^{k-1} p_{u_{k'}}}. \tag{4.9}$$

Proof. We are looking for the probability that a node $i \in \mathcal{V}$ from the graph belongs to a drawn subset $\mathcal{V}_{(S)}$, that contains $n_{(S)}$ distinct nodes. For $\mathcal{V}_{(S)}$ to include i, $\mathcal{V}_{(S)}$ should match any of the possible ordered subsets of $n_{(S)}$ nodes that include node i. In this setting where we sample without replacement, the probability to draw node i *depends on nodes previously drawn*. All possible orders of sampling the nodes should be considered. Let:

$$\mathbf{U}(i) = \Big\{\mathcal{U} \subset \mathcal{V}, |\mathcal{U}| = n_{(S)} \text{ and } i \in \mathcal{U}\Big\}$$

denote the set of all **ordered** subsets of $n_{(S)}$ distinct nodes that include node i. With such a notation, we have:

$$\mathbb{P}\Big(i \in \mathcal{V}_{(S)}\Big) = \mathbb{P}\Big(\mathcal{V}_{(S)} \in \mathbf{U}(i)\Big) = \sum_{\mathcal{U} \in \mathbf{U}(i)} \mathbb{P}\Big(\mathcal{V}_{(S)} = \mathcal{U}\Big).$$

The summation comes from the fact that events are *disjoint* ($\mathcal{V}_{(S)}$ can not match two of these ordered subsets simultaneously).

Now, for a given set $\mathcal{U} \in \mathbf{U}(i)$, let us denote by $(u_1, u_2, ..., u_{n_{(S)}})$ its **ordered** elements. Also,

let $(\mathcal{V}_{(S)1}, \mathcal{V}_{(S)2}, ..., \mathcal{V}_{(S)n_{(S)}})$ be the $n_{(S)}$ ordered nodes of set $\mathcal{G}_{(S)}$ (i.e., $\mathcal{V}_{(S)1}$ is the first drawn node, $\mathcal{V}_{(S)2}$ is the second one, etc). We have:

$$\mathbb{P}\Big(\mathcal{V}_{(S)} = \mathcal{U}\Big) = \mathbb{P}\Big(\mathcal{V}_{(S)1} = u_1, \mathcal{V}_{(S)2} = u_2, ..., \mathcal{V}_{(S)n_{(S)}} = u_{n_{(S)}}\Big)$$

$$= \mathbb{P}(\mathcal{V}_{(S)1} = u_1) \prod_{k=2}^{n_{(S)}} \mathbb{P}(\mathcal{V}_{(S)k} = u_k | \mathcal{V}_{(S)k-1} = u_{k-1}, ..., \mathcal{V}_{(S)1} = u_1)$$

$$= p_{u_1} \prod_{k=2}^{n_{(S)}} \frac{p_{u_k}}{1 - \sum_{k'=1}^{k-1} p_{u_{k'}}}.$$

Therefore, by summing elements to come back to $\mathbb{P}(i \in \mathcal{V}_{(S)})$:

$$\mathbb{P}\Big(i \in \mathcal{V}_{(S)}\Big) = \sum_{\mathcal{U} \in \mathbf{U}(i)} p_{u_1} \prod_{k=2}^{n_{(S)}} \frac{p_{u_k}}{1 - \sum_{k'=1}^{k-1} p_{u_{k'}}}.$$

Moreover, let i and j denote two distinct nodes from the original graph $\mathcal{G}$: $(i, j) \in \mathcal{V}^2$. Using a similar notation and reasoning, we get:

$$\mathbb{P}\Big((i, j) \in \mathcal{V}_{(S)}^2\Big) = \mathbb{P}\Big(i \in \mathcal{V}_{(S)}, j \in \mathcal{V}_{(S)}\Big) = \sum_{\mathcal{U} \in \mathbf{U}(i) \cap \mathbf{U}(j)} \mathbb{P}\Big(\mathcal{V}_{(S)} = \mathcal{U}\Big).$$

Therefore:

$$\mathbb{P}\Big((i, j) \in \mathcal{V}_{(S)}^2\Big) = \sum_{\mathcal{U} \in \mathbf{U}(i) \cap \mathbf{U}(j)} p_{u_1} \prod_{k=2}^{n_{(S)}} \frac{p_{u_k}}{1 - \sum_{k'=1}^{k-1} p_{u_{k'}}}.$$

And, for self-loops, $\mathbb{P}((i, i) \in \mathcal{V}_{(S)}^2) = \mathbb{P}(i \in \mathcal{V}_{(S)})$. □

Despite different formulations, both variants share a similar behavior in practice on most real-world graphs. In this paper, as explained in Section 4.2.2, we sample nodes *without replacement*. One can derive from the above expressions that the probability to draw a node i, or an edge incident to i, increases with $n_{(S)}$, with p_i and with $f(i)$ for $\alpha > 0$. This also leads to the following formulation of the expected (re-weighted) loss that FastGAE stochastically optimizes.

Proposition 4.3. Using the expressions of Proposition 4.1 (when sampling with replacement) or Proposition 4.2 (when sampling without replacement):

$$\mathbb{E}\Big[\mathcal{L}_{\text{FastGAE}}\Big] = \frac{1}{n_{(S)}^2} \sum_{(i,j) \in \mathcal{V}^2} \mathbb{P}\Big((i, j) \in \mathcal{V}_{(S)}^2\Big) \mathcal{L}_{ij}(A_{ij}, \hat{A}_{ij}). \tag{4.10}$$

Proof. We have:

$$\mathbb{E}\Big[\mathcal{L}_{\text{FastGAE}}\Big] = \mathbb{E}\Big[\frac{1}{n_{(S)}^2} \sum_{(i,j)\in\mathcal{V}^2} \mathbb{1}_{((i,j)\in\mathcal{V}_{(S)}^2)}\mathcal{L}_{ij}(A_{ij},\hat{A}_{ij})\Big]$$

$$= \frac{1}{n_{(S)}^2} \sum_{(i,j)\in\mathcal{V}^2} \mathbb{E}\Big[\mathbb{1}_{((i,j)\in\mathcal{V}_{(S)}^2)}\Big]\mathcal{L}_{ij}(A_{ij},\hat{A}_{ij})$$

$$= \frac{1}{n_{(S)}^2} \sum_{(i,j)\in\mathcal{V}^2} \mathbb{P}\Big((i,j)\in\mathcal{V}_{(S)}^2\Big)\mathcal{L}_{ij}(A_{ij},\hat{A}_{ij}).$$

By replacing $\mathbb{P}((i,j)\in\mathcal{V}_{(S)}^2)$ by the expressions of Proposition 4.1 (with replacement) or Proposition 4.2 (without replacement), we obtain an explicit formulation for $\mathbb{E}\Big[\mathcal{L}_{\text{FastGAE}}\Big]$. $\qquad\square$

On the Selection of $n_{(S)}$

While our experiments will tend to show that stochastically minimizing $\mathbb{E}(\mathcal{L}_{\text{FastGAE}})$ (Equation (4.10)) instead of $\mathcal{L}$ (Equation (4.4)) might be beneficial, we also acknowledge that, for small values of $n_{(S)}$, the actual loss $\mathcal{L}_{\text{FastGAE}}$ computed at a given training iteration (Equation (4.5)) might significantly deviate from its expectation.

We propose to use these deviations as a criterion to automatically select a relevant subgraph size. More precisely, let us rewrite $\mathcal{L}_{\text{FastGAE}}$ from Equation (4.5) as follows:

$$\mathcal{L}_{\text{FastGAE}} = \frac{1}{n_{(S)}} \sum_{i\in\mathcal{V}} \mathbb{1}_{(i\in\mathcal{V}_{(S)})}\mathcal{L}_{\text{FastGAE}}(i), \tag{4.11}$$

where the node-level terms $\mathcal{L}_{\text{FastGAE}}(i)$ are defined as:

$$\mathcal{L}_{\text{FastGAE}}(i) = \frac{1}{n_{(S)}} \sum_{j\in\mathcal{V}} \mathbb{1}_{(j\in\mathcal{V}_{(S)})}\mathcal{L}_{ij}(A_{ij},\hat{A}_{ij}), \tag{4.12}$$

and where $\mathcal{L}_{ij}$ denotes the cross entropy loss as in Equation (4.4). In the following, we leverage concentration inequalities [143] to derive a theoretically-grounded threshold size, denoted $n_{(S)}^*$ in the following, for which, under mild assumptions, the (random) node-level deviation $|\mathcal{L}_{\text{FastGAE}}(i) - \mathbb{E}[\mathcal{L}_{\text{FastGAE}}(i)]|$ at each training iteration is proven to be bounded with a high probability, for any node i. This proposed subgraph size is of the form:

$$n_{(S)}^* = C\sqrt{n}, \tag{4.13}$$

where the constant $C > 0$ depends on the deviation magnitude and probability, and is explicitly presented in Proposition 4.6. In our empirical analysis, this criterion will allow us to significantly improve the scalability and training speed of GAE and VGAE models (see discussion on complexity in Section 4.2.4), while reaching fairly competitive performances in a majority of

experiments (see Section 4.3). To prove our bounds, we require a technical assumption on $\hat{A}$:

Assumption 4.4. Let $(i, j) \in \mathcal{V}^2$. We thereafter assume that $\hat{A}_{ij} = \sigma(z_i^T z_j)$ can actually be *capped*, and that $\hat{A}_{ij} \in [\varepsilon, 1 - \varepsilon]$, where $0 < \varepsilon < 1$ is a constant that can be arbitrarily close to 0.

Under this assumption, we derive Propositions 4.5 and 4.6.

Proposition 4.5. Let us consider a training iteration of the FastGAE framework, a sampled subgraph $\mathcal{G}_{(S)} = (\mathcal{V}_{(S)}, \mathcal{E}_{(S)})$, with $|\mathcal{V}_{(S)}| = n_{(S)} < n$ nodes sampled without replacement, and the corresponding node-level approximate reconstruction computed for a given node i: $\mathcal{L}_{\text{FastGAE}}(i)$ from Equation (4.12), with $\mathcal{L}_{ij}(A_{ij}, \hat{A}_{ij}) = -[A_{ij} \log(\hat{A}_{ij}) + (1 - A_{ij}) \log(1 - \hat{A}_{ij})]$ in this same equation. Then, under Assumption 4.4, for any $\gamma \geq 0$, we have:

$$\mathbb{P}\Big(|\mathcal{L}_{\text{FastGAE}}(i) - \mathbb{E}[\mathcal{L}_{\text{FastGAE}}(i)]| \geq \gamma\Big) \leq 2 \exp\Big(-2(\frac{\gamma}{\log(\varepsilon)})^2 \frac{n_{(S)}^2}{n}\Big). \tag{4.14}$$

Proof. As a preliminary, let us recall Hoeffding's inequality [143]. Let $X_1, X_2..., X_n$ be real independent random variables verifying, for some $(a_k)_{1 \leq k \leq n}$ and $(b_k)_{1 \leq k \leq n}$ with $a_k < b_k$: $\forall k, \mathbb{P}(a_k \leq X_k \leq b_k) = 1$. Let $S_n = \sum_{i=1}^n X_i$. Then, for all $\gamma > 0$, we have:

$$\mathbb{P}\Big(|S_n - \mathbb{E}(S_n)| \geq t\Big) \leq 2 \exp\Big(-\frac{2\gamma^2}{\sum_{i=1}^n (b_i - a_i)^2}\Big).$$

Hoeffding [143] also proves that the inequality holds when the X_i are samples without replacement from a finite population (and therefore not independent). In the setting of Proposition 4.5, that falls into this second case, we have $\mathcal{L}_{\text{FastGAE}}(i) = \sum_{j \in \mathcal{V}} X_{ij}$, where, under Assumption 4.4:

$$X_{ij} = \frac{1}{n_{(S)}} \underbrace{\mathbb{1}_{(j \in \mathcal{V}_{(S)})}}_{\in \{0,1\}} \mathcal{L}_{ij}(A_{ij}, \hat{A}_{ij}) = \mathbb{1}_{(j \in \mathcal{V}_{(S)})} \frac{-1}{n_{(S)}} \underbrace{\underbrace{[A_{ij} \log(\hat{A}_{ij}) + (1 - A_{ij}) \log(1 - \hat{A}_{ij})]}_{\in [\log(\varepsilon), \log(1-\varepsilon)]}}_{\in [-\log(1-\varepsilon)/n_{(S)}, -\log(\varepsilon)/n_{(S)}]} \in \Big[0, \frac{-\log(\varepsilon)}{n_{(S)}}\Big].$$

We note that $\frac{-\log(\varepsilon)}{n_{(S)}} > 0$, as $0 < \varepsilon < 1$. Applying [143], at each sampling step and for all $\gamma > 0$:

$$\mathbb{P}\Big(|\mathcal{L}_{\text{FastGAE}}(i) - \mathbb{E}[\mathcal{L}_{\text{FastGAE}}(i)]| \geq \gamma\Big) \leq 2 \exp\Big(-\frac{2\gamma^2}{\sum_{j \in \mathcal{V}}(\frac{-\log(\varepsilon)}{n_{(S)}})^2}\Big)$$

$$= 2 \exp\Big(\frac{-2\gamma^2}{n \frac{(-\log(\varepsilon))^2}{n_{(S)}^2}}\Big) = 2 \exp\Big(-2(\frac{\gamma}{\log(\varepsilon)})^2 \frac{n_{(S)}^2}{n}\Big).$$

$\square$

We note that it exhibits the link between the loss deviation and the $\frac{n_{(S)}^2}{n}$ ratio. Also, the right-hand side term tends to 0 exponentially fast w.r.t. γ and $n_{(S)}$.

Proposition 4.6. For any confidence level $\alpha \in]0, 1[$ and node $i \in \mathcal{V}$, selecting a subgraph size $n_{(S)}$ such that:

$$n_{(S)} \geq n_{(S)}^* = \sqrt{n} \underbrace{\sqrt{\frac{-\log(\frac{\alpha}{2})\log(\varepsilon)^2}{2\gamma^2}}}_{\text{denoted } C}, \tag{4.15}$$

guarantees that:

$$\mathbb{P}\Big(|\mathcal{L}_{\text{FastGAE}}(i) - \mathbb{E}[\mathcal{L}_{\text{FastGAE}}(i)]| \geq \gamma\Big) \leq \alpha. \tag{4.16}$$

Proof. This is a corollary of Proposition 4.5, from which we derive that, for any $\alpha \in]0, 1[$:

$$2\exp\Big(-2(\frac{\gamma}{\log(\varepsilon)})^2\frac{n_{(S)}^2}{n}\Big) \leq \alpha \Rightarrow \mathbb{P}(|\mathcal{L}_{\text{FastGAE}} - \mathbb{E}[\mathcal{L}_{\text{FastGAE}}]| \geq \gamma) \leq \alpha.$$

Then:

$$2\exp\Big(-2(\frac{\gamma}{\log(\varepsilon)})^2\frac{n_{(S)}^2}{n}\Big) \leq \alpha \Leftrightarrow n_{(S)} \geq \sqrt{n}\sqrt{\frac{-\log(\frac{\alpha}{2})\log(\varepsilon)^2}{2\gamma^2}}.$$

$\square$

As an opening, we note that, while the current bounds are empirically effective (see Section 4.3), future research will aim to directly bound the deviation of $\mathcal{L}_{\text{FastGAE}}$ instead of the node-level terms $\mathcal{L}_{\text{FastGAE}}(i)$, which would be more ambitious and challenging due to the inherent dependencies among sampled node pairs in FastGAE. Also, while Propositions 4.5 and 4.6 focus on the case of the *cross entropy loss* for consistency w.r.t. the content presented in this chapter, a similar analysis (omitted here) could be performed to obtain comparable bounds for other *bounded* reconstruction losses. For instance, in the case of the Frobenius loss, where $\mathcal{L}_{ij}(A_{ij}, \hat{A}_{ij}) = (A_{ij} - \hat{A}_{ij})^2$, and without Assumption 4.4, one can obtain similar concentration guarantees as Proposition 5, with C being replaced by the constant $C' = \sqrt{\frac{-\log(\alpha/2)}{2\gamma^2}}$.

Numerical Application In our experiments, all $n_{(S)}^*$ thresholds are computed by evaluating Equation (4.15), setting $\gamma = 1$, $\alpha = 0.1$ and $\varepsilon = 0.001$.

On the hyperparameter α

To finish, we report the additional Figure 4.4 presenting the optimal values of the hyperparameter α, for all graphs, and for both core-based and degree-based sampling.
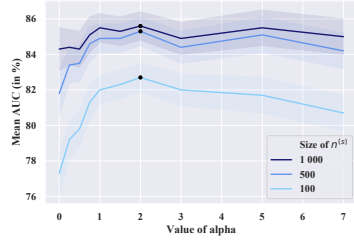
(a) Cora - Degree Sampling
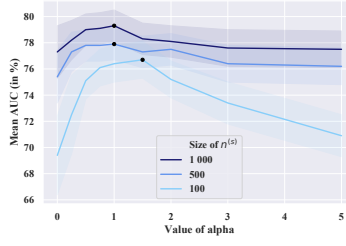
(b) Citeseer - Degree Sampling
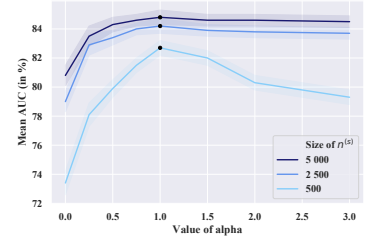
(c) Pubmed - Degree Sampling
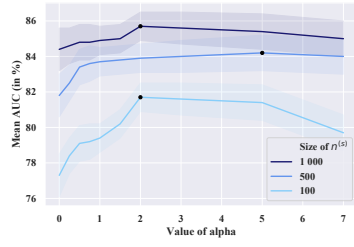
(d) Cora - Core Sampling
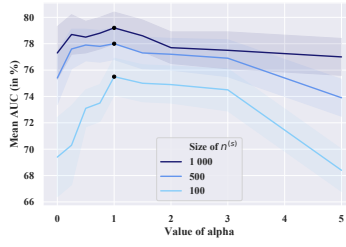
(e) Citeseer - Core Sampling
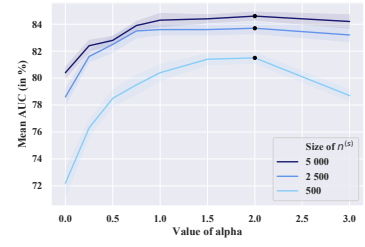
(f) Pubmed - Core Sampling
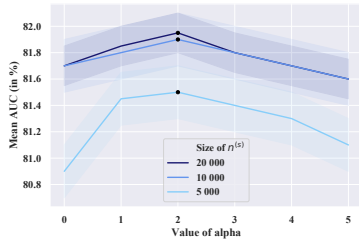
(g) SBM - Degree Sampling
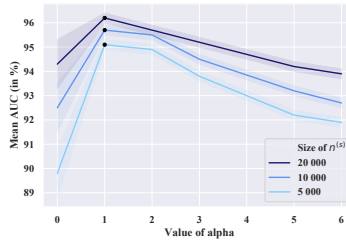
(h) Google - Degree Sampling
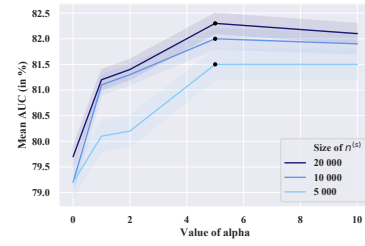
(i) Youtube - Degree Sampling
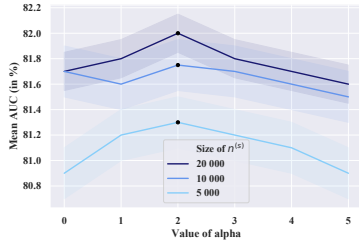
(j) SBM - Core Sampling
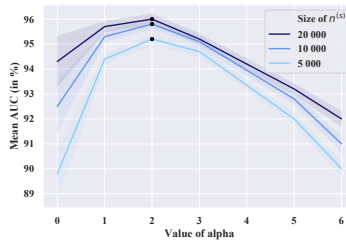
(k) Google - Core Sampling
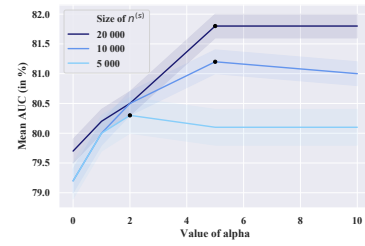
(l) Youtube - Core Sampling
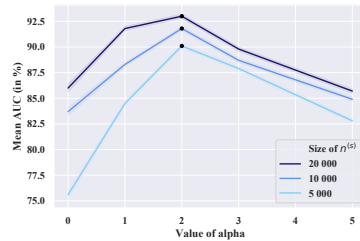
(m) Patent - Degree Sampling
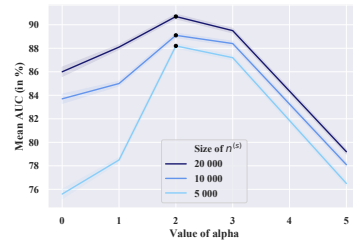
(n) Patent - Core Sampling

Figure 4.4: Optimal values of the hyperparameter α for degree-based and core-based node sampling w.r.t. mean AUC scores on validation sets, for Variational FastGAE models and for all graphs.

5

Gravity-Inspired Graph Autoencoders for Directed Link Prediction

This chapter presents research conducted with Stratis Limnios, Romain Hennequin, Viet-Anh Tran, and Michalis Vazirgiannis, and published in the proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM 2019) [311].

5.1 Introduction

While GAEs and VGAEs emerged as powerful node embedding models, their original versions were designed for *undirected* graphs [187] and ignore the potential direction of edges during the decoding step. As we will explain in Section 5.2, if a standard GAE/VGAE model predicts that a node $i \in \mathcal{V}$ is connected to a node $j \in \mathcal{V}$, then the same model will also predict that j is connected to i with the same probability. As a consequence, in our own experiments from Chapters 3 and 4, we also only analyzed undirected versions of the graphs under consideration.

This is limiting for numerous real-world applications, as *directed* graphs are ubiquitous. For instance, web graphs are made up of directed hyperlinks. In social networks such as Twitter, opinion leaders are usually followed by many users, but only few of these connections are reciprocal. Moreover, directed graphs are efficient abstractions in many domains where data are not explicitly structured as graphs. For instance, in Chapter 8, we will study top-k similar artists graphs, that artificially connect Deezer artists to their top-k most similar ones according to usage-based similarity metrics, e.g., the proportion of the artist's fans that also listened to these other artists. By nature, such a graph is directed. While most fans of a little known reggae band

might listen to Bob Marley (Marley thus appearing among their top-k similar artists according to the aforestated criterium), Bob Marley's fans will rarely listen to this band, which is unlikely to appear back among Bob Marley's own top-k most similar artists.

In this chapter, we aim to extend GAEs and VGAEs to directed graphs, with a particular focus on applications to *directed link prediction* tasks. Predicting the location of directed links has been historically performed by leveraging graph mining-based asymmetric measures [103, 318, 396] and, recently, a few attempts at capturing asymmetric proximity when learning node embedding spaces were proposed [259, 276, 408]. However, the question of how to reconstruct directed graphs from vector space representations to effectively perform directed link prediction remains widely open. In particular, at the time of this work, i.e., in 2019 [311], it was unclear how to extend GAE and VGAE models to directed graphs, and to which extent the promising performances of these models on undirected graphs could also be achieved on directed link prediction tasks.

In this chapter, we address these research questions. We present a new method, referred to as *Gravity-Inspired GAE or VGAE*, to effectively learn node embedding representations from directed graphs using the GAE and VGAE frameworks. We draw inspiration from Newton's theory of universal gravitation [269] to introduce a new decoding scheme, able to reconstruct asymmetric relations from embedding vectors. We empirically evaluate our method on three different directed link prediction tasks, for which standard GAE and VGAE models perform poorly. We achieve competitive results on three real-world datasets, outperforming popular baselines. To the best of our knowledge, our work provided the first GAE/VGAE experiments on directed graphs. Our implementation of Gravity-Inspired GAE and VGAE is publicly available [311].

This chapter is organized as follows. In Section 5.2, we explain why standard GAEs and VGAEs are not suitable for directed link prediction. In Section 5.3, we introduce our proposed Gravity-Inspired GAE and VGAE models. We present and discuss our experimental analysis in Section 5.4, and we conclude in Section 5.5.

5.2 Extending Graph Autoencoders to Directed Graphs

In this section, we present the limitations of standard GAE and VGAE models in the presence of directed graphs. We also mention the "source-target" paradigm, a strategy previously used in the scientific literature to reconstruct asymmetric links from an embedding space [276, 408].

5.2.1 On the Limitations of Standard Decoders for Directed Graphs

The GAE and VGAE models presented in Section 2.4.1 and 2.4.2 from Chapter 2 respectively, as well as the extensions of these models cited in Section 2.4.3 from this same chapter, all assume

that the input graph $\mathcal{G}$ is *undirected*. By design, these GAEs and VGAEs are *not* suitable for directed graphs, as they are ignoring directions when reconstructing the adjacency matrix from an embedding space. In particular, due to the symmetry of the inner product decoder, we have:

$$\hat{A}_{ij} = \sigma(z_i^T z_j) = \sigma(z_j^T z_i) = \hat{A}_{ji}. \tag{5.1}$$

In other words, if the inner product decoder predicts the existence of an edge (i, j) from a node $i \in \mathcal{V}$ to a node $j \in \mathcal{V}$, then it also necessarily predicts the existence of the reverse edge (j, i), *with the same probability*. This is undesirable for directed graphs, where $A_{ij} \neq A_{ji}$ in general. Replacing inner product decoders by any L_p distance in the embedding (e.g., the Euclidean distance, if $p = 2$) or by existing more refined decoders [124, 329] would lead to the same problem, as they are symmetric functions as well.

Therefore, as we will empirically show in Section 5.4, standard GAE and VGAE models significantly underperform on link prediction tasks in directed graphs. In 2019, Zhang et al. [403] proposed D-VAE, a variational autoencoder for small Directed Acyclic Graphs (DAG) such as neural networks architectures or Bayesian networks, focusing on neural architecture search and structure learning. However, at the time of this work, the question of how to extend GAE and VGAE to general directed graphs, such as citation graphs or web graphs, remained open.

5.2.2 The Source-Target Paradigm

Out of the GAE and VGAE frameworks, a few recent studies did tackle directed link prediction tasks using node embedding methods [276, 408]. These studies actually proposed to learn, not one, but *two embedding vectors* for each node $i \in \mathcal{V}$ of the graph: a *source* vector $z_i^{(s)} \in \mathbb{R}^d$ and a *target* vector $z_i^{(t)} \in \mathbb{R}^d$. More precisely:

- HOPE, short for *High-Order Proximity preserved Embedding* [276], aims to preserve high-order node-level proximity and to capture asymmetric transitivity. Nodes are represented by source vectors, stacked up in an $n \times d$ matrix $Z^{(s)}$, and by target vectors stacked up in another $n \times d$ matrix $Z^{(t)}$. For a given $n \times n$ node-level similarity matrix S, Ou et al. [276] learn these vectors by approximately minimizing:

$$\|S - Z^{(s)} Z^{(t)T}\|_F \tag{5.2}$$

using a generalized SVD (we refer to Section 2.2.2 from Chapter 2 for an introduction to node embeddings from matrix factorization). For directed graphs, an usual choice for S is the Katz matrix [177]:

$$S^{\text{Katz}} = \sum_{i=1}^{\infty} \beta^i A^i, \tag{5.3}$$

with $S^{\text{Katz}} = (I - \beta A)^{-1} \beta A$ if the parameter $\beta > 0$ is smaller than the spectral radius of A [177]. It computes the number of paths from a node to another one, these paths being exponentially weighted according to their length. Then, for directed link prediction, one can assess the likelihood of a missing edge from node i to node j using the asymmetric reconstruction:

$$\hat{A}_{ij} = \sigma(z_i^{(s)T} z_j^{(t)}), \tag{5.4}$$

with verifies $\hat{A}_{ij} \neq \hat{A}_{ji} = \sigma(z_j^{(s)T} z_i^{(t)})$ in general;

- APP, for *Asymmetric Proximity Preserving*, is a related node embedding method [408], that instead aims to conserve the Rooted PageRank score [278] for any node pair. APP leverages *random walk* with restart strategies to learn, as HOPE, a source vector and a target vector for each node (we refer to Section 2.2.3 from Chapter 2 for an introduction to node embeddings from random walks). As above, one can predict whether node i is connected to node j through a directed edge by computing the inner product of the source vector of i with the target vector of j, with a sigmoid activation.

We can derive a straightforward extension of this *source/target vectors* paradigm for GAE and VGAE models. Indeed, considering GCN encoders returning d-dimensional embedding vectors z_i, with d being even, we can assume that the $d/2$ first dimensions (respectively the $d/2$ last dimensions) of z_i actually correspond to the source (resp. to the target) vector of node i, i.e.:

$$z_i^{(s)} = z_i[1 : \frac{d}{2}] \ \text{ and } \ z_i^{(t)} = z_i[(\frac{d}{2} + 1) : d]. \tag{5.5}$$

Then, we can replace the symmetric decoder $\hat{A}_{ij} = \hat{A}_{ji} = \sigma(z_i^T z_j)$ by $\hat{A}_{ij} = \sigma(z_i^{(s)T} z_j^{(t)})$ and $\hat{A}_{ji} = \sigma(z_j^{(s)T} z_i^{(t)})$ as in APP and HOPE, to reconstruct directed links from GAE/VGAE-based encoded representations. In Section 5.4, we will provide an experimental evaluation of such an approach, which we will refer to as *Source/Target GAE (or VGAE)*.

Nonetheless, in most of this chapter, we will consider a different approach. We will come back to the original idea consisting in learning a *single* node embedding space, and therefore represent each node by a single embedding vector. Such an approach has a stronger interpretability power. It permits to better visualize representations, and is preferable for other tasks than link prediction (e.g., for community detection, as one can directly run a k-means algorithm on such representations). Besides, we will show in Section 5.4 that this single vector approach significantly outperforms Source/Target GAE and VGAE on various directed link prediction tasks.

5.3   Gravity-Inspired GAE and VGAE

In this section, we introduce our proposed method to learn node embedding spaces from directed graphs, and subsequently perform directed link prediction, using the GAE and VGAE frameworks. Our main challenge is the following: *how to effectively reconstruct asymmetric relations from encoded representations that are (unique) vectors in a node embedding where inner product and common distances are symmetric?*

To overcome this challenge, we resort to classical mechanics and especially to Newton's theory of universal gravitation [269]. We propose an analogy between vectors in a node embedding space and celestial objects in space. Specifically, even if the Earth-Moon distance is symmetric, the *acceleration* of the Moon towards the Earth due to gravity is larger than the acceleration of the Earth towards the Moon. As explained below, this is due to the fact that the Earth is more massive. In the remainder of this section, we transpose these notions of *mass* and *acceleration* to node embedding spaces, to build up our asymmetric graph decoding scheme.

5.3.1   From Physics to Node Representations

Newton's Theory of Universal Gravitation   According to Newton's theory of universal gravitation [269], each particle in the universe attracts the other particles through a force called *gravity*. This force is proportional to the product of the masses of the particles, and inversely proportional to the squared distance between their centers. More formally, let us denote by m_1 and m_2 the positive masses of two objects 1 and 2 and by r the distance between their centers. Then, the gravitational force F attracting the two objects is:

$$F = \frac{Gm_1m_2}{r^2},$$ (5.6)

where G is the gravitational constant [45]. Then, using Newton's second law of motion [269], we derive $a_{1\to2}$, the acceleration of object 1 towards object 2 due to gravity:

$$a_{1\to2} = \frac{F}{m_1} = \frac{Gm_2}{r^2}.$$ (5.7)

Likewise, the acceleration $a_{2\to1}$ of 2 towards 1 due to gravity is:

$$a_{2\to1} = \frac{F}{m_2} = \frac{Gm_1}{r^2}.$$ (5.8)

We note that $a_{1\to2} \neq a_{2\to1}$ when $m_1 \neq m_2$. More precisely, we have $a_{1\to2} > a_{2\to1}$ when $m_2 > m_1$ and conversely, i.e., the acceleration of the least massive object towards the most massive object due to gravity is higher.

Despite being superseded in modern physics by Einstein's theory of general relativity [80], de-

scribing gravity not as a force but as a consequence of spacetime curvature, Newton's law of universal gravitation is still used in many applications, as the theory provides precise approximations of the effect of gravity when gravitational fields are not extreme. In this chapter, we draw inspiration from this theory, notably from the formulation of acceleration, to build our proposed autoencoders. We highlight that Newtonian gravity concepts were already successfully leveraged for graph visualization [22] and to build symmetric node similarity scores [369].

From Physics to Node Embedding Spaces   We come back to our initial analogy between celestial objects in space and node embedding representations. In this paragraph, let us assume that we have at our disposal a model that is able to learn, for each node $i \in \mathcal{V}$ of a directed graph:

- a node embedding vector $z_i \in \mathbb{R}^d$, of dimension $d \ll n$, as before;

- but also a new *mass parameter* $m_i \in \mathbb{R}^+$.

. We explain how to learn m_i in the next sections. Such a parameter m_i would capture the propensity of i to attract other nodes from its neighborhood in this graph, i.e., to make them point towards i through a directed edge. From such an augmented model, we could apply Newton's equations in the resulting embedding. Specifically, we propose to use the *acceleration* $a_{i \to j} = \frac{Gm_j}{r^2}$ of a node i towards a node j due to gravity in the embedding as an indicator of the likelihood that i is connected to j in the directed graph, with $r^2 = \|z_i - z_j\|_2^2$. In a nutshell:

- the numerator captures the fact that some nodes are more influential than others in the graph. For instance, in a citation network of scientific publications, seminal groundbreaking articles are more influential and should be more cited than others. Here, the larger m_j the more likely i will be connected to j via the (i, j) directed edge;

- the denominator highlights that nodes with structural proximity in the graph, typically with a common neighborhood, are more likely to be connected, provided that the model effectively manages to embed these nodes close to each other in the embedding space. For instance, in a scientific publications citation network, an article i will more likely cite an article j if it comes from a similar field of study.

More precisely, instead of directly dealing with $a_{i \to j}$, we use $\log a_{i \to j}$ in the remainder of this chapter. Using the logarithm has two advantages. Firstly, thanks to its concavity we limit the potentially large values resulting from the acceleration towards very central nodes. Also, $\log a_{i \to j}$ can be negative, which is more convenient to reconstruct an unweighted edge using a sigmoid activation function, as follows:

$$\hat{A}_{ij} = \sigma(\log a_{i \to j}) = \sigma(\underbrace{\log Gm_j}_{\text{denoted } \tilde{m}_j} - \log \|z_i - z_j\|_2^2). \tag{5.9}$$

5.3.2   Gravity-Inspired GAE

For pedagogical purposes, we assumed that we had, at our disposal, a model providing the m_i masses. In this section, we detail how to actually learn them, using graph autoencoders.

Encoder   For the GAE encoder, we still leverage a multi-layer GCN, as defined in Definition 2.11 and processing an adjacency matrix A potentially combined with a node features matrix X. However, such a GCN will now assign a vector of size $(d + 1)$ to each node of the graph, instead of d as in standard GAE models. The first d dimensions correspond to the embedding vector of the node, i.e., z_i, where $d \ll n$ still denotes the dimension of the node embedding space. The last dimension will correspond to the model's estimate of $\tilde{m}_i = \log G m_i$. To sum up, we have:

$$\tilde{Z} = (Z, \tilde{M}) = \text{GCN}(A, X), \tag{5.10}$$

where, as in previous chapters, Z is the $n \times d$ node embedding matrix, and where $\tilde{M}$ is the n-dimensional vector of all values of $\tilde{m}_i$. $\tilde{Z} = (Z, \tilde{M})$ denotes the $n \times (d + 1)$ matrix row-concatenating Z and $\tilde{M}$. We note that learning $\tilde{m}_i$ is equivalent to learning m_i, but is also more convenient since we get rid of the gravitational constant G and of the logarithm.

In this GCN encoder, as we process directed graphs, we need to differentiate between incoming and outcoming edges during message passing. In this chapter, we therefore replace the usual symmetric normalization of A from Definition 2.5 by the out-degree normalization $\tilde{A}_{\text{out}}$:

Definition 5.1. The *out-degree normalization* of the adjacency matrix A of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with diagonal out-degree matrix D_{out} (as defined in Definition 2.4) is:

$$\tilde{A}_{\text{out}} = (D_{\text{out}} + I_n)^{-1}(A + I_n). \tag{5.11}$$

Therefore, at each layer of the GCN, the vector of a node becomes a weighted average of hidden vectors from the previous layer of the neighbors *to which it points*, together with its own vector.

Decoder and Optimization   We leverage the previously defined logarithmic version of acceleration, together with a sigmoid activation, to reconstruct an estimation of the adjacency matrix A from Z and $\tilde{M}$. Denoting $\hat{A}$ the reconstruction of A, we have:

$$\hat{A}_{ij} = \sigma(\tilde{m}_j - \log \|z_i - z_j\|_2^2). \tag{5.12}$$

For weighted graphs, $\hat{A}_{ij}$ is an estimation of some true normalized weight on the edge connecting i to j. For unweighted graphs, it corresponds to the probability of a missing edge from i to j.

Contrary to the inner product decoder, we generally have $\hat{A}_{ij} \neq \hat{A}_{ji}$. This approach is therefore suitable for directed graph reconstruction. During the training phase, we tune the GCN weights of this model in a similar fashion w.r.t. standard GAEs (see Section 2.4.1), i.e., we iteratively minimize a weighted cross entropy reconstruction loss [187], by gradient descent [117].

5.3.3 Gravity-Inspired VGAE

In this chapter, we also extend our proposed decoder to variational graph autoencoders.

Encoder We extend the VGAE inference model from Kipf and Welling [187]. Formally, using the same notation as in Section 2.4.2 from Chapter 2, we set:

$$q(\tilde{Z}|A, X) = \prod_{i=1}^{n} q(\tilde{z}_i|A, X), \text{ with } q(\tilde{z}_i|A, X) = \mathcal{N}(\tilde{z}_i|\mu_i, \text{diag}(\sigma_i^2)), \tag{5.13}$$

where $\tilde{z}_i = (z_i, \tilde{m}_i)$ is the $(d+1)$-dimensional vector concatening the d-dimensional embedding vector z_i and the scalar $\tilde{m}_i$. As is the case for standard VGAE, Gaussian parameters are learned from two GCNs, i.e., $\mu = \text{GCN}_\mu(A, X)$, with μ denoting the $n \times (d+1)$ matrix stacking up mean vectors μ_i for each node. Similarly, $\log \sigma = \text{GCN}_\sigma(A, X)$.

Decoder and Optimization As in Section 2.4.2, the actual embedding vectors $\tilde{z}_i$ are sampled from the above distributions. Then, we incorporate our gravity-inspired decoding scheme into the generative model attempting to reconstruct A:

$$p(A|\tilde{Z}) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(A_{ij}|\tilde{z}_i, \tilde{z}_j), \tag{5.14}$$

with:

$$p(A_{ij}|\tilde{z}_i, \tilde{z}_j) = \hat{A}_{ij} = \sigma(\tilde{m}_j - \log \|z_i - z_j\|_2^2). \tag{5.15}$$

During training, weights of the two GCN encoders are tuned by iteratively maximizing, by gradient ascent, the corresponding ELBO objective acting as a reconstruction quality measure.

5.3.4 Generalization of the Decoding Scheme

One can improve the flexibility of our decoders, both in the GAE and in the VGAE settings, by introducing an additional parameter $\lambda \in \mathbb{R}^+$ and by reconstructing $\hat{A}_{ij}$ as follows:

$$\hat{A}_{ij} = \sigma(\tilde{m}_j - \lambda \log \|z_i - z_j\|_2^2). \tag{5.16}$$

Decoders from Sections 5.3.2 and 5.3.3 correspond to special cases of Equation (5.16) where we set $\lambda = 1$. Such a parameter can be tuned by cross-validation on link prediction tasks (See section 5.4). Our interpretation of λ is twofold. Firstly, it balances the relative importance of distances in the embedding for reconstruction w.r.t. the mass attraction parameter. Then, from a "physics" point of view, it is equivalent to replacing the squared distance in Newton's formula with a distance to the power of 2λ. In our experimental analysis on link prediction, we will provide insights on when and why deviating from Newton's actual theory (i.e., $\lambda = 1$) is relevant.

5.3.5 On Complexity and Scalability

Assuming featureless nodes, a sparse representation of the adjacency matrix A with m non-zero entries, and considering that our models return a dense $n \times (d+1)$ embedding matrix $\tilde{Z}$, then the space complexity of our approach is $O(m+n(d+1))$, both in the GAE and VGAE frameworks. If nodes also have features summarized in the $n \times f$ matrix X, then the space complexity becomes $O(m + n(f + d + 1))$, with $d \ll n$ and $f \ll n$ in practice. Therefore, as is the case for standard GAE and VGAE models [187], space complexity increases linearly w.r.t. the size of the graph.

Moreover, due to the pairwise computations of L_2 distances between all d-dimensional vectors z_i and z_j involved in our gravity-inspired decoding scheme, our models have a quadratic time complexity $O(dn^2)$ w.r.t. the number of nodes in the graph, similarly to standard GAE and VGAE models. As our experiments from Section 5.4 will focus on medium-size graphs, such a quadratic complexity will be computationally affordable.

One can extend our method to larger graphs, e.g., with millions of nodes and edges, by applying the degeneracy framework proposed in Chapter 3, or a variant of this approach involving *directed graph degeneracy* a.k.a. "D-Cores" concepts [106]. One can also apply the FastGAE method from Chapter 4, either with degree-based or core-based sampling, or alternatively with a refined sampling strategy taking into account the directionality of edges. Floyd Everest (`fleverest` on GitHub) recently combined FastGAE with Gravity-Inspired GAE/VGAE models in one of his own repositories[1]. In Chapter 8, we will also mention a similar combination when leveraging Gravity-Inspired GAE and VGAE models for graph-based similar artists ranking on Deezer.

5.4 Experimental Analysis

5.4.1 Three Directed Link Prediction Tasks

We consider the following three learning tasks for our experimental evaluation of our models.

---

[1]https://github.com/fleverest/Gravity_FastGAE

5.4. Experimental Analysis

Task 1: General Directed Link Prediction   The first task is referred to as *general directed link prediction*. Similarly to experiments from Chapter 3 and 4, we train models on incomplete versions of graphs where 15% of edges were randomly removed. We take directionality into account in the masking process. In other words, if a link between node i and j is reciprocal, we can possibly remove the (i, j) edge but still observe the reverse (j, i) edge in the training graph. Then, we create validation and test sets from removed edges and from the same number of randomly sampled pairs of unconnected nodes. In the following, the validation set contains 5% of edges, and the test set contains 10% of edges. As in previous chapters, we evaluate the performance of our models on a binary classification task consisting in discriminating the actually removed edges from the fake ones, and we compare results using the AUC and AP scores.

This setting corresponds to the most general formulation of link prediction. However, due to the large number of unconnected pairs of nodes in numerous real-world graphs, we expect the impact of directionality on performances to be limited. Indeed, for each actual unidirectional edge (i, j) from the graph, it is unlikely to retrieve the reverse (unconnected) pair (j, i) among negative samples in the test set. As a consequence, models focusing on graph proximity and ignoring the direction of the link, such as standard GAEs and VGAEs, might still perform fairly well on such a task. For this reason, in the remainder of this section, we also propose and study two additional learning tasks, designed to reinforce the importance of directionality learning.

Task 2: Biased Negative Samples (B.N.S.) Link Prediction   For the second task, we also train models on incomplete versions of graphs where 15% of edges were removed: 5% initially extracted as a validation set, and 10% acting as a test set. However, the removed edges are all *unidirectional*, i.e., (i, j) exists but not (j, i). In this task, the reverse node pairs are included in validation and test sets and constitute negative samples. In other words, all node pairs from validation and test sets are included in *both* directions. As for Task 1, we evaluate the performance of our models on a binary classification task consisting in discriminating actual edges from fake ones, and therefore evaluate the ability of our models to correctly reconstruct $A_{ij} = 1$ and $A_{ji} = 0$ *simultaneously*.

This task has been presented by Zhou et al. [408] under the name *biased negative samples link prediction*. It is more challenging than Task 1, as the ability to reconstruct asymmetric relations is more crucial. Models ignoring directionality, such as standard GAEs and VGAEs who always predict $\tilde{A}_{ij} = \tilde{A}_{ji}$, will fail in such a setting.

Task 3: Bidirectionality Prediction   As a third task, we evaluate the ability of our models to discriminate *bidirectional* edges, i.e., reciprocal connections, from *unidirectional* edges. We create a training graph by removing at random one of the two directions of all bidirectional edges for an initial graph. Therefore, the training graph only has unidirectional connections. Then, we once again consider a binary classification problem, aiming to retrieve bidirectional edges in a

test set composed of their removed direction and of the same number of reverse directions from unidirectional edges (that are therefore fake edges a.k.a. negative samples). In other words, for each pair of nodes (i, j) from the test set, we observe a connection from j to i in the incomplete training graph, but only half of them are reciprocal. This third evaluation task, referred to as *bidirectionality prediction* in this chapter, also strongly relies on directionality learning. As a consequence, as for Task 2, standard GAEs and VGAEs are expected to perform poorly.

5.4.2 Experimental Setting

Datasets We provide experiments on three publicly available real-world directed graphs. Firstly, we consider the Cora and Citeseer *citation graphs* already described in Chapter 3, and consisting of scientific publications citing one another. Contrary to experiments from Chapters 3 and 4, we deal with the original *directed* versions of these graphs.

We also consider the Google directed web graph from Konect[2]. The 15 763 nodes of this graph are web pages, and its 171 206 directed edges represent hyperlinks between these pages. We point out that this graph is different from the large Google web graph from SNAP, that we mentioned in Chapter 3 and 4. To avoid confusion, we refer to it as "Google-Medium" in the remainder of this chapter, due to its smaller size w.r.t. the Google graph from previous chapters.

The Google-Medium graph is denser than Cora and Citeseer. It also has a higher proportion of bidirectional edges. Specifically, we have 2.86%, 1.20%, and 14.55% of bidirectional edges in Cora, Citeseer, and Google-Medium respectively. The three graphs are unweighted and featureless.

Models: Standard and Gravity-Inspired GAE and VGAE We train Gravity-Inspired GAE and VGAE models for each graph. For comparison purposes, we also train standard GAE and VGAE from Kipf and Welling [187]. Each of these four models includes a two-layer GCN encoder with a 64-dimensional hidden layer and with out-degree left normalization of A as defined in mentioned in Section 5.3.2. All models are trained for 200 epochs and return 32-dimensional embedding vectors (i.e., $d = 32$). We used the Adam optimizer [184], apply a learning rate of 0.1 for Cora and Citeseer and 0.2 for Google-Medium, trained models without dropout, performing full-batch gradient descent (or ascent for VGAE), and using the reparameterization trick [185] in the case of VGAEs. Also, for Task 1 and Task 3 we picked $\lambda = 1$ (respectively $\lambda = 10$) for Cora and Citeseer (resp. for Google-Medium). For Task 2 we picked $\lambda = 0.05$ for all three graphs, which we interpret and discuss in the next subsections. All hyperparameters were tuned from validation AUC scores on Task 1, i.e., on the general directed link prediction task.

---

[2]https://konect.cc/networks/

Models: Other Baselines Besides standard GAEs and VGAEs, we also compare the performance of our models to the graph embedding methods introduced in Section 5.2.2:

- our own Source/Target GAE and VGAE, extending the source/target paradigm to GAE and VGAE, and trained with similar settings w.r.t. standard and gravity-inspired models;

- HOPE [276], setting $\beta = 0.01$ and with source and target vectors of dimension 16, in order to learn 32-dimensional node representations.

- APP [408], training models over 100 iterations to learn 16-dimensional source and target vectors, i.e., 32-dimensional node representations. We adopted a similar setting and similar hyperparameters w.r.t. Zhou et al. [408]'s public implementation.

- for comparison purposes, in our experiments we also train *node2vec* models [123] that, while dealing with directionality in random walks, only return one 32-dimensional embedding vector per node. We rely on symmetric inner products with sigmoid activation for link prediction, and we therefore expect node2vec to underperform w.r.t. APP and HOPE on Tasks 2 and 3. We trained models with consistent hyperparameters w.r.t. Chapter 3.

We used Python and especially the TensorFlow library [1], except for APP where we used the authors' Java implementation [408]. We trained models on an NVIDIA GTX 1080 GPU and ran other operations on a double Intel Xeon Gold 6134 CPU.

5.4.3 Results and Discussion

We now present our experimental results. Our source code is publicly available on GitHub[3].

Results Table 5.1 reports mean AUC and AP scores, along with standard errors over 100 runs with different test sets, for each dataset and the three tasks under consideration. Overall, our Gravity-Inspired GAE and VGAE models achieve very competitive results.

On Task 1, standard GAE and VGAE models, despite ignoring directionality for graph reconstruction, still perform fairly well (e.g., with an 82.79% AUC score for the standard VGAE on Cora). This emphasizes the limited impact of directionality on performances for such a task, as expected in Section 5.4.1. Nonetheless, our gravity-inspired models significantly outperform their standard counterparts (e.g., with a 91.92% AUC score for our Gravity-Inspired VGAE on Cora), confirming the relevance of capturing both proximity *and* directionality for general directed link prediction. Moreover, our models often reach comparable results w.r.t. the other baselines specifically designed for directed graphs. Among them, APP is the best on the three datasets, together with the Source/Target GAE on the Google-Medium graph.

---
[3]https://github.com/deezer/gravity_graph_autoencoders

Table 5.1: Directed link prediction on the Cora, Citeseer, and Google-Medium graphs, using our Gravity-Inspired GAE/VGAE, standard GAE/VGAE, and other baselines. Scores are reported for the three tasks described in Section 5.4.1, and are averaged over 100 runs. All models learn embedding vectors of dimension $d = 32$. **Bold** numbers correspond to the best scores. Scores *in italic* are within one standard deviation range from the best ones.

Dataset	Model	Task 1: General Link Prediction		Task 2: B.N.S. Link Prediction		Task 3: Bidirectionality Prediction	
		AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Cora	*Gravity-Inspired VGAE (ours)*	91.92 ± 0.75	*92.46 ± 0.64*	**83.33 ± 1.11**	**84.50 ± 1.24**	*75.00 ± 2.10*	**73.87 ± 2.82**
	Gravity-Inspired GAE (ours)	87.79 ± 1.07	90.78 ± 0.82	*83.18 ± 1.12*	*84.09 ± 1.16*	**75.57 ± 1.90**	*73.40 ± 2.53*
	Standard VGAE	82.79 ± 1.20	86.69 ± 1.08	50.00 ± 0.00	50.00 ± 0.00	58.12 ± 2.62	59.70 ± 2.08
	Standard GAE	81.34 ± 1.47	82.10 ± 1.46	50.00 ± 0.00	50.00 ± 0.00	53.07 ± 3.09	54.60 ± 3.13
	Source/Target VGAE	85.34 ± 1.29	88.35 ± 0.99	63.00 ± 1.05	64.62 ± 1.37	*75.20 ± 2.62*	*73.86 ± 3.04*
	Source/Target GAE	82.67 ± 1.42	83.25 ± 1.51	57.81 ± 2.64	57.66 ± 3.35	65.83 ± 3.87	63.15 ± 4.58
	APP	**93.92 ± 1.01**	**93.26 ± 0.60**	69.20 ± 0.65	67.93 ± 1.09	72.85 ± 1.91	70.97 ± 2.60
	HOPE	80.82 ± 1.63	81.61 ± 1.08	61.84 ± 1.84	63.73 ± 1.12	65.11 ± 1.40	64.24 ± 1.18
	node2vec	79.01 ± 2.00	84.20 ± 1.62	50.00 ± 0.00	50.00 ± 0.00	66.97 ± 1.41	67.61 ± 1.80
Citeseer	*Gravity-Inspired VGAE (ours)*	*87.67 ± 1.07*	*89.79 ± 1.01*	**76.19 ± 1.35**	**79.27 ± 1.24**	**71.61 ± 3.20**	**71.87 ± 3.87**
	Gravity-Inspired GAE (ours)	78.36 ± 1.55	84.75 ± 1.10	*75.32 ± 1.53*	*78.47 ± 1.27*	*71.48 ± 3.64*	*71.50 ± 3.62*
	Standard VGAE	78.56 ± 1.43	83.66 ± 1.09	50.00 ± 0.00	50.00 ± 0.00	47.66 ± 3.73	50.31 ± 3.27
	Standard GAE	75.23 ± 2.13	75.16 ± 2.04	50.00 ± 0.00	50.00 ± 0.00	45.01 ± 3.75	49.79 ± 3.71
	Source/Target VGAE	79.45 ± 1.75	83.66 ± 1.32	57.32 ± 0.92	61.02 ± 1.37	69.67 ± 3.12	67.05 ± 4.10
	Source/Target GAE	73.97 ± 3.11	75.03 ± 3.37	56.97 ± 1.33	57.62 ± 2.62	54.88 ± 6.02	55.81 ± 4.93
	APP	**88.70 ± 0.92**	**90.29 ± 0.71**	64.35 ± 0.45	63.70 ± 0.51	64.16 ± 1.90	63.77 ± 3.28
	HOPE	72.91 ± 0.59	71.29 ± 0.52	60.24 ± 0.51	61.28 ± 0.57	52.65 ± 3.05	54.87 ± 1.67
	node2vec	71.02 ± 1.78	77.70 ± 1.22	50.00 ± 0.00	50.00 ± 0.00	61.08 ± 1.88	63.63 ± 2.77
Google Medium	*Gravity-Inspired VGAE (ours)*	**97.84 ± 0.25**	*98.18 ± 0.14*	**88.03 ± 0.25**	**91.04 ± 0.14**	84.69 ± 0.31	84.89 ± 0.30
	Gravity-Inspired GAE (ours)	*97.77 ± 0.10*	**98.43 ± 0.10**	*87.71 ± 0.29*	*90.84 ± 0.16*	**85.82 ± 0.63**	**85.91 ± 0.50**
	Standard VGAE	87.14 ± 1.20	88.14 ± 0.98	50.00 ± 0.00	50.00 ± 0.00	40.03 ± 4.98	44.69 ± 3.52
	Standard GAE	91.34 ± 1.13	92.61 ± 1.14	50.00 ± 0.00	50.00 ± 0.00	41.35 ± 1.92	41.92 ± 0.81
	Source/Target VGAE	96.33 ± 1.04	96.24 ± 1.06	85.30 ± 3.18	84.69 ± 4.42	75.11 ± 2.07	73.63 ± 2.06
	Source/Target GAE	*97.76 ± 0.41*	*97.74 ± 0.40*	86.16 ± 2.95	86.26 ± 3.33	82.27 ± 1.29	80.10 ± 1.80
	APP	97.04 ± 0.10	96.97 ± 0.11	83.06 ± 0.46	85.15 ± 0.42	73.43 ± 0.16	68.74 ± 0.19
	HOPE	81.16 ± 0.67	83.02 ± 0.35	74.23 ± 0.80	72.70 ± 0.79	70.45 ± 0.18	70.84 ± 0.22
	node2vec	83.11 ± 0.27	85.79 ± 0.30	50.00 ± 0.00	50.00 ± 0.00	78.99 ± 0.35	76.72 ± 0.53

On Task 2, our models consistently achieve the best performances (e.g., with a top 76.19% AUC score on Citeseer, 11+ points above the best baseline). Models ignoring directionality for prediction, i.e., node2vec and standard GAE/VGAE, totally fail (50.00% AUC and AP scores on all graphs, corresponding to the random classifier level) which was expected since test sets include both directions of each node pair. Experiments on Task 3 confirm the superiority of our approach when dealing with challenging tasks where directionality learning is crucial. On this last task, gravity-inspired models also outperform alternative approaches (e.g., with a top 85.82% AUC score for Gravity-Inspired GAE on Google-Medium).

We hardly found any consistent and significant performance gap between *determinic* GAEs and their *variational* counterparts. This result is in phase with previous empirical results in the literature [187, 279] on undirected graphs, as well as in our own previous insights from Chapters 3 and 4. In futures studies, we might investigate alternative prior distributions for VGAE, to challenge the Gaussian hypothesis that, despite being convenient for computations, might not be optimal [187]. Last, we note that all GAE and VGAE models required a comparable training time of roughly 7 seconds (respectively 8 seconds, 5 minutes) for Cora (resp. for Citeseer, for Google-Medium) on our machine. Baselines were faster: for instance, on Google-Medium, 1 minute (resp. 1.30 minutes, 2 minutes) were required to train HOPE (resp. APP, node2vec).
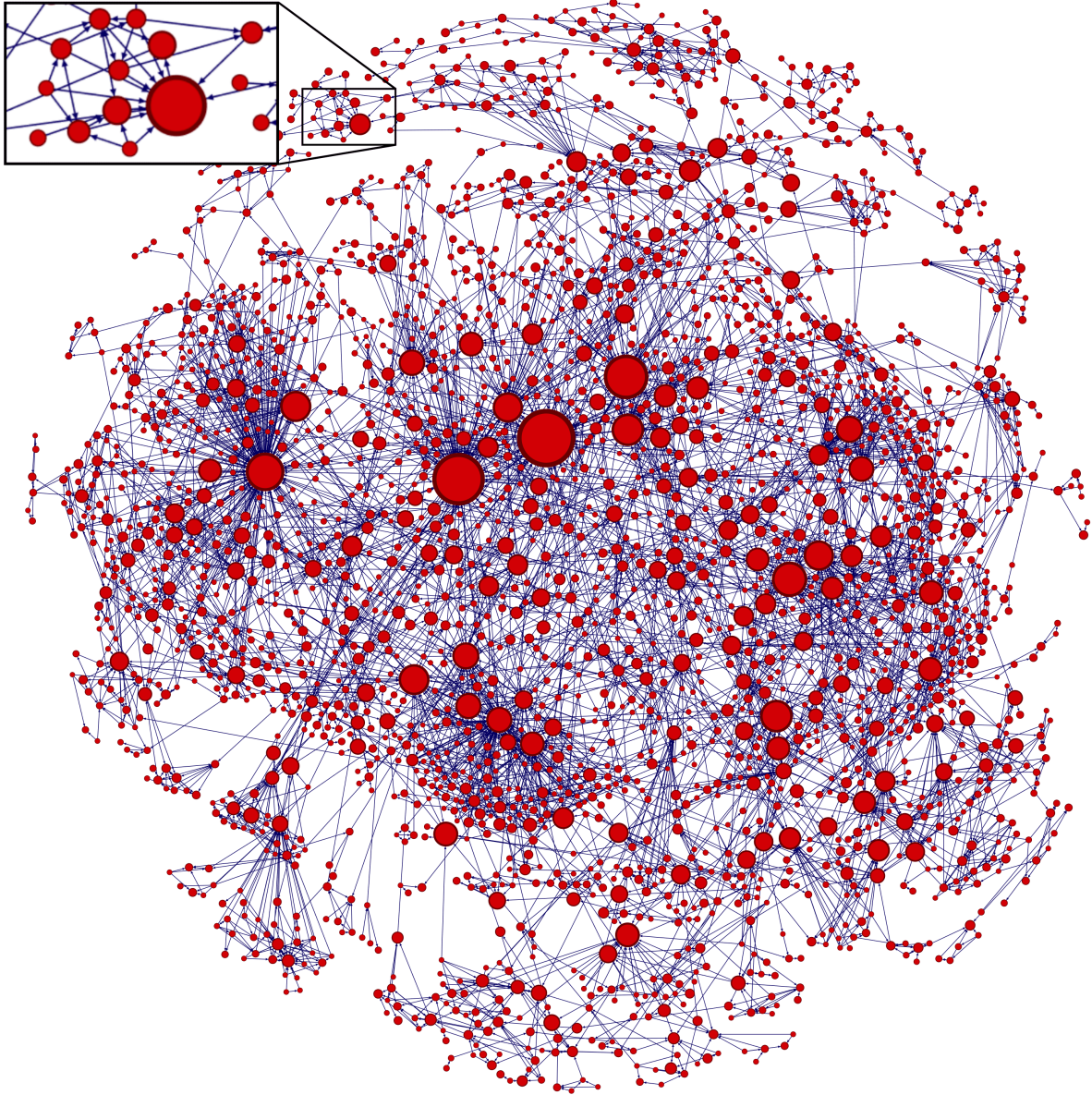
Figure 5.1: Visualization of the Cora graph based on node embedding representations learned from our Gravity-Inspired VGAE model. In this graph, nodes are scaled using the mass parameters $\tilde{m}_i$. The node separation is based on distances in the embedding, using the method described by Fruchterman and Reingold [100] on Gephi.

More insights on $\tilde{m}_i$ To pursue our experimental analysis, we propose a discussion on the nature of $\tilde{m}_i$ (in this paragraph), followed by discussions on the role of λ to balance node proximity and influence and on some possible extensions of our work (in the next paragraphs).

Figure 5.1 provides a visualization of the Cora graph, using node embedding representations and $\tilde{m}_i$ masses learned by our Gravity-Inspired VGAE model. In such a visualization, we observe that nodes with smaller masses tend to be connected to nodes with larger masses from their embedding neighborhood, which was expected by the design of our decoding scheme.

From Figure 5.1, one might argue that $\tilde{m}_i$ tend to reflect the node *centrality* in the graph. In this direction, we compared $\tilde{m}_i$ to the most common graph centrality measures. Specifically, in Table 5.2 we report Pearson correlation coefficients of $\tilde{m}_i$ w.r.t. the following measures, computed using the NetworkX [127] Python library:

- the *in-degree* and *out-degree* of the node, i.e., the number of edges coming into and going out of the node respectively;

- the *betweenness centrality*, which is, for a node $i \in \mathcal{V}$, the sum of the fraction of shortest paths between node pairs going through i, i.e., $c_B(i) = \sum_{(s,t)\in\mathcal{V}^2} \frac{\text{sp}(s,t|i)}{\text{sp}(s,t)}$, where $\text{sp}(s,t)$ is the number of shortest paths from node s to node t, and $\text{sp}(s,t|i)$ is the number of those paths going through the node i [37];

- the *PageRank* [278], computing a node "importance" ranking based on the structure of incoming links. It was originally designed to rank web pages;

- the *Katz centrality*, a generalization of the eigenvector centrality [177]. The Katz centrality of a node $i \in \mathcal{V}$ is: $c_K(i) = \alpha \sum_{j\in\mathcal{V}} A_{ij} c_K(j) + \beta$, where A is the adjacency matrix with largest eigenvalue $\lambda_{\max}$. Usually, $\beta = 1$ and $\alpha < \frac{1}{\lambda_{\max}}$ [177].

We observe in Table 5.2 that $\tilde{m}_i$ is positively correlated to all of these centrality measures, except for the out-degree where the correlation is negative (or almost null for Google-Medium), meaning that nodes with few edges going out of them tend to have larger values of $\tilde{m}_i$. Correlations are not perfect, which emphasizes that our models do *not* exactly learn one of these measures. We also note that centralities are lower for Google-Medium, which might be due to the structure of this graph and especially to its density. In our experiments, we tried to replace $\tilde{m}_i$ by any of these (normalized) centrality measures when performing link prediction.

Table 5.2: Pearson correlation coefficients of $\tilde{m}_i$ w.r.t. common centrality measures, for our Gravity-Inspired VGAE model. Correlation w.r.t. the Katz score is not reported on the Google-Medium graph due to its computational complexity.

Centrality Measures	Cora	Citeseer	Google Medium
In-degree	0.5960	0.6557	0.1571
Out-degree	-0.2662	-0.1994	0.0559
Betweenness	0.5370	0.4945	0.2223
Pagerank	0.4143	0.3715	0.1831
Katz	0.5886	0.6428	-

Specifically, we tried to learn optimal vectors z_i while these scores were fixed as masses values, achieving underperforming results. For instance, we reached an 89.05% main AUC score by using betweenness centrality on Cora instead of the actual $\tilde{m}_i$ learned by the VGAE, which is above standard VGAE (82.79% AUC) but below the Gravity-Inspired VGAE with optimal $\tilde{m}_i$ (91.92% AUC). Also, using centrality measures as initial values for $\tilde{m}_i$ before model training did not significantly impact performances in experiments.

(a) Task 1   (b) Task 2

(c) Task 3
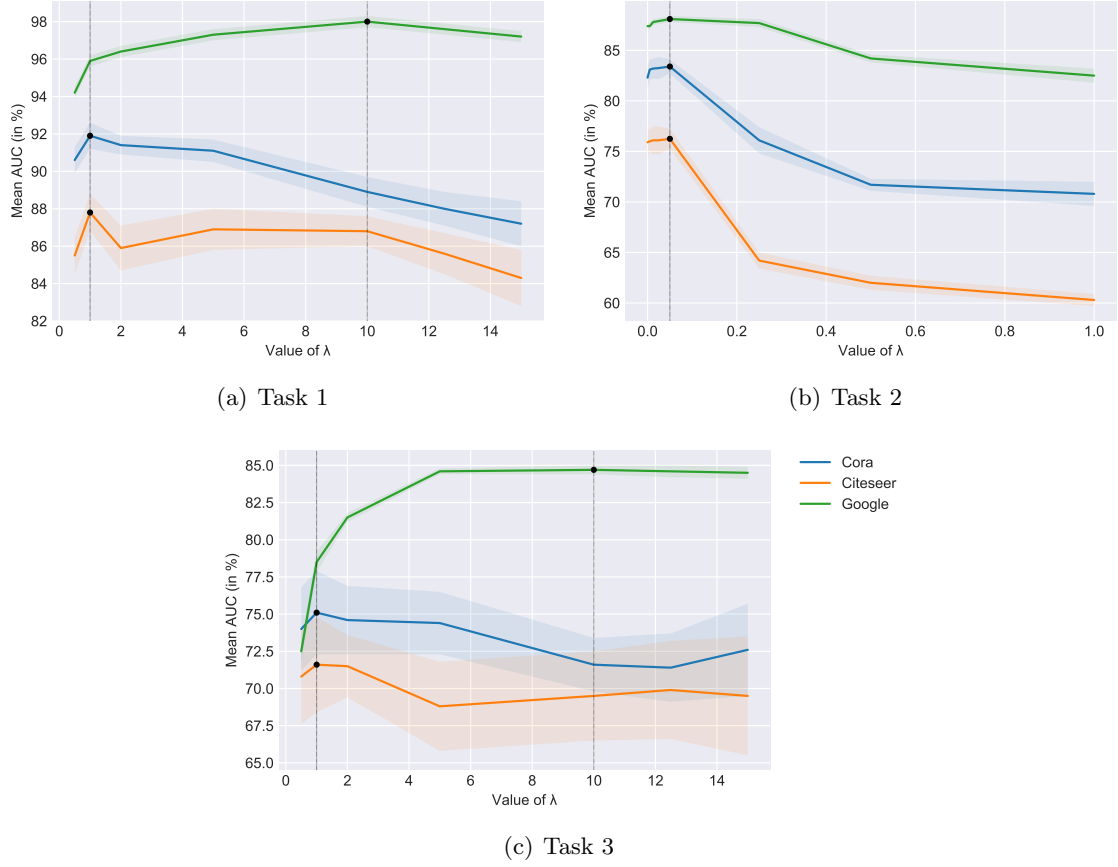
Figure 5.2: Optimal values of the hyperparameter λ w.r.t. validation AUC scores, for the Gravity-Inspired VGAE model, for all three tasks and for all three graphs under consideration.

Impact of λ In Equation (5.16), we introduced the hyperparameter $\lambda \in \mathbb{R}^+$ to tune the relative importance of node proximity in the embedding space w.r.t. mass attraction, leading to the reconstruction scheme $\hat{A}_{ij} = \sigma(\tilde{m}_j - \lambda \log \|z_i - z_j\|_2^2)$. In Figure 5.2, we show the impact of λ on mean AUC scores for the Gravity-Inspired VGAE model and for all three graphs. For Cora and Citeseer, on Task 1 and Task 3, $\lambda = 1$ is an optimal choice, consistently with Newton's formula. However, for Google-Medium, on Task 1 and Task 3, we obtained better performances for higher values of λ, notably for $\lambda = 10$ that we used in our experiments. Increasing λ reinforces the relative importance of the node pair's (symmetric) distance, measured by $\log \|z_i - z_j\|_2^2$, in the decoder w.r.t. parameter $\tilde{m}_j$ capturing the global influence of a node on its neighbors and therefore asymmetries in links. Since the Google-Medium graph is denser than Cora and Citesser, and has a higher proportion of symmetric relations (see Section 5.4.2), putting the emphasis on node proximity appears as a relevant strategy.

On a contrary, on Task 2 we achieved optimal performances by setting $\lambda = 0.05$, for all three graphs. As $\lambda < 1$, we improved scores by assigning more relative importance to the mass parameter $\tilde{m}_j$. Such a result is not surprising as, for *biased negative samples link prediction*

task, capturing the directionality is more crucial than the proximity, as nodes pairs from test sets are all included in both directions. As illustrated in Figure 5.2(b), increasing λ significantly deteriorates performances.

Extensions   Throughout these experiences, we focused on featureless graphs, to fairly compete with the HOPE, APP, and node2vec methods (see Section 2.2). However, as explained in Section 5.3.2, our models can easily leverage node features X, in addition to the graph structure summarized in A. Moreover, we recall that the Gravity-Inspired GAE and VGAE models are not limited to GCN encoders, and can be generalized to any alternative GNN models. However, so far, we assumed a fixed graph structure. Future research on directed link prediction in *dynamic* graphs [179] could definitely improve our approach. Also, while we focused on directed link prediction, we neglected the community detection problem in this chapter. Community detection in directed graphs is a challenging problem [248], which could deserve further attention in our future research studies.

5.5   Conclusion

In this chapter, we extended GAEs and VGAEs to directed graphs. We drew inspiration from physics to introduce a new gravity-inspired decoder, that can effectively reconstruct asymmetric relations from node embedding spaces. We achieved competitive experimental results on three different directed link prediction tasks, for which standard GAE and VGAE models perform poorly. We also pinpointed several research directions that, in the future, should lead to the improvement of our approach.

Since the publication of this work in late 2019 [311], various recent papers explicitly mentioned our Gravity-Inspired GAE and VGAE models. Several of them included these models into their own experiments, including [142, 299, 304, 395, 401]. In Chapter 8, we will ourselves leverage these models once again, to address music recommendation problems. Specifically, in this chapter, we will model the *cold start similar artists ranking* problem [313] as a link prediction task in a directed and attributed graph, connecting music artists to their top-k most similar neighbors and incorporating side musical information as node features. We will show how the gravity-inspired decoder can be used to automatically rank the top-k most similar neighbors of new artists. Such an application will also emphasize how Gravity-Inspired GAE/VGAE models, when equipped with node features, can be used in *inductive* settings that involve generalizing representations to new unseen nodes after training.

6

Simplifying Graph Autoencoders
with One-Hop Linear Models

This chapter presents research conducted with Romain Hennequin and Michalis Vazirgiannis, and published in the proceedings of the 2020 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2020) [310]. A preliminary version of this work has also been presented at the "Graph Representation Learning" workshop of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019) [309].

6.1   Introduction

Despite the prevalent use of multi-layer GCN encoders in the recent literature (see Section 6.2), at the time of this work the relevance of this architecture choice had never been thoroughly studied nor challenged. The actual benefit of incorporating multi-layer GCNs, or even more complex GNNs, in GAE and VGAE models remained unclear. This is an important question, as simpler encoding strategies are easier to understand, to train, to deploy in production, and to debug, and might therefore be preferred for real-world industrial applications.

In this chapter, we propose to tackle this important aspect, showing that GCN-based GAE and VGAE models are often unnecessarily complex for numerous applications. Our work falls into a family of recent efforts questioning the systematic use of complex deep learning methods without a clear comparison to less fancy but simpler baselines [93, 228, 326].

More precisely, in this chapter, we introduce simpler versions of the GAE and VGAE models, referred to as *Linear GAE* and *Linear VGAE*. We propose to replace multi-layer GCN encoders

with linear models w.r.t. the direct neighborhood (one-hop) adjacency matrix of the graph, involving a unique weight matrix to tune, fewer operations, and no activation function.

Then, through an extensive empirical analysis on 17 real-world graphs with various sizes and characteristics, we show that these simplified models consistently reach competitive performances w.r.t. GCN-based GAE and VGAE models on link prediction and community detection tasks. We identify the settings where simple linear encoders appear as an effective alternative to GCNs, and as a first relevant baseline to implement before considering more complex models. In this chapter, we also question the relevance of the current benchmark datasets (Cora, Citeseer, Pubmed) commonly used in the literature to evaluate GAE and VGAE models.

This chapter is organized as follows. In Section 6.2 we introduce our proposed Linear GAE and VGAE models, and mention some related work aiming to simplify GCNs. In Section 6.3, we present our experimental results, and provide discussions on the simplification and the evaluation of GAE and VGAE models. We conclude in Section 6.4.

6.2 Simplifying Graph Autoencoders

We adopt a consistent notation with respect to previous chapters but, for the sake of simplicity, we assume in this section that the graph $\mathcal{G}$ is *undirected*. $\tilde{A}$ therefore denotes the symmetric normalization of A, consistently with Definition 2.5 from Chapter 2.

Nonetheless, our models could be straightforwardly extended to *directed* graphs by replacing $\tilde{A}$ by the out-degree normalization $\tilde{A}_{\text{out}}$, as in Chapter 5, in all equations of this section. Our analysis from Section 6.3 will include experiments related to such an extension, in order to simplify as well our Gravity-Inspired GAE and VGAE models from Chapter 5.

6.2.1 One-Hop Linear Encoders

To this day, multi-layer GCNs remain the most popular encoders for GAE and VGAE models building upon the seminal work of Kipf and Welling [187], including (but not limited to) the recent research of [55, 72, 124, 135, 156, 187, 279, 299, 329, 361]. We ourselves mainly considered multi-layer GCN encoders in our previous experiments from Chapters 3, 4, and 5, although our contributions are not restricted to this architecture choice.

We recall that, in a multi-layer GCN [188] with $L \geq 2$ layers, with an input layer $H^{(0)} = I_n$ (in the absence of node features) or $H^{(0)} = X$, and with an output layer $H^{(L)}$ (with $H^{(L)} = Z$ for

a GAE, and $H^{(L)} = \mu$ or $\log \sigma$ for a VGAE), embedding vectors are computed as follows:

$$
\begin{cases}
H^{(0)} = X \ (\text{or } I_n) \\
H^{(l)} = \text{ReLU}(\tilde{A}H^{(l-1)}W^{(l-1)}), \ \ \text{for } l \in \{1, ..., L-1\} \\
H^{(L)} = \tilde{A}H^{(L-1)}W^{(L-1)},
\end{cases}
\tag{6.1}
$$

using the notation from Definition 2.11. In this chapter, we consider a simpler linear model w.r.t. the normalized one-hop (i.e., direct neighborhood) adjacency matrix of the graph. More precisely, the term *linear encoder* will refer to the following function.

Definition 6.1. A *linear encoder* is a function taking as input an adjacency matrix A, potentially equipped with a node feature matrix $X \in \mathbb{R}^{n \times f}$, and returning an output matrix $Z \in \mathbb{R}^{n \times d}$ computed as follows:

$$
Z = \text{Linear}(A, X) = \tilde{A}XW,
\tag{6.2}
$$

for some trainable weight matrix $W \in \mathbb{R}^{f \times d}$. In particular, in the absence of node features (i.e., $X = I_n$), we have:

$$
Z = \text{Linear}(A, I_n) = \tilde{A}W,
\tag{6.3}
$$

and the weight matrix W is then of dimension $n \times d$.

In Sections 6.2.2 and 6.2.3, we incorporate this straightforward one-hop linear encoder in the GAE and VGAE frameworks, and discuss the implications of such a modification.

6.2.2   Linear GAE

In this chapter, we propose to replace the multi-layer GCN encoder of standard GAE and VGAE models with the linear encoder from Definition 6.1. Firstly, in the GAE framework, we set:

$$
Z = \text{Linear}(A, X) = \tilde{A}XW, \ \text{ then } \hat{A} = \sigma(ZZ^T).
\tag{6.4}
$$

In the absence of node features X, the model is simplified as follows:

$$
Z = \tilde{A}W, \ \text{ then } \hat{A} = \sigma(ZZ^T).
\tag{6.5}
$$

We refer to this model as *Linear Graph Autoencoder (Linear GAE)*. In the absence of node features, embedding vectors are obtained by multiplying the $n \times n$ normalized adjacency matrix $\tilde{A}$ by a *unique $n \times d$* weight matrix W. In the presence of node features, the linear encoder also simply consists in multiplying $\tilde{A}X$ with a unique $f \times d$ weight matrix W.

We tune this matrix W in a similar fashion w.r.t. standard GAE models considered in the previous chapters, i.e., we iteratively minimize a weighted cross-entropy loss capturing the quality
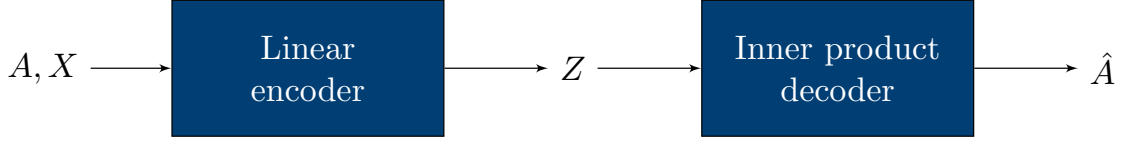
Figure 6.1: Schematic representation of a Linear GAE model.

of the reconstruction $\hat{A}$ w.r.t. the original matrix A, by gradient descent [187].

This encoder is a straightforward linear mapping. It can be interpreted as the simplest possible GCN (with a single layer). Each element of the z_i embedding vector is a weighted average from node i's direct one-hop connections. Contrary to multi-layer GCN encoders (with $L \geq 2$ layers):

- we *ignore the higher-order information* from the k-hop neighborhood with $k > 1$;

- also, we do not include any *non-linear activation function*.

In Section 6.3, we will highlight the very limited impact of these two simplifications on empirical performances. The above Figure 6.1 provides an illustration of a Linear GAE model.

Our proposed linear encoder runs in a linear time w.r.t. the number of edges m of the graph using a sparse representation for $\tilde{A}$, and involves fewer matrix operations than a multi-layer GCN. It includes nd parameters in the absence of node features (resp. fd parameters with node features), i.e., fewer than the $nd + (L-1)d^2$ parameters (resp. $fd + (L-1)d^2$ parameters) required by a L-layer GCN where all layers are of dimension d.

As is the case for standard GAEs, the inner product decoder has a $O(dn^2)$ complexity, as it involves the multiplication of the two dense matrices Z and Z^T. Such a complexity can be significantly reduced by leveraging the scalable methods we proposed in Chapter 3 and 4. Our evaluation from Section 6.3 will include experiments on large graphs requiring the use of such methods.

To emphasize that the Linear GAE model is not restricted to inner product decoders, the evaluation from Section 6.3 will also consider two alternative decoders to replace inner products [124, 311], including our proposed gravity-inspired decoder from Chapter 5.

6.2.3 Linear VGAE

We adopt a similar approach to replace the two multi-layer GCN encoders of standard VGAE models [187] by:

$$\mu = \text{Linear}_\mu(A, X) = \tilde{A}XW_\mu \text{ and } \log \sigma = \text{Linear}_\sigma(A, X) = \tilde{A}XW_\sigma, \quad (6.6)$$
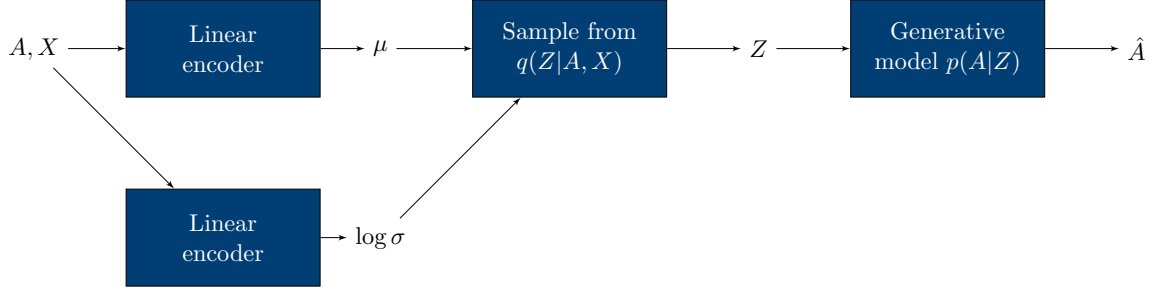
Figure 6.2: Schematic representation of a Linear VGAE model.

with $f \times d$ weight matrices W_μ and W_σ. In the absence of node features X, the encoding step is simplified as follows:

$$\mu = \tilde{A}W_\mu \text{ and } \log \sigma = \tilde{A}W_\sigma, \tag{6.7}$$

with $n \times d$ weight matrices W_μ and W_σ. Then:

$$\forall i \in \mathcal{V}, z_i \sim \mathcal{N}(\mu_i, \text{diag}(\sigma_i^2)), \tag{6.8}$$

with a similar decoder a.k.a. generative process w.r.t. standard VGAEs (see Section 2.4.2 from Chapter 2). We refer to this model as *Linear Variational Graph Autoencoder (Linear VGAE)*. During the learning phase, as is the case for standard VGAEs, we iteratively optimize the ELBO variational lower bound from Equation (2.40), w.r.t. W_μ and W_σ, by gradient ascent [187]. Figure 6.2 provides a schematic representation of the proposed Linear VGAE model.

6.2.4   Related Work

Our work falls into a family of research efforts aiming to challenge and question the prevalent use of complex deep learning methods without a clear comparison to simpler baselines [93, 228, 326]. In particular, in a concurrent study, Wu et al. [383] also recently proposed to simplify GCN models, notably by removing non-linearities between layers and collapsing some weight matrices during training. Their simplified model, referred to as Simple Graph Convolution (SGC), empirically rivals standard GCNs on several large-scale classification tasks [383]. While our work also focuses on simplifications of GCNs, we argue that the two research studies actually tackle different and complementary problems:

- Wu et al. [383] focus on supervised and semi-supervised settings. They consider the GCN as the model itself, optimized to classify node-level labels. On the contrary, we consider two *unsupervised* settings, in which GCNs are only a building part (the encoder) of a larger framework (the GAE or the VGAE), and where we optimize reconstruction losses/objectives from GCN-based embedding vectors (for GAE) or from vectors drawn from distributions learned through two GCNs (for VGAE);

- besides, our encoders only capture one-hop interactions, i.e., nodes only aggregate information from their direct neighbors during message passing. On the contrary, Wu et al. [383] still rely on a stacked layers design that, although simplified, permits learning from higher-order interactions. Contrary to us, considering such relations is crucial in their model for good performances [383]. We will show in Section 6.3 that, in our setting, it would mainly increase running times while bringing few to no improvement.

6.3 Experimental Analysis

In this section, we propose an in-depth empirical evaluation of our Linear GAE and VGAE models. Our Python/TensorFlow implementation of these models is publicly available on GitHub[1].

6.3.1 Experimental Setting

Tasks We consider two tasks. Firstly, we focus on *link prediction*, similarly to Chapters 3 to 5. As in these chapters, we train models on incomplete versions of graphs where 15% of edges were randomly removed, and used for validation (5%) and test (10%), together with the same number of randomly sampled pairs of unconnected nodes. We evaluate the ability of our models to classify edges from non-edges in these sets using, as in previous chapters, the mean AUC and AP scores averaged over 100 runs (resp. 10 runs for datasets with $n > 100\,000$). Models were trained on 100 (resp. 10) different random train/test splits.

As a second task, we perform *community detection*. When datasets include node-level ground truth communities. Similarly to experiments from Chapters 3 and 4, we train models on complete graphs, then run k-means algorithms in node embedding spaces. We compare the resulting clusters to ground truth communities via mean AMI scores.

Datasets We provide experiments on 17 publicly available real-world graphs. For each graph, Tables 6.1 and 6.2 report the number of nodes n, the number of edges m, and the dimension f of node feature vectors, when such vectors are available:

- we first consider the Cora, Citeseer, and Pubmed citation graphs[2] [323], with and without node features corresponding to f-dimensional bag-of-words vectors. These three graphs were used in the original experiments of Kipf and Welling [187], and subsequently in the wide majority of recent works on GAE and VGAE models including (but not limited to) [124, 135, 279, 282, 308, 329, 359, 361, 370], becoming the *de facto* benchmark datasets for evaluating GAE and VGAE. We ourselves considered these citation networks in our

---

[1]https://github.com/deezer/linear_graph_autoencoders
[2]https://linqs.soe.ucsc.edu/data

own experiments from Chapters 3 to 4. Therefore, we saw value in comparing Linear and GCN-based models on these graphs;

- we also report results on 14 alternative graphs, including some of those from the previous chapters. We consider four other citations networks: Patent[3] from Chapters 3 and 4, DBLP[4], Arxiv-HepTh[3], and a larger version of Cora[4], that we denote Cora-larger. We add the WebKD[2], Blogs[4] and Stanford[4] web graphs, where hyperlinks connect web pages, as well as two Google web graphs: the medium-size one[4] from Chapter 5, denoted Google-Medium, and the larger one[3] from Chapters 3 and 4, denoted Google. We complete the list with two social networks (Hamsterster[4] and LiveMocha[4]), the Flickr[4] image graph (nodes represent images, connected when sharing metadata), the Proteins[4] network of proteins interactions and the Amazon[4] products co-purchase network. Therefore, in these experiments, we consider a wide variety of real-world graphs of various origins, characteristics and sizes (from 877 to 2.7 million nodes, from 1 608 to 13.9 million edges).

Models   In all experiments, we compare the Linear GAE and VGAE models to 2-layer and 3-layer GCN-based graph GAE and VGAE models. We do not report performances of deeper models, due to a significant deterioration of all scores. For a comparison to other methods and notably to non-GAE/VGAE methods, which is out of the scope of this study, we refer to experiments from the previous chapters, as well as to experiments from the next Chapter 7 for a stronger emphasis on community detection.

All models were trained for 200 epochs (resp. 300 epochs) for graphs with $n < 100\,000$ (resp. $n \geq 100\,000$). We thoroughly checked the convergence of all models, in terms of mean AUC performances on validation sets, for these epochs numbers. As in Chapters 3 and 4, we ignored edges directions when initial graphs were directed, as we focus on symmetric inner product decoders in most of this chapter[5]. For Cora, Citeseer, and Pubmed, we set identical hyperparameters w.r.t. Kipf and Welling [187] to reproduce their results, i.e., we had $d = 16$, 32-dimensional hidden layer(s) for GCNs, and we used the Adam optimizer [184] with a learning rate of 0.01.

For other datasets, we tuned hyperparameters by performing a grid search on the validation set. We adopted a learning rate of 0.1 for Arxiv-HepTh, Patent, and Stanford; of 0.05 for Amazon, Flickr, LiveMocha, and Google; of 0.01 for Blogs, Cora-larger, DBLP, Google-Medium, Hamsterster, and Proteins (GAE models); of 0.005 for WebKD (except Linear GAE and VGAE where we used 0.001 and 0.01) and Proteins (VGAE models). We set $d = 16$ (but we reached similar conclusions with $d = 32$ and 64), with 32-dimensional hidden layer(s) and without dropout.

---

[3]https://snap.stanford.edu/data/index.html

[4]https://konect.cc/networks/

[5]We will nonetheless relax this restriction and consider directed edges during training and decoding later in Section 6.3.2, when we will extend our linear encoders to Gravity-Inspired GAE and VGAE models.

Table 6.1: Link prediction on the Cora, Citeseer, and Pubmed common benchmark datasets, with Linear GAE and VGAE models, and with their multi-layer GCN-based counterparts. We report details on hyperparameters for each model and dataset in Section 6.3.1. Cells are colored in blue when Linear GAE/VGAE models are reaching competitive results w.r.t. standard GCN-based models, i.e., results that are at least as good as GCN-based models $\pm$ 1 standard deviation. Note: as we aim to evaluate whether linear models are *as good* as others, and not necessarily *better*, we do not report **bold** numbers in this table contrary to tables from previous chapters.

Model	Cora (n = 2 708, m = 5 429)		Citeseer (n = 3 327, m = 4 732)		Pubmed (n = 19 717, m = 44 338)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear GAE (ours)	83.19 ± 1.13	87.57 ± 0.95	77.06 ± 1.81	83.05 ± 1.25	81.85 ± 0.32	87.54 ± 0.28
2-layer GCN-based GAE	84.79 ± 1.10	88.45 ± 0.82	78.25 ± 1.69	83.79 ± 1.24	82.51 ± 0.64	87.42 ± 0.38
3-layer GCN-based GAE	84.61 ± 1.22	87.65 ± 1.11	78.62 ± 1.74	82.81 ± 1.43	83.37 ± 0.98	87.62 ± 0.68
Linear VGAE (ours)	84.70 ± 1.24	88.24 ± 1.02	78.87 ± 1.34	83.34 ± 0.99	84.03 ± 0.28	87.98 ± 0.25
2-layer GCN-based VGAE	84.19 ± 1.07	87.68 ± 0.93	78.08 ± 1.40	83.31 ± 1.31	82.63 ± 0.45	87.45 ± 0.34
3-layer GCN-based VGAE	84.48 ± 1.42	87.61 ± 1.08	79.27 ± 1.78	83.73 ± 1.13	84.07 ± 0.47	88.18 ± 0.31

Model	Cora, with features (n = 2 708, m = 5 429, f = 1 433)		Citeseer, with features (n = 3 327, m = 4 732, f = 3 703)		Pubmed, with features (n = 19 717, m = 44 338, f = 500)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear GAE (ours)	92.05 ± 0.93	93.32 ± 0.86	91.50 ± 1.17	92.99 ± 0.97	95.88 ± 0.20	95.89 ± 0.17
2-layer GCN-based GAE	91.27 ± 0.78	92.47 ± 0.71	89.76 ± 1.39	90.32 ± 1.62	96.28 ± 0.36	96.29 ± 0.25
3-layer GCN-based GAE	89.16 ± 1.18	90.98 ± 1.01	87.31 ± 1.74	89.60 ± 1.52	94.82 ± 0.41	95.42 ± 0.26
Linear VGAE (ours)	92.55 ± 0.97	93.68 ± 0.68	91.60 ± 0.90	93.08 ± 0.77	95.91 ± 0.13	95.80 ± 0.17
2-layer GCN-based VGAE	91.64 ± 0.92	92.66 ± 0.91	90.72 ± 1.01	92.05 ± 0.97	94.66 ± 0.51	94.84 ± 0.42
3-layer GCN-based VGAE	90.53 ± 0.94	91.71 ± 0.88	88.63 ± 0.95	90.20 ± 0.81	92.78 ± 1.02	93.33 ± 0.91

Lastly, due to the prohibitive cost of decoding the entire matrix $\hat{A}$ for large graphs (i.e., Amazon, Flickr, Google, LiveMocha, Patent, and Stanford that all verify with $n > 100\,000$), we adopted a stochastic sampling strategy for these graphs. At each training iteration, we estimated losses by only reconstructing a subgraph of 10 000 nodes from the original graph. These 10 000 nodes were randomly picked during training at each iteration. This strategy corresponds to a special case of the FastGAE method from Chapter 4, with $n_{(S)} = 10\,000$ and with *uniform* sampling[6].

6.3.2 Results and Discussion

Cora, Citeseer, and Pubmed benchmarks Table 6.1 reports link prediction results for Cora, Citeseer, and Pubmed. For standard 2-layer GAEs and VGAEs, we reproduce similar performances w.r.t. Kipf and Welling [187]. We show that Linear GAE and VGAE models consistently reach competitive performances w.r.t. 2-layer and 3-layer GCN-based models, i.e., they are at least as good ($\pm$ 1 standard deviation). In some settings, Linear GAE and VGAE models are even slightly better (e.g., +1.25 points in AUC for the Linear VGAE on Pubmed

---

[6]Experiments from this chapter and from the papers associated with this work [309, 310] were actually done *before* our study of FastGAE from Chapter 4. We acknowledge that using FastGAE with a non-uniform sampling, e.g., with degree-based sampling as in Section 4.2.2, should improve results w.r.t. a uniform sampling.

with features, w.r.t. a 2-layer GCN-based VGAE). These results emphasize the effectiveness of our proposed simple encoder on these datasets, where the empirical benefit of multi-layer GCNs is very limited. In Table 6.3, we consolidate our results by reaching similar conclusions on the community detection task. As explained in Chapter 3, in these graphs nodes are documents clustered in respectively 6, 7, and 3 topic classes, acting as communities. In almost all settings, Linear GAE and VGAE models rival their multi-layer GCN-based counterparts (e.g., +4.31 AMI points for the Linear VGAE on Pubmed with features, w.r.t. 2-layer GCN-based VGAE).

Alternative graph datasets   Table 6.2 reports link prediction results for all other graphs. Linear GAE models are competitive in 13 cases out of 15, and sometimes even achieve better performances (e.g., +1.72 AUC points for Linear GAE on the largest dataset, Patent, w.r.t. 3-layer GCN-based GAE). Moreover, Linear VGAE models rival or outperform GCN-based models in 10 cases out of 15. Overall, Linear GAE and VGAE models also achieve very close results w.r.t. GCN-based models in all remaining datasets (e.g., on Google-Medium, with a mean AUC score of 96.02% ± 0.14 for Linear GAE, only slightly below the mean AUC score of 96.66% ± 0.24 of 2-layer GCN-based GAE). This confirms the empirical effectiveness of simple node encoding schemes, that appear as a suitable alternative to more complex multi-layer encoders for many real-world applications. Regarding community detection in Table 6.3, Linear GAE and VGAE models are competitive on the Cora-larger graph, in which nodes are documents clustered in 70 topic classes. However, 2-layer and 3-layer GCN-based models are significantly outperforming on the Blogs graph, where political blogs are classified as either left-leaning or right-leaning (e.g., −23.42 AMI points for Linear VGAE w.r.t. 2-layer GCN-based VGAE).

On running times and extensions to k-hop linear encoders   While this work puts the emphasis on performance and not on training speed, we also note that Linear GAE and VGAE models are 10% to 15% faster than their GCN-based counterparts. For instance, on our NVIDIA GTX 1080 GPU, we report a 6.03 seconds (vs 6.73 seconds) mean running time for training our Linear VGAE (vs a 2-layer GCN-based VGAE) on the featureless Citeseer dataset, and roughly 800 seconds (vs 900+ seconds) on the Patent dataset, using our sampling strategy from Section 6.3.1. This gain comes from the fewer parameters and matrix operations required by one-hop linear encoders and from the sparsity of the one-hop matrix $\tilde{A}$ in our graphs.

Besides, while in this work we only learn embedding vectors from direct neighbors, variants of our models could capture higher-order links by considering polynomials of the matrix A. For instance, we could learn node embedding representations from one-hop and two-hop links by replacing $\tilde{A}$ by the normalized version of $A + \alpha A^2$ (with $\alpha > 0$), or simply A^2, in the linear encoders of Section 6.2. However, we observed few to no improvement for our graphs in some preliminary tests, consistently with our claim on the effectiveness of simple one-hop strategies. Such variants also tend to increase running times, as A^2 is usually denser than A.

Table 6.2: Link prediction on alternative datasets, with Linear GAE and VGAE models, and with their multi-layer GCN-based counterparts. We report details on hyperparameters for each model and dataset in Section 6.3.1. Cells are colored in blue when Linear GAE/VGAE models are reaching competitive results w.r.t. standard GCN-based models, i.e., results that are at least as good as GCN-based models ± 1 standard deviation. Note: as we aim to evaluate whether linear models are *as good* as others, and not necessarily *better*, we do not report **bold** numbers in this table contrary to tables from previous chapters.

Model	WebKD (n = 877, m = 1 608)		WebKD, with features (n = 877, m = 1 608, f = 1 703)		Hamsterster (n = 1 858, m = 12 534)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear GAE (ours)	77.20 ± 2.35	83.55 ± 1.81	84.15 ± 1.64	87.01 ± 1.48	93.07 ± 0.67	94.20 ± 0.58
2-layer GCN-based GAE	77.88 ± 2.57	84.12 ± 2.18	86.03 ± 3.97	87.97 ± 2.76	92.07 ± 0.63	93.01 ± 0.69
3-layer GCN-based GAE	78.20 ± 3.69	83.13 ± 2.58	81.39 ± 3.93	85.34 ± 2.92	91.40 ± 0.79	92.22 ± 0.85
Linear VGAE (ours)	83.50 ± 1.98	86.70 ± 1.53	85.57 ± 2.18	88.08 ± 1.76	91.08 ± 0.70	91.85 ± 0.64
2-layer GCN-based VGAE	82.31 ± 2.55	86.15 ± 2.03	87.87 ± 2.48	88.97 ± 2.17	91.62 ± 0.60	92.43 ± 0.64
3-layer GCN-based VGAE	82.17 ± 2.70	85.35 ± 2.25	89.69 ± 1.80	89.90 ± 1.58	91.06 ± 0.71	91.85 ± 0.77

Model	DBLP (n = 12 591, m = 49 743)		Cora-larger (n = 23 166, m = 91 500)		Arxiv-HepTh (n = 27 770, m = 352 807)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear GAE (ours)	90.11 ± 0.40	93.15 ± 0.28	94.64 ± 0.08	95.96 ± 0.10	98.34 ± 0.03	98.46 ± 0.03
2-layer GCN-based GAE	90.29 ± 0.39	93.01 ± 0.33	94.80 ± 0.08	95.72 ± 0.05	97.97 ± 0.09	98.12 ± 0.09
3-layer GCN-based GAE	89.91 ± 0.61	92.24 ± 0.67	94.51 ± 0.31	95.11 ± 0.28	94.35 ± 1.30	94.46 ± 1.31
Linear VGAE (ours)	90.62 ± 0.30	93.25 ± 0.22	95.20 ± 0.16	95.99 ± 0.12	98.35 ± 0.05	98.46 ± 0.05
2-layer GCN-based VGAE	90.40 ± 0.43	93.09 ± 0.35	94.60 ± 0.20	95.74 ± 0.13	97.75 ± 0.08	97.91 ± 0.06
3-layer GCN-based VGAE	89.92 ± 0.59	92.52 ± 0.48	94.48 ± 0.28	95.30 ± 0.22	94.57 ± 1.14	94.73 ± 1.12

Model	LiveMocha (n = 104 103, m = 2 193 083)		Flickr (n = 105 938, m = 2 316 948)		Patent (n = 2 745 762, m = 13 965 410)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear GAE (ours)	93.35 ± 0.10	94.83 ± 0.08	96.38 ± 0.05	97.27 ± 0.04	85.49 ± 0.09	87.17 ± 0.07
2-layer GCN-based GAE	92.79 ± 0.17	94.33 ± 0.13	96.34 ± 0.05	97.22 ± 0.04	82.86 ± 0.20	84.52 ± 0.24
3-layer GCN-based GAE	92.22 ± 0.73	93.67 ± 0.57	96.06 ± 0.08	97.01 ± 0.05	83.77 ± 0.41	84.73 ± 0.42
Linear VGAE (ours)	93.23 ± 0.06	94.61 ± 0.05	96.05 ± 0.08	97.12 ± 0.06	84.57 ± 0.27	85.46 ± 0.30
2-layer GCN-based VGAE	92.68 ± 0.21	94.23 ± 0.15	96.35 ± 0.07	97.20 ± 0.06	83.77 ± 0.28	83.37 ± 0.26
3-layer GCN-based VGAE	92.71 ± 0.37	94.01 ± 0.26	96.39 ± 0.13	97.16 ± 0.08	85.30 ± 0.51	86.14 ± 0.49

Model	Blogs (n = 1 224, m = 19 025)		Amazon (n = 334 863, m = 925 872)		Google (n = 875 713, m = 5 105 039)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear GAE (ours)	91.71 ± 0.39	92.53 ± 0.44	90.70 ± 0.09	93.46 ± 0.08	95.37 ± 0.05	96.93 ± 0.05
2-layer GCN-based GAE	91.57 ± 0.34	92.51 ± 0.29	90.15 ± 0.15	92.33 ± 0.14	95.06 ± 0.08	96.40 ± 0.07
3-layer GCN-based GAE	91.74 ± 0.37	92.62 ± 0.31	88.54 ± 0.37	90.47 ± 0.38	93.68 ± 0.15	94.99 ± 0.14
Linear VGAE (ours)	91.34 ± 0.24	92.10 ± 0.24	84.53 ± 0.08	87.79 ± 0.06	91.13 ± 0.14	93.79 ± 0.10
2-layer GCN-based VGAE	91.85 ± 0.22	92.60 ± 0.25	90.14 ± 0.22	92.33 ± 0.23	95.04 ± 0.09	96.38 ± 0.07
3-layer GCN-based VGAE	91.83 ± 0.48	92.65 ± 0.35	89.44 ± 0.25	91.23 ± 0.23	93.79 ± 0.22	95.12 ± 0.21

Model	Stanford (n = 281 903, m = 2 312 497)		Proteins (n = 6 327, m = 147 547)		Google-Medium (n = 15 763, m = 171 206)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear GAE (ours)	97.73 ± 0.10	98.37 ± 0.10	94.09 ± 0.23	96.01 ± 0.16	96.02 ± 0.14	97.09 ± 0.08
2-layer GCN-based GAE	97.05 ± 0.63	97.56 ± 0.55	94.55 ± 0.20	96.39 ± 0.16	96.66 ± 0.24	97.45 ± 0.25
3-layer GCN-based GAE	92.19 ± 1.49	92.58 ± 1.50	94.30 ± 0.19	96.08 ± 0.15	95.10 ± 0.27	95.94 ± 0.20
Linear VGAE (ours)	94.96 ± 0.25	96.64 ± 0.15	93.99 ± 0.10	95.94 ± 0.16	91.11 ± 0.31	92.91 ± 0.18
2-layer GCN-based VGAE	97.60 ± 0.11	98.02 ± 0.10	94.57 ± 0.18	96.18 ± 0.33	96.11 ± 0.59	96.84 ± 0.51
3-layer GCN-based VGAE	97.53 ± 0.13	98.01 ± 0.10	94.27 ± 0.25	95.71 ± 0.28	95.10 ± 0.54	96.00 ± 0.44

6.3. Experimental Analysis

Table 6.3: Community detection on graphs with communities, with Linear GAE and VGAE models, and with their multi-layer GCN-based counterparts. We report details on hyperparameters for each model and dataset in Section 6.3.1. Cells are colored in blue when Linear GAE/VGAE models are reaching competitive results w.r.t. standard GCN-based models, i.e., results that are at least as good as GCN-based models ± 1 standard deviation. Note: as we aim to evaluate whether linear models are *as good* as others, and not necessarily *better*, we do not report **bold** numbers in this table contrary to tables from previous chapters.

Model	Cora (n = 2 708, m = 5 429)	Cora with features (n = 2 708, m = 5 429, f = 1 433)	Citeseer (n = 3 327, m = 4 732)	Citeseer with features (n = 3 327, m = 4 732, f = 3 703)
	AMI (in %)	AMI (in %)	AMI (in %)	AMI (in %)
Linear GAE (ours)	26.31 ± 2.85	47.02 ± 2.09	8.56 ± 1.28	20.23 ± 1.36
2-layer GCN-based GAE	30.88 ± 2.56	43.04 ± 3.28	9.46 ± 1.06	19.38 ± 3.15
3-layer GCN-based GAE	33.06 ± 3.10	44.12 ± 2.48	10.69 ± 1.98	19.71 ± 2.55
Linear VGAE (ours)	34.35 ± 1.42	48.12 ± 1.96	12.67 ± 1.27	20.71 ± 1.95
2-layer GCN-based VGAE	26.66 ± 3.94	44.84 ± 2.63	9.85 ± 1.24	20.17 ± 3.07
3-layer GCN-based VGAE	28.43 ± 2.83	44.29 ± 2.54	10.64 ± 1.47	19.94 ± 2.50

Model	Pubmed (n = 19 717, m = 44 338)	Pubmed with features (n = 19 717, m = 44 338, f = 500)	Cora-larger (n = 23 166, m = 91 500)	Blogs (n = 1 224, m = 19 025)
	AMI (in %)	AMI (in %)	AMI (in %)	AMI (in %)
Linear GAE (ours)	10.76 ± 3.70	26.12 ± 1.94	40.34 ± 0.51	46.84 ± 1.79
2-layer GCN-based GAE	16.41 ± 3.15	23.08 ± 3.35	39.75 ± 0.79	72.58 ± 4.54
3-layer GCN-based GAE	23.11 ± 2.58	25.94 ± 3.09	35.67 ± 1.76	72.72 ± 1.80
Linear VGAE (ours)	25.14 ± 2.83	29.74 ± 0.64	43.32 ± 0.52	49.70 ± 1.08
2-layer GCN-based VGAE	20.52 ± 2.97	25.43 ± 1.47	38.34 ± 0.64	73.12 ± 0.83
3-layer GCN-based VGAE	21.32 ± 3.70	24.91 ± 3.09	37.30 ± 1.07	70.56 ± 5.43

Experiments on more complex decoders   So far, we compared different encoders but the standard inner product decoder was fixed. As a robustness check, in the next Table 6.4, we report complementary link prediction experiments, on variants of GAE and VGAE models with two more complex decoders:

- the "Graphite" GAE and VGAE models from Grover et al. [124] for iterative generative modeling of graphs, and already mentioned in our experiments from Chapter 3. Graphite models still process undirected graphs only. They rely on an iterative graph refinement strategy inspired by low-rank approximations for decoding, instead of a simple inner product with a sigmoid activation as in Section 6.2;

- our own Gravity-Inspired GAE and VGAE models from Chapter 5 that, contrary to the other models, incorporate an asymmetric decoder, i.e., $\hat{A}_{ij} \neq \hat{A}_{ji}$. As explained in Chapter 5, this model is designed for directed link prediction. Therefore, contrary to previous experiments, we do not ignore edges directions when initial graphs were directed. The models from Table 4 all process the out-degree normalization of A a.k.a. $\tilde{A}_{\text{out}}$ instead of the symmetric normalization $\tilde{A}$.

Overall, we draw similar conclusions w.r.t. Tables 6.1 and 6.2, consolidating our findings. For

Table 6.4: Link prediction with Linear GAE and VGAE models incorporating the "Graphite" [124] and Gravity-Inspired [311] decoders, and with their multi-layer GCN-based counterparts. We report details on hyperparameters for each model and dataset in Section 6.3.1. Cells are colored in blue when Linear GAE/VGAE models are reaching competitive results w.r.t. standard GCN-based models, i.e., results that are at least as good as GCN-based models $\pm$ 1 standard deviation. Note: as we aim to evaluate whether linear models are *as good* as others, and not necessarily *better*, we do not report **bold** numbers in this table contrary to tables from previous chapters.

Model	Cora (n = 2 708, m = 5 429)		Citeseer (n = 3 327, m = 4 732)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear Graphite GAE (ours)	83.42 ± 1.76	87.32 ± 1.53	77.56 ± 1.41	82.88 ± 1.15
2-layer Graphite GAE	81.20 ± 2.21	85.11 ± 1.91	73.80 ± 2.24	79.32 ± 1.83
3-layer Graphite GAE	79.06 ± 1.70	81.79 ± 1.62	72.24 ± 2.29	76.60 ± 1.95
Linear Graphite VGAE (ours)	83.68 ± 1.42	87.57 ± 1.16	78.90 ± 1.08	83.51 ± 0.89
2-layer Graphite VGAE	84.89 ± 1.48	88.10 ± 1.22	77.92 ± 1.57	82.56 ± 1.31
3-layer Graphite VGAE	85.33 ± 1.19	87.98 ± 1.09	77.46 ± 2.34	81.95 ± 1.71
Linear Gravity-Inspired GAE (ours)	90.71 ± 0.95	92.95 ± 0.88	80.52 ± 1.37	86.29 ± 1.03
2-layer Gravity-Inspired GAE	87.79 ± 1.07	90.78 ± 0.82	78.36 ± 1.55	84.75 ± 1.10
3-layer Gravity-Inspired GAE	87.76 ± 1.32	90.15 ± 1.45	78.32 ± 1.92	84.88 ± 1.36
Linear Gravity-Inspired VGAE (ours)	91.29 ± 0.70	93.01 ± 0.57	86.65 ± 0.95	89.49 ± 0.69
2-layer Gravity-Inspired VGAE	91.92 ± 0.75	92.46 ± 0.64	87.67 ± 1.07	89.79 ± 1.01
3-layer Gravity-Inspired VGAE	90.80 ± 1.28	92.01 ± 1.19	85.28 ± 1.33	87.54 ± 1.21

Model	Pubmed (n = 19 717, m = 44 338)		Google-Medium (n = 15 763, m = 171 206)	
	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Linear Graphite GAE (ours)	80.28 ± 0.86	85.81 ± 0.67	94.30 ± 0.22	95.09 ± 0.16
2-layer Graphite GAE	79.98 ± 0.66	85.33 ± 0.41	95.54 ± 0.42	95.99 ± 0.39
3-layer Graphite GAE	79.96 ± 1.40	84.88 ± 0.89	93.99 ± 0.54	94.74 ± 0.49
Linear Graphite VGAE (ours)	79.59 ± 0.33	86.17 ± 0.31	92.71 ± 0.38	94.41 ± 0.25
2-layer Graphite VGAE	82.74 ± 0.30	87.19 ± 0.36	96.49 ± 0.22	96.91 ± 0.17
3-layer Graphite VGAE	84.56 ± 0.42	88.01 ± 0.39	96.32 ± 0.24	96.62 ± 0.20
Linear Gravity-Inspired GAE (ours)	76.78 ± 0.38	84.50 ± 0.32	97.46 ± 0.07	98.30 ± 0.04
2-layer Gravity-Inspired GAE	75.84 ± 0.42	83.03 ± 0.22	97.77 ± 0.10	98.43 ± 0.10
3-layer Gravity-Inspired GAE	74.61 ± 0.30	81.68 ± 0.26	97.58 ± 0.12	98.28 ± 0.11
Linear Gravity-Inspired VGAE (ours)	79.68 ± 0.36	85.00 ± 0.21	97.32 ± 0.06	98.26 ± 0.05
2-layer Gravity-Inspired VGAE	77.30 ± 0.81	82.64 ± 0.27	97.84 ± 0.25	98.18 ± 0.14
3-layer Gravity-Inspired VGAE	76.52 ± 0.61	80.73 ± 0.63	97.32 ± 0.23	97.81 ± 0.20

brevity, in Tables 6.4 we only report results for the Cora, Citeseer, and Pubmed graphs, where linear models are competitive, and for the Google-Medium graph, where GCN-based GAE and VGAE models slightly outperform their linear counterparts. We stress out that scores from Graphite and Gravity-Inspired GAEs/VGAEs are *not* directly comparable, as the former ignores the directionality of edges while the latter processes directed graphs, i.e., the learning task is a *directed* link prediction problem for Gravity-Inspired GAEs/VGAEs.

When (not) to use multi-layer GCN encoders? Linear GAE and VGAE models reached strong empirical results on all graphs under consideration in this work despite their various sizes and characteristics, and rival or outperform GCN-based GAE and VGAE models in a

majority of experiments. These models are also simpler and more interpretable, each element of z_i being interpreted as a weighted average from node i's direct neighborhood. We recall that, in Chapter 3, we also showed that multi-layer GCNs are themselves often competitive (or very close) to more complex GNN encoders. Therefore, we recommend the systematic use of Linear GAE and VGAE models as a first baseline, before diving into more complex encoding schemes whose actual benefit might be unclear.

From our experiments, we also conjecture that multi-layer GCN encoders *can* bring an empirical advantage in the presence of graphs with *intrinsic non-trivial high-order interactions*. Notable examples of such graphs include the Amazon co-purchase graph (+5.61 AUC points for 2-layer GCN-based VGAE) and web graphs such as Blogs, Google, and Stanford, in which two-hop hyperlinks connections of pages usually include relevant information on the global network structure. On such graphs, capturing this information tends to improve results, especially 1) for the probabilistic VGAE framework, and 2) when evaluating embeddings via the community detection task (e.g., 20+ AMI points on Blogs for 2-layer GCN-based GAE and VGAE) which is, by design, a more *global* learning task than the quite *local* link prediction problem.

On the contrary, in citation graphs such as Cora, Citeseer, and Pubmed, the relevance of two-hop links is more limited. Indeed, if a reference A in an article B cited by some authors is relevant to their work, authors will likely also cite this reference A, thus creating a one-hop link. Lastly, while the impact of the graph *size* is unclear in our experiments (linear models achieve strong results even on large graphs, such as Patent), we note that graphs where multi-layer GCN encoders tend to outperform linear models are all relatively *dense*.

As a consequence, we conjecture that denser graphs with intrinsic high-order interactions (e.g., the aforestated web graphs) should be better suited than the sparse Cora, Citeseer, and Pubmed citation networks to evaluate and compare complex GAE and VGAE models. Our recommendation to the scientific community is not to completely avoid these three graphs, but rather to stop using them *exclusively* in experiments.

6.4 Conclusion

In this chapter, we proposed to simplify GAE and VGAE models. While most existing variants of these methods rely on multi-layer GCNs encoders to learn embedding vectors, we emphasized that these encoders are unnecessarily complex for many applications. We suggested replacing them with simpler linear models w.r.t. the (one-hop) normalized adjacency matrix of the graph, involving fewer operations, fewer parameters, and no activation function. We provided an in-depth experimental evaluation of such an approach, showing that our simplified models consistently reach competitive or very close performances w.r.t. multi-layer GCN-based GAE and VGAE models on link prediction and community detection, and on numerous real-world

graphs. We also aimed to identify the settings where simple one-hop linear encoders appear as an effective alternative to multi-layer GCNs.

Based on these experiments, and on previous results from Chapter 3 showing that multi-layer GCNs can themselves be empirically close to more complex GNN encoders, we recommended the systematic use of Linear GAE and VGAE models as a first baseline, before diving into more complex encoding schemes whose actual benefit might be unclear. Since the publication of this research in late 2019 (for the NeurIPS workshop paper [309]) and then in mid-2020 (for the ECML-PKDD conference paper [310]), several recent articles including [9, 56, 73, 142, 332, 385] considered Linear GAE and VGAE models in their own experiments, either as a baseline or as a component of a larger model. These experiments confirmed that one-hop linear models often reach comparable results w.r.t. multi-layer GCNs.

Besides research papers, we emphasize that such a result is valuable in the context of an industrial PhD thesis. Indeed, in the industry, e.g., for music streaming services, simple models are often preferred in production environments, as they are easier to understand, to deploy, and to debug. In Chapters 9 and 10, we will provide a broader overview of how Deezer internally leverages similarity graphs and graph ontologies for music recommendation, involving Linear GAE and VGAE models. Moreover, to encourage the future usage of our method, we also recently implemented a PyTorch version of Linear GAE and VGAE, now available in the popular PyTorch-Geometric[7] library [96], in addition to our initial TensorFlow implementation[8].

Last, but not least, in this chapter we also questioned the relevance of repeatedly using the same sparse medium-size citation networks (Cora, Citeseer, Pubmed) to evaluate and compare complex GAE and VGAE models. In Section 6.3.2, we recommended to stop using these three graphs *exclusively* in experiments. While we admit that finding "challenging" alternative datasets might be difficult, we also praise the recent efforts from the graph learning community to release large, various, and realistic benchmark datasets. This includes the Open Graph Benchmark [151] initiative, aiming to provide challenging datasets for various graph-based machine learning problems, together with standardized dataset splits and evaluators to properly compare the performances of different models.

---

[7]https://github.com/pyg-team/pytorch_geometric
[8]https://github.com/deezer/linear_graph_autoencoders

7

Improving Community Detection
with Graph Autoencoders

This chapter presents research conducted with Johannes F. Lutzeyer, George Dasoulas, Romain Hennequin, and Michalis Vazirgiannis, and currently under review for publication in Elsevier's Neural Networks journal in 2022 [314].

7.1 Introduction

GAE and VGAE models were originally mainly designed for link prediction, at least in their modern formulation [187]. The overall effectiveness of GAE and VGAE models and of their extensions on this specific task has been widely experimentally confirmed over the past few years [124, 135, 279, 329, 359, 361], including in our own experiments from Chapters 3 to 6.

On the other hand, several concurring studies [55, 56, 307] have simultaneously pointed out the limitations of these models on community detection. They emphasized that standard GAEs and VGAEs are often outperformed by simpler clustering alternatives, such as the popular Louvain method [31]. We ourselves observed and discussed this limitation in some of our community detection experiments from Chapters 3, 4 and 6.

While some recent studies worked on this issue (see Section 7.2 for an overview), their solutions strongly relied on *clustering-oriented probabilistic* priors that only fit the VGAE setting and can not be directly transposed to deterministic GAEs. They also benefited greatly from the presence of *node features* complementing the graph structure, but provided only little to no empirical gain on featureless graphs that are nonetheless ubiquitous. Thirdly, they did not explicitly try to

119

preserve the good performances of GAE and VGAE models on link prediction. In practice, as we will argue in this chapter, learning node embedding spaces that jointly enable good link prediction and community detection performances is often desirable, both for real-world applications and in pursuit of learning accurate and general representations of a graph structure.

In summary, the question of how to improve community detection with GAE and VGAE models remains incompletely addressed, especially in the absence of node features, and it is still unclear to which extent one can improve community detection with these models without simultaneously deteriorating link prediction. In this chapter, we propose to tackle these important problems by investigating the following two research questions:

- **Question 1:** can we improve community detection for *both* the GAE *and* VGAE settings? And does this improvement persist for *featureless* graphs?

- **Question 2:** do improvements on the community detection task necessarily incur a loss in the link prediction performance or can they be *jointly* addressed with high accuracy?

In this chapter we propose several novel contributions to both the GAE and VGAE frameworks, which allow us to answer both of these research questions positively. More precisely, our contributions are listed as follows. We first diagnose the reasons why GAE and VGAE models tend to perform well on link prediction but to underperform on community detection. Then, based on insights from this diagnosis, we improve GAE and VGAE models for graph-based community detection while preserving their ability to identify missing edges. Our strategy leverages concepts inspired by *modularity-based* clustering [31, 38, 333]. Specifically, we first present and theoretically study a novel *community-preserving message passing scheme*, doping our GAE and VGAE encoders by considering both the initial graph structure and modularity-based prior communities when computing embedding spaces. We also introduce revised training and optimization strategies w.r.t. current practices in the scientific literature, including the introduction of modularity-inspired losses complementing the existing reconstruction losses with the aim of jointly ensuring good performances on link prediction and community detection. Backed by in-depth experiments on several real-world graphs, we demonstrate the empirical effectiveness of our approach at addressing 1) pure community detection problems, and 2) joint community detection and link prediction problems. We publicly release our source code on GitHub.

The remainder of this chapter is organized as follows. In Section 7.2, we point out the limitations of current GAE and VGAE models on community detection. In Section 7.3, we diagnose the reasons explaining these limits. We subsequently introduce and theoretically study our proposed solution, referred to as *Modularity-Aware GAE and VGAE*, to overcome these limitations. We report and discuss our experimental evaluation in Section 7.4, and we conclude in Section 7.5. We provide additional proofs in Section 7.6, placed out of the "main" chapter for the sake of brevity and readability.

7.2 On the Limitations of GAE/VGAE-Based Community Detection

While link prediction remains the most prominent evaluation task for GAE and VGAE models, they have also shown promising results on (semi-supervised) node classification [135, 359], canonical correlation analysis [174] and, in the case of VGAE, graph generation especially in the context of molecular graph data [233, 243, 336]. However, their performances are less impressive on *community detection* [55, 56], on which we focus in this section.

7.2.1 Community Detection with GAE and VGAE

Throughout this chapter, we consider the community detection problem formulated in Section 2.1.2 from Chapter 2, which we briefly remind in this paragraph. Among the fundamental problems in graph-based machine learning, community detection consists in identifying $K < n$ clusters a.k.a. *communities* of nodes that, in some sense, are more similar to each other than to the other nodes [55, 248]. More formally, we aim to obtain a partition of $\mathcal{V}$ into K sets:

$$C_1 \subseteq \mathcal{V}, \ldots, C_K \subseteq \mathcal{V}, \tag{7.1}$$

with cardinality $|C_k| = n_k \leq n$ for $k \in \{1, \ldots, K\}$. The quality of such a partition is usually assessed through some predefined similarity metrics, e.g., unsupervised density-based metrics calculated from the intra- and inter-cluster edge density [248], or scores such as the *Adjusted Mutual Information* (AMI) [366] and *Adjusted Rand Index* (ARI) [159] scores, that compare the partition to some ground truth node labels hidden during training.

In the presence of node embedding representations, community detection boils down to the more standard problem of clustering n vectors in a d-dimensional Euclidean space into K groups [244]. With this goal in mind, several studies specifically tried to perform community detection with GAE and VGAE by:

- learning an embedding vector z_i for each $i \in \mathcal{V}$, as described in the previous chapters;

- clustering the resulting vectors z_i into K groups, through one of the numerous clustering methods for Euclidean data, such as the popular k-means algorithm [244].

However, concurring experimental evaluations [55, 56, 307, 308] recently pointed out the limitations of such an approach. They emphasized its lower performance w.r.t. simpler community detection alternatives, that sometimes even directly operate on the graph structure without considering node features, such as the popular Louvain method [31].

For instance, Choong et al. [56] show that, on the (featureless) Cora citation network [323],

a VGAE+k-means strategy reaches a mean normalized mutual information score of 23.84%, way below the Louvain method (43.36%). In the previous Chapter 4 on FastGAE, we showed that, on the same graph, a GAE+k-means also reaches an underwhelming 30.88% mean AMI score. We reached comparable conclusions from experiments on several other graphs, such as the featureless versions [323] of Citeseer (9.85% AMI for VGAE+k-means vs 16.39% for Louvain, in our Chapter 4) and Pubmed (20.41% for VGAE+k-means in the study of Choong et al. [56], which is comparable to Louvain, but significantly below the 29.46% obtained by running a k-means on node embedding vectors learned via DeepWalk [287]).

7.2.2 Community Detection with Extensions of GAE and VGAE

Several studies have worked on the issue of the underwhelming performance of GAE and VGAE models in the community detection task [55, 56, 217, 262]. Choong et al. [55] introduced VGAECD, a *VGAE for Community Detection (CD)* model that replaces Gaussian priors by learnable *Gaussian mixtures*. Such a choice permits recovering communities from node embedding spaces without relying on an additional k-means step. In a subsequent study [56], the same authors proposed VGAECD-OPT, an improved version of VGAECD. Specifically, VGAECD-OPT replaces GCN encoders with the simpler linear models we proposed in Chapter 6. It also adopts a different optimization procedure based on neural expectation-maximization [122], which guarantees that communities do not collapse during training [56] and experimentally leads to better performances.

More recently, Li et al. [217] introduced *Dirichlet Variational Graph Autoencoder* (DVGAE), another extension of VGAE which uses Dirichlet distributions as priors on latent vectors, acting as indicators of community memberships. The *Marginalized GAE* (MGAE) model from Wang et al. [370] is also evaluated on community detection. However, the MGAE model does not explicitly leverage embedding representations for this task; instead the *spectral clustering* [368] is applied to the decoded graphs. Last, while community detection was not the main focus in [156, 279, 282, 329, 361], these works all proposed various encoding-decoding methods that, to different extents, seem to outperform standard GAE and VGAE models on the community detection task, in the reported evaluations. They consider alternatives encoders or training choices, which we further discuss and investigate in Section 7.3.

7.2.3 Current Limitations

While the models discussed in Section 7.2.2 will constitute relevant baselines in our experiments (see Section 7.4), they still suffer from several fundamental limitations that motivate our work:

- firstly, all extensions explicitly designed for community detection [55, 56, 217] *rely on clustering-oriented probabilistic priors*. They are only applicable in the VGAE paradigm,

7.3. On the Limitations of GAE/VGAE-Based Community Detection

Table 7.1: Normalized mutual information scores (in %) for community detection on the Cora and Pubmed citation networks, *with* and *without* node features. Results are directly taken from the evaluation of Choong et al. [56]. **Bold** numbers correspond to the best score for each graph. This table emphasizes that, in the absence of node features, VGAECD and VGAECD-OPT bring little (to no) advantage w.r.t. standard VGAE, and remain below the Deepwalk and/or Louvain baselines. Scores of VGAECD and VGAECD-OPT significantly increase when adding features to the graph. Recall: in this table, Deepwalk and Louvain both ignore node features.

Dataset	VGAE	VGAECD	VGAECD-OPT	DeepWalk	Louvain
Cora *without* node features	23.84	28.22	37.35	37.96	**43.36**
Pubmed *without* node features	20.41	16.42	25.05	**29.46**	19.83
Cora *with* node features	31.73	50.72	**54.37**	37.96	43.36
Pubmed *with* node features	19.81	32.53	**35.52**	29.46	19.83

and cannot be directly transposed to the deterministic GAE setting. The question of how to design clustering-efficient GAE models thus remains widely open;

- more importantly, a closer look at these models reveals that their *empirical gains often mostly stem from the addition of node features* to the graph. As an illustration, Table 7.1 displays the reported performances of VGAECD and VGAECD-OPT on the *featureless* versions of two graphs [56]. We observe that they offer little to no empirical advantage when features are absent. This draws into question the extent to which these models are able to capture communities from (only) graph data.

The important role of node features has subsequently been confirmed (e.g., Park et al. [282] show that, on the Pubmed dataset, a straightforward k-means on the node features alone reaches comparable AMI scores w.r.t. VGAE and MGAE). On the other hand, most of the aforementioned other studies with empirical improvements [156, 262, 279, 282, 329, 361] only reported results on graphs equipped with node features. This motivates the need for a proper investigation of the *featureless* case where models cannot rely on the additional node feature information;

- lastly, previous studies centered around community detection [55, 56, 217, 370] *did not explicitly try to preserve the good performances of GAE and VGAE models on link prediction.* Overall, most of the aforementioned existing works learn node representations specific to a particular learning task. Therefore, it is still unclear whether one can improve community detection with these models without simultaneously deteriorating link prediction. With the general aim of learning high-quality node embedding representations, one can wonder to which extent these models can learn representations that are *jointly* useful for several tasks. Besides providing a more accurate summary of the graph structure under consideration, such representations could also lead to significant resource savings in real-world applications, as we will illustrate in Section 7.4.

In conclusion to this section, the question of how to effectively improve community detection with GAE and VGAE remains incompletely addressed.
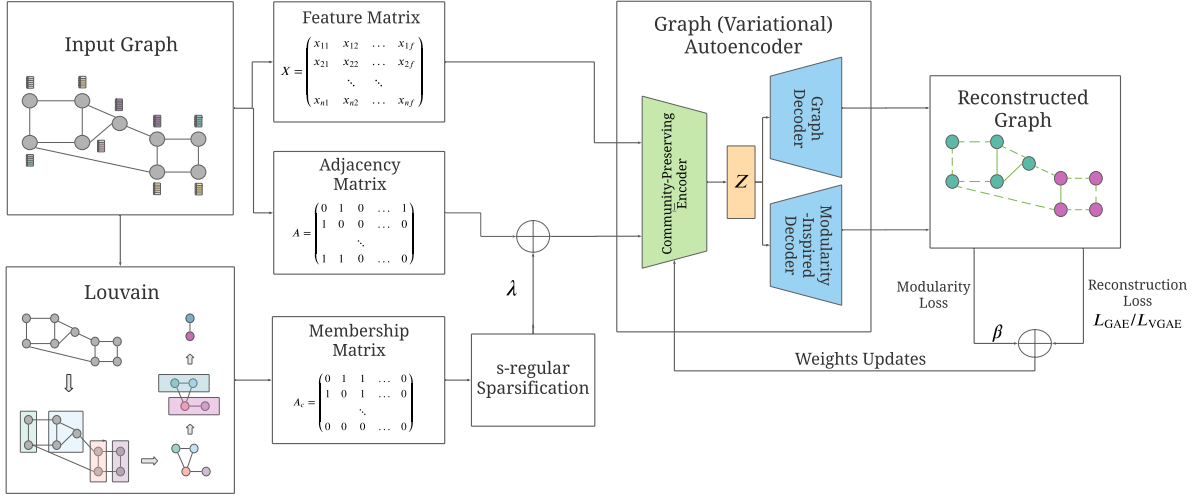
Figure 7.1: Overview of our proposed Modularity-Aware GAE and VGAE. Firstly, input graph data A and X are combined with the s-regular sparsified prior community membership matrix A_s, derived through iterative modularity maximization via the Louvain algorithm, as described in Section 7.3.2. Then, they are processed by our revised community-preserving (linear or multi-layer GCN) encoders, encoding each node $i \in \mathcal{V}$ as an embedding vector z_i of dimension $d \ll n$. Neural weights of encoders are optimized through a procedure combining reconstruction and modularity-inspired losses, and described in Section 7.3.3. Furthermore, other hyperparameters from this model are tuned via the method described in Section 7.3.3 and designed for joint link prediction and community detection applications.

7.3 Modularity-Aware Graph Autoencoders

We now introduce our approach, referred to as *Modularity-Aware GAE and VGAE* in the following, to address the aforementioned limitations. In Section 7.3.1, we first provide a general overview of the key components of our solution. They transpose concepts from *modularity-based* clustering [31, 38, 333] to GAEs and VGAEs, and are illustrated in Figure 7.1. We subsequently detail these solution components in Sections 7.3.2 and 7.3.3.

7.3.1 Diagnostic and Overview of our Framework

Based on our literature review, we diagnose three main reasons that can explain why previous GAE and VGAE models still suffer from the limitations described in Section 7.2.3:

- firstly, they leveraged *encoders* that were not specifically designed to preserve the intrinsic communities from the graph structure under consideration in the node embedding space. This includes the popular GCN, as well as refined neural models that rather aimed to preserve clusters from node features (but not necessarily the actual communities from the graph under consideration). In Modularity-Aware GAE and VGAE, we overcome this issue by incorporating a novel encoding scheme for graph community-preserving representation learning. It consists in an improvement of the GCN *message passing operator*, boosting

both GAE and VGAE models by simultaneously considering the initial graph structure and *modularity-based node communities* when computing node embedding spaces. We present and theoretically study this encoder in Section 7.3.2;

- besides the encoder's architecture, previous models were often *optimized* in a fashion that, by design, favors link prediction over community detection. In particular, the standard cross-entropy and ELBO losses, used to learn neural weight matrices, directly involve the reconstruction of *node pairs* from the embedding space[1]. However, as we will detail, a good reconstruction of *local* pairwise connections does not necessarily imply a good reconstruction of the *global* community structure. In Modularity-Aware GAE and VGAE, we instead optimize an alternative loss inspired by the *modularity* [31]. Such a loss acts as a simple yet effective global regularization over pairwise reconstruction losses, with desirable properties for joint link prediction and community detection. It will empirically enable a refined optimization of the weight matrices from our encoders. We present this aspect in detail in Section 7.3.3;

- lastly, in addition to these weight matrices, GAE and VGAE models involve several other hyperparameters, ranging from the number of training iterations to the learning rate [187]. While they also impact the model's performance, the selection procedure for such hyperparameters was sometimes omitted in previous works [56] or based on link prediction validation sets, as in experiments from our previous chapters (while, intuitively, the best hyperparameters for community detection might differ from those for link prediction). For the Modularity-Aware GAE and VGAE, we adopt an alternative graph-based model selection procedure. It completes the previous two aspects, by providing the most relevant GAE/VGAE hyperparameters for joint link prediction and community selection. We present and discuss this procedure in Section 7.3.3 as well.

7.3.2 Community-Aware Encoders for GAE and VGAE

Following this diagnosis and overview, we now detail the first of the three bullet points from Section 7.3.1, i.e., our proposed revised *encoding* strategy. We recall that our proposed solution aims to encode nodes as embedding vectors z_i *more suitable for community detection*. Essentially, intrinsic communities in the graph under consideration should be easily retrievable from these representations, e.g., from their L_2 distances via a straightforward k-means clustering. These vectors should also simultaneously remain relevant for *link prediction*, i.e., as for existing GAE

---

[1]In the case of the probabilistic VGAE paradigm, another limitation of the ELBO loss - and of the underlying generative decoder - lies in the use of standard Gaussian priors. Replacing these priors by for example *Gaussian mixtures* as in [55, 56], appears to be an intuitive approach for community-based learning. However, as this approach 1) does not extend to deterministic GAE, and 2) has been extensively studied in [55, 56], we do not further develop it in this work. We will nonetheless compare to [55, 56] in experiments, and will argue in Section 7.3.3 that Gaussian mixtures could straightforwardly be incorporated in our proposed Modularity-Aware VGAE.

and VGAE models, the likelihood of a missing edge between two nodes should also be inferred from the learned representations z_i.

Preliminary remark on the notation   In the following, for simplicity, we mainly consider *undirected* graphs and continue using the symmetric inner product $\hat{A}_{ij} = \sigma(z_i^T z_j)$ as the probability of an edge between nodes i and j. Also, in this chapter, we will use $\mathcal{F}(\cdot)$ to denote the symmetrically normalized adjacency matrix of a graph, i.e.:

$$\mathcal{F}(A) = (D + I_n)^{-\frac{1}{2}}(A + I_n)(D + I_n)^{-\frac{1}{2}} \tag{7.2}$$

instead of $\tilde{A}$ as in Definition 2.5 and other chapters of the thesis. This slight discrepancy is due to the fact that, in this chapter, the symmetric normalization will be simultaneously applied to several matrices and to sums of matrices, which makes the *"tilde"* notation too heavy.

7.3.2.1   Revising the Message Passing Operator

Existing graph encoders usually involve normalized versions of the adjacency matrix A, or some generalized *message passing operator* matrix that also captures each node's direct connections in the graph under consideration [66]. For instance, in the popular multi-layer GCN from Definition 2.11, the symmetric normalization $\tilde{A}$ a.k.a. $\mathcal{F}(A)$ from Definition 2.5 is used such that at each layer l a vectorial representation for each node is computed by averaging the representations from layer $l - 1$ of its direct neighbors and of itself. In this chapter, we adopt an alternative strategy that consists in averaging, at each layer:

- representations from the direct neighbors of each node, as above;

- but also representations from other *unconnected nodes* that, according to some prior available knowledge and criteria, belong to the same graph community.

More precisely, let us assume that we have, at our disposal, a pre-processing *graph mining* technique that, based on the graph structure and on some fixed criteria, learns an initial *prior partition of the node set* $\mathcal{V}$ into K sets $C_1, \ldots, C_K$, with $|C_k| = n_k$ for $k \in \{1, \ldots, K\}$. Here, K acts as a hyperparameter, that can differ from the actual number of communities eventually used for the community detection downstream evaluation task (i.e., the number of clusters in the k-means operated on the final vectors z_i). A concrete example of such a technique will be provided in the upcoming Subsection 7.3.2.3. We simply assume its availability at this stage.

We propose to leverage such an initial partition as a *prior node clustering signal* from which the GAE/VGAE encoder should benefit, but also can deviate during training, when learning the embedding space. Specifically, we propose to replace the standard input adjacency matrix

A by:

$$A + \lambda A_c, \tag{7.3}$$

where $\lambda \geq 0$ is a scalar hyperparameter, and where A_c is the community membership matrix defined as follows:

Definition 7.1. Let us consider a partition of the node set $\mathcal{V}$ into K sets $C_1, \ldots, C_K$. The corresponding *community membership matrix* is defined as:

$$A_c = MM^T - I_n, \tag{7.4}$$

with $M \in \{0, 1\}^{n \times K}$ denoting the $n \times K$ matrix where elements $M_{ik} = 1$ if and only if $i \in C_k$ according to the prior clustering.

We interpret A_c as the adjacency matrix of an alternative graph in which each cluster of our prior partition is represented by a fully connected graph, without self-loops. Since nodes are only allocated to one cluster, there exists a node ordering such that the matrix A_c is block-diagonal. In essence, $A + \lambda A_c$ aims to capture refined node similarities, by simultaneously considering some *local* information from direct neighborhoods, and some *global* information from prior node communities. The hyperparameter λ helps to balance these two aspects. In particular, setting $\lambda = 0$ results in the standard adjacency matrix.

7.3.2.2   From Message Passing Operators to Encoding Schemes

At first glance, $A + \lambda A_c$ could straightforwardly be incorporated as a refined message passing operator in popular GAE and VGAE encoders. For instance, one could consider its direct incorporation in:

- variants of *2-layer GCN encoders*, as initially proposed by Kipf and Welling [187], since this neural architecture remains the most popular GAE/VGAE encoder in the literature (see Chapters 2 and 6 for a review). Specifically, one could consider:

  - a version incorporating $A + \lambda A_c$ in both layers. Then, in the case of a GAE[2] as described in Section 2.4.1, we would have: $Z = \text{GCN}^{(1)}(A + \lambda A_c, X) = \mathcal{F}(A + \lambda A_c)\text{ReLU}(\mathcal{F}(A + \lambda A_c)XW^{(0)})W^{(1)}$;

  - a version incorporating the prior communities only on the first layer, i.e., $Z = \text{GCN}^{(2)}(A + \lambda A_c, X) = \mathcal{F}(A)\text{ReLU}(\mathcal{F}(A + \lambda A_c)XW^{(0)})W^{(1)}$;

---

[2]For clarity of exposition we discuss the deterministic GAE framework (Section 2.4.1). However, the changes are equally applicable to the VGAE framework (Section 2.4.2), for which Z has to be replaced by μ and $\log \sigma$ as in Equation (2.35).

- or, a variant of the *linear encoder* proposed in Chapter 6, as this simplified one-hop GCN without activation reached competitive performances w.r.t. multi-layer GCNs for GAE/VGAE-based community detection in this previous Chapter 6 as well as in the study of Choong et al. [56]. In this case: $Z = \text{Linear}(A + \lambda A_c, X) = \mathcal{F}(A + \lambda A_c)XW^{(0)}$.

However, the computational cost of evaluating each layer of a GCN or a linear encoder depends linearly on the number of edges $|\mathcal{E}| = m$ in the message passing operator [188]. As the graph represented by $A + \lambda A_c$ contains at least $\sum_{k=1}^{K} n_k^2$ edges, such a direct incorporation of $A + \lambda A_c$ in encoders could incur a large computational expense.

To alleviate this cost, we will instead consider a *s-regular sparsification of A_c*, denoted by A_s in the following. In A_s, each node $i \in C_k$ is only connected to $s < n_k$ randomly selected nodes in C_k (instead of all other nodes in C_k). Therefore, the $A + \lambda A_s$ message passing operator still contains some of the prior clustering information without necessarily incurring the cost implied by the use of A_c. In particular, selecting $s \approx \frac{m}{n}$ ensures that $A + \lambda A_s$ has $O(2m)$ non-null elements, preserving the linear complexity w.r.t. m of the aforementioned encoders. Note that we only sample A_s once at the beginning of the model training and then keep it fixed throughout training and testing. To sum up, in our upcoming experiments in Section 7.4 we will instead consider the following two[3] encoding schemes:

- $Z = \text{GCN}^{(2)}(A + \lambda A_s, X) = \mathcal{F}(A)\text{ReLU}(\mathcal{F}(A + \lambda A_s)XW^{(0)})W^{(1)}$;

- $Z = \text{Linear}(A + \lambda A_s, X) = \mathcal{F}(A + \lambda A_s)XW^{(0)}$.

In these encoders, our altered message passing scheme allows practitioners to incorporate information from prior communities in the resulting node embedding space. A given node $i \in C_k \subset \mathcal{V}$, for $k \in \{1, ..., K\}$, will aggregate information from its direct neighbors and from some nodes in C_k. By design, i will thus have an embedding vector z_i more similar to the embedding vectors of the other nodes in C_k than would be the case for the standard encoders based on $\mathcal{F}(A)$. We recall that the choice of linear and 2-layer GCN encoders is made without loss of generality. $A + \lambda A_s$ could be incorporated into other encoders including deeper GCNs, ChebNets [69] or graph attention networks [364].

The remainder of this Section 7.3.2 on encoders is organized as follows. In Subsection 7.3.2.3 we detail how we derive the matrix A_c (that has loosely been assumed to be "available" so far) in our work. Then, in Subsection 7.3.2.4, we provide a theoretical analysis of our novel encoding strategy. It notably aims to better understand our newly introduced operators A_c and A_s in terms of the spectral filtering they induce, as well as to assess the impact of the *s-regular* sparsification of A_c.

---

[3]We will favor $\text{GCN}^{(2)}$ over $\text{GCN}^{(1)}$ in the remainder of this work, as the former outperformed the latter in our experiments. To simplify the notation $\text{GCN}^{(2)}$ will be referred to as GCN in experiments.

7.3.2.3   Learning A_c and A_s with Modularity-Based Clustering

So far, for pedagogical purposes, we loosely assumed the availability of the A_c and A_s prior community membership matrices. In practice, how these matrices are *learned* plays an important role, as the empirical performance of our strategy will directly depend on the quality of the underlying prior node clusters. Throughout this chapter, we will rely on *modularity* concepts to learn A_c - hence the name *Modularity-Aware GAE and VGAE*. Specifically, we will leverage the popular *Louvain* algorithm [31].

In the absence of node feature information, the Louvain greedy algorithm remains a popular and powerful approach for community detection [31]. It iteratively aims to maximize the *modularity* value [267], defined as follows:

Definition 7.2. Let us consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix A and nodes $i \in \mathcal{V}$ of degree $d_i = D_{ii} = \sum_{j=1}^{n} A_{ij}$. We denote a partition of these nodes into $K \leq |\mathcal{V}|$ communities by $\{C_1, ..., C_K\}$. Then, the *modularity* associated with this partition is:

$$Q = \frac{1}{2m} \sum_{i,j=1}^{n} \left[A_{ij} - \frac{d_i d_j}{2m} \right] \delta(i, j), \tag{7.5}$$

where m is the sum of all edge weights in the graph, i.e., the number of edges for unweighted graphs, and where $\delta(i, j) = 1$ if nodes i and j belong to the same community and 0 otherwise.

In essence, the modularity compares the density of connections inside communities to connections between communities. More specifically, Equation (7.5) returns a scalar Q in the range $[-\frac{1}{2}, 1]$, that measures the difference between the observed fraction of (potentially weighted) edges that occur within the same community and the expected fraction of edges in a configuration model graph, which matches our observed degree distribution but allocates edges randomly without any specified community structure.

The Louvain algorithm   In the Louvain greedy algorithm, aiming to maximize the modularity of a graph over the set of possible cluster assignments, each node is initialized in its own community. Then, the algorithm iteratively completes two phases:

- in phase 1, for each node $i \in \mathcal{V}$, one computes changes in modularity occurring from placing i into the community of each of its neighbors. Then, i is either placed into the community leading to the greatest modularity increase, or remains in its original group if no increase is possible;

- in phase 2, a new graph is constructed. Nodes correspond to communities obtained in phase 1 and edges are formed by summing edge weights occurring between communities.

Edges within a community are represented by self-loops in this new graph. One repeats phase 1 on this new graph until no further modularity improvement is possible.

Our justification for the use of this method to derive A_c is threefold. First and foremost, it automatically selects the relevant number of prior communities K, by iteratively maximizing the modularity value. Secondly, it runs in $O(n \log n)$ time [31]. It therefore scales to large graphs with millions of nodes. Thirdly, such a modularity criterion complements the encoding-decoding paradigm of standard GAE and VGAE models. We argue that learning node embedding spaces from complementary criteria is beneficial. Our experiments will confirm that leveraging prior modularity-based node clusters in the GAE/VGAE outperforms the individual use of the Louvain or of the GAE/VGAE *alone*.

Note that, the use of the Louvain method is made without loss of generality as our framework remains valid for alternative graph mining methods deriving A_c and A_s.

7.3.2.4 Theoretical Analysis of the Encoder's Message Passing Operator

We now conduct a theoretical analysis of our newly introduced message passing operator, which we begin by motivating the spectral analysis of the matrices involved. We recall that the computations performed by a GCN at a given layer are the following:

$$\text{ReLU}(\mathcal{F}(A)H^{(l-1)}W^{(l-1)}). \tag{7.6}$$

If we consider the spectral decomposition of the message passing operator that is used in Equation (7.6), $\mathcal{F}(A) = U\Theta U^T$, where $U = [u_1, \ldots, u_n]^T$ denotes the matrix containing the eigenvectors u_i of $\mathcal{F}(A)$ and Θ is a diagonal matrix containing the eigenvalues θ_i of $\mathcal{F}(A)$. Then, the computation performed in Equation (7.6) can be reformulated as follows:

$$\text{ReLU}(U\Theta U^T H^{(l-1)}W^{(l-1)}) = \text{ReLU}\left(\sum_{i=1}^{n} \theta_i u_i u_i^T H^{(l-1)}W^{(l-1)}\right). \tag{7.7}$$

Therefore, performing one message passing step of the hidden states H on a graph given by $\mathcal{F}(A)$, i.e., $\mathcal{F}(A)H^{(l-1)}$, can be interpreted as a Fourier transform of H, called *graph Fourier transform* [334], where the eigenvectors of $\mathcal{F}(A)$ act as a Fourier basis and the eigenvalues of $\mathcal{F}(A)$ define the Fourier coefficients.

When trying to perform a theoretical analysis of the message passing step in Equation (7.6) it often turns out to be more insightful to consider Equation (7.7) instead and analyze the eigenvalues and eigenvectors of the used message passing operator. Such a spectral perspective has given rise to a variety of architectures proposing learnable functions applied to the diagonal terms of Θ [41, 69, 214]. Historically, the study of spectral graph theory [58, 342], and in

particular the area of graph signal processing [275, 315], has yielded much insight in the study of graphs. Therefore, it is somewhat unsurprising that also in the study of the GNNs the spectral analysis of these architectures is a promising avenue of analysis [21, 66, 102].

Therefore, we now provide spectral results allowing us to gain a better understanding of our proposed message passing operator and compare our proposed message passing operator to the standard message passing operators. To characterize the eigenvectors of our newly introduced $\mathcal{F}(A_c)$ we rely on the concept of 2-sparse eigenvectors.

Definition 7.3. (from Teke and Vaidyanathan [354]) The entries of *2-sparse eigenvectors* are all equal to 0 except for the ith and j^{th} entry which equal to 1 and -1, where i and j denote two nodes which share all their neighbors, i.e., $A_{ih} = A_{jh}$ for $h \in \{1, \ldots, n\} \backslash \{i, j\}$.

An extended discussion of the literature related to such 2-sparse eigenvectors and their corresponding vertices, which are sometimes referred to as twin vertices, can be found in the thesis of Lutzeyer [240]. We are now able to characterize the spectrum and eigenvectors of $\mathcal{F}(A_c)$.

Proposition 7.4. The matrix $\mathcal{F}(A_c)$ has eigenvalues $\{\{1\}^K, \{0\}^{n-K}\}$, where we denote the multiset containing a given element x, y times, by $\{x\}^y$. Each non-zero eigenvalue has an associated eigenvector v_k, with $k \in \{1, \ldots, K\}$, with entries $(v_k)_i = 1$ for $i \in C_k$ and $(v_k)_i = 0$ for $i \notin C_k$. The eigenspace corresponding to the zero eigenvalue has dimension $n - K$ and is spanned by, for example, a set of two-sparse eigenvectors on each of the connected components in the graph.

The proof of Proposition 7.4 can be found in Section 7.6. The informal take-away from Proposition 7.4 is that *the cluster membership of nodes is encoded clearly and compactly in the spectrum and eigenvectors of $\mathcal{F}(A_c)$*. More formally, in Proposition 7.4 we observe that in the spectral domain the operator $\mathcal{F}(A_c)$, which we introduce to the encoder's message passing scheme, directly encodes the cluster membership of the different nodes and all other signals are filtered out by the 0 eigenvalues. Also in the graph domain the matrix $\mathcal{F}(A_c)$ encodes the cluster structure by representing each cluster by a fully connected component of the graph. Therefore, the matrix $\mathcal{F}(A_c)$ is an appropriate choice to introduce cluster information into the message passing scheme and does so clearly in both the graph and spectral domains.

In general, the spectral filtering steps performed by our message passing operator, $\mathcal{F}(A + \lambda A_c)$, and the standard message passing operator, $\mathcal{F}(A)$, are different. $\mathcal{F}(A + \lambda A_c)$ accounts for the clustering information which we introduce. In the following theorem, we provide a result allowing us to establish under which conditions the spectral filtering performed by $\mathcal{F}(A)$ and $\mathcal{F}(A + \lambda A_c)$ are equal, to gain a better understanding of the action of $\mathcal{F}(A + \lambda A_c)$.

Proposition 7.5. If $\mathcal{G}$ is composed of regular connected components, i.e., connected components containing only vertices of equal degree, and the partition of the node set defining these regular

components equals the partition defining A_c, then the matrices $\mathcal{F}(A_c)$ and $\mathcal{F}(A + \lambda A_c)$ have a shared set of eigenvectors and the spectrum of $\mathcal{F}(A + \lambda A_c)$, denoted by $\mathcal{S}(\mathcal{F}(A + \lambda A_c))$, can be expressed in terms of the eigenvalues of $\mathcal{F}(A)$ and $\mathcal{F}(A_c)$, denoted by θ and η, respectively, as:

$$\mathcal{S}(\mathcal{F}(A + \lambda A_c)) = \{g_1(\theta_1) + g_2(\eta_{s(1)}), \ldots, g_1(\theta_n) + g_2(\eta_{s(n)})\}, \tag{7.8}$$

for affine functions g_1, g_2 parameterized by the node degrees and some permutation $s(\cdot)$ defined on the set $\{1, \ldots, n\}$.

The proof of Proposition 7.5 can be found in Section 7.6 as well. Hence, we observe that for graphs consisting of regular connected components the spectral filtering performed by our proposed message passing operator $\mathcal{F}(A + \lambda A_c)$ is equal to that of the standard operator $\mathcal{F}(A)$. For graphs consisting of regular connected components the clustering information is already contained in the spectrum of $\mathcal{F}(A)$ and therefore its further addition does not affect the eigenvectors of our proposed message passing operator.

Note that in general the spectrum of the sum of two matrices cannot be characterized by the individual spectra of the two matrices, meaning that, in general, there does not exist an exact relation between the spectra of $\mathcal{F}(A)$ and $\mathcal{F}(A_c)$ to $\mathcal{F}(A + \lambda A_c)$. We can, however, make direct use of existing results such as Weyl's inequality [381] and the extended Davis–Kahan theorem [241], which, respectively, upper bound the distance of the eigenvalues and spaces spanned by the eigenvectors of the sum of matrices and the individual matrices.

s-regular sparsification   We now turn to the analysis of our sparsified message passing operator A_s, which, as we will see now, still contains the external cluster information without incurring the large computation cost implied by the use of A_c.

Proposition 7.6. If the partition defining the connected components of A_s is a refinement of the partition defining the components of A_c, then the multiplicity of the largest eigenvalue of $\mathcal{F}(A_s)$ is greater or equal to the multiplicity of the largest eigenvalue of $\mathcal{F}(A_c)$. Further, the largest eigenvalue of both $\mathcal{F}(A_s)$ and $\mathcal{F}(A_c)$ equals 1 and the eigenvectors corresponding to the eigenvalue 1 of $\mathcal{F}(A_c)$ are also eigenvectors corresponding to the eigenvalue 1 of $\mathcal{F}(A_s)$.

The proof of Proposition 7.6 can be found in Section 7.6. Informally, Proposition 7.6 can be interpreted to show that *the sparsification of A_c producing A_s does not impact the "informative" part of the spectrum.*   We recall that the eigenvectors corresponding to the largest eigenvalue of $\mathcal{F}(A_c)$ and $\mathcal{F}(A_s)$ are indicator vectors of our introduced cluster membership. Since the remaining eigenvectors are orthogonal to these indicator vectors we know that none of them encode our cluster membership as compactly as the eigenvectors corresponding to the largest eigenvalue.

For $\mathcal{F}(A_s)$ the eigenvalues corresponding to the less informative eigenvectors correspond to nonzero eigenvalues in general and we expect the choice of s to influence the impact of this

uninformative part of the spectrum. This uninformative part of the spectrum can be upper bounded by adapting the bound by Friedman [99] to our message passing operator. However, in our work, we choose a more practice-oriented approach by treating s as a hyperparameter of our model and find its optimal values using the procedure which is described in Section 7.3.3.2.

7.3.3 Modularity-Inpired Loss and Training Strategy

So far, our work considered improvements of the encoder's *architecture*. While this aspect is crucial, we also argue that previously proposed models were often *optimized* in a fashion that, by design, favors link prediction over community detection. With this in mind, this Section 7.3.3 now complements our contributions from Section 7.3.2 with revised training and optimization strategies.

7.3.3.1 Modularity-Inspired Losses for GAE and VGAE

As explained in Section 2.4 from Chapter 2, neural weight matrices of standard GAE and VGAE encoders were tuned by optimizing *reconstruction losses/objectives*, capturing the similarity between the decoded graph and the original one. Usually, these losses directly evaluate the quality of reconstructed node pairs $\hat{A}_{ij}$ w.r.t. their ground truth counterpart A_{ij}. This includes the cross-entropy loss $\mathcal{L}_{\text{GAE}}$ from Equation (2.28) and the ELBO loss $\mathcal{L}_{\text{VGAE}}$ from Equation (2.40). We argue that this optimization strategy also contributes to explaining the underwhelming performance of some GAE and VGAE models on community detection tasks:

- by design, existing optimization strategies favor good performances on link prediction tasks, that precisely consist in accurately reconstructing connected/unconnected node pairs. However, some recent studies emphasized that a good reconstruction of *local* pairwise connections does not always imply a good reconstruction of the *global* community structure from the graph under consideration [235, 376]. This motivates the need for a revised loss function capturing some global community information;

- besides, GAE/VGAE-based community detection experiments often consisted in running k-means algorithms in the final node embedding space (and, as stated at the beginning of Section 7.3.2, we also adopt this strategy). However, this results in clustering embedding vectors based on their L_2 *distances* $\|z_i - z_j\|_2$, whereas the aforementioned reconstruction losses instead often involve *inner products* $(\hat{A}_{ij} = \sigma(z_i^T z_j))$. There is thus a discrepancy between the criterion ultimately used for k-means clustering, and the one used during training to assess node similarities.

To address these issues, we propose to complement standard GAE and VGAE losses with an additional loss term, involving L_2 distances and inspired by the *modularity*[4] from Equation (7.5). In the case of the GAE, we will iteratively minimize by gradient descent:

$$\tilde{\mathcal{L}}_{\text{GAE}} = \mathcal{L}_{\text{GAE}} - \frac{\beta}{2m} \sum_{i,j=1}^{n} \left[A_{ij} - \frac{d_i d_j}{2m} \right] e^{-\gamma \|z_i - z_j\|_2^2}, \tag{7.9}$$

with hyperparameters $\beta \geq 0$, $\gamma \geq 0$. Also, for VGAE we will maximize[5]:

$$\tilde{\mathcal{L}}_{\text{VGAE}} = \mathcal{L}_{\text{VGAE}} + \frac{\beta}{2m} \sum_{i,j=1}^{n} \left[A_{ij} - \frac{d_i d_j}{2m} \right] e^{-\gamma \|z_i - z_j\|_2^2}. \tag{7.10}$$

In Equations (7.9) and (7.10), the exponential term (taking values in $[0,1]$) acts as a *soft counterpart of the common community indicator* $\delta(i,j) \in \{0,1\}$ from Equation (7.5). It tends to 1 when nodes i and j get closer in the embedding space, and tends to 0 when they move apart.

In essence, we expect the addition of such a *global regularizer* to $\mathcal{L}_{\text{GAE}}$ and $\mathcal{L}_{\text{VGAE}}$ to encourage closer embedding vectors (in the L_2 distance) of densely connected parts of the original graph, and therefore to permit a k-means-based *detection of communities with higher modularity values*. On the other hand, the remaining presence of the original $\mathcal{L}_{\text{GAE}}$ or $\mathcal{L}_{\text{VGAE}}$ term[6] in the loss aims to *preserve good performances on link prediction*. The hyperparameter β balances the relative importance of the modularity regularizer w.r.t. the pairwise node pairs reconstruction loss, while the hyperparameter γ regulates the magnitude of $\|z_i - z_j\|_2^2$ in the exponential term. Our experiments will show that proper tuning of β and γ permits us to improve community detection while jointly preserving performances on link prediction.

The use of a modularity-inspired regularizer in the loss of the Modularity-Aware GAE and VGAE builds upon several studies, which were not studying the GAE/VGAE frameworks but emphasized the benefits of various modularity-inspired losses for learning community-preserving node embedding representations [237, 376, 391]. In our setting, we favor the use of a *soft modularity* instead of the term in Equation (7.5), as it permits 1) to obtain a differentiable loss, and 2) to avoid the actual reconstruction of node communities at each training iteration, which would incur a larger computational expense.

To conclude we note that, as for a complete evaluation of $\mathcal{L}_{\text{GAE}}$ and $\mathcal{L}_{\text{VGAE}}$, computing the modularity-inspired terms in Equations (7.9) and (7.10) *on the entire graph* would be of quadratic

---

[4]We emphasize that this new term does *not* involve the prior Louvain clusters used in A_c.

[5]We recall that $\mathcal{L}_{\text{GAE}}$ is *minimized* while $\mathcal{L}_{\text{VGAE}}$ is *maximized*, hence the occurrence of a *minus* term in Equation (7.9) but a *plus* term in Equation (7.10).

[6]Our experiments will consider the original $\mathcal{L}_{\text{GAE}}$ or $\mathcal{L}_{\text{VGAE}}$ from Equations (2.28) and (2.40) and originally formulated by Kipf and Welling [187]. Nonetheless, one can observe that our modularity-inspired global regularizer term could be optimized *in conjunction with other reconstruction losses*. For instance, modularity-inspired terms could be added to the variant formulation of ELBO loss from Choong et al. [55], that incorporates *Gaussian mixtures* in the Kullback-Leibler divergence.

complexity w.r.t. the number of nodes in the graph. In some of our experiments where such a complexity would be unaffordable (roughly, when $n \geq 30\,000$), we will rely on the FastGAE method from Chapter 4 to approximate modularity-inspired terms on random subgraphs and scale our method to larger graphs with up to millions of nodes and edges.

7.3.3.2 On the Selection of Hyperparameters

We expect our modularity-inspired losses to improve the training of our Linear/GCN encoders for community detection, i.e., the tuning of their *weight matrices*. However, in addition to these weight matrices, our Modularity-Aware GAE and VGAE models involve several other hyperparameters, that also play a key role. This includes the standard hyperparameters of GAE and VGAE models (e.g., the number of training iterations, the learning rate, the dimensions of encoding layers, and, potentially, the dropout rate [343]), but also our newly introduced hyperparameters: λ and s from our encoders, as well as β and γ from our losses.

In previous research, the selection procedure for such important hyperparameters was sometimes omitted [55, 56]. In our previous community detection experiments from Chapters 3, 4 and 6, it was solely based on the optimization of AUC scores on link prediction *validation* sets, following the train/validation/test splitting procedure initially adopted by Kipf and Welling [187] and described in Section 2.1.2 from Chapter 2. However, intuitively, the best hyperparameters for community detection might differ from the best ones for link prediction. Such a selection procedure might therefore be suboptimal for community detection problems.

To tackle this issue, and to complement our novel encoders (Section 7.3.2) and losses (Subsection 7.3.3.1), we propose an alternative hyperparameters selection procedure w.r.t. previous practices. As community detection is an *unsupervised* downstream task, we cannot rely on train/validation/test splits as for the *supervised* link prediction binary classification task[7]. Consistently with our already described contributions, we rather propose to rely on *modularity* scores, as it is an unsupervised criterion computed independently of the unobserved ground truth clusters. More precisely, to select relevant hyperparameters, we will:

- firstly, construct link prediction train/validation/test sets, as in Section 2.1.2;

- then, select hyperparameters that maximize the average of:

  - the *AUC score* obtained for link prediction on the validation set;
  - the *modularity score* Q defined in Equation (7.5). This score is obtained from the communities extracted by running a k-means on the final vectors z_i, learned from the train graph (all nodes are visible but edges from validation and test sets are masked).

---

[7]We recall that the ground truth communities of each node will be *unavailable* during training. They will only be ultimately revealed for model evaluation, to compare the agreement of the node partition proposed by our GAE or VGAE model to the ground truth partition.

We expect this dual criterion to facilitate the identification of hyperparameters that will be jointly relevant for link prediction and community detection downstream applications.

7.4 Experimental Analysis

We now present an in-depth experimental evaluation of our proposed Modularity-Aware GAE and VGAE models together with relevant baselines. In Section 7.4.1 we first describe our experimental setting. Then in Section 7.4.2, we report and discuss our results.

7.4.1 Experimental Setting

Datasets In the following, we provide an experimental evaluation on seven graphs of various origins, characteristics, and sizes. First and foremost, we consider the Cora ($n = 2\,708$ and $m = 5\,429$), Citeseer ($n = 3\,327$ and $m = 4\,732$), and Pubmed ($n = 19\,717$ and $m = 44\,338$) citation networks already used in previous chapters and described in Chapter 3. As in our previous experiments, we study two versions of each of these datasets, with and without node features that correspond to bag-of-words vectors of dimensions $f = 1433$, 3703, and 500, respectively. We recall that, in these datasets, nodes are clustered in 6, 7, and 3 topic classes. As these three citations networks remain the most commonly used graph datasets to evaluate GAE and VGAE models (see our review in Chapter 6), we see value in studying them as well, especially in their *featureless* version where, as explained in Section 7.2.3, previous GAE and VGAE extensions fall short on community detection.

Consistently with our concluding recommendation from Chapter 6, we complete our experimental evaluation with four other datasets. Firstly, we consider the ten times larger version of Cora already used in Chapter 6 for community detection, and referred to as Cora-Larger in the following ($n = 23\,166$ and $m = 91\,500$). Nodes are documents clustered in 70 topic-related communities. Additionally, we consider the Blogs web graph ($n = 1\,224$ and $m = 19\,025$) also used in Chapter 6, where nodes correspond to webpages of political blogs connected through hyperlinks. The blogs are clustered in two communities corresponding to politically left-leaning or right-leaning blogs. Thirdly, we examine the SBM graph ($n = 100\,000$ and $m = 1\,498\,844$) presented in Chapter 4, and generated from a stochastic block model [2]. We recall that nodes from this graph are clustered in 100 ground truth communities of 1 000 nodes each.

Lastly, in the paper associated with this work [314], we also considered an industrial-scale private graph provided by Deezer. As we will further emphasize in Part III of this thesis, graph-based methods are at the core of Deezer's recommender systems. In the graph under consideration in this study, denoted Albums ($n = 2\,503\,985$ and $m = 25\,039\,155$) nodes correspond to *music albums* available on the service. They are connected through an undirected edge when they

are regularly *co-listened* by Deezer users (as assessed by internal usage metrics computed from millions of users, but undisclosed in this work for privacy reasons). Deezer is jointly interested in 1) predicting new connections in the graph corresponding to new albums pairs that users would enjoy listening to together, and is achieved by performing the *link prediction* task; and 2) learning groups of similar albums, with the aim of providing usage-based recommendations (i.e., if users listen to several albums from a community, other unlistened albums from this same community could be recommended to them), which is achieved by performing the *community detection* task. In such an industrial application, learning high-quality album representations that would jointly enable effective link prediction and community detection would therefore be desirable. For evaluation, node communities will be compared to a ground truth clustering of albums in 20 groups defined by their main *music genre*, allowing us to assess the musical homogeneity of the node communities proposed by each model.

Tasks For each of these seven graphs, we assess the performance of our models on two tasks:

- **Task 1:** we first consider a pure *community detection* task, consisting in the extraction of a partition of the node set $\mathcal{V}$ which ideally agrees with the ground truth communities of each graph. This task corresponds to the community detection problem considered in experiments from the previous Chapters 3, 4 and 6. Communities are retrieved by running the k-means algorithm (with k-means++ initialization [17]) in the final embedding space of each model to cluster the vectors z_i (with k matching the true number of communities); except for some baseline methods that explicitly incorporate another strategy to partition nodes (see thereafter). We compare the obtained partitions to the ground truth using the AMI score already used in previous chapters, that we complete in this study (more focused on community detection than others) by the Adjusted Rand Index (ARI).

- **Task 2:** we also consider a *joint link prediction and community detection* task. In such a setting, we learn all node embedding spaces from *incomplete* versions of the seven graphs, where 15% of edges were randomly masked. We create a validation and a test set from these masked edges (resp. from 5% and 10% of edges, as in Chapters 3 to 6) and the same number of randomly picked unconnected node pairs acting as "non-edge" negative pairs. Then, as in previous chapters, we evaluate the ability to distinguish edges from non-edges, i.e., *link prediction*, from the embedding space, using once again the AUC and AP scores on test sets. Jointly, we also evaluate the community detection performance obtained from such incomplete graphs, using the same methodology and AMI/ARI scores as in Task 1.

In the case of Task 2, we expect AMI and ARI scores to slightly decrease w.r.t. Task 1, as models will only observe *incomplete* versions of the graphs when learning embedding spaces. Task 2 will further assess whether empirically improving community detection inevitably leads to deteriorating the original good performances of GAE and VGAE models on link prediction.

Table 7.2: Complete list of optimal hyperparameters of Modularity-Aware GAE and VGAE models.

Dataset	Learning rate	Number of iterations	Dropout rate	Use of FastGAE (if yes: subgraphs size)	λ	β	γ	s
Blogs	0.01	200	0.0	No	0.5	0.75	2	10
Cora (featureless)	0.01	500	0.0	No	0.25	1.0	0.25	1
Cora (with features)	0.01	300	0.0	No	0.001	0.01	1	1
Citeseer (featureless)	0.01	500	0.0	No	0.75	0.5	0.5	2
Citeseer (with features)	0.01	500	0.0	No	0.75	0.5	0.5	2
Pubmed (featureless)	0.01	500	0.0	No	0.1	0.5	0.1	5
Pubmed (with features)	0.01	700	0.0	No	0.1	0.5	10	2
Cora-Large	0.01	500	0.0	No	0.001	0.1	0.1	10
SBM	0.01	300	0.0	Yes (10 000)	0.5	0.1	2	10
Albums	0.005	600	0.0	Yes (10 000)	0.25	0.25	1	5

As our proposed modularity-inspired GAE and VGAE models are designed for *joint link prediction and community detection*, we expect them to 1) reach comparable (or, ideally, identical) link prediction scores w.r.t. standard GAE and VGAE models, while 2) reaching better community detection scores.

Details on Models   For the aforementioned evaluation tasks and graphs, we will compare the performances of our proposed *Modularity-Aware GAE and VGAE* models to standard GAE and VGAE and to several other baselines. All results reported below will verify $d = 16$, i.e., all node embedding models will learn embedding vectors z_i of dimension 16. We also tested models with $d \in \{32, 64\}$ by including them in our grid search space and reached similar conclusions to the $d = 16$ setting (we report and further discuss the impact of d in Section 7.4.2):

- **Modularity-Aware GAE and VGAE models**: we trained two versions of Modularity-Aware GAE and VGAE: one with the *linear encoder* described in Section 7.3.2, and one with the *2-layer GCN encoder* ($\text{GCN}^{(2)}$). The latter encoder includes a 32-dimensional hidden layer. We recall that link prediction is performed from inner product decoding $\hat{A}_{ij} = \sigma(z_i^T z_j)$, and that community detection is performed via a k-means on the final vectors z_i learned by each model. During training, as in previous chapters, we used the Adam optimizer [184], without dropout (but we tested models with dropout values in $\{0, 0.1, 0.2\}$ in our grid search optimization). All hyperparameters were carefully tuned following the procedure described in Section 7.3.3.2. For each graph, we tested learning rates from the grid $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.2\}$, number of training iterations in $\{100, 200, 300, ..., 800\}$, with $\lambda \in \{0, 0.01, 0.05, 0.1, 0.2, 0.3, ..., 1.0\}$, $\beta \in \{0, 0.01, 0.05, 0.1, 0.25, 0.5, 1.0, 1.5, 2.0\}$, $\gamma \in \{0.1, 0.2, 0.5, 1.0, 2, 5, 10\}$ and $s \in \{1, 2, 5, 10\}$. The best hyperparameters for each graph are reported in Table 7.2. We adopted the same optimal hyperparameters for GAE *and* VGAE variants (a result which is consistent with the literature [187]). Lastly, as exact loss computation was computationally unaffordable for our two largest graphs, SBM and Albums, their corresponding models were trained by using the FastGAE method from

Chapter 4, approximating losses by reconstructing degree-based sampled subgraphs of $n_{(S)} = 10\,000$ nodes, with $\alpha = 1$.

As in previous chapters, we used Tensorflow [1], training our models (as well as GAE/VGAE baselines described below) on an NVIDIA GTX 1080 GPU, and running other operations on a double Intel Xeon Gold 6134 CPU[8]. Along with the publication of our paper [314] (which is still under review), we will publicly release our source code on GitHub.

- **Standard GAE and VGAE:** we compare these models to two variants of GAEs and VGAEs: one with 2-layer GCN encoders with a 32-dimensional hidden layer (which is equal to the standard GAE and VGAE models from Kipf and Welling [187]) and one with a linear encoder (corresponding to our models from Chapter 6). We note that these are particular cases of our Modularity-Aware GAE/VGAE with GCN or linear encoder and with $\lambda = 0$ and $\beta = 0$. As for our Modularity-Aware models, link prediction is performed from inner product decoding, and community detection via a k-means on vectors z_i. We also adopt a similar model selection procedure as for our Modularity-Aware GAE and VGAE to select hyperparameters (see Section 7.3.3.2). We selected similar learning rates and number of iterations to the values reported in Table 7.2.

- **Other baselines:** for completeness, we also compare the standard and Modularity-Aware GAE/VGAE to several other relevant baselines. First and foremost, we report experiments on the VGAECD [55] and VGAECD-OPT [56] models, designed for community detection and discussed in Section 7.2.2. We use our own Tensorflow implementation of these models[9]. We set similar hyperparameters to the above other GAE/VGAE-based models. In all models, the number of Gaussian mixtures matches the ground truth number of communities of each graph. Besides, we also report experiments on the DVGAE [217] model also discussed in Section 7.2.2, setting similar learning rates and layer dimensions to the above GAE/VGAE-based models, and using the authors' public implementation. In the case of DVGAE, we use 2-layer GCN encoders for consistency with other models of our experiments; we nonetheless acknowledge that Li et al. [217] also proposed another encoding scheme, denoted Heatts in their paper (but unavailable in their public code at the time of writing) that could replace GCNs both in DVGAE and in Modularity-Aware GAE and VGAE. We also report experiments on the ARGA and ARVGA models from Pan et

---

[8]On our machines, running times of the Modularity-Aware GAE and VGAE models were comparable to running times of their standard GAE and VGAE counterparts. For example, training each variant of VGAE on the Pubmed graph for 500 training iterations and with $s = 5$ approximately takes 25 minutes on a single GPU (without the FastGAE method which significantly speeds up training [307]). This is consistent with our claims on the comparable complexity of Modularity-Aware and standard models.

[9]Authors of VGAECD/VGAECD-OPT did not release any public implementation of their models, and we were unable to reach them by e-mail. We note that we obtained some inconsistent results w.r.t. their original performances (specifically, we reached better performances on featureless graphs, and lower performances on graphs with node features), even when adopting their set of hyperparameters. Authors followed an experimental setting and pure community detection task similar to ours.

al. [279] that incorporate an adversarial regularization scheme, with similar hyperparameters, and using the authors' implementation. We already considered these models in Chapter 3. While they were not specifically introduced for community detection, Pan et al. [279] reported empirical gains on this task w.r.t. standard GAE/VGAE (on graphs with node features).

We furthermore consider three additional baselines not utilizing the autoencoder paradigm. Firstly, we report results from node2vec [123] and DeepWalk [287], using similar settings w.r.t. Chapter 3. We use a similar strategy to our aforementioned GAE/VGAE models (k-means/inner products) for community detection and link prediction from embedding spaces. Lastly, we also compare to the *Louvain* community detection method, using the authors' implementation [31]. We see value in comparing our methods to a direct use of Louvain, as this method 1) often emerged as a simple but competitive alternative to GAE/VGAE for community detection (see Section 7.2), and 2) is directly leveraged in our proposed Modularity-Aware GAE and VGAE models as a pre-processing step for the computation of A_c and A_s (see Section 7.3.2).

7.4.2 Results and Discussion

We now present our experimental results. Firstly, we analyze the impact of our proposed hyperparameter selection procedure. Then, we discuss results on Task 1 and then on Task 2. Finally, we mention the limitations and possible extensions of our approach.

On the Selection of Hyperparameters In Subsection 7.3.3.2, we proposed an alternative hyperparameter selection procedure w.r.t. previous practices in the literature. Based on the joint maximization of AUC validation scores for link prediction and modularity scores Q, it aims to identify more relevant GAE/VGAE hyperparameters for joint link prediction and community detection. We recall that the resulting optimal hyperparameters are displayed in Table 7.2.

In our experiments, this procedure did not modify our choices of learning rates and dropout rates for the different GAE/VGAE models under consideration, w.r.t. a standard selection solely relying on AUC validation scores. It had a more noticeable impact on the choices of clustering-related hyperparameters in Modularity-Aware GAE and VGAE (i.e., λ, β, γ, and s) as well as on the required number of training iterations in the gradient descent/ascent.

Figure 7.2 provides an example of this phenomenon, for the number of training iterations required to train Modularity-Aware VGAE models on the featureless Cora and Pubmed graphs. The figure shows that, unlike our proposed procedure jointly based on AUC and Q, a hyperparameter selection based solely on AUC validation scores leads to earlier stopping of the model training and suboptimal performances on community detection. This reaffirms the empirical relevance
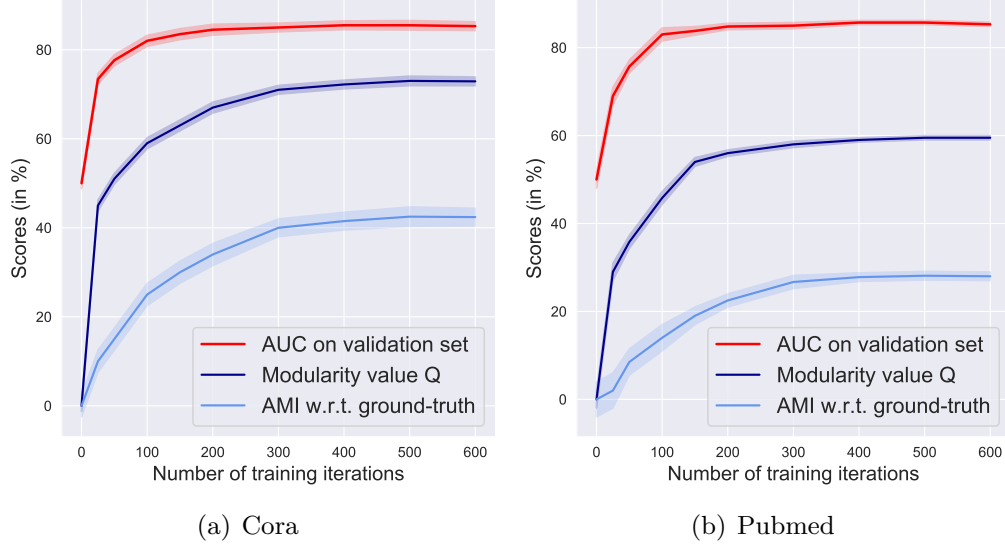
(a) Cora (b) Pubmed

Figure 7.2: Identification of the required number of training iterations, for Modularity-Aware VGAE with linear encoders trained on the featureless (a) Cora, and (b) Pubmed graphs. The plots report the evolution of the modularity Q (dark blue) and AUC link prediction scores on validation sets (red) w.r.t. the number of model training iterations in gradient descent. By looking at the red curves only, one might choose to stop training models after 200 iterations as Kipf and Welling [187], as the AUC validation scores have almost stabilized. However, the dark blue curves emphasize that Q still increases up to 400-500 training iterations for both graphs. By also using Q for hyperparameter selection (as we proposed), one will therefore continue training VGAE models up to 400-500 iterations. The light blue curves confirm that such a strategy eventually leads to better AMI final scores w.r.t. ground truth communities. Note, that the light blue curves could *not* be directly used for tuning, as ground truth communities are assumed to be unavailable at training time.

of our proposed procedure, and that optimal hyperparameters for joint link prediction and community detection might differ from those for link prediction only. Moreover, we note that, while Figure 7.2 focuses on two Modularity-Aware VGAE models, our procedure also leads to the selection of a larger number of training iterations for the other GAE/VGAE-based methods under consideration in this work (values are similar to those in Table 7.2), which explains why, on some occasions, we will report slightly improved results w.r.t. those obtained in the original papers for these models (for instance, some results on Linear GAE and VGAE models will be slightly better than those reported in Chapter 6).

Results for Community Detection on Original Graphs (Task 1) We now focus on the "pure" community detection task (Task 1), performed by models trained on graphs where no edges are removed for model training, as previously introduced in Section 7.4.1. The second and third columns of Table 7.3 report mean AMI and ARI scores on Cora for this task along with standard deviations over 100 runs, for Modularity-Aware GAE and VGAE models (with linear or with 2-layer GCN encoders), their standard counterparts and other baselines. We draw several conclusions from Table 7.3. Foremost, previous conclusions [55, 56, 307, 308] on the limitations of standard GAE and VGAE models for community detection are confirmed:

Table 7.3: Results for Task 1 and Task 2 on the featureless Cora graph, using Modularity-Aware GAE and VGAE with Linear and GCN encoders, their standard GAE and VGAE counterparts, and other baselines. All node embedding models learn embedding vectors of dimension $d = 16$, with other hyperparameters set as described in Section 7.4.1. Scores are averaged over 100 runs. For Task 2, link prediction results are reported from test sets (edges masked for the original graph in addition to the same number of randomly picked unconnected node pairs). **Bold** numbers correspond to the best performance for each score. Scores *in italic* are within one standard deviation range from the best score.

Model	Task 1: Community Detection on complete graph		Task 2: Joint Link Prediction and Community Detection on graph with 15% of edges being masked			
	AMI (in %)	ARI (in %)	AMI (in %)	ARI (in %)	AUC (in %)	AP (in %)
Modularity-Aware GAE/VGAE Models						
Linear Modularity-Aware VGAE	**46.65 ± 0.94**	*39.43 ± 1.15*	*42.86 ± 1.65*	*34.53 ± 1.97*	*85.96 ± 1.24*	*87.21 ± 1.39*
Linear Modularity-Aware GAE	*46.58 ± 0.40*	**39.71 ± 0.41**	**43.48 ± 1.12**	**35.51 ± 1.20**	**87.18 ± 1.05**	*88.53 ± 1.33*
GCN-based Modularity-Aware VGAE	43.25 ± 1.62	35.08 ± 1.88	41.03 ± 1.55	*33.43 ± 2.17*	84.87 ± 1.14	85.16 ± 1.23
GCN-based Modularity-Aware GAE	44.39 ± 0.85	38.70 ± 0.94	41.13 ± 1.35	*35.01 ± 1.58*	*86.90 ± 1.16*	*87.55 ± 1.26*
Standard GAE/VGAE Models						
Linear VGAE	37.12 ± 1.46	26.83 ± 1.68	32.22 ± 1.76	21.82 ± 1.80	85.69 ± 1.17	**89.12 ± 0.82**
Linear GAE	35.05 ± 2.55	24.32 ± 2.99	28.41 ± 1.68	19.45 ± 1.75	84.46 ± 1.64	*88.42 ± 1.07*
GCN-based VGAE	34.36 ± 3.66	23.98 ± 5.01	28.62 ± 2.76	19.70 ± 3.71	85.47 ± 1.18	*88.90 ± 1.11*
GCN-based GAE	35.64 ± 3.67	25.33 ± 4.06	31.30 ± 2.07	19.89 ± 3.07	85.31 ± 1.35	*88.67 ± 1.24*
Other Baselines						
Louvain	42.70 ± 0.65	24.01 ± 1.70	39.09 ± 0.73	20.19 ± 1.73	–	–
VGAECD	36.11 ± 1.07	27.15 ± 2.05	33.54 ± 1.46	24.32 ± 2.25	83.12 ± 1.11	84.68 ± 0.98
VGAECD-OPT	38.93 ± 1.21	27.61 ± 1.82	34.41 ± 1.62	24.66 ± 1.98	82.89 ± 1.20	83.70 ± 1.16
ARGVA	34.97 ± 3.01	23.29 ± 3.21	28.96 ± 2.64	19.74 ± 3.02	85.85 ± 0.87	*88.94 ± 0.72*
ARGA	35.91 ± 3.11	25.88 ± 2.89	31.61 ± 2.05	20.18 ± 2.92	85.95 ± 0.85	*89.07 ± 0.70*
DVGAE	35.02 ± 2.73	25.03 ± 4.32	30.46 ± 4.12	21.06 ± 5.06	85.58 ± 1.31	*88.77 ± 1.29*
DeepWalk	36.58 ± 1.69	27.92 ± 2.93	30.26 ± 2.32	20.24 ± 3.91	80.67 ± 1.50	80.48 ± 1.28
node2vec	41.64 ± 1.25	34.30 ± 1.92	36.25 ± 1.38	29.43 ± 2.21	82.43 ± 1.23	81.60 ± 0.91

in Table 7.3, these methods are notably outperformed by a direct use of the Louvain method (e.g., 42.70% vs 34.36% mean AMI scores for Louvain vs GCN-based VGAE). We also observe that previous GAE/VGAE extensions, reported as baselines, actually provide few empirical benefits w.r.t. standard GAE and VGAE models for this *featureless* graph (e.g., only +1.81 AMI points for VGAECD-OPT[10] vs Linear VGAE). Such a result, in conjunction with the improved performances of these same baselines on graphs *with features* (see thereafter), tends to confirm our initial diagnosis that various GAE/VGAE extensions for community detection mainly benefit from the presence of node features.

On the contrary, our proposed Modularity-Aware GAE and VGAE models, incorporating Louvain clusters as a prior signal in the GAE's and VGAE's encoders, significantly outperform both the use of the Louvain method alone, and the use of GAE and VGAE alone (e.g., with a top 46.65% mean AMI for Modularity-Aware VGAE with linear encoders, and a top 39.71% mean ARI score for Modularity-Aware GAE with linear encoders). Modularity-Aware models also compare favorably to the baselines under consideration (e.g., with +12.1 ARI points for Linear Modularity-Aware GAE w.r.t. VGAECD-OPT), while also providing less volatile results w.r.t. standard GAE/VGAE models. Furthermore, we note that Modularity-Aware models with linear

---

[10]The increase is even smaller when replacing AMI scores obtained via our re-implementation of VGAECD and VGAECD-OPT (i.e., 36.11% and 38.93%, respectively) by AMI scores originally reported in [56] for these methods (i.e., 28.22% and 37.35%, respectively), which are lower than ours.

(a) Linear Standard VGAE

(b) Linear Modularity-Aware VGAE
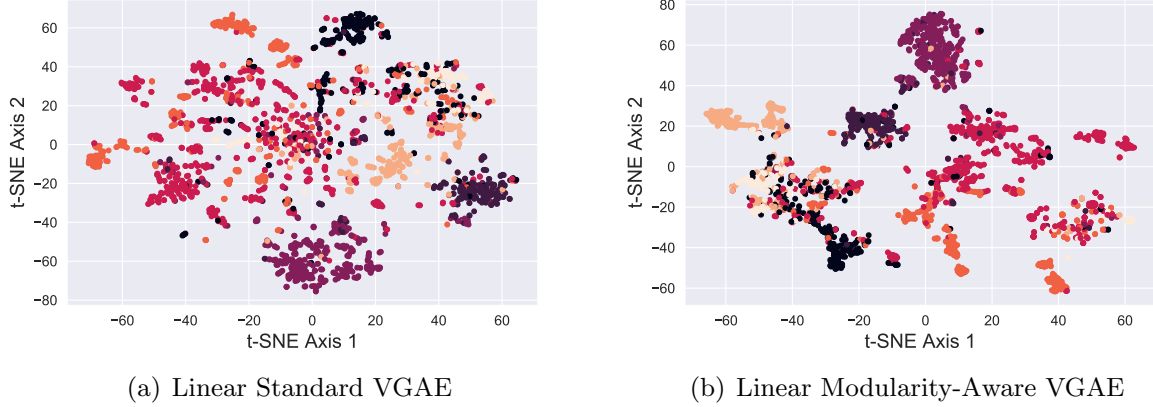
Figure 7.3: Visualization of node embedding representations for the featureless Cora graph, learned by (a) Standard VGAE, and (b) Modularity-Aware VGAE, with linear encoders. The plots were obtained using the t-SNE method for high-dimensional data visualization. Colors denote ground truth communities, that were not available during training. Although community detection is not perfect (both methods return AMI scores < 50% in Table 7.3), node embedding representations from (b) provide a more visible separation of these communities. Specifically, in Table 7.3, using Linear Modularity-Aware VGAE for community detection leads to an increase of 9 AMI points (Task 1) to 10 AMI points (Task 2) for community detection w.r.t. Linear Standard VGAE, while preserving comparable performances on link prediction (Task 2).

encoders tend to outperform their GCN-based counterparts and that GAE and VGAE reach comparable scores. In addition to these results, Figure 7.3 visualizes the representations learned by our models using t-SNE[11] [363].

Overall, we obtain similar conclusions from the other graph datasets. Following the format of Table 7.3, columns two and three of Table 7.4 present detailed community detection results for the featureless Pubmed graph. Table 7.5 reports more summarized results for all other graph datasets under consideration, with and without node features (when available). While Louvain outperforms standard GAE/VGAE models in 5 featureless graphs out of 7 in Table 7.5 (e.g., 19.81% vs 15.79% mean AMI scores for Louvain vs GCN-based VGAE on Albums), our Modularity-Aware models manage to achieve either comparable or better performances w.r.t. standard models, Louvain and other baselines in the wide majority of experiments. Furthermore, throughout Table 7.5, we observe that linear encoders outperform their GCN-based counterparts in 8/10 experiments, and that VGAE models outperform GAE models in 8/10 experiments (even though performances are often relatively close, as for Cora). Lastly we emphasize that, while all tables report results for fixed embedding dimensions of $d = 16$, we reached similar conclusions for $d \in \{32, 64\}$. Although performances sometimes improved by increasing d, the *ranking* of methods under consideration remained similar. Such a result is consistent with experiments from previous chapters. For instance, by setting $d = 64$, Deepwalk's mean AMI score increased from 36.58% to roughly 41% on the featureless Cora graph, while the mean AMI score from our

---

[11]We used the scikit-learn [284] implementation of this data visualization method: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Table 7.4: Results for Task 1 and Task 2 on the featureless Pubmed graph, using Modularity-Aware GAE and VGAE with Linear and 2-layer GCN encoders, their standard GAE and VGAE counterparts, and other baselines. All node embedding models learn embedding vectors of dimension $d = 16$, with other hyperparameters set as described in Section 7.4.1. Scores are averaged over 100 runs. For Task 2, link prediction results are reported from test sets (edges masked during training + same number of randomly picked unconnected node pairs). **Bold** numbers correspond to the best performance for each score. Scores *in italic* are within one standard deviation range from the best score.

Model	Task 1: Community Detection on complete graph		Task 2: Joint Link Prediction and Community Detection on graph with 15% of edges being masked			
	AMI (in %)	ARI (in %)	AMI (in %)	ARI (in %)	AUC (in %)	AP (in %)
Modularity-Aware GAE/VGAE Models						
Linear Modularity-Aware VGAE	28.12 ± 0.29	29.01 ± 0.51	25.93 ± 0.65	23.76 ± 0.49	$\mathbf{85.76 \pm 0.37}$	87.77 ± 0.31
Linear Modularity-Aware GAE	*28.54 ± 0.24*	26.36 ± 0.34	$\mathbf{26.38 \pm 0.43}$	21.30 ± 0.59	84.39 ± 0.32	*87.92 ± 0.40*
GCN-based Modularity-Aware VGAE	28.08 ± 0.27	28.14 ± 0.33	25.70 ± 0.86	22.65 ± 0.80	84.70 ± 0.24	86.64 ± 0.15
GCN-based Modularity-Aware GAE	$\mathbf{28.74 \pm 0.28}$	26.71 ± 0.47	25.52 ± 0.45	20.52 ± 0.31	85.07 ± 0.35	*88.27 ± 0.39*
Standard GAE/VGAE Models						
Linear VGAE	22.16 ± 2.02	13.90 ± 3.47	21.78 ± 2.57	13.81 ± 3.17	84.57 ± 0.51	$\mathbf{88.31 \pm 0.44}$
Linear GAE	12.61 ± 4.61	6.37 ± 3.86	12.60 ± 4.67	6.21 ± 1.75	82.03 ± 0.32	87.71 ± 0.24
GCN-based VGAE	20.11 ± 3.05	13.12 ± 3.10	17.34 ± 2.99	8.71 ± 3.05	82.19 ± 0.88	87.51 ± 0.55
GCN-based GAE	20.12 ± 2.89	14.21 ± 2.78	16.75 ± 3.36	9.18 ± 2.71	82.33 ± 1.32	87.20 ± 0.58
Other Baselines						
Louvain	20.06 ± 0.27	10.34 ± 0.99	16.71 ± 0.46	8.32 ± 0.79	–	–
VGAECD	20.32 ± 2.95	13.54 ± 2.98	17.39 ± 3.04	9.21 ± 3.12	82.05 ± 0.90	87.30 ± 0.53
VGAECD-OPT	22.50 ± 1.99	14.58 ± 2.86	21.98 ± 2.46	15.22 ± 2.92	82.03 ± 0.82	87.41 ± 0.53
ARGVA	20.73 ± 3.10	13.94 ± 3.12	17.63 ± 3.19	9.19 ± 3.09	84.07 ± 0.55	87.73 ± 0.49
ARGA	20.98 ± 2.90	14.79 ± 2.80	17.21 ± 3.01	9.59 ± 2.76	83.73 ± 0.53	*87.90 ± 0.45*
DVGAE	23.15 ± 2.52	15.02 ± 3.33	22.10 ± 2.50	14.62 ± 2.96	83.21 ± 0.92	*88.17 ± 0.49*
DeepWalk	*28.53 ± 0.43*	29.61 ± 0.33	15.80 ± 1.05	16.16 ± 1.75	80.63 ± 0.42	81.03 ± 0.54
node2vec	*28.52 ± 1.12*	$\mathbf{30.63 \pm 1.14}$	23.88 ± 0.54	$\mathbf{25.90 \pm 0.65}$	81.03 ± 0.30	82.33 ± 0.41

Linear Modularity-Aware GAE model simultaneously increased from 46.58% to 47.80%.

Interestingly, we also observe that combining the Louvain method and GAE/VGAE in our Modularity-Aware models might be empirically beneficial *even when standard GAE/VGAE models initially outperform the Louvain method.* For instance in Table 7.4, our Linear Modularity-Aware VGAE outperforms the Linear Standard VGAE (e.g., with 28.12% vs 22.16% mean AMI scores), despite the fact that this standard model initially outperformed the Louvain method (20.06% mean AMI score). This tends to confirm that modularity-based clustering *à la* Louvain complements the encoding-decoding paradigm of GAE and VGAE models, and that learning node embedding spaces from complementary criteria is empirically beneficial. On a more negative note, we nonetheless acknowledge that, on Cora, Citeseer, and Pubmed in Table 7.5, empirical gains of Modularity-Aware models are less visible on graphs *equipped with node features* than on featureless graphs, which we will further discuss in our limitation section.

As our Modularity-Aware models include two main novel components (namely our community-preserving encoders from Section 7.3.2 and our revised loss from Section 7.3.3), one might wonder what the contribution of each of these components to the performance gains is. To study this question, we report in Figure 7.4 the results of an *ablation study*, that consisted in training variant versions of our models leveraging one of these components only[12] (i.e., the encoder but

---

[12] A Modularity-Aware GAE or VGAE model that leverages the novel encoder only (respectively, the novel
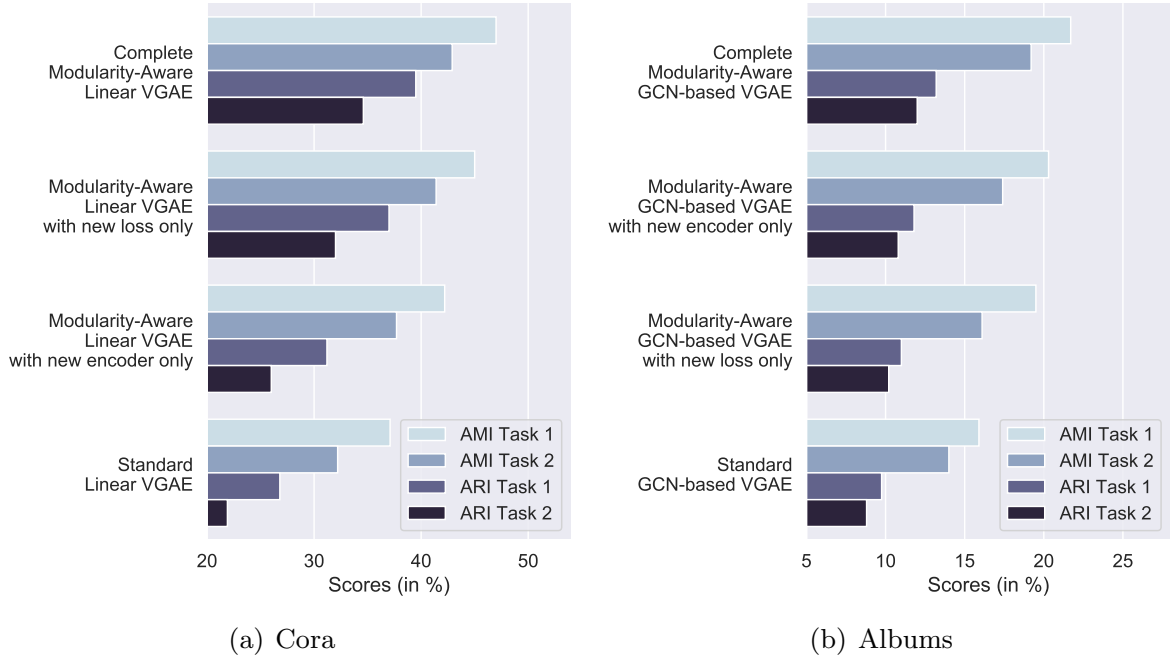
Figure 7.4: Comparison of two "complete" Modularity-Aware VGAE, trained on (a) featureless Cora and (b) Albums with variants of these models only leveraging the new *encoder* from Section 7.3.2, or the new *loss* from Section 7.3.3. We observe that incorporating any of these two components improves community detection on these two graphs w.r.t. Standard VGAE. Moreover, using both components *simultaneously* leads to the best results. Note, the optimal pair (λ, β) for complete models might differ from the optimal λ (resp. β) when incorporating the new encoder (resp. loss) only.

not the loss, or the loss but not the encoder). Figure 7.4 shows that incorporating any of these two individual components into the model improves community detection. The gain is larger for the *loss* in the Cora example from Figure 7.4(a), while it is larger for the *encoder* in the Albums example from Figure 7.4(b). A simultaneous use of the encoder and the loss leads to the best results in both examples, which we also confirmed on the other graphs under consideration.

Results for Joint Link Prediction and Community Detection (Task 2) We now study results for Task 2, the joint link prediction and community detection task described in Section 7.4.1, and performed on incomplete versions of the graph datasets where 15% of edges are randomly masked. Results for this task (i.e., AMI and ARI scores from community detection on incomplete graphs, and AUC and AP scores from link prediction on test sets) are reported in the four rightmost columns of Tables 7.3, 7.4 and 7.5. AMI and ARI scores from this task are also included in the ablation study in Figure 7.4.

We draw several conclusions from these additional experiments. First of all, we confirm that AMI and ARI scores decrease slightly w.r.t. Task 1, which was expected due to the absence of

---

loss only) corresponds to a particular case of a "complete" Modularity-Aware GAE or VGAE model, where the hyperparameter β (respectively, the hyperparameter λ) is set to 0.

part of the graph structure during the training phase (e.g., from 46.65% to 42.86% mean AMI, for Linear Modularity-Aware VGAE on Cora in Table 7.3). Nonetheless, the ranking of the different methods under consideration remains consistent with Task 1. In particular, our Modularity-Aware models still outperform baselines in cases where they were already outperforming in Task 1 (e.g., with a top 43.48% mean AMI for Linear Modularity-Aware GAE on Cora in Table 7.3, vs 28.41% for the standard Linear GAE and 39.09% for the Louvain method).

Besides these confirmations, the main goal of Task 2 was to address our second research question stated in Section 7.1: *do improvements on the community detection task necessarily incur a loss in the link prediction performance or can they be jointly addressed with high accuracy?* Indeed, as GAE and VGAE models were originally recognized as effective link prediction methods, improving community detection while deteriorating link prediction might be undesirable, especially in problems requiring effective node embedding representations for multitask applications (see the Deezer example from Section 7.4.1). By design, our proposed encoders, losses, and selection procedure specifically aimed to avoid such a deterioration.

Empirical results confirm the ability of Modularity-Aware models to preserve comparable link prediction performances w.r.t standard GAE and VGAE models. For instance in Table 7.3, our Linear Modularity-Aware GAE model reaches mean AUC and AP scores of 87.18% and 88.53%, respectively, which is comparable (or even slightly better in the case of AUC) to Linear Standard GAE (84.46% and 88.42%, respectively). We reach similar results for the three other Modularity-Aware models in Table 7.3, while scores of several baselines deteriorate by a few points. Overall, all other Modularity-Aware models reported in the complete Table 7.5 achieve comparable (either better, identical, or only a few points below) AUC and AP scores w.r.t. their GAE or VGAE counterparts.

Limitations and Possible Extensions   As observed in the previous paragraphs, empirical gains of Modularity-Aware models are less pronounced on graphs equipped with node features (although non null in 2/3 cases). In Table 7.5, for Cora with features, we "only" report increases in Task 1 AMI scores of +2.63 points w.r.t. the corresponding standard VGAE model. For comparison, in the featureless case, we reported increases in Task 1 AMI scores of +11.53 points. Furthermore, our Modularity-Aware VGAE model does not surpass standard VGAE models at all on Pubmed with features. We hypothesize that the incorporation of Louvain-based prior clusters in Modularity-Aware models might be less relevant for these attributed graphs. Indeed, while Louvain only leverages the graph structure for node clustering, node features seem to play a strong role in the identification of ground truth communities for these graphs.

7.4. Experimental Analysis

Table 7.5: Summarized results for Tasks 1 and 2 on all graphs. For each graph, for brevity, we only report the **best** Modularity-Inspired model (best on Task 2, among GCN **or** linear encoder, and GAE **or** VGAE), its standard counterpart, and a comparison to the Louvain baseline as well as the best other baseline (among VGAECD, VGAECD-OPT, ARGA, ARGVA, DVGAE, DeepWalk and node2vec). All node embedding models learn embedding vectors of dimension $d = 16$, with other hyperparameters set as described in Section 7.4.1. Scores are averaged over 100 runs except for the larger SBM and Albums graphs (10 runs). **Bold** numbers correspond to the best performance for each score. Scores *in italic* are within one standard deviation range from the best score.

Dataset	Model	Task 1: Community Detection on complete graph		Task 2: Joint Link Prediction and Community Detection on graph with 15% of edges being masked			
		AMI (in %)	ARI (in %)	AMI (in %)	ARI (in %)	AUC (in %)	AP (in %)
Blogs	GCN-based Modularity-Aware VGAE	**74.23 ± 0.95**	**83.13 ± 0.79**	**70.42 ± 1.28**	**79.80 ± 1.12**	**91.67 ± 0.39**	*92.37 ± 0.41*
	GCN-based Standard VGAE	73.42 ± 0.65	82.58 ± 0.52	66.90 ± 3.32	*77.23 ± 3.89*	*91.64 ± 0.42*	**92.52 ± 0.51**
	Louvain	63.43 ± 0.86	76.66 ± 0.70	57.25 ± 1.67	73.00 ± 1.56	–	–
	Best other baseline:						
	node2vec	72.88 ± 0.87	82.08 ± 0.73	67.64 ± 1.23	77.03 ± 1.85	83.63 ± 0.34	79.60 ± 0.61
Cora	Linear Modularity-Aware GAE	**46.58 ± 0.40**	**39.71 ± 0.41**	**43.48 ± 1.12**	**35.51 ± 1.20**	**87.18 ± 1.05**	**88.53 ± 1.33**
	Linear Standard GAE	35.05 ± 2.55	24.32 ± 2.99	28.41 ± 1.68	19.45 ± 1.75	84.46 ± 1.64	*88.42 ± 1.07*
	Louvain	42.70 ± 0.65	24.01 ± 1.70	39.09 ± 0.73	20.19 ± 1.73	–	–
	Best other baseline:						
	node2vec	41.64 ± 1.25	34.30 ± 1.92	36.25 ± 1.38	29.43 ± 2.21	82.43 ± 1.23	81.60 ± 0.91
Cora with features	Linear Modularity-Aware VGAE	**52.61 ± 1.41**	**45.74 ± 2.02**	**49.70 ± 2.04**	**43.64 ± 3.51**	**93.10 ± 0.88**	**94.06 ± 0.75**
	Linear Standard VGAE	49.98 ± 2.40	*43.15 ± 4.35*	46.90 ± 1.43	38.24 ± 3.56	*93.04 ± 0.80*	*94.04 ± 0.75*
	Louvain	42.70 ± 0.65	24.01 ± 1.70	39.09 ± 0.73	20.19 ± 1.73	–	–
	Best other baseline:						
	VGAECD-OPT	50.32 ± 1.95	*43.54 ± 3.23*	47.83 ± 1.64	39.45 ± 3.53	92.25 ± 1.07	92.60 ± 0.91
Citeseer	Linear Modularity-Aware VGAE	21.28 ± 1.03	**15.39 ± 1.06**	19.05 ± 1.47	**12.19 ± 1.38**	**80.84 ± 1.64**	**84.21 ± 1.21**
	Linear Standard VGAE	13.83 ± 1.00	8.31 ± 0.89	11.11 ± 1.43	5.87 ± 0.87	78.26 ± 1.55	*82.93 ± 1.39*
	Louvain	**24.72 ± 0.27**	9.21 ± 0.75	**22.71 ± 0.47**	7.70 ± 0.67	–	–
	Best other baseline:						
	node2vec	18.68 ± 1.13	*14.93 ± 1.15*	14.40 ± 1.18	*12.13 ± 1.53*	76.05 ± 2.12	79.46 ± 1.65
Citeseer with features	Linear Modularity-Aware VGAE	**25.11 ± 0.94**	**15.55 ± 0.60**	*22.21 ± 1.24*	**12.59 ± 1.25**	86.54 ± 1.20	88.07 ± 1.22
	Linear Standard VGAE	17.80 ± 1.61	6.01 ± 1.46	17.38 ± 1.43	6.10 ± 1.51	**89.08 ± 1.19**	**91.19 ± 0.98**
	Louvain	24.72 ± 0.27	9.21 ± 0.75	**22.71 ± 0.47**	7.70 ± 0.67	–	–
	Best other baseline:						
	DVGAE	20.09 ± 2.84	12.16 ± 2.74	16.02 ± 3.32	*10.03 ± 4.48*	86.85 ± 1.48	88.43 ± 1.23
Pubmed	Linear Modularity-Aware GAE	**28.54 ± 0.24**	26.36 ± 0.34	**26.38 ± 0.43**	21.30 ± 0.59	**84.39 ± 0.32**	**87.92 ± 0.40**
	Linear Standard GAE	12.61 ± 4.61	6.37 ± 3.86	12.60 ± 4.67	6.21 ± 1.75	82.03 ± 0.32	*87.71 ± 0.24*
	Louvain	20.06 ± 0.27	10.34 ± 0.99	16.71 ± 0.46	8.32 ± 0.79	–	–
	Best other baseline:						
	node2vec	*28.52 ± 1.12*	**30.63 ± 1.14**	23.88 ± 0.54	**25.90 ± 0.65**	81.03 ± 0.30	82.33 ± 0.41
Pubmed with features	Linear Modularity-Aware VGAE	30.09 ± 0.63	**29.11 ± 0.65**	**29.60 ± 0.70**	**28.54 ± 0.74**	*97.10 ± 0.21*	**97.21 ± 0.18**
	Linear Standard VGAE	29.98 ± 0.41	*29.05 ± 0.20*	*29.51 ± 0.52*	*28.50 ± 0.36*	**97.12 ± 0.20**	*97.20 ± 0.17*
	Louvain	20.06 ± 0.27	10.34 ± 0.99	16.71 ± 0.46	8.32 ± 0.79	–	–
	Best other baseline:						
	VGAECD-OPT	**32.47 ± 0.45**	*29.09 ± 0.42*	*29.46 ± 0.52*	*28.43 ± 0.61*	94.27 ± 0.33	94.53 ± 0.36
Cora-Larger	Linear Modularity-Aware VGAE	**48.55 ± 0.18**	**22.21 ± 0.39**	**46.10 ± 0.29**	**20.53 ± 0.38**	**95.76 ± 0.17**	**96.31 ± 0.12**
	Linear Standard VGAE	46.07 ± 0.54	20.01 ± 0.90	43.38 ± 0.37	18.02 ± 0.66	*95.55 ± 0.22*	*96.30 ± 0.18*
	Louvain	44.72 ± 0.50	19.46 ± 0.66	43.41 ± 0.52	19.29 ± 0.68	–	–
	Best other baseline:						
	DVGAE	46.63 ± 0.56	20.72 ± 0.96	43.48 ± 0.61	18.45 ± 0.67	94.97 ± 0.23	95.98 ± 0.21
SBM	Linear Modularity-Aware VGAE	**36.08 ± 0.13**	**8.11 ± 0.10**	**35.85 ± 0.20**	**8.06 ± 0.11**	*82.34 ± 0.38*	*86.76 ± 0.41*
	Linear Standard VGAE	35.01 ± 0.21	7.88 ± 0.15	30.79 ± 0.21	6.50 ± 0.13	80.11 ± 0.35	83.40 ± 0.36
	Louvain	*36.06 ± 0.12*	**8.11 ± 0.10**	*35.84 ± 0.18*	*8.03 ± 0.09*	–	–
	Best other baseline:						
	DVGAE	*35.90 ± 0.18*	*8.07 ± 0.15*	35.53 ± 0.23	*7.95 ± 0.19*	**82.59 ± 0.36**	**87.08 ± 0.40**
Albums	GCN-Based Modularity-Aware VGAE	**21.64 ± 0.18**	**13.19 ± 0.09**	**19.10 ± 0.21**	**12.00 ± 0.17**	**85.40 ± 0.14**	*86.38 ± 0.15*
	GCN-Based Standard VGAE	15.79 ± 0.32	9.75 ± 0.21	13.98 ± 0.35	8.81 ± 0.32	*85.37 ± 0.12*	**86.41 ± 0.11**
	Louvain	19.81 ± 0.19	12.21 ± 0.09	17.68 ± 0.20	11.02 ± 0.13	–	–
	Best other baseline:						
	node2vec	20.03 ± 0.24	12.20 ± 0.19	18.34 ± 0.29	11.27 ± 0.28	83.51 ± 0.17	84.12 ± 0.15

Nevertheless, we recall that the use of the Louvain method was made without loss of generality. As explained in Section 7.3.2, our revised message passing operators would remain valid for other methods that alternatively derive a prior clustering signal. Future experiments on such alternatives (e.g., methods processing node features) could therefore improve community detection performances on these three attributed graphs. Overall, the empirical performance of our method directly depends on the quality of the underlying prior clustering method used to compute A_c and A_s, which should therefore be carefully selected.

More broadly, our framework could also straightforwardly incorporate alternative encoders (besides linear and multi-layer GCN encoders), alternative decoders (e.g., decoders replacing inner products by more refined graph reconstruction methods [124, 329] including our gravity-inspired decoder from Chapter 5) and alternative losses (for instance, as explained in Section 7.3.3, our modularity-inspired regularizer could be optimized in conjunction with the ELBO loss from VGAECD and VGAECD-OPT [55, 56] involving Gaussian mixtures). One could also replace our k-means step, to cluster vectors z_i, with another method such as k-medoids [281] or spectral clustering [368] (although our preliminary experiments in this direction did not reach significantly better results). Future work considering such alternative architectures for Modularity-Aware GAE/VGAE could definitely lead to the improvement of our models.

Lastly, we would also need to extend Modularity-Aware GAE and VGAE to dynamic graphs. Indeed, while our work considered fixed graph structures, real-world graphs often evolve over time. For instance, as we will further explain and study in Chapter 9, on the Deezer service, new albums regularly appear in the musical catalog. New nodes will therefore appear in the Albums graph. Capturing such changes, e.g., through dynamic embedding methods [271], might permit learning more refined representations and provide effective dynamic community detection.

7.5 Conclusion

In this chapter, we introduced an effective method for simultaneous link prediction and community detection, compatible with both the GAE and VGAE frameworks. This approach, referred to as Modularity-Aware GAE and VGAE, is based on a rigorous diagnosis of the shortcomings of existing approaches to this problem.

Modularity-Aware GAE and VGAE take advantage of two elements: a theoretically grounded variant of message passing operator in the GAE and VGAE encoders, that incorporates prior cluster information, and the addition of a modularity-based loss component to the usual existing loss functions. Both elements were experimentally shown to have an individual impact on the community detection performances. We furthermore introduced a revised hyperparameter selection procedure specifically designed for joint link prediction and community detection. We experimentally demonstrated the effectiveness of the approach on several real-world datasets,

both with node features and, crucially, featureless graphs. The results are consistently on par or better than popular baselines for both link prediction and community detection.

Last, but not least, we identified several research directions that, in future studies, should lead to the extension and the improvement of our work. In particular, we mentioned potential extensions of our approach to dynamic graphs. In Chapter 9, we will present some applications of Modularity-Aware GAE and VGAE models to recommendation problems on Deezer, that inherently incorporate a dynamic aspect.

7.6   Appendices

In this supplementary section, we prove the propositions of Subsection 7.3.2.4. These proofs were placed out of the main content of Chapter 7 for the sake of brevity and readability.

Preliminaries

We begin by introducing several theoretical results which we will use in our proofs. The specific formulations of the results in this section, i.e., Definition 7.7 and Propositions 7.8, 7.9 and 7.10, are adapted from Lutzeyer [240]. When considering regular graphs, i.e., graphs containing only nodes of equal degree, their different graph representation matrices, such as the adjacency matrix, Laplacian matrices, and the GCN's message passing operator, are related via polynomial matrix transformations. These are now defined.

Definition 7.7. Horn and Johnson [147] define the evaluation of a polynomial $p(x) = c_l x^l + c_{l-1} x^{l-1} + \ldots + c_1 x + c_0$ at a matrix Φ as

$$p(\Phi) = c_l \Phi^l + c_{l-1} \Phi^{l-1} + \ldots + c_1 \Phi + c_0 I. \tag{7.11}$$

Horn and Johnson [147] further include a discussion of the influence of a polynomial matrix transformation on the matrices' eigenvalues and eigenvectors, which we reproduce below.

Proposition 7.8. Let $p(\cdot)$ be a given polynomial. If ϕ is an eigenvalue of $\Phi \in \mathbb{R}^{n \times n}$, while u is an associated eigenvector, then $p(\phi)$ is an eigenvalue of the matrix $p(\Phi)$ and u is an eigenvector of $p(\Phi)$ associated with $p(\phi)$.

Since we consider graphs consisting of several connected components in a multitude of our propositions, we now provide a theorem which relates the eigenvalues and eigenvectors of the whole graph to those of its connected components.

Proposition 7.9. Let $\mathcal{G}$ be a graph with corresponding adjacency matrix A and assume $\mathcal{G}$ to consist of K connected components each with corresponding adjacency matrix A_k for $k \in$

$\{1, \ldots, K\}$. Then, the eigenvalues of $\mathcal{F}(A)$ are equal to the union of the eigenvalues of $\mathcal{F}(A_k)$ over $k \in \{1, \ldots, K\}$. Further, a set of eigenvectors of $\mathcal{F}(A)$ can be constructed from the eigenvector sets of $\mathcal{F}(A_k)$ for $k \in \{1, \ldots, K\}$.

We refer to [314] for the proof of Proposition 7.9. To allow us to relate the spectra and eigenvectors of A to the more novel GCN message passing operator $\mathcal{F}(A)$ we frequently make use of the matrix similarity relationship. The consequence of a matrix similarity relationship between matrices on their eigenvalues and eigenvectors is discussed in Proposition 7.10.

Proposition 7.10. [147] If two matrices Φ and Ψ are related via a nonsingular matrix S as follows, $\Phi = S^{-1}\Psi S$. Then, Φ and Ψ have the same multiset of eigenvalues. Further, for eigenvector v with corresponding eigenvalue ϕ of Φ gives rise to an eigenvector Sv of Ψ with equal corresponding eigenvalue ϕ.

Equipped with such a background, we can now prove the propositions of Subsection 7.3.2.4.

Proof of Proposition 7.4

Teke and Vaidyanathan [354], among others, state that the unnormalized Laplacian matrix $L = D - A$ corresponding to a complete graph has eigenvalue 0 with multiplicity 1 and eigenvalue n with multiplicity $n-1$. Furthermore, the eigenspace corresponding to the eigenvalue n is spanned by a 2-sparse set of orthogonal eigenvectors [354]. In addition, the eigenvector corresponding to the eigenvalue 0 of the unnormalized graph Laplacian describing a connected graph is well known to be the constant eigenvector [368].

Since the complete graph is regular, its degree matrix is a multiple of the identity matrix, i.e., $D = (n-1)I_n$. Therefore, for complete graphs the following relationship holds $\mathcal{F}(A) = I_n - \frac{1}{n}L$. Hence, from Proposition 7.8 the eigenvectors of $\mathcal{F}(A)$ and L are equal and $\mathcal{F}(A)$ has the eigenvalue 1 with multiplicity 1 and eigenvalue 0 with multiplicity $n - 1$. Now, since A_c corresponds to a graph composed of several complete graphs, we can invoke Proposition 7.9 to construct the spectrum and eigenvectors of $\mathcal{F}(A_c)$ from the the spectrum and eigenvectors of $\mathcal{F}(A)$ corresponding to a complete graph, which we just derived. Consequently, $\mathcal{F}(A_c)$ has eigenvalues $\{\{1\}^K, \{0\}^{n-K}\}$ and a set of eigenvectors as described in the statement of Proposition 7.4.

Proof of Proposition 7.5

Proposition 7.9 can be used to extend the required result from one connected component of $\mathcal{G}$ to the full graph. Therefore, we consider only one connected component on the graph from now on. Let A' denote the adjacency matrix of this connected component, containing nodes of degree b, and A'_c denote the corresponding complete component of A_c, containing nodes of degree $n' - 1$.

Then, the adjacency matrix of our connected component under consideration $A' + \lambda A'_c$ is related to $\mathcal{F}(A' + \lambda A'_c)$ as follows:

$$\mathcal{F}(A' + \lambda A'_c) = \frac{1}{b + \lambda(n' - 1) + 1} \left(A' + \lambda A'_c + I \right). \tag{7.12}$$

Therefore, from Proposition 7.8 it follows that $A' + \lambda A'_c$ and $\mathcal{F}(A' + \lambda A'_c)$ share eigenvectors. Similarly, the relations $\mathcal{F}(A') = \frac{1}{b+1}(A' + I)$ and $\mathcal{F}(A'_c) = \frac{1}{n'}(A'_c + I)$ together with Proposition 7.8 allow us to establish that both $\mathcal{F}(A')$ and A' as well as $\mathcal{F}(A'_c)$ and A'_c each have a common set of eigenvectors.

We now make use of a result by Godsil [110], which states that the adjacency matrix of a graph commutes with the matrix of all ones, i.e., $A_c + I_n$, if and only if the graph under consideration is regular. Further, a family of matrices is a commuting family if and only if they are simultaneously diagonalizable, i.e., they share a set of eigenvectors [147]. Hence, A' and A'_c share a set of eigenvectors. Furthermore, this shared set of eigenvectors is also a valid set of eigenvectors for $A' + \lambda A'_c$, which, in conjunction with the above polynomial relationships, establishes the needed eigenvector relation.

In addition the eigenvalues of the sum of two simultaneously diagonalizable matrices Φ, Ψ with eigenvalues denoted by ϕ and ψ, respectively, are related [147] as follows

$$\mathcal{S}(\Phi + \Psi) = \{\phi_1 + \psi_{s(1)}, \ldots, \phi_n + \psi_{s(n)}\}, \tag{7.13}$$

for some permutation $s(\cdot)$ defined on the set $\{1, \ldots, n\}$. Since A' and $A'_c + I_n$ are simultaneously diagonalizable their eigenvalues follow the relation in Equation (7.13). Now the above polynomial relationships of the GCN message passing operators to the corresponding adjacency matrices gives us the desired eigenvalue result and establish that $g_1(\mu) = \frac{b+1}{b+\lambda(n'-1)+1}(\mu - 1)$ and $g_2(\eta) = \frac{\lambda(n'-1)+1}{b+\lambda(n'-1)+1}(\eta - 1) + 1$.

Proof of Proposition 7.6

Hoory et al. [146] state that for s-regular graphs the largest eigenvalue of the corresponding adjacency matrix A' equals s and the corresponding eigenvector is constant. Now the relation $\mathcal{F}(A') = \frac{1}{o+1}(A' + I_n)$ in conjunction with Proposition 7.8 establish that the largest eigenvalue of $\mathcal{F}(A')$ equals 1 with a corresponding constant eigenvector. This spectrum and eigenvectors can be extended to the matrix $\mathcal{F}(A_s)$ corresponding to a graph of several s-regular connected components using Proposition 7.9. The comparison of the derived spectrum and eigenvectors to those derived in Proposition 7.4 completes this proof.

Part III

Applications to Music Recommendation

8

Graph-Based Cold Start Similar Artists Ranking

This chapter presents research conducted with Romain Hennequin, Benjamin Chapus, Viet-Anh Tran, and Michalis Vazirgiannis, and published in the proceedings of the 15th ACM Conference on Recommender Systems (RecSys 2021) [313] where it received a "best student paper" honorable mention.

8.1 Introduction

The previous Chapter 7 concluded the second part of this thesis, which detailed our technical contributions to improving node representation learning with GAE and VGAE models. In this Chapter 8, we now start the Part III, which provides five additional and more "applied" chapters, presenting several industrial applications to music recommendation problems arising on music streaming services such as Deezer. Firstly, in this Chapter 8, we propose a graph-based approach to tackle the *cold start similar artists ranking* problem.

Music streaming services heavily rely on recommender systems to help users discover and enjoy new musical content within large catalogs of millions of songs, artists, and albums, with the general aim of improving their experience and engagement [39, 253, 319]. In particular, these services frequently recommend, on an artist's profile page, a ranked list of related artists that fans also listened to or liked [75, 170, 189]. Referred to as *"Fans Also Like"* on Spotify and Soundcloud and as *"Related"* or *"Similar Artists"* on Amazon Music, Apple Music and Deezer, such a feature typically leverages *learning to rank* models [175, 296, 319]. It retrieves the most relevant artists according to similarity measures usually computed from usage data, e.g., from the proportion of shared listeners across artists [75, 170], or from more complex *collaborative filtering* models

153

[165, 195, 319] that predict similarities from the known preferences of an artist's listeners. It has been described as *"one of the easiest ways"* to let *"users discover new music"* by Spotify [170].

However, filling up such ranked lists is especially challenging for new artists. Indeed, while music streaming services might have access to some general descriptive information on these artists, listening data will however not be available upon their first release. This prevents computing the aforementioned usage-based similarity measures. As a consequence of this problem, which we refer to as *cold start similar artists ranking*, music streaming services usually do not propose any *"Fans Also Like"* section for these artists, until (and if ever) a sufficiently large number of usage interactions, e.g., listening sessions, has been reached. Besides new artists, this usage-based approach also excludes from recommendations a potentially large part of the existing catalog with too few listening data, which raises fairness concerns [61]. Furthermore, while we will focus on music streaming applications, this problem encompasses the more general *cold start similar items ranking* issue, which is also crucial for media recommending other items such as videos [63].

In this chapter, we address this problem by exploiting the fact that, as detailed in Section 8.3, such *"Fans Also Like"* features can naturally be summarized as a directed and attributed *graph*, that connects each item *node*, e.g., each artist, to their most similar neighbors via directed *links*. Such a graph also incorporates additional descriptive information on nodes from the graph, e.g., musical information on artists. In this direction, we model cold start similar items ranking as a *directed link prediction* problem [311], for new nodes gradually added into this graph.

Then, we solve this problem by leveraging our recent advances from Chapter 5, and specifically our Gravity-Inspired GAEs and VGAEs. We propose a flexible framework which permits retrieving similar neighbors of new items from GAE/VGAE-based node embedding spaces, and where the gravity-inspired decoder acts as a ranking mechanism. Backed by in-depth experiments on industrial data from Deezer, we show the effectiveness of our approach at addressing a real-world cold start similar artists ranking problem, outperforming several popular baselines for cold start recommendation. We publicly released our code and the industrial data from our experiments.

This chapter is organized as follows. In Section 8.2, we introduce the cold start similar items ranking problem more precisely and mention previous works on related topics. In Section 8.3, we present our graph-based framework to address this problem. We report and discuss our experiments on Deezer's data in Section 8.4, and we conclude in Section 8.5.

8.2   Ranking Similar Artists/Items on Music Streaming Services

In this section, we introduce our ranking problem more precisely, and mention some related efforts to address cold start item recommendation problems. We voluntarily use a notation that heavily overlaps the one from the previous chapters, as the concepts introduced in this Section 8.2 will be transposed to graphs in the remainder of the chapter (e.g., n, which denotes
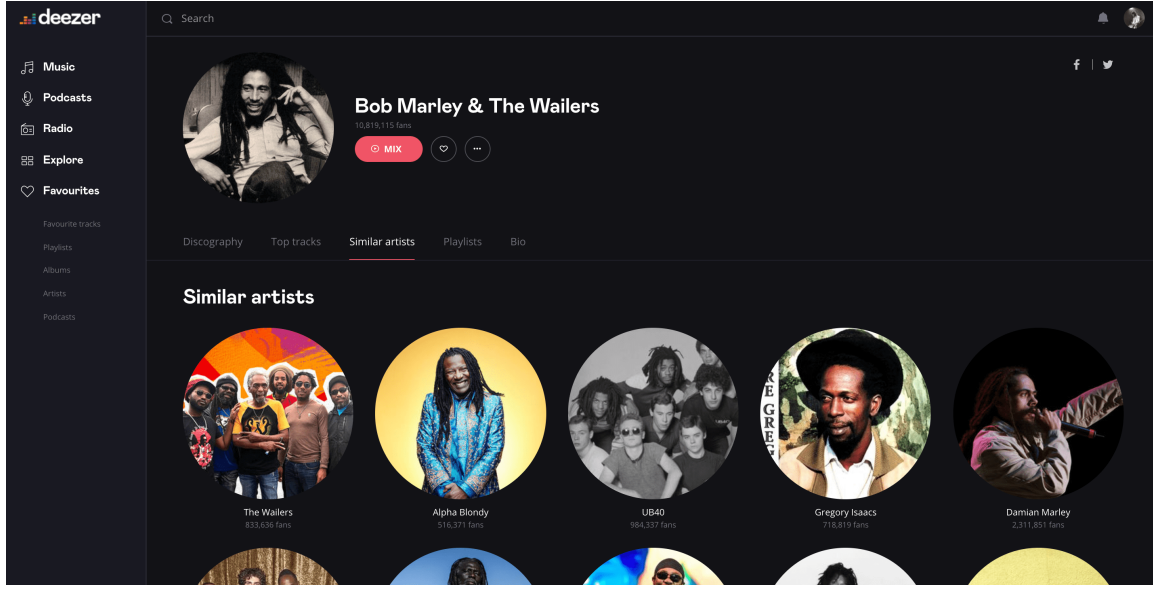
Figure 8.1: Some examples of "similar artists" recommended on the website version of Deezer. The mobile app proposes identical recommendations.

the number of items to recommend, will correspond to the number of nodes in Section 8.3).

8.2.1 Problem Formulation

Throughout this chapter, we consider a catalog of n recommendable items on an online service, such as music artists in our application. Each item i is described by some side information summarized in an f-dimensional vector x_i. For artists, such a vector could for instance capture information related to their country of origin or their music genres. These n items are assumed to be "*warm*", meaning that the service considers that a sufficiently large number of interactions with users, e.g., likes or streams, has been reached for these items to ensure reliable usage data analyses.

From these usage data, the service learns an $n \times n$ *similarity matrix* S, where the element $S_{ij} \in [0, 1]$ captures the similarity of item j w.r.t. item i. Examples of some possible usage-based similarity scores[1] S_{ij} include the percentage of users interacting with item i that also interacted with item j (e.g., users listening to or liking both items [170]), mutual information scores [324], or more complex measures derived from collaborative filtering [165, 195, 319]. Throughout this chapter, we assume that similarity scores are fixed over time, which we later discuss.

Leveraging these scores, the service proposes a *similar items* feature comparable to the *"Fans Also Like/Similar Items"* described in the introduction, and illustrated in Figure 8.1. Specifically, along with the presentation of an item i, it recommends a ranked list of k similar items to

---

[1] Details on the computation of similarities at Deezer are provided in Section 8.4.1 - without loss of generality, as our framework is valid for any $S_{ij} \in [0, 1]$.

users. They correspond to the top-k items j such as $j \neq i$ and with the highest scores S_{ij}.

On a regular basis, *"cold"* items will appear in the catalog. While the service might have access to descriptive side information on these items, *no usage data* will be available upon their first online release. This hinders the computation of usage-based similarity scores, thus excluding these items from recommendations until they become warm - if ever.

In this chapter, we study the feasibility of effectively predicting their future similar items ranked lists, from the delivery of these items, i.e., without any usage data. This would enable offering such an important feature quicker and on a larger part of the catalog. More precisely, we answer the following research question: *using only 1) the known similarity scores between warm items, and 2) the available descriptive information, how, and to which extent, can we predict the future "Fans Also Like" lists that would ultimately be computed once cold items become warm?*

8.2.2 Related Work

While collaborative filtering methods effectively learn item proximities, e.g., via the factorization of user-item interaction matrices [195, 362], these methods usually become unsuitable for cold items without any interaction data and thus absent from these matrices [362]. In such a setting, the simplest strategy for similar items ranking would consist in relying on *popularity* metrics [319], e.g., to recommend the most listened artists. In the presence of descriptive information on cold items, one could also recommend items with the closest descriptions [161]. These heuristics are usually outperformed by *hybrid* models, leveraging both item descriptions and collaborative filtering on warm items [137, 149, 362, 373]. They consist in:

- learning a vector space representation (an *embedding*) of warm items, where proximity aims to reflect user preferences;

- then, projecting cold items into this embedding, typically by learning a model to map descriptive vectors of warm items to their embedding vectors, and then applying this mapping to cold items' descriptive vectors.

Albeit under various formulations, this strategy has been transposed to Matrix Factorization [39, 362], Collaborative Metric Learning [149, 209] and Bayesian Personalized Ranking [23, 137]; in practice, a deep neural network often acts as the mapping model. The retrieved *similar items* are then the closest ones in the embedding. Other deep learning approaches were also recently proposed for item cold start, with promising performances. DropoutNet [367] processes both usage and descriptive data, and is explicitly trained for cold start through a dropout [343] simulation mechanism. MeLU (for Meta-Learned User preference estimator) [208] deploys a meta-learning paradigm to learn embeddings in the absence of usage data. CVAE (for Collaborative Variational Autoencoder) [222] leverages a Bayesian generative process to sample cold embedding

vectors, via a variational autoencoder.

While they constitute relevant baselines, these models do not rely on *graphs*, contrary to our work. Graph-based recommendation has recently grown at a fast pace (see the surveys of [375, 384]), including in industrial applications [373, 394]. Existing research widely focuses on bipartite user-item graphs [375]. Notably, STAR-GCN [399] addresses cold start by reconstructing user-item links using STAcked and Reconstructed GCNs, enhancing ideas from the GCN of Kipf and Welling [188] and a related extension for graph convolutional matrix completion by Berg et al. [28]. Instead, recent efforts [292] emphasized the relevance of leveraging - as we will - graphs connecting items together, along with their attributes. In this direction, the work closest to ours might be the recent DEAL (for Dual-Encoder graph embedding with ALignment) model by Hao et al. [133]. Thanks to an alignment mechanism, DEAL predicts links in such graphs for new nodes having only descriptive information. We will also compare to DEAL; we nonetheless point out that their work focused on undirected graphs, and did not consider ranking settings.

8.3 A Graph-Based Framework for Cold-Start Similar Items Ranking

In this section, we present our graph-based framework to address this cold start ranking problem.

8.3.1 Similar Items Ranking as a Directed Link Prediction Task

We argue that *"Fans Also Like"* features can naturally be summarized as a graph structure with n nodes and $n \times k$ edges. Nodes are warm recommendable items from the catalog, e.g., music artists with enough usage data according to the service's internal rules. Each item node points to its k most similar neighbors via a link, i.e., an edge. This graph is:

- *directed*: edges have a direction, leading to asymmetric relations. For instance, while most fans of a little known reggae band might listen to Bob Marley (Marley thus appearing among their similar artists), Bob Marley's fans will rarely listen to this band, which is unlikely to appear back among Bob Marley's own similar artists.

- *weighted*: among the k neighbors of node i, some items are more similar to i than others (hence the ranking). We capture this aspect by equipping each directed edge (i, j) from the graph with a *weight* corresponding to the similarity score S_{ij}. More formally, we summarize our graph structure by the $n \times n$ *adjacency matrix* A, where the element $A_{ij} = S_{ij}$ if j is one of the k most similar items w.r.t. i, and where $A_{ij} = 0$ otherwise[2].

---

[2]Alternatively, one could consider a *dense* matrix where $A_{ij} = S_{ij}$ for all pairs (i, j). However, besides acting as a data cleaning process on lowest scores, sparsifying A speeds up computations for the encoder introduced thereafter, whose complexity evolves linearly w.r.t. the number of edges (see Section 8.3.3).

- *attributed*: as explained in Section 8.2.1, each item i is also described by a vector $x_i \in \mathbb{R}^f$. In the following, and consistently with previous chapters, we denote by X the $n \times f$ matrix stacking up all descriptive vectors a.k.a. feature vectors from the graph, i.e., the i-th row of X is x_i.

Then, we model the release of a cold recommendable item in the catalog *as the addition of a new node* in the graph, together with its side descriptive vector. As usage data and similarity scores are unavailable for this item, it is *observed as isolated*, i.e., it does not point to k other nodes. In our framework, we assume that these k missing directed edges - and their weights - are actually *masked*. They point to the k nodes with the highest similarity scores, as would be identified by the service once it collects enough usage data to consider the item as warm, according to the service's criteria. These links and their scores, ultimately revealed, are treated as *ground truth* in the remainder of this work.

From this perspective, the cold start similar items ranking problem consists in a *directed link prediction* task [239, 318]. Specifically, we aim to predict the locations and weights - i.e., estimated similarity scores - of these k missing directed edges, and at comparing predictions with the actual ground truth edges ultimately revealed, both in terms of:

- *prediction accuracy:* do we retrieve the correct locations of missing edges in the graph?

- *ranking quality:* are the retrieved edges correctly ordered, in terms of similarity scores?

8.3.2 From Similar Items Graphs to Directed Node Embeddings

As explained in Chapter 2, locating missing links in graphs has been the objective of significant research efforts from various fields [225, 239, 318]. While this problem has been historically addressed via the construction of hand-engineered node similarity metrics [225], we explained in this same chapter that significant improvements were recently achieved by methods directly *learning* node representations summarizing the graph structure [128, 386]. This includes the GAE and VGAE models at the center of this thesis, representing each node i as a vector $z_i \in \mathbb{R}^d$ (with $d \ll n$) in a node embedding space where structural proximity should be preserved.

In this work, we build upon these advances and propose to learn node embeddings to tackle link prediction in our similar items graph. Specifically, we leverage our *Gravity-Inspired GAEs and VGAEs* from Chapter 5. These models are doubly advantageous for our application:

- foremost, they can effectively *process node attribute vectors* in addition to the graph, contrary to some popular alternatives such as DeepWalk [287] and standard Laplacian eigenmaps [25]. As we explain thereafter in Section 8.3.3, this will help us add some cold nodes, isolated but equipped with some descriptions, into an existing warm node

embedding. In other words, these models will be used in an *inductive* setting that involves generalizing representations to new unseen nodes after training;

- simultaneously, our Gravity-Inspired GAE and VGAE models were specifically designed to address *directed* link prediction from node embedding spaces, contrary to the aforementioned alternatives or to standard GAE and VGAE models from Kipf and Welling [187].

8.3.3   Cold Start Similar Items Ranking using Gravity-Inspired GAE/VGAE

In this section, we now explain how we build upon our Gravity-Inspired GAE and VGAE models to address the cold start similar items ranking problem. As VGAEs emerged as competitive alternatives to GAEs on some link prediction experiments, including some of those from Part II, we therefore saw value in considering both deterministic GAEs *and* VGAEs in this chapter. We assume a good understanding of these models, and we refer in particular to our previous Chapter 5 for a broader introduction to our Gravity-Inspired GAE and VGAE models.

Encoding Cold Nodes with GCNs   In this chapter, and consistently with Chapter 5, our encoders (both for Gravity-Inspired GAEs and VGAEs) will be 2-layer GCNs [188], incorporating the *out-degree normalized*[3] version of A denoted $\tilde{A}_{\text{out}}$ and a ReLU activation function [311]. Therefore, adopting the notation from Chapter 5, we have:

$$\tilde{Z} = \tilde{A}_{\text{out}}\text{ReLU}(\tilde{A}_{\text{out}}XW^{(0)})W^{(1)}, \tag{8.1}$$

in the GAE setting. We recall that the i-th row of $\tilde{Z} \in \mathbb{R}^{n \times (d+1)}$ is a $(d+1)$-dimensional vector $\tilde{z}_i$. The d first dimensions of $\tilde{z}_i$ correspond to the embedding vector z_i of node i; the last dimension corresponds to the mass $\tilde{m}_i$ as defined in Section 5.3.2.

Regarding the gravity-inpired VGAE model, and consistently with Chapter 5 once again, we have $\mu = \tilde{A}_{\text{out}}\text{ReLU}(\tilde{A}_{\text{out}}XW_\mu^{(0)})W_\mu^{(1)}$, $\log\sigma = \tilde{A}_{\text{out}}\text{ReLU}(\tilde{A}_{\text{out}}XW_\sigma^{(0)})W_\sigma^{(1)}$, and $\tilde{Z}$ is then sampled from μ and $\log\sigma$.

As all outputs are $n \times (d+1)$ matrices and X is an $n \times f$ matrix, then $W^{(0)}$ (or, $W_\mu^{(0)}$ and $W_\sigma^{(0)}$) is an $f \times d_{\text{hidden}}$ matrix, with d_{hidden} the hidden layer dimension, and $W^{(1)}$ (or, $W_\mu^{(1)}$ and $W_\sigma^{(1)}$) is a $d_{\text{hidden}} \times (d+1)$ matrix. These GCN weights are optimized from the graph of warm recommendable items, in a similar fashion w.r.t. Chapter 5, i.e., by iteratively optimizing a cross-entropy reconstruction loss (for Gravity-Inspired GAE) or an ELBO variational lower bound objective (for Gravity-Inspired VGAE), by gradient descent.

We rely on GCN encoders as they permit *incorporating new nodes*, attributed but isolated in the

---

[3] We recall that $\tilde{A}_{\text{out}} = (D_{\text{out}} + I_n)^{-1}(A + I_n)$ where I_n is the $n \times n$ identity matrix and D_{out} is the diagonal out-degree matrix of A.

graph, into an existing embedding. Indeed, let us consider a model already trained on a graph of warm items with some A and X, leading to optimized weights $W^{(0)}$ and $W^{(1)}$ for some GCN encoder. If $m \geq 1$ cold nodes appear, along with their f-dimensional descriptive vectors, then:

- A becomes A', an $(n+m) \times (n+m)$ adjacency matrix[4], with m new rows and columns filled with zeros;

- X becomes X', an $(n+m) \times f$ attribute matrix, concatenating X and the f-dimensional descriptions of the m new nodes;

- we derive embedding vectors and masses of new nodes through a *forward pass*[5] into the GCN previously trained on warm nodes, i.e., by computing the $(n+m) \times (d+1)$ new embedding matrix $\tilde{Z}' = \tilde{A}'_{\text{out}} \operatorname{ReLU}(\tilde{A}'_{\text{out}} X'W^{(0)})W^{(1)}$.

We emphasize that the choice of GCN encoders is made without loss of generality. Our framework remains valid for any inductive encoder processing new attributed nodes. In our experiments, 2-layer GCNs reached better or comparable results w.r.t. some considered alternatives, namely deeper GCNs, graph attention networks [364] and our linear encoders from Chapter 6.

Ranking Similar Items After projecting cold nodes into the warm embedding, we use the gravity-inspired decoder to predict their masked connections. More precisely, in our experiments, we add the hyperparameter $\lambda \in \mathbb{R}^+$ from Equation (5.16) for flexibility. The *estimated similarity weight* $\hat{A}_{ij}$ between some cold node i and another node j is thus:

$$\hat{A}_{ij} = \sigma(\ \underbrace{\tilde{m}_j}_{\text{influence of } j}\ -\lambda \times \underbrace{\log \|z_i - z_j\|_2^2}_{\text{proximity of } i \text{ and } j}\ ). \tag{8.2}$$

Then, the predicted top-k most similar items of i will correspond to the k nodes j with the highest estimated weights $\hat{A}_{ij}$.

As in Chapter 5, we interpret Equation (8.2) in terms of influence/proximity trade-off:

- the *influence* part of Equation (8.2) indicates that, if two nodes j and l are equally close to i in the embedding space (i.e., $\|z_i - z_j\|_2 = \|z_i - z_l\|_2$), then i will more likely points towards the node with the largest mass (i.e., the largest *"influence"*; we will compare these masses to popularity metrics in experiments from Section 8.4.3);

---

[4]Its $(n+m) \times (n+m)$ out-degree normalized version is $\tilde{A}'_{\text{out}} = (D'_{\text{out}} + I_{(n+m)})^{-1}(A' + I_{(n+m)})$, where D'_{out} is the diagonal out-degree matrix of A'.

[5]We note that such GCN forward pass is possible since dimensions of weight matrices $W^{(0)}$ and $W^{(1)}$ are *independent* of the number of nodes. In the case of featureless nodes, i.e., with $X = I_n$ as in Equation (2.20) from Chapter 2, the dimension of $W^{(0)}$ becomes $n \times d_{\text{hidden}}$ which therefore depends on n. This prevents computing forward passes in the presence of new nodes, thus making the GCN encoder *transductive* by design.

- the *proximity* part of Equation (8.2) indicates that, if j and l have the same mass (i.e., $\tilde{m}_j = \tilde{m}_l$), then i will more likely points towards its closest neighbor, which could, e.g., capture a closer musical similarity for artists.

As illustrated in Section 8.4.3, tuning λ will help us flexibly balance between these two aspects, and thus control for *popularity biases* [319] in our recommendations.

On Complexity   As extensively explained throughout this thesis, training GAE/VGAE models via full-batch gradient descent requires reconstructing the entire matrix $\hat{A}$ at each training iteration, which has an $O(dn^2)$ time complexity due to the evaluation of pairwise distances [311].

While we will follow such a full-batch training strategy in Section 8.4, our released code also implements the FastGAE method from Chapter 4 to approximate losses by decoding random subgraphs of $O(n)$ size. This permits scaling our method to large graphs with millions of nodes and edges. In our experiments, using FastGAE with degree-based sampling permits reducing training times from roughly 30 minutes to 1 minute on our machine and for the Deezer graph presented in Section 8.4.1, while preserving performances on the task presented in Section 8.4.2.

Moreover, projecting cold nodes in an embedding only requires a single forward GCN pass, with linear time complexity w.r.t. the number of edges [188, 310]. This is another advantage of using GCNs w.r.t. more complex encoders. Lastly, retrieving the top-k accelerations boils down to a nearest neighbors search in a $O(ndk)$ time, which could even be improved in our future research with approximate search methods [161].

8.4   Experimental Analysis

We now present the experimental evaluation of our graph-based framework on music artists data from the Deezer production system. We publicly released the private graph dataset used in these experiments on GitHub[6], as well as our source code for the experiments described thereafter. Besides making our results reproducible, such a release publicly provides a new benchmark dataset to the research community, permitting the evaluation of comparable graph-based recommender systems on real-world resources.

8.4.1   Ranking Similar Artists on Deezer

Dataset   We consider a directed graph of 24 270 artists with various musical characteristics (see below), extracted from the Deezer service. Each artist points towards $k = 20$ other artists. They correspond, up to internal business rules, to the top-20 artists from the same graph that

---

[6]https://github.com/deezer/similar_artists_ranking

would be recommended by our production system on top of the *"Fans Also Like/Similar Artists"* feature illustrated in Figure 8.1. Each directed edge (i, j) has a weight A_{ij} normalized to lie in the $[0, 1]$ set; for unconnected pairs, $A_{ij} = 0$. It corresponds to the similarity score of artist j w.r.t. i, computed on a weekly basis from usage data of millions of Deezer users. More precisely, weights are based on *mutual information* scores [324] from *artist co-occurrences among streams*. Roughly, they compare the probability that a user listens to the two artists, to their global listening frequencies on the service, and they are normalized at the artist level through internal heuristics and business rules (some details on exact score computations are voluntarily omitted for confidentiality reasons). In the graph, edges correspond to the 20 highest scores for each node. In general, $A_{ij} \neq A_{ji}$. In particular, j might be the most similar artist of i while i does not even appear among the top-20 of j.

We also have access to descriptions of these artists, either extracted through the musical content or provided by record labels. Here, each artist i will be described by an attribute vector x_i of dimension $f = 56$, concatenating:

- a 32-dimensional *genre* vector. Deezer artists are described by music genres [83], among more than 300. 32-dimensional embedding vectors are learned from these genres, by factorizing a co-occurrence matrix based on listening usages with SVD [196]. Then, the genre vector of an artist is the average of embedding vectors of his/her music genres.

- a 20-dimensional *country* vector. It corresponds to a one-hot encoding vector, indicating the country of origin of an artist, among the 19 most common countries on Deezer, and with a 20$^{\text{th}}$ category gathering all other countries.

- a 4-dimensional *mood* vector. It indicates the average and standard deviations of the *valence* and *arousal* scores across an artist's discography. In a nutshell, valence captures whether each song has a positive or negative mood, while arousal captures whether each song has a calm or energetic mood [70, 305]. These scores are computed internally, from audio data and using a deep neural network inspired by the work of Delbouys et al. [70].

While some of these features are quite general, we emphasize that the actual Deezer app also gathers more refined information on artists, e.g., from audio or textual descriptions. They are undisclosed and unused in these experiments.

Problem　We consider the following similar artists ranking problem. Artists are split into a training set, a validation set, and a test set gathering 80%, 10%, and 10% of artists, respectively. The training set corresponds to warm artists. Artists from the validation and test sets are the cold nodes: their edges are *masked*, and they are therefore observed as isolated in the graph. Their 56-dimensional descriptions are available.

We evaluate the ability of our models at retrieving these edges, with correct weight ordering. As a measure of *prediction accuracy*, we will report *Recall@K* scores. They indicate, for various K, which proportion of the 20 ground truth similar artists appear among the top-K artists with the highest estimated weights. Moreover, as a measure of *ranking quality*, we will also report the widely used *Mean Average Precision at K (MAP@K)* and *Normalized Discounted Cumulative Gain* at K (*NDCG@K*) scores[7].

8.4.2 List of Models and Baselines

We now describe all methods considered in our experiments. All embedding vectors have $d = 32$, which we will discuss. Also, all hyperparameters mentioned thereafter were tuned by optimizing NDCG@20 scores on the validation set.

Gravity-Inspired GAE/VGAE We follow our framework from Section 8.3 to embed cold nodes. For both GAE and VGAE, we use 2-layer GCN encoders with a 64-dimensional hidden layer, and a 33-dimensional output layer (i.e., a 32-dimensional z_i vectors, plus the mass), trained for 300 epochs. We use the Adam optimizer [184], with a learning rate of 0.05, without dropout, performing full-batch gradient descent, and using the reparameterization trick [185] for VGAE. We set $\lambda = 5$ in the decoder of Equation (8.2) and discuss the impact of λ thereafter. Our adaptation of these models builds upon the Tensorflow code that we developed for the experiments of Chapter 5, which is publicly available on GitHub[8].

Other Methods based on the Directed Artist Graph We compare our Gravity-Inspired GAE and VGAE models to the *standard GAE and VGAE* models from Kipf and Welling [187], with a similar setting as above. These models use symmetric inner product decoders, i.e., $\hat{A}_{ij} = \sigma(z_i^T z_j)$, therefore ignoring directionalities. Moreover, we implement the *Source-Target GAE and VGAE* models used as a baseline in Chapter 5. We recall that these models are similar to standard GAE and VGAE, except that they decompose the 32-dimensional z_i vectors into a *source* vector $z_i^{(s)} = z_{i[1:16]}$ and a *target* vector $z_i^{(t)} = z_{i[17:32]}$, and then decode edges as follows: $\hat{A}_{ij} = \sigma(z_i^{(s)T} z_j^{(t)})$ and $\hat{A}_{ji} = \sigma(z_j^{(s)T} z_i^{(t)})$ ($\neq \hat{A}_{ij}$ in general). They reconstruct directed links, as gravity-inspired models, and are therefore relevant baselines for our evaluation. Lastly, we also test the recent DEAL model [133] mentioned in Section 8.2.2, and designed for inductive link prediction on new isolated but attributed nodes. We used the authors' PyTorch implementation, with similar attribute and structure encoders, alignment mechanism, loss and cosine predictions as their original model [133].

---

[7]MAP@K and NDCG@K are computed as in Equation (4) of [319] (averaged over all cold artists) and in Equation (2) of [378] respectively.

[8]https://github.com/deezer/gravity_graph_autoencoders

Other Baselines In addition, we compare our framework to four popularity-based baselines. *Popularity* recommends the K most popular[9] artists on Deezer (with K as in Section 8.4.1); *Popularity by Country* recommends the K most popular artists from the country of origin of the cold artist; *In-Degree* recommends the K artists with the highest in-degrees in the graph, i.e., sum of weights pointing to them; lastly, *In-Degree by Country* proceeds as *In-Degree*, but on warm artists from the country of origin of the cold artist.

We also consider three baselines only or mainly based on descriptions x_i and not on usage data. Firstly, *K-NN* recommends the K artists with closest x_i vectors, from a nearest neighbors search with Euclidean distance. Besides, *K-NN + Popularity* and *K-NN + In-degree* retrieve the 200 artists with closest x_i vectors, then recommends the K most popular ones among these 200 artists, ranked according to popularity and in-degree values respectively.

We also implement *SVD+DNN*, which follows the *"embedding+mapping"* strategy from Section 8.2.2 by 1) computing an SVD [196] of the warm artists similarity matrix, learning 32-dimensional z_i vectors, 2) training a 3-layer neural network (with layers of dimension 64, 32 and 32, trained with Adam [184] and a learning rate of 0.01) to map warm x_i vectors to z_i vectors, and 3) projecting cold artists into the SVD embedding through this mapping.

Lastly, among deep learning approaches from Section 8.2.2 (CVAE, DropoutNet, MeLU, STAR-GCN), we report results from the two best methods on our dataset, namely *DropoutNet* [367] and *STAR-GCN* [399], using the authors' implementations with careful fine-tuning on validation artists[10]. Similar artists ranking is done via a nearest neighbors search in embedding spaces.

8.4.3 Results and Discussion

Performances Table 8.1 reports mean performance scores for all models, along with standard deviations over 20 runs for models with randomness due to weights initialization in GCNs or neural networks. Popularity and In-degree appear as the worst baselines. Their scores significantly improve by focusing on the country of origin of cold artists (e.g., with a Recall@100 of 12.38% for Popularity by country, v.s. 0.44% for Popularity).

Besides, we observe that methods based on a direct K-NN search from x_i attributes are outperformed by the more elaborated cold start methods leveraging both attributes and warm usage data. In particular, DropoutNet, as well as the graph-based DEAL, reach stronger results than SVD+DNN and STAR-GCN. They also surpass standard GAE and VGAE (e.g., with a +6.46 gain in average NDCG@20 score for DEAL w.r.t. GAE), but not the GAE/VGAE ex-

---

[9]Our dataset includes the *popularity* rank (from 1st to n^{th}) of warm artists. It is out of the x_i vectors, as it is usage-based and thus unavailable for cold artists.

[10]These last models do not process similar artists graphs, but raw *user-item usage data*, either as a bipartite user-artist graph or as an interaction matrix. While Deezer can not release such fine-grained data, we nonetheless provide embedding vectors from these baselines along with our code, to reproduce our scores.

8.4. Experimental Analysis

Table 8.1: Cold start similar artists ranking with Gravity-Inspired GAE and VGAE and with all baselines. Performances are computed on test set, and averaged over 20 runs. All embedding methods verify $d = 32$. **Bold** numbers correspond to the best scores.

Method $(d = 32)$	Recall@K (in %)			MAP@K (in %)			NDCG@K (in %)		
	$K = 20$	$K = 100$	$K = 200$	$K = 20$	$K = 100$	$K = 200$	$K = 20$	$K = 100$	$K = 200$
Popularity	0.02	0.44	1.38	¡0.01	0.03	0.12	0.01	0.17	0.44
Popularity by country	2.76	12.38	18.98	0.80	3.58	6.14	2.14	6.41	8.76
In-degree	0.91	3.43	6.85	0.15	0.39	0.86	0.67	1.69	2.80
In-degree by country	5.46	16.82	23.52	2.09	5.43	7.73	5.00	10.19	12.64
K-NN on x_i	4.41	13.54	19.80	1.14	3.38	5.39	4.29	8.83	11.22
K-NN + Popularity	5.73	15.87	19.83	1.66	4.32	5.74	4.86	10.03	11.76
K-NN + In-degree	7.49	17.29	18.76	2.78	5.60	6.18	7.41	12.48	13.14
SVD + DNN	6.42 ± 0.96	21.83 ± 1.21	35.01 ± 1.41	2.25 ± 0.67	6.36 ± 1.19	11.52 ± 1.98	6.05 ± 0.75	12.91 ± 0.92	17.89 ± 0.95
STAR-GCN	10.03 ± 0.56	31.45 ± 1.09	43.92 ± 1.10	3.10 ± 0.32	10.64 ± 0.54	16.62 ± 0.68	10.07 ± 0.40	21.17 ± 0.69	25.99 ± 0.75
DropoutNet	12.96 ± 0.54	37.59 ± 0.76	49.93 ± 0.82	4.18 ± 0.30	13.61 ± 0.55	20.12 ± 0.67	13.12 ± 0.68	25.61 ± 0.72	30.52 ± 0.78
DEAL	12.80 ± 0.52	37.98 ± 0.59	50.75 ± 0.72	4.15 ± 0.25	14.01 ± 0.44	20.92 ± 0.54	12.78 ± 0.53	25.70 ± 0.62	30.69 ± 0.70
GAE	7.30 ± 0.51	25.92 ± 0.95	40.37 ± 1.11	2.81 ± 0.29	7.97 ± 0.47	14.24 ± 0.67	6.32 ± 0.39	15.54 ± 0.66	20.94 ± 0.72
VGAE	10.01 ± 0.52	34.00 ± 1.06	49.72 ± 1.14	3.53 ± 0.27	11.68 ± 0.52	19.46 ± 0.70	10.09 ± 0.58	21.37 ± 0.73	27.31 ± 0.75
Sour.-Targ. GAE	12.21 ± 1.30	39.52 ± 3.53	56.25 ± 3.57	4.62 ± 0.81	14.67 ± 2.33	23.60 ± 2.85	12.42 ± 1.39	25.45 ± 3.37	31.80 ± 3.38
Sour.-Targ. VGAE	13.52 ± 0.64	42.68 ± 0.69	59.51 ± 0.76	5.19 ± 0.31	16.07 ± 0.40	25.48 ± 0.55	13.60 ± 0.73	27.81 ± 0.56	34.19 ± 0.59
Gravity GAE	**18.33 ± 0.45**	**52.26 ± 0.90**	**67.85 ± 0.98**	**6.64 ± 0.25**	**21.19 ± 0.55**	**30.67 ± 0.68**	**18.64 ± 0.47**	**35.77 ± 0.66**	**41.42 ± 0.68**
Gravity VGAE	16.59 ± 0.50	49.51 ± 0.78	65.70 ± 0.75	5.66 ± 0.35	19.07 ± 0.57	28.66 ± 0.59	16.74 ± 0.55	33.34 ± 0.66	39.29 ± 0.64

tensions that explicitly model edges directionalities, i.e., source-target GAE/VGAE and, even more, Gravity-Inspired GAE/VGAE, that provide the best recommendations. It emphasizes the effectiveness of our framework, both in terms of prediction accuracy (e.g., with a top 67.85% average Recall@200 for Gravity-Inspired GAE) and of ranking quality (e.g., with a top 41.42% average NDCG@200 for this same method).

Moreover, while all embedding methods from Table 8.1 verify $d = 32$, we point out that gravity-inspired models remained superior on our tests with $d = 64$ and $d = 128$. Also, VGAE methods tend to outperform their deterministic counterparts for standard and source-target models, while the contrary conclusion emerges on gravity-inspired models. This confirms the value of considering both settings when testing these models on novel applications.

On the mass parameter In Figure 8.2, we visualize some artists and their estimated masses. At first glance, one might wonder whether nodes i with the largest masses $\tilde{m}_i$, as Bob Marley in Figure 8.2, simply correspond to the most popular artists on Deezer. Table 8.2 shows that, while masses are indeed positively correlated to popularity and to various graph-based node importance measures, these correlations are not perfect, which highlights that our models do not exactly learn any of these metrics.

Furthermore, as in Chapter 5, replacing masses $\tilde{m}_i$ by any of these measures during training (i.e., by optimizing z_i vectors with the mass of i being fixed, e.g., as its PageRank [278] score) diminishes performances (e.g., more than -6 points in NDCG@200, in the case of PageRank), which confirms that jointly learning embedding vectors and masses is optimal.

Lastly, qualitative investigations also tend to reveal that masses, and thus relative node attractions, vary across the location in the graph. Local influence correlates with popularity but is also impacted by various culture or music-related factors such as countries and genres. As an
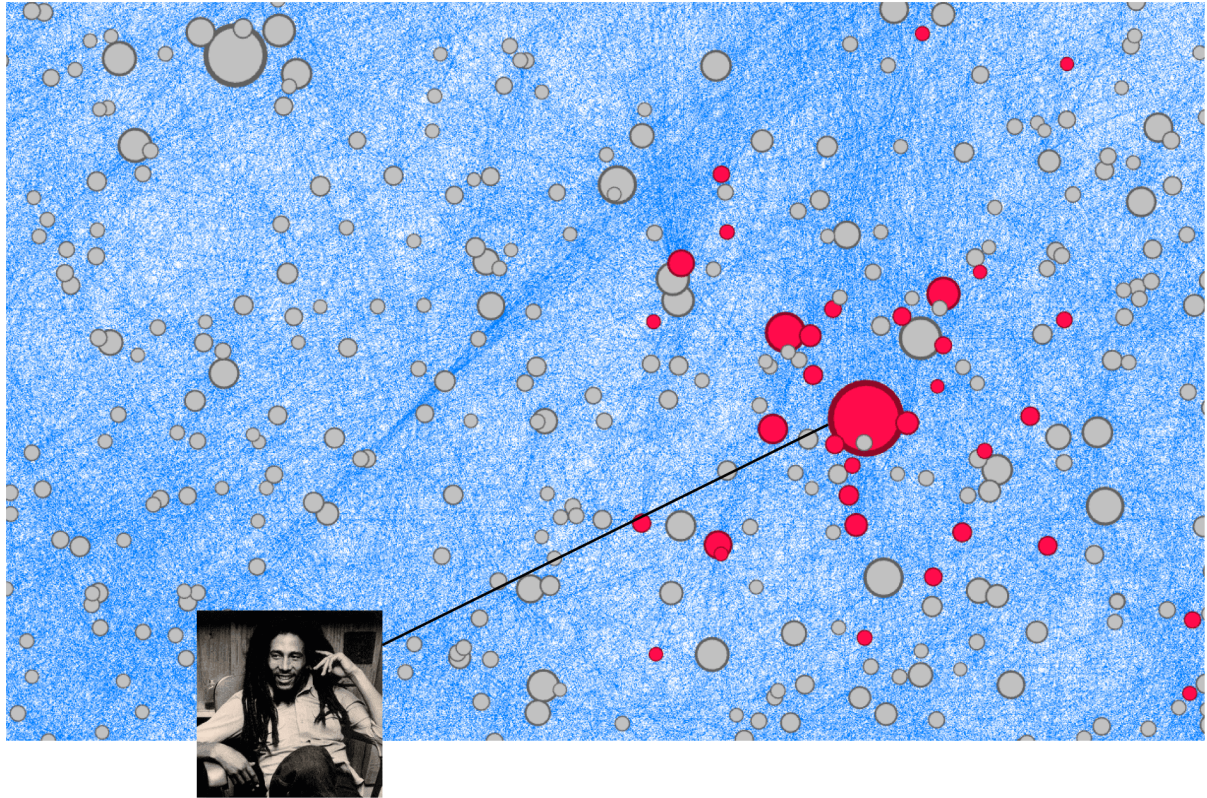
Figure 8.2: Visualization of embedding representations of some music artists from the Gravity-Inspired GAE model. Nodes are scaled using masses $\tilde{m}_i$, and node separation is based on distances in the embedding, using multidimensional scaling and with [100]. Red nodes correspond to *Jamaican reggae* artists, appearing in the same neighborhood.

illustration, the successful samba/pagode Brazilian artist Thiaguinho, out of the top-100 most popular artists from our training set, has a larger mass than American pop star Ariana Grande, appearing among the top-5 most popular ones. While numerous Brazilian pagode artists point towards Thiaguinho, American pop music is much broader and all pop artists do not point towards Ariana Grande despite her popularity.

Popularity/diversity trade-off The gravity-inspired decoder from Equation (8.2) enables us to flexibly address popularity biases when ranking similar artists. More precisely:

- setting $\lambda \to 0$ increases the relative importance of the *influence* part of equation (8.2). Thus, the model will highly rank the most massive nodes. As illustrated in Figure 8.3, this results in recommending *more popular* music artists;

- on the contrary, increasing λ diminishes the relative importance of masses in predictions, in favor of the actual node *proximity*. As illustrated in Figure 8.3, this tends to increase the recommendation of *less popular* content.

Table 8.2: Pearson and Spearman correlation coefficients of masses $\tilde{m}_i$ from the Gravity-Inspired GAE model, w.r.t. artist-level reversed popularity ranks (i.e., the higher the more popular) on Deezer and to three node importance measures: the in-degree (i.e., the sum of edges coming into the node), the betweenness centrality [311] and the PageRank [278]. Coefficients were computed on the training set.

| Node-level measures	Pearson correlation	Spearman correlation
Popularity Rank	0.208	0.206
Popularity Rank by Country	0.290	0.336
In-degree Centrality	0.201	0.118
Betweenness Centrality	0.109	0.079
PageRank Score	0.272	0.153
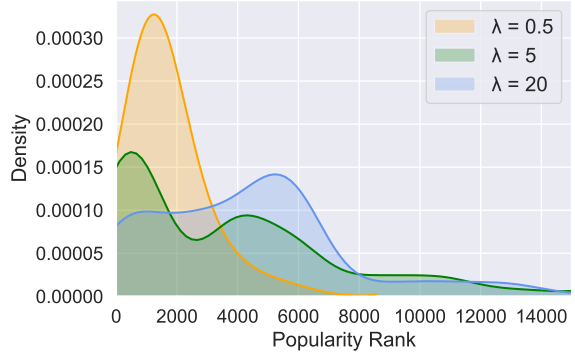
Figure 8.3: Popularity rank of the *most popular* artist recommended to each test artist, among their top-20. Distributions obtained from Gravity-Inspired GAE models with varying hyperparameters λ.

Setting $\lambda = 5$ leads to optimal scores in our application (e.g., with a 41.42% NDCG@200 for our AE, v.s. 35.91% and 40.31% for the same model with $\lambda = 1$ and $\lambda = 20$ respectively). Balancing between popularity and diversity is often desirable for industrial-level recommender systems [319]. Gravity-inspired decoders flexibly permit such a balancing.

Impact of attributes So far, all models processed the complete 56-dimensional attribute vectors x_i, concatenating information on music genres, countries, and moods.

In Figure 8.4, we assess the actual impact of each of these descriptions on performances, for our Gravity-Inspired GAE. Assuming only one attribute (genres, countries *or* moods) is available during training, genres-aware models return the best performances, in terms of MAP, NDCG, and Recall scores.

Moreover, we observe that adding moods to country vectors leads to larger gains than adding moods to genre vectors. This could reflect how some of our music genres, such as *speed metal*, already capture some valence or arousal characteristics. Lastly, Figure 8.4 confirms that gathering all three descriptions provides the best performances, corresponding to those reported in Table 8.1.
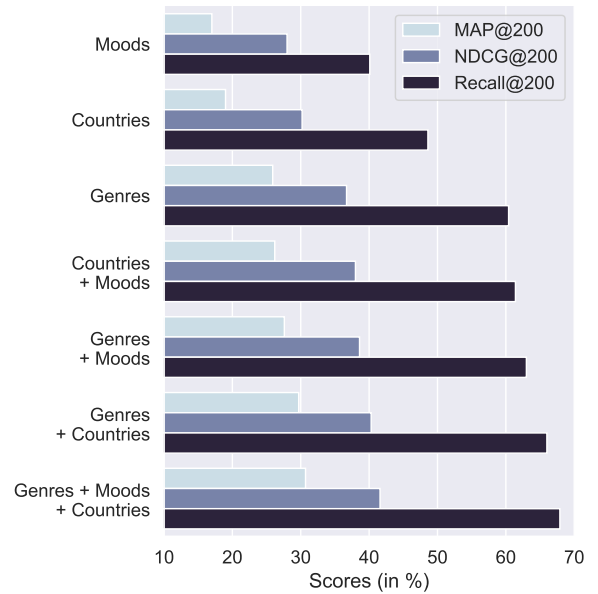

Figure 8.4: Performances of Gravity-Inspired GAE, when only trained from subsets of artist-level attributes, among countries, music genres and moods.

Possible Improvements (on Models)   Despite promising results in this chapter, assuming fixed similarity scores over time might sometimes be unrealistic, as some user preferences could actually evolve. Capturing such changes, e.g., through *dynamic graph embeddings*, might permit providing even more refined recommendations. Also, during training we kept $k = 20$ edges for each artist, while one could consider varying k at the artist level, i.e., adding more (or fewer) edges, depending on the actual musical relevance of each link. One could also compare different metrics, besides mutual information, when constructing the ground truth graph. Lastly, we currently rely on a single GCN forward pass to embed cold artists which, while being fast and simple, might also be limiting. Future studies on more elaborated approaches, e.g., to incrementally update GCN weights when new nodes appear, could also improve our framework.

Possible Improvements (on Evaluation)   Our evaluation focused on the prediction of *ranked lists for cold artists*. This permits filling up their *"Fans Also Like/Similar Artists"* sections, which was our main goal in this chapter and of the paper associated with this work [313]. On the other hand, future internal investigations could also aim to measure to which extent the inclusion of new nodes in the embedding space impacts the existing *ranked lists for warm artists*. Such an additional evaluation, e.g., via an online test on Deezer, could assess which cold artists actually enter these lists, and whether the new recommendations 1) are more diverse, according to some music or culture-based criteria, and 2) improve user engagement on the service.

8.5   Conclusion

In this chapter, we modeled the challenging cold start similar items ranking problem as a link prediction task, in a directed and attributed graph summarizing information from *"Fans Also Like"* features. We presented an effective framework to address this task, transposing our recent advances on Gravity-Inspired GAE and VGAE models to recommender systems.

Backed by in-depth experiments on artists available on the Deezer service, we emphasized the practical benefits of our approach, both in terms of recommendation accuracy, ranking quality, and flexibility. As a consequence, we plan to A/B test such models in production on the actual Deezer app in 2022, in combination with the FastGAE method from Chapter 4 to address scalability concerns.

Last, but not least, along with the paper associated with this work [313], we publicly released our source code, as well as the industrial data from our experiments. We hope that this release of industrial resources will benefit future research on graph-based cold start recommendation. In particular, we already identified several directions that, in future studies, should lead to the improvement of our approach.

9

Music Graph Embedding and Clustering for Recommendation

This chapter provides a broader overview of how Deezer internally leverages similarity graphs for music recommendation. It presents and discusses practical (ongoing and future) applications of GAE and VGAE models.

9.1   Introduction

In Chapter 8, we focused on quite specific top-k directed graphs. In this Chapter 9, we now give a more general overview of how Deezer leverages other pre-processed *similarity graphs* to recommend musical content to users on the service. Unlike the previous chapters of this thesis, the work presented below is not associated with a research publication. It rather aims to discuss more practical industrial applications and, as a consequence, some technical details on production-facing algorithms will be voluntarily omitted for confidentiality reasons.

Firstly, in Section 9.2, we explain how these similarity graphs have historically played a central role in Deezer's recommender systems, to detect communities of similar artists. Then, in Section 9.3, we show the benefits of adopting GAEs and VGAEs, integrating our advances from previous chapters. We focus on community detection in graphs of artists and albums with recommendation purposes, through offline experiments on data extracted from Deezer. At the end of Section 9.3, we also mention several ongoing and future plans to A/B test these methods, but also to extend them to other graphs, e.g., to graphs of music tracks and of users, as well as to culture-specific graphs and to dynamic graphs. We conclude in Section 9.4.
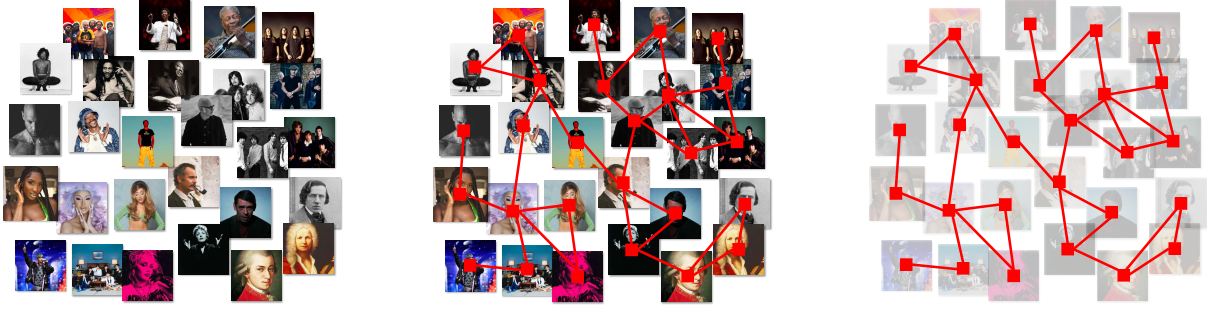
Figure 9.1: An example of a similar artists graph. While connections between artists are not naturally given (*left*), a graph of similar artists can be constructed, e.g., by connecting artists that are simultaneously listened to or liked by numerous users (*middle*). This graph can be used as an abstraction (*right*), e.g., for music recommendation.

9.2 Music Graph Representations at Deezer

In this section, we describe how, at the beginning of this PhD project, similar *artists* graphs were already used at Deezer to recommend music to millions of users.

9.2.1 Connecting Artists based on Usage Data

Deezer historically leveraged two types of similar artists graphs computed from usage data:

- the first ones correspond to the top-k *directed* graphs previously described in Section 8.4 from Chapter 8. We recall that, in such graphs, each node is an artist, pointing towards its k most similar neighbors (for a pre-selected k, e.g., $k = 20$ in Chapter 8, and up to $k = 150$ in Deezer's internal databases) in terms of co-listening data. As described in this previous chapter, each directed edge (i, j) has a weight corresponding to a "similarity score" for the artist j w.r.t. the artist i;

- the second ones are refined versions of the above top-k directed graphs. The Deezer team performs several pre-processing operations, consisting in adding or removing edges between artists from these graphs. The general objective of these operations is to facilitate the use of these graphs for recommendation purposes and, in particular, to ease the extraction of homogeneous artist communities (see Section 9.2.2). Among others, several internal heuristics permit consolidating the existing links through external information, e.g., by using "likes" or the country of origin of each artist. Moreover, some pre-processing steps aim to limit popularity biases [319] and to ensure that node degrees are relatively homogeneous[1]. The exact procedure will not be detailed in this thesis. We nonetheless emphasize that the graphs resulting from these operations are *undirected*, contrary to those from Chapter 8 but consistently with the illustrative example reported in the above Figure 9.1.

---

[1]This objective contrasts with the ranking problem from Chapter 8 where, on the contrary, having nodes of various degrees was desirable and was (explicitly or implicitly) leveraged by several methods considered in this chapter.
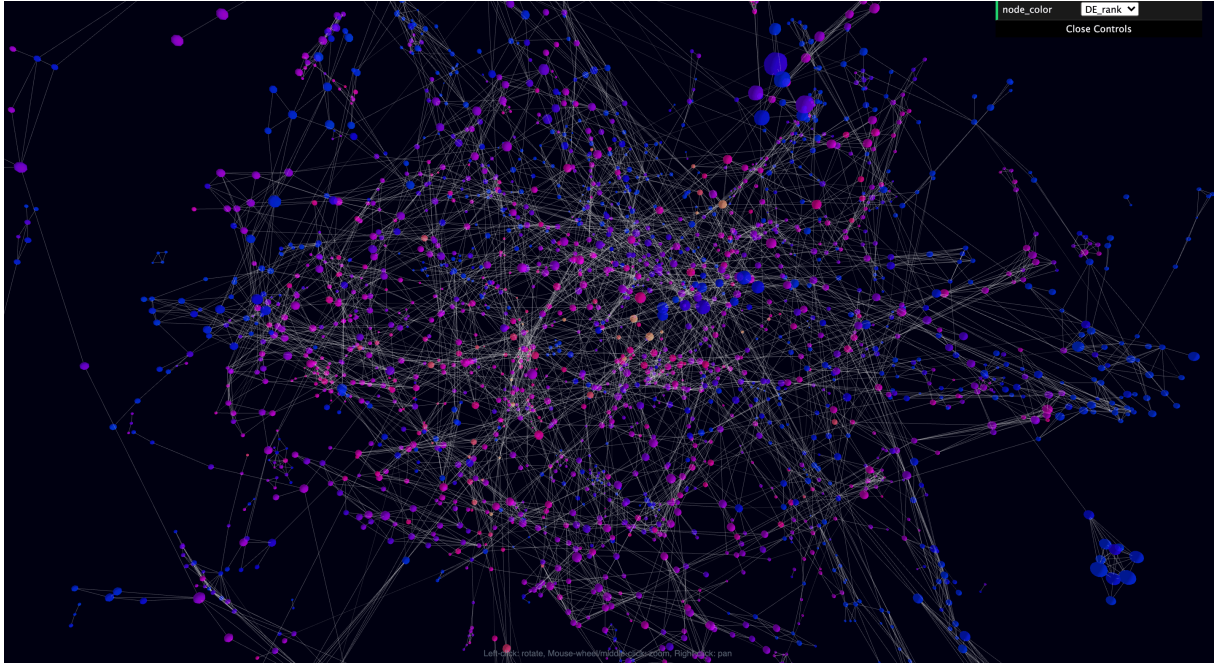
Figure 9.2: Visualization of a similar artists graph, computed internally at Deezer and based on the "3D Force-Directed Graph" visualization tool, using the ThreeJS and WebGL JavaScript libraries and available on GitHub: https://github.com/vasturiano/3d-force-graph. By clicking on a node, one can display information about the corresponding artist. In this figure, colors denote the popularity rank of each artist on Deezer, in Germany.

These graphs are computed and updated on a weekly basis, from the usage data of millions of active users on Deezer. While the catalog includes millions of artists[2], at the beginning of this PhD project the Deezer team had only considered smaller versions of these graphs, with roughly 100 000 nodes corresponding to the most popular artists. Such a restriction was due to scalability constraints, but also to the desire to focus on the most reliable data for the recommendation applications described in the next sections. Figures 9.2 and 9.3 provide some illustrative 3D visualizations of such similar artists graphs (specifically, of the refined undirected graphs described in the second bullet point of the previous page).
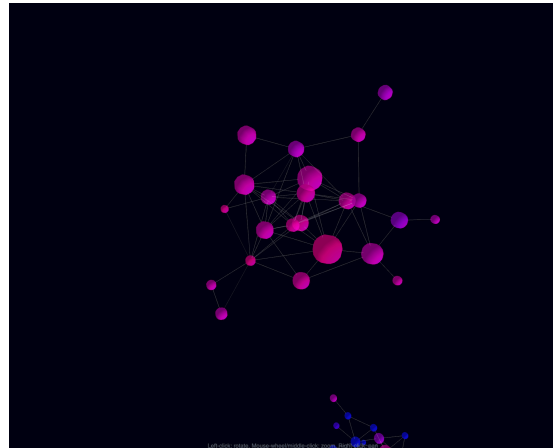
Figure 9.3: Zoom on an isolated group of interconnected artists, extracted from the similar artists graphs visualized in Figure 9.2. All nodes from this group correspond to Turkish pop-folk artists, such as Tarkan and Yavuz Bingöl. They are connected in the graph under consideration as they are simultaneously listened to and liked by the same users on Deezer.

[2]While Deezer publicly advertises a catalog of 73 million tracks, the exact number of artists on the service remains a private and undisclosed information at the time of writing (i.e., in late 2021).

9.2.2 Artist Community Detection for Music Recommendation

The undirected graphs described in Section 9.2.1 have historically played a central role in Deezer's recommender systems. At the time of writing, the Deezer team still leverages these graphs to extract clusters of similar artists using graph-based community detection methods, with the general aim of providing recommendations based on usage data. The intuition behind this strategy is the following: if some users already listened to several artists from a cluster (i.e., a community), then other unlistened artists from this same cluster could be recommended to them.

For instance, graph-based artist communities are a key component of Deezer's *"Flow"* feature. As illustrated in Figure 9.4, the Flow takes the appearance of a button on the homepage. A click on this button launches a virtually infinite playlist mixing the user's favorite music tracks with some recommended new tracks. Tracklists are generated by a recommendation algorithm processing the aforementioned artist communities, through a pipeline involving several operations and other data sources. While technical details on the Flow are omitted in this thesis, we emphasize that previous internal investigations at Deezer revealed the benefits of relying on graph-based methods to learn these communities, w.r.t.
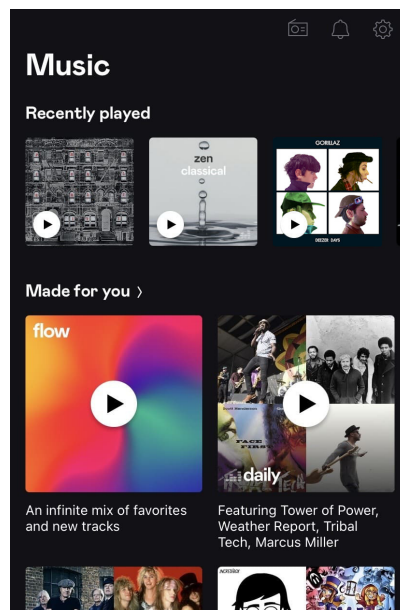
Figure 9.4: The "Flow" feature on Deezer, along with other daily mixes of recommended music tracks.

more standard methods, e.g., some strategies consisting in clustering artists from some usage-based input features via a k-means algorithm[3]. Specifically, at Deezer, artist communities are currently computed using the Louvain algorithm [31], that we presented in Chapter 7.

Such a choice has several advantages. As explained in Chapter 7, the Louvain algorithm is relatively fast and scalable, as it runs in $O(n \log n)$ time [31]. Besides, its iterative modularity maximization procedure permits retrieving a hierarchical clustering of artists. This is convenient as Deezer often wants to merge or, on the contrary, to split some artist communities in post-processing steps, for recommendation purposes. Using the Louvain algorithm for artist community detection also suffers from several limitations, which will be presented and discussed in Section 9.3. In Figure 9.5, we provide an example of an artist community, visualized through an internal tool developed by data scientists from Deezer's Recommendation team.

---

[3]We refer to the examples of Figure 2.4 from Chapter 2 for an illustration of how graph-based methods (specifically, spectral clustering, in this figure) can leverage the "connectivity" between data points and identify community structures in settings where a standard k-means would fail.

Figure 9.5: An example of an artist community, computed at Deezer using the Louvain algorithm [31] on a similar artists graph, and visualized through an internal tool developed by our Recommendation team. This community mainly gathers DJs and "house music" artists. This internal tool also generates a playlist mixing music tracks from artists of the community under consideration, using algorithms that we do not describe in this thesis.

9.2.3 Graphs of Albums, Users, Songs

At the beginning of this PhD project, Deezer had mainly studied graphs of similar *artists*. Nonetheless, we emphasize that the general approach from Section 9.2.1, consisting in connecting artists based on usage data (e.g., artists simultaneously listened to by numerous users), can be extended to create other graphs. For instance, one could consider different granularity levels from the musical catalog, and aim to create usage-based graphs of similar *albums* or similar *music tracks*. One could also study graphs of similar *users*, e.g., by connecting users with the most similar listening histories on the service. As explained in Chapter 1, Deezer users can also "follow" each other on the service, hence creating a large *social graph*. These two graphs could be used in conjunction, e.g., to derive graph-based communities of similar users. While we did not examine such an approach so far, it constitutes a relevant direction for future work[4].

9.3 Improving Music Recommendation with GAE and VGAE

In this section, we study the potential benefits of leveraging the GAE and VGAE models developed in Part II of this thesis for music recommendation.

---

[4]We refer in particular to the two projects presented in Chapters 11 and 12. Both of them will directly rely on clusters of users, and could therefore benefit from such graph-based user communities.

9.3.1 From Music Graphs to Node Embedding Spaces

We consider the replacement of the Louvain algorithm by a GAE/VGAE-based detection of communities in the similarity graphs described in Section 9.2. In addition to obtaining communities, running GAE and VGAE models instead of the Louvain algorithm would permit learning low-dimensional node embedding representations of the musical entities under consideration (e.g., of artists or albums). These representations could act as feature vectors for these entities in other machine learning problems, and be helpful for data visualization.

More importantly, the standard Louvain algorithm only relies on the graph structure to extract communities, which is limiting as our similarity graphs are often enriched by node-level descriptions. For instance, in the graph of Chapter 8, artists were also described by their own vectors providing information on their music genres, moods, and countries of origin.

On the other hand, as explained in previous chapters, GAE and VGAE models can process such node *attributes* a.k.a. node *features*. Chapter 8 already emphasized how GAE and VGAE models can effectively learn artist embeddings that simultaneously capture usage (through edges in the graph) and musical (through node features) information on artists. In this section, we now aim to analyze to which extent this aspect could also improve the detection of communities of artists, or of other musical entities, and enhance Deezer's recommender systems.

9.3.2 Experimental Setting

In this direction, we now present the experimental setting of some offline community detection experiments, conducted on data extracted from Deezer's production system.

Task and Datasets   As is the case for the previous community detection experiments from this thesis, our task will consist in extracting a partition of nodes from a graph into K communities, for a given value of K. We consider two *undirected* graphs:

- a similar *artists* graph, including $n = 77\ 656$ artists from the Deezer catalog, connected via $m = 377\ 591$ edges. This graph was computed in September 2021 from usage data of millions of active users, and using the pre-processing steps from Section 9.2.1;

- a larger similar *albums* graph, including $n = 2\ 503\ 985$ albums from the Deezer catalog, connected via $m = 25\ 039\ 155$ edges. This graph was also computed in September 2021. It corresponds to the "Albums" graphs used in the experiments of Chapter 7.

Artists and albums are also described by 128-dimensional vectors x_i, which will act as node features in GAE/VGAE models. We obtained these vectors from the factorization of a *pointwise mutual information* matrix among music tracks, using singular value decomposition (SVD). The

matrix is constructed from the co-occurrences of music tracks in diverse musical collections on Deezer, e.g., music playlists, listening sessions, and lists of favorite tracks. The SVD returns a 128-dimensional vector for each track; we subsequently derive the 128-dimensional vector of each artist/album by averaging the vectors from their corresponding music tracks.

Evaluation Metrics We do not have access to a list of artist/album communities that should actually be recommended together to users. As a consequence, in this chapter, we assume the unavailability of ground truth node labels for evaluation. This prevents computing AMI and ARI scores, as we did in Part II of this thesis. In these offline experiments, we instead report three alternative metrics, to evaluate the models described in the next paragraph:

- first and foremost, we report the *modularity* score Q associated with each node partition into K communities. The modularity, defined in Definition 7.2 from Chapter 7, acts as an unsupervised graph-based measure of the quality of a node partition, in terms of intra- and inter-community edge density;

- then, we evaluate the *musical homogeneity* of each community, in terms of music genres. As in Section 8.4.1 from Chapter 8, nodes are associated with 32-dimensional music genre embedding vectors, computed internally at Deezer. For each community detection method, we compute H, the average intra-community standard deviation of these vectors, as a measure of intra-community musical homogeneity;

- lastly, we consider the artists and albums liked by a set of 2 000 Deezer users in October 2021, i.e., the month after computing the two similarity graphs of this study. We denote by $\mathcal{F}_u$ the set of artists (or, of albums) liked by the user u during the period. Furthermore, we denote by $N_{uk} = \sum_{i \in \mathcal{F}_u} \mathbb{1}_{(i \in C_k)}$ the number of artists (or albums) from $\mathcal{F}_u$ belonging to the community C_k, with $k \in \{1, \ldots, K\}$. We compute:

$$P_u = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}_{N_{uk} > 0}, \tag{9.1}$$

i.e., the percentage of communities including at least one favorite artist (or album) of u. A low value of P_u indicates that a few communities summarize the musical preferences of u. This is desirable in the context of our recommender systems (see Section 9.2), as unlistened artists/albums from the same few communities would more likely constitute relevant recommendations for this user. Specifically, in our experiments, we consider $P = \frac{1}{2000} \sum_u P_u$, the average value of P_u over the set of users.

Models Using these evaluation metrics, we compare the communities obtained from the Louvain method [31] to those obtained via several GAE and VGAE models:

- the *standard GAE and VGAE* models from Kipf and Welling [187] with two-layer GCN encoders and inner product decoders;

- their simplified counterparts, *Linear GAE and VGAE*, introduced in Chapter 6;

- our improved models for community detection, *Modularity-Aware GAE and VGAE*, with consistent decoders/losses w.r.t. Chapter 7. We consider two variants of these models, with either 1) the linear encoders from Chapter 6, or 2) 2-layer GCN encoders, incorporating our revised message passing operator $A + \lambda A_s$ on the first layer only, as in Chapter 7.

All hyperparameters were optimized according to the model selection procedure proposed in Section 7.3.3 from Chapter 7. We set $d = 32$ for all models. Multi-layer GCN encoders include a 64-dimensional hidden layer. We trained models for 500 iterations (resp., 600 iterations), with a learning rate of 0.01 (resp., of 0.005) for the similar artists graph (resp., for the similar albums graph), without dropout and using the Adam optimizer [184]. For all Modularity-Aware GAE and VGAE models and for both graphs, we used the same values of λ, β, γ and s as those reported in Table 7.2 from Chapter 7. To overcome scalability issues, we used the FastGAE method from Chapter 4, and trained all models by decoding stochastic subgraphs of size $n_{(S)} = 10\,000$, with degree-based sampling and with $\alpha = 1$ for both graphs. Our experiments therefore incorporate and combine several of the technical contributions presented throughout this thesis.

9.3.3 Offline Evaluation of GAE and VGAE Models

Table 9.1 reports our results on the two graphs. For the sake of readability, we report *relative* scores w.r.t. the Louvain method, currently used in our production systems. We recall that, in these experiments, we aim to *maximize Q* but to *minimize H* and *P*.

Results   The Louvain method produces the communities associated with the highest modularity (Q) values. We underline that Q is the criterion explicitly optimized by the Louvain greedy algorithm [31]. While the GAE and VGAE models underperform according to this criterion, most of them reach better H and P scores. Better H scores indicate that the GAE/VGAE-based communities are more homogeneous in terms of music genres. Better P scores indicate that user preferences tend to be summarized by fewer GAE/VGAE-based communities.

We now compare the different GAE and VGAE models. Firstly, we observe that our Modularity-Aware GAEs and VGAEs, specifically designed for improved community detection, provide the best H and P scores. They also obtain better Q scores than standard GAEs and VGAEs. Such a result is not surprising, as these models incorporate the Louvain method as a pre-processing step (see Section 7.3.2) as well as a modularity-inspired regularization term (see Section 7.3.3). Moreover, as in Chapter 6, the results obtained using linear encoders are either on par or

9.3. Improving Music Recommendation with GAE and VGAE

Table 9.1: Community detection on similar Deezer artists/albums graphs, using the Louvain method and several GAE and VGAE models. All GAE and VGAE models learn embedding vectors of dimension $d = 32$, with other hyperparameters described in Section 9.3.2. All models learn $K = 100$ (resp., $K = 1000$) communities of artists (resp., of albums). Scores are averaged over 10 runs. We report relative scores (in %) w.r.t. the Louvain baseline. We recall that we aim to *maximize Q* but to *minimize H and P*. **Bold** numbers correspond to the best scores.

Model	Artists			Albums		
	Modularity Q	Homogeneity H	Coverage P	Modularity Q	Homogeneity H	Coverage P
Louvain	**100**	100	100	**100**	100	100
Linear Standard GAE	88.2	96.9	97.6	78.3	101.4	95.1
Linear Standard VGAE	89.4	96.8	97.4	80.1	98.5	94.8
GCN-based Standard GAE	89.4	96.6	97.6	79.7	100.9	95.4
GCN-based Standard VGAE	90.9	96.0	97.5	80.6	98.2	94.7
Linear Mod.-Aware GAE	93.6	96.0	97.1	92.4	98.9	94.3
Linear Mod.-Aware VGAE	94.2	96.0	**96.9**	92.7	97.8	93.7
GCN-based Mod.-Aware GAE	93.9	95.3	97.2	92.6	98.7	94.1
GCN-based Mod.-Aware VGAE	94.6	**94.4**	97.0	93.1	**97.5**	**93.6**

relatively close to those obtained using 2-layer GCN encoders. Lastly, on both graphs, VGAE models tend to slightly outperform their GAE counterparts, although most results are still close as in our previous experiments from this thesis.

In our experiments, all GAE and VGAE models processed the aforementioned SVD-based node features. In future tests, these features could be completed with some additional information on artists and albums. For instance, one could incorporate information on music genres, as we did in Chapter 8 (in this Chapter 9, music genres were alternatively used for *evaluation*). We would expect the addition of music genres into node features to improve the homogeneity scores H. Lastly, while our experiments set $K = 100$ or 1000 in Table 9.1, we note that we obtained consistent conclusions with other values, in our preliminary tests. At the time of writing, Deezer selects the optimal number and sizes of communities through heuristics undisclosed in this thesis.

Discussion Overall, these results are promising. The best communities for music recommendation (see Section 9.2) are not always the densest ones, e.g., the ones associated with the highest modularity. At the price of a lower modularity, GAEs and VGAEs from Table 9.1 manage to learn artist/album communities with a slightly higher musical homogeneity. More importantly, their lower P scores suggest that GAE/VGAE-based communities are more in phase with our recommendation strategy, i.e., that if users already listened to several artists from a community, then other unlistened artists from the same community could be recommended to them.

Undoubtedly, these results remain very preliminary. To verify whether GAEs and VGAEs indeed learn "better" communities for recommendation, the Deezer team might try to incorporate them into production-facing recommender systems such as the Flow. For instance, an online A/B test could measure whether the GAE/VGAE-based communities improve some key performance indicators (e.g., whether Deezer users listen to the recommended tracks longer, or, skip the recommended tracks less often) w.r.t. Louvain-based communities. At the time of writing, such

investigations were still undone. They open the way for some future work and online tests, that might lead to a model deployment on the Deezer service in 2022.

Extensions   In addition to these online A/B tests on Deezer, we also plan to extend these analyses to graphs of *users*, previously mentioned in Section 9.3.1. As the FastGAE method permits scaling GAE and VGAE models to large graphs with millions of nodes and edges, future work could also aim to train models on graphs with a larger number of artists or of albums, as well as on large similar *music tracks* graphs with up to 73 million nodes, and on more general *knowledge graphs* of musical entities. In such graphs, as detailed in Chapter 1, music tracks would be connected to artists, albums, music genres, or record labels, that would themselves be connected together through various semantic links.

Another research direction could consist in learning *country-specific* or *culture-specific* music graph representations. As we will develop in Chapter 10, people from different countries and/or different cultures perceive music differently [340]. This can result in different listening behaviors on music streaming services. For instance, two similar artists graphs, constructed only from the usage data of French users and Japanese users, respectively, might be structurally different. Taking into account these differences by considering country/culture-specific graphs might permit providing more refined recommendations. In a similar fashion, future work could also consider *context-specific* similarity graphs. As music consumption highly depends on the listening context [360], some node pairs (e.g., some artist pairs) might only be connected (i.e., co-listened) in some contexts (e.g., based on the activity, mood, or time of the day).

9.3.4   Towards Dynamic Music Graph Embedding and Clustering

In this section, we continue to discuss extensions of our methods. We provide more details on an ongoing project, related to GAE/VGAE-based dynamic node embedding and clustering.

From Static to Dynamic Graphs   In the experiments of Section 9.3.3, the graph structures under consideration were *fixed*. In reality, as previously mentioned in Chapter 8, new artists (as well as new albums, music tracks, etc) regularly appear on the Deezer service. As a consequence, new nodes should be frequently incorporated into the corresponding similarity graphs. On the contrary, some artists (or albums, music tracks, etc) are sometimes removed from the catalog. In addition, users' listening habits on Deezer might evolve over time, resulting in new or removed edges between these nodes. Therefore, these similarity graphs are *dynamic* by nature.

Our (static) definition of a graph from Definition 2.1 can be straightforwardly extended to represent dynamic structures. In the scientific literature, *dynamic graphs* are often summarized

by T discrete snapshots of standard graphs, depicting the structural evolution over time [179]:

$$\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \ \mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2), \ \ldots, \ \mathcal{G}_T = (\mathcal{V}_T, \mathcal{E}_T), \tag{9.2}$$

They are associated with T adjacency matrices $A_1, A_2, \ldots, A_T$. As an illustration, Deezer computes similarity graphs on a weekly basis (see Section 9.2). Hence, each similar artist graph $\mathcal{G}_t$ corresponds to a snapshot of the Deezer catalog and users' listening data during the t-th week.

Dynamic Music Graph Clustering   Currently, our production system computes artist communities every week *from scratch*, by running the Louvain algorithm on every new similarity graphs. In a similar fashion, one could consider training a completely new GAE/VGAE model every week. While being simple, this approach suffers from two limitations:

- firstly, this approach is computationally *inefficient*. In our applications, $\mathcal{G}_t$ will often be relatively similar to $\mathcal{G}_{t-1}$. For instance, the new nodes (e.g., the new artists) appearing during a week usually represent much less than 1% of the catalog. Intuitively, the information learned on $\mathcal{G}_{t-1}$ should still be relevant to extract communities from $\mathcal{G}_t$. Training an entire model from scratch every week might not be necessary;

- secondly, this approach is not *stable* over time. By training a new model on $\mathcal{G}_t$, there is no theoretical guarantee that the resulting communities will be "close" to those obtained at week $t-1$. Yet, for consistency reasons, ensuring relatively stable recommendations over time is usually desirable for music streaming services such as Deezer.

Extending GAE and VGAE Models to Dynamic Graphs   In 2020, the research internship of Raphaël Ginoulhac [108], that we supervised at Deezer with Benjamin Chapus, aimed to address these limitations. During his internship at Deezer, Raphaël reviewed the recent advances on dynamic node embedding and clustering (we refer the interested reader to the recent survey of Kazemi et al. [179]). He subsequently investigated several strategies to extend GAEs and VGAEs to dynamic graphs. Through an experimental analysis on similar artists graphs, he revealed the empirical effectiveness of two simple methods:

- **HotStart GAE/VGAE**: this method is inspired by the work of Goyal et al. [120]. It consists in 1) training a GAE/VGAE model on an initial graph $\mathcal{G}_0$, then 2) using the optimized weights of such a model as the *initial weights* of the GAE/VGAE model associated with a new graph $\mathcal{G}_1$, and 3) fine-tuning this new model for a few training iterations only[5]. The procedure is repeated over time, i.e., the final weights of the $(t-1)$-th model correspond to the initial weights of the t-th model, for any $t > 1$;

---

[5]In Chapter 8, we trained our models on a graph of "warm" artists, and used the same model to learn embeddings for new "cold" nodes from this graph. This approach actually constitutes a particular case of HotStart GAE/VGAE, where the model is *not* fine-tuned after a modification of the graph structure.

- **KL-Reg VGAE**: this alternative method for VGAE models consists in modifying the ELBO objective from Equation (2.40), so that the Kullback-Leibler divergence [199] term forces the posterior distribution at time t, say $q^{(t)}$, to be close to the one computed at time $t-1$, say $q^{(t-1)}$, instead of the unit Gaussian prior $\mathcal{N}(0, I_d)$. Specifically, we firstly train a VGAE model, with a standard ELBO objective, on an initial graph $\mathcal{G}_0$. Then, when training the t-th VGAE on $\mathcal{G}_t$ with $t > 0$, we maximize, by gradient ascent:

$$\mathcal{L}_{\text{VGAE}}^{(t)} = \mathbb{E}_{q^{(t)}(Z|A,X)}\Big[\log p^{(t)}(A|Z,X)\Big] - \mathcal{D}_{KL}\Big(q^{(t)}(Z|A,X)||q^{(t-1)}(Z|A,X)\Big). \qquad (9.3)$$

Raphaël's experiments tend to confirm that both methods are computationally efficient, i.e., that only a few training iterations are required for the t-th model to converge when $\mathcal{G}_t$ is relatively similar to $\mathcal{G}_{t-1}$. Simultaneously, and in a comparable way, both methods are more stable than the standard approach consisting in training new models from scratch every week. Specifically, in a majority of experiments, the AMI scores of communities extracted from $\mathcal{G}_t$ w.r.t. those extracted on $\mathcal{G}_{t-1}$ increase when using the HotStart GAE/VGAE or KL-Reg VGAE methods.

Nonetheless, such results still require further theoretical analyses, as well as an empirical validation on external datasets and a proper comparison to existing baselines [120, 179]. While we chose not to fully report these results in their preliminary state in this thesis, they will motivate more investigations in the upcoming months, potentially leading to a scientific publication and/or to a model deployment on the Deezer service.

9.4   Conclusion

In this chapter, we provided a broader overview of how Deezer has historically leveraged similarity graphs for community-based music recommendation. Then, through offline experiments on data extracted from Deezer's production system, we showed the benefits of adopting GAE and VGAE models, integrating our technical contributions from Part II.

While our conclusions are promising, we acknowledge that the results presented in this chapter are the most preliminary of this thesis. They still require further empirical validation, including through online A/B tests on the service. With the Deezer team, we will consider launching such tests in the upcoming months, i.e., after the end of this PhD project.

Lastly, in this chapter, we mentioned several ongoing and future plans to extend our methods and our analyses to other graphs. This includes our graphs of users, which have been quite neglected so far, as well as more general knowledge graphs, culture-specific graphs, and context-specific graphs. As the structures under consideration in our work are dynamic by nature, we also aim to pursue our efforts towards more efficient dynamic node embedding methods.

10

Modeling the Music Genre Perception across Language-Bound Cultures

This chapter presents research conducted with Elena V. Epure, Manuel Moussallam, and Romain Hennequin, and published in the proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020) [83]. This chapter also mentions the Muzeeglot prototype [84], internally developed with Felix Voituret and based on this research.

10.1 Introduction

The first two chapters of this Part III, i.e., Chapters 8 and 9, presented several applications of GAE and VGAE models to similarity graphs. In this Chapter 10, we now focus on quite different graph structures, that correspond to music genre *ontologies*. As explained in Chapter 1 and further detailed in the following sections, these ontologies are graphs of conceptually related music genres, connected through various relation-specific edges [321]. In this chapter, we propose to leverage these representations to model the music genre perception across cultures.

Specifically, we consider the following problem: the music genre perception expressed through human annotations of artists or albums varies significantly across (language-bound) cultures [95]. As we will show, these variations cannot be modeled as mere translations since we also need to account for cultural differences in the music genre perception [83]. This is an important issue for music streaming services such as Deezer, as these variations impact a wide range of music information retrieval and recommendation tasks, ranging from language-aware music genre auto-tagging to localized playlist captioning and music genre-driven recommendations [82]. To address

this problem, we would need to tag music items, such as artists, albums, or tracks, with music genre representations that capture such differences in perception, which is challenging [83, 85].

Previous research works at Deezer already studied the relations between music genres and the way music items are associated with music genres. Hennequin et al. [141] managed to learn music genre embedding representations from audio data, with application to audio-based disambiguation of music genre annotations (i.e., identifying situations where different tags actually refer to the same genre, or, on the contrary, where the same genre is identified by different tags). Epure et al. [81] proposed to use knowledge bases and parallel annotations for music genre "translation" across several inconsistent genre tag systems, but only considered English genres.

In this work, published in a more NLP-focused venue (EMNLP 2020), we aim to complement these studies, by *analyzing the feasibility of obtaining relevant cross-lingual, culture-specific music genre annotations based only on language-specific semantic representations*, namely on 1) *graph ontologies*, and 2) NLP-based *distributed word embeddings* of music genres, acting as attribute vectors of genres/nodes in these graphs. Our study, focused on six languages and on genre annotations obtained from the Wikipedia online encyclopedia, shows that unsupervised cross-lingual music genre annotation is feasible with high accuracy when combining both types of representations, using either GAE/VGAE-inspired models or *retrofitting* [87]. Besides, along with the paper associated with this work [83], we publicly released our source code, as well as the dataset (scraped and processed from the internet) that we used for our experiments.

This chapter is organized as follows. In Section 10.2, we introduce the problem of subjectivity in the music genre perception in detail. We also formally describe the cross-lingual music genre annotation problem we aim to solve in this work. To address this problem, we subsequently present, in Section 10.3, our methods to learn music genre representations by combining graph ontologies and word embeddings. We report and discuss our experimental setting and our results in Section 10.4. At the end of this same section, we also mention the Muzeeglot prototype [84], internally developed at Deezer to visualize our multilingual music genre embedding representations. We conclude in Section 10.5. In Section 10.6, we provide an additional proof, placed out of the "main" chapter for the sake of brevity and readability.

10.2   The Cross-Lingual Music Genre Annotation Problem

In this section, we detail the research problem we aim to address in this chapter.

10.2.1   Modeling the Music Genre Perception

A prevalent approach to culturally study music genres starts with a common set of music items, e.g., artists, albums, or music tracks, and assumes that the same music genres would be asso-

ciated with the items in all cultures [95, 337]. However, music genres are subjective. Cultures themselves and individual musicological backgrounds influence the music genre perception, which can differ among individuals [210, 340]. For instance, a Westerner may relate *funk* to *soul* and *jazz*, while a Brazilian to *baile funk*, which is a type of *rap* [141]. Thus, accounting for cultural differences in the music genre perception could give a more grounded basis for such cultural studies. It could also help music streaming services associate their musical catalog with more relevant music genres tags, and, among other applications, provide better genre-driven recommendations to users (see, e.g., our experiments from Chapters 8 and 9, that both involved music genre-related data). However, ensuring both a common set of music items and culture-sensitive annotations with a broad coverage of music genres is strenuous [33].

To address this challenge, we study the feasibility of cross-culturally annotating music items with music genres, without relying on a parallel corpus. In this work, culture is related to *a community speaking the same language*[1] [198]. The specific research question we build upon is: *assuming consistent patterns of music genres association with music items within cultures, can a mapping between these patterns be learned by relying on language-specific semantic representations?* It is worth noting that, since music genres fall within the class of Culture-Specific Items [10, 268], cross-lingual annotation, in this case, cannot be framed as standard translation, as one also needs to model the dissimilar perception of music genres across cultures.

Our work focuses on six languages from four language families: Germanic (English and Dutch), Romance (Spanish and French), Japonic (Japanese), and Slavic (Czech); and on two types of language-specific semantic representations: music genre ontologies and word embeddings. In the following Section 10.2.2, we formalize the cross-lingual annotation task. Then, in Section 10.2.3, we describe the Wikipedia-based test corpus we used in this work, to evaluate the music genre representations that we will subsequently present in Section 10.3.

10.2.2 Problem Formulation

Our cross-lingual music genre annotation task consists of inferring, for music items, tags in a target language L_t, knowing tags in a source language L_s according to some source (e.g., Wikipedia, in this study). For instance, knowing the English music genres of *Fatboy Slim* (*big beat, electronica, alternative rock*), the goal is to predict *rave* and *rock alternativo* in Spanish. As shown in this example and in the one from Section 10.2.1, the problem goes beyond translation and instead targets a model able to map concepts, potentially dissimilar, across languages and cultures.

Formally, given S a set of tags in a language L_s, $\mathcal{P}$ the partitions of S and T a set of tags in a language L_t, a mapping scoring function $f : \mathcal{P}(S) \to \mathbb{R}^{|T|}$ can attribute a prediction score to each target tag, relying on subsets of source tags drawn from S [81, 83, 141]. The produced score

---

[1]This assumption conveniently frames the problem. Nonetheless, we acknowledge that it is quite strong for some languages, e.g., French and Spanish, if the origin of music genres annotators is not specified.

Figure 10.1: Wikipedia infoboxes of Puerto Rican artist Milton Cardona, in English, Spanish, and Japanese. Some music genre annotations are culture-specific, such as *World*/ワールドミュージック which is present in English and Japanese but not in Spanish, or 実験音楽 (*Experimental Music*) in Japanese only.

incorporates the degree of relatedness of each particular input source tag to the target tag. A common approach to compute relatedness in distributional semantics relies on cosine similarity. Thus, for some source tags $\{s_1, ..., s_K\}$ and a target tag t, f can be defined as:

$$f_t(\{s_1, s_2, \ldots, s_K\}) = \sum_{k=1}^{K} \frac{z_{s_k}^T z_t}{||z_{s_k}||_2 ||z_t||_2}, \tag{10.1}$$

where z_{s_k} and z_t are some d-dimensional embedding vectors representing each s_k and t, respectively (see Section 10.3), and where $|| \cdot ||_2$ denotes the Euclidean norm.

10.2.3 Evaluation Corpus

We identify Wikipedia[2], the online multilingual encyclopedia, to be particularly relevant to our study. Indeed, Wikipedia records worldwide music artists and their discographies, with a frequent mention of their music genres. By manually checking the Wikipedia pages of music items, we observed that their music genres vary significantly across languages. For instance, *Knights of Cydonia*, a single by *Muse*, was annotated in Spanish as *progressive rock*, while in Dutch as *progressive metal* and *alternative rock*. In Figure 10.1, we provide another example of different annotations in English, Spanish, and Japanese from Wikipedia infoboxes.

---

[2]https://en.wikipedia.org

Table 10.1: Number of music items for each language pair.

Language	Dutch	French	Spanish	Czech	Japanese
English	12 604	28 252	32 891	4 772	14 752
Dutch		7 139	7 689	1 885	3 426
French			15 616	3 046	8 622
Spanish				3 245	7 644
Japanese					2 065

As Wikipedia's writing is localized, the various cultures of its contributors can lead to differences in the multilingual content on the same topic, particularly for subjective matters. We refer the interested reader to the study of Pfeil et al. [289], arguing that Wikipedia's contributions expose cultural differences aligned with the real-world ones. Therefore, Wikipedia appears as a suitable source for assembling a test corpus.

Using DBpedia [19] as a proxy to Wikipedia, we collected music items such as artists and albums, annotated with music genres in at least two of the six languages (English, Dutch, French, Spanish, Czech, and Japanese). We targeted the *MusicalWork*, *MusicalArtist* and *Band* DBpedia resource types, and we only kept music items that were annotated with music genres which appeared at least 15 times in the corpus. Our final corpus includes 63 246 music items, and was publicly released along with the paper associated with this work [83]. The number of annotations for each language pair is presented in Table 10.1. We also show in Table 10.2 the number of unique music genres per language in the corpus and the average number of tags for each music item.

The English and Spanish languages use the most diverse tags. This can be because more annotations exist in these languages, in comparison to Czech, which has the least annotations and least diverse tags. However, the mean number of tags per item appears relatively high for Czech, while Japanese has the smallest average number of tags per item.

10.3 Learning Music Genre Representations

This work assesses the possibility of obtaining relevant cross-lingual music genre annotations, able to capture cultural differences too, by relying on language-specific semantic representations. Again, such representations would be valuable for music streaming services such as Deezer, e.g., to improve localized content tagging and music genre-driven recommendation.

10.3.1 General Overview

The first step of our method consists in obtaining initial embedding representations for music genres by leveraging *pre-trained multilingual word embeddings* (Section 10.3.2). However,

directly using these embeddings for cross-lingual genre annotation is prone to underperform, as:

- the embeddings often correspond to the most common word senses (e.g., *country* can refer to *nation*, and *rock* could be closer in meaning to *stone*), and not to their musical senses;

- some music genres could contain rare words which are absent from the pre-trained model vocabulary, resulting in potentially unknown tag embeddings.

To address these issues, we complement our word embeddings with semantics from knowledge bases that expose concept relations. Specifically, in a second step, we assemble *graph ontologies* to represent music genre relations, as described in Section 10.3.3. Then, we aim to adjust the initial genre embeddings to encode relations from these graphs, ensuring the domain adaptation. For this, but also to learn embeddings for concepts with unknown vocabulary words, we consider two different approaches based on retrofitting and on GAE/VGAE-inspired models, and presented in Sections 10.3.4 and 10.3.5, respectively.

We note that, in contrast to our unsupervised approach, mapping patterns of associating music genres with music items could have also been enabled with a parallel corpus [81]. However, gathering a corpus that includes all music genres for each pair of languages is challenging [83].

10.3.2 Initializing Music Genre Embedding Representations

Multilingual Word Embeddings Under the formalism introduced in Section 10.2.2, our main goal relates to quantifying the degree of relatedness of two textual tags. This task is widely popular in the NLP community and contemporary approaches resort to expressing their relatedness via the distance between their corresponding word embeddings [121, 256, 285]. The mapping of words with embedding vectors is guided by the *distributional hypothesis* [134], which states that words occurring in similar contexts are likely to have similar meanings.

To measure the relatedness of multilingual words using embeddings obtained from monolingual corpora, an *alignment* [171] between the language-specific embedding spaces is required. Through this alignment, we ensure that multilingual word embeddings are projected into a common space where they are comparable. Practically, a mapping function between two monolingual word embedding spaces is learned, for instance by using a bilingual lexicon [257].

In the experiments reported in this chapter, we leverage word embeddings obtained from the multilingual *fastText* (FT) model proposed by Grave et al. [121], which we aligned by using the method described by Joulin et al. [172]. Embedding vectors are of dimension $d = 300$. We emphasize that this choice is not restrictive. In our EMNLP 2020 paper, we provide additional results for music genre word embeddings obtained from the recent BERT [71], LASER [16] and XLM [60] models, together with a comparison of these different methods. We omit these additional results in this thesis, for the sake of brevity.

Music Genre Embeddings   Starting from these multilingual fastText word vectors, we now discuss our strategy to initialize our music genre embedding representations. Music genres can contain multiple words. We claim that the *compositionality principle*, stating that the meaning of a multi-word expression is dictated by the meaning of each word, often holds for our case. For instance, *Dance pop* is related to *dance* and *pop* or *Balada romántica* is a type of *ballad* which is *romantic*[3]. The contemporary approach for compositional embeddings consists in learning a function which derives the embeddings of a multi-word expression from the embeddings of its words [335]. In this work, we consider two music genre initialization strategies:

- the first one, denoted *avg*, simply consists in *averaging word embeddings*. Formally, let $\mathcal{V} = \{c_1, c_2, \ldots, c_n\}$ be the multilingual vocabulary, c_i being a concept (i.e., a genre) composed of at least one word. We aim to compute $\hat{Q} \in \mathbb{R}^{n \times d}$, the embedding matrix for the vocabulary $\mathcal{V}$, where the i-th row $\hat{q}_i \in \mathbb{R}^d$ denotes the embedding of c_i. If c_i is composed of the following words, $\{w_1, w_2, \ldots, w_M\}$, $\hat{q}_i$ can be computed as $\frac{1}{M} \sum_{m=1}^{M} z_{w_m}$, where z_{w_m} is the embedding vector of w_m. We note that, if c_i contains words absent from the pretrained fastText vocabulary, the d-dimensional null vector 0_d is used by default;

- the second one, denoted *sif*, exploits the fact that some words in a compounded expression may be more illustrative than others. The more frequently a word is observed in a corpus, the more likely it is that the word is common for a language and semantically less informative (e.g., *music* in *post industrial music*). Thus, the compositional embedding computation of a multi-word expression can be modified such that the contribution of each word embedding is inversely proportional to its frequency. Pre-trained word embeddings are generally released sorted by decreasing word corpus frequency. Let r_{w_m} be the rank of w_m in this vocabulary. Then, based on Mandelbrot's generalization [249] of Zipf's law [411], its frequency f_{w_m} can be estimated as follows: $f_{w_m} = 1/(r_{w_m} + 2.7)$.

In the following, we rely on the *smooth inverse frequency* (*sif*) based averaging proposed by Arora et al. [15] to compute the multi-word expression embeddings. This method is aligned with our previous observations and proven highly effective compared to more complex neural network-based models on a large diversity of NLP tasks [15]. Given f_{w_m} the estimated frequency of the word w_m and a a fixed hyperparameter[4], $\hat{q}_i$ is computed as:

$$\overline{q}_i = \frac{1}{M} \sum_{m=1}^{M} \frac{a}{a + f_{w_m}} z_{w_m}, \tag{10.2}$$

$$\hat{q}_i = \overline{q}_i - uu^T \overline{q}_i, \tag{10.3}$$

where u is the first singular vector from the SVD of $\overline{Q}$ obtained with the Equation (10.2).

---

[3]Exceptions from the principle also exist, e.g., *hard rock*.
[4]Arora et al. [15] argue that $a = 10^{-3}$ is a suitable choice for various pre-trained models.

Using one of these two methods, we can derive initial music genre embedding representations. In the remainder of Section 10.3, we will improve these (*avg-* or *sif*-based) embeddings using retrofitting or GAE/VGAE-inspired models, to encode relations from the ontologies presented in the next Section 10.3.3.

10.3.3   Music Genre Graph Ontologies

Conceptually, music genres are interconnected entities. Researchers often use *ontologies* to represent music genre relations and enrich the music genre definitions [230, 297, 321]. These ontologies are graphs of conceptually related music genres, acting as nodes and connected through various relation-specific edges. For instance, *rap west coast* is a subgenre of *hip hop*, while *punk* and *electronic music* are the origin of *synthpunk*.

In this chapter, we consider Wikipedia-based ontologies, extracted from the internet using the procedure described below. Wikipedia extensively documents worldwide music genres, relating them through a coherent set of relation types across languages, including the aforementioned subgenre and origin relations. Though the relations types are the same per language, the actual music genres and the way they are related can differ and expose cultural differences in the music genre perception, as emphasized by Pfeil et al. [289]. We further describe how we crawled these ontologies, by relying on DBpedia as in Section 10.2.3.

Table 10.2: Number of unique music genres in the evaluation corpus (presented in Section 10.2.3) and in the graph ontologies (presented in Section 10.3.3).

Language	Number of unique music genres	
	in the evaluation corpus *(avg. per item)*	in the ontologies
English	558 *(2.12 ± 1.34)*	10 748
Dutch	204 *(1.71 ± 1.06)*	1 529
French	364 *(1.75 ± 1.06)*	2 905
Spanish	525 *(2.11 ± 1.34)*	3 988
Czech	133 *(2.23 ± 1.34)*	1 418
Japanese	192 *(1.51 ± 1.11)*	1 609

Firstly, for each language, we constituted a seed list using two sources: the DBpedia resources of type *MusicGenre* and their aliases linked through the *wikiPageRedirects* relation; the music genres discovered when collecting the evaluation corpus (introduced in Section 10.2.3) and their aliases. Then, music genres were fetched by visiting the DBpedia resources linked to the seeds through the relations: *wikiPageRedirects*, *musicSubgenre*, *stylisticOrigin*, *musicFusionGenre* and *derivative*[5]. The seed list was updated each time, allowing the crawling to continue until no new resource was found. In DBpedia, resources are sometimes linked to their equivalents in other languages through the relation *sameAs*. In most experiments, we will rely on monolingual music genres ontologies (i.e., on six isolated graphs). However, we also collected the cross-lingual links between music genres to include a translation baseline for cross-lingual annotation, i.e., for

---

[5]We present these relations by their English names, which may be translated in DBpedia in other languages.

each music genre in a source language, we predicted its equivalent in a target language using DBpedia's *sameAs* (see Section 10.4).

The number of unique music genres discovered in each language is presented in Table 10.2. As our evaluation corpus from Section 10.2.3, these graphs were publicly released along with the paper associated with this work [83]. We note that the number of nodes is significantly larger than the number of genres in the evaluation corpus, emphasizing the challenge of constituting a parallel corpus that would cover all language-specific music genres.

10.3.4 Improving Music Genre Representations with Retrofitting

In this section, and in the following one, we present our two strategies to improve the initial music genre embedding vectors from Section 10.3.2, by encoding information from the graph ontologies of Section 10.3.3 into these vectors. Firstly, we present a method referred to as *retrofitting* [87] in this section, while the following one will focus on GAE/VGAE-inspired approaches.

Retrofitting [87] is a method to refine word embedding vectorial representations by considering the relations between words as defined by a graph, e.g., an ontology. It consists in modifying the word embeddings to become closer to the representations of the concepts to which they are related. Ever since the original work of Faruqui et al. [87], many uses of retrofitting have been explored to semantically specialize word embeddings in relations such as synonyms or antonyms [182, 183], in other languages than a source one [291] or in specific domains [132].

Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph including the concepts $\mathcal{V}$ and the semantic relations between these concepts $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The goal of retrofitting is to learn new concept embeddings $q_i \in \mathbb{R}^d$, stacked up in the matrix $Q \in \mathbb{R}^{n \times d}$, with $n = |\mathcal{V}|$ and d the embedding dimension. The learning starts by initializing each q_i, i.e., the new embedding vector for a concept $i \in \mathcal{V}$, to $\hat{q}_i$, the initial embedding vector. Then, it consists in iteratively updating q_i until convergence, as follows:

$$q_i \leftarrow \frac{\sum_{j:(i,j)\in\mathcal{E}} (\beta_{ij} + \beta_{ji})q_j + \alpha_i \hat{q}_i}{\sum_{j:(i,j)\in\mathcal{E}} (\beta_{ij} + \beta_{ji}) + \alpha_i}. \tag{10.4}$$

The α and β terms are positive scalars weighting the importance of the initial embedding vector and of the ones from related concepts in computations, respectively. Equation (10.4) was reached through the minimization of the convex retrofitting loss:

$$\Phi(Q) = \sum_{i\in\mathcal{V}} \alpha_i ||q_i - \hat{q}_i||_2^2 + \sum_{i=1}^{n} \sum_{(i,j)\in\mathcal{E}} \beta_{ij} ||q_i - q_j||_2^2, \tag{10.5}$$

using the Jacobi method [306]. We note that Equation (10.4) is actually a corrected version of the iterative procedure initially proposed by Faruqui et al. [87]. For a concept i, not only β_{ij} appears in it, but also β_{ji} (which was omitted in the original formula of Faruqui et al [87]).

The further modifications that we make regard the parameters α and β. For each $i \in \mathcal{V}$, Faruqui et al. [87] originally fixed α_i to 1, and β_{ij} to $\frac{1}{D_{ii}}$ (with D_{ii} the degree of i, as in Definition 2.4) for $(i,j) \in \mathcal{E}$ or 0 otherwise. However, while many word embedding models can handle unknown words, some concepts may still have unknown initial vectors, depending on the model's choice. For this case, expanded retrofitting [341] has been proposed, considering $\alpha_i = 0$, for each concept i with unknown initial distributed vector, and $\alpha_i = 1$ for the other ones: q_i is initialized to 0_d and updated by averaging the embeddings of its related concepts at each iteration.

We adopt the same approach to initialize the α_i parameters. We also adjust the β_{ij} to weight the importance of each related concept embedding depending on the relation semantics in our genre ontologies. Specifically, we distinguish between "equivalence" and "relatedness" as follows:

$$
\overline{\beta}_{ij} = \begin{cases} 1 & : (i,j) \in \mathcal{E}_{\text{equi}} \subset \mathcal{E}, \\ \beta_{ij} & : (i,j) \in \mathcal{E}_{\text{rel}} = \mathcal{E} \setminus \mathcal{E}_{\text{equi}}, \\ 0, & : (i,j) \notin \mathcal{E}, \end{cases} \tag{10.6}
$$

where $\mathcal{E}_{\text{equi}}$ contains the "equivalence" relation type *wikiPageRedirects*, and $\mathcal{E}_{\text{rel}}$ contains the "relatedness" relation types *stylisticOrigin*, *musicSubgenre*, *derivative* and *musicFusionGenre*.

Finally, we want to highlight an important aspect of retrofitting. Previous studies [86, 136, 341] claim that, while the retrofitting updating procedure converges, the results depend on the order in which the updates are made. We prove in the supplementary Section 10.6 that the retrofitting loss is actually *strictly convex* when at least one initial concept vector is known in each connected component. Hence, with this condition satisfied, retrofitting always converges to the same solution, independently of the ordering of updates.

10.3.5 Improving Music Genre Representations with GAE and VGAE

In essence, the retrofitting procedure consists in learning reinforced embedding representations of some nodes (corresponding to some music genres), by jointly leveraging 1) their connections in a graph, and 2) some initial vectorial representations of these nodes, in an unsupervised fashion. In the previous chapters, and on numerous occasions, we also achieved such a combination by training GAE and VGAE models. Therefore, while our EMNLP 2020 paper [83] mainly focused on improving genre representations with retrofitting, leveraging GAE and VGAE models appears as a natural alternative strategy to consider, in the context of this PhD thesis. Formally:

- the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, on which the GAE or VGAE model should be trained, is the one gathering all music genres from the six languages under consideration, as nodes, and their connections in the six ontologies, as edges. For consistency with the previous retrofitting method, we consider the *wikiPageRedirects*, *stylisticOrigin*, *musicSubgenre*, *derivative* and *musicFusionGenre* links, but not the DBpedia's *sameAs* links. Therefore, $\mathcal{G}$ actually

includes six isolated subgraphs, corresponding to our six monolingual genre ontologies;

- the feature vector $x_i \in \mathbb{R}^f$ of each node $i \in \mathcal{V}$, which acts as the input layer representation of i in the context of a GAE or VGAE model, corresponds, in this application, to the initial word embedding vector of the music genre i, i.e., $x_i = \hat{q}_i \in \mathbb{R}^d$. We observe that, in this application, we have $f = d$, i.e., the input and output representations will have the same dimension;

- the final embedding representation $z_i \in \mathbb{R}^d$ of each node $i \in \mathcal{V}$ corresponds to the vector denoted as $q_i \in \mathbb{R}^d$ in the previous section.

Nonetheless, as we argue in the next two paragraphs, a direct use of standard GAEs or VGAEs is prone to underperform in this setting. This motivates several modifications of the encoder and decoder components of these models, which we describe in the same paragraphs.

Encoder Unlike previous chapters, in our training graph ontologies, nodes can be connected through edges of five *different natures*. Our definition of a graph from Definition 2.1 can be extended to represent such structures, by replacing the notation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by: $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_5)$, where each $\mathcal{E}_i \subset \mathcal{E}$ (with $i \in \{1, \ldots, 5\}$ and $\bigcup_i \mathcal{E}_i = \mathcal{E}$) only includes the edges of a particular type, among *wikiPageRedirects, stylisticOrigin, musicSubgenre, derivative* and *musicFusionGenre*. For the sake of simplicity, and as in Section 10.3.4, we can also alternatively consider:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}_{\text{equi}}, \mathcal{E}_{\text{rel}}), \tag{10.7}$$

where, as previously defined, $\mathcal{E}_{\text{equi}}$ contains edges of "equivalence" relation type *wikiPageRedirects*, and $\mathcal{E}_{\text{rel}}$ contains edges of the "relatedness" relation types *stylisticOrigin, musicSubgenre, derivative* and *musicFusionGenre*. Then, extending our Definition 2.2, we summarize $\mathcal{G}$ by two adjacency matrices A_{equi} and A_{rel}, indicating edges of each subgroup.

In this work, we aim to capture the dichotomy of Equation (10.7) in the message passing operations of our GAE/VGAE encoders. We expect such an aspect to improve our representations, w.r.t. a standard encoding scheme ignoring the different natures of edges. Specifically, in our experiments, we employ the *linear encoders* from Chapter 6 (both for the sake of simplicity, and as they returned comparable results w.r.t. multi-layer GCNs in our preliminary tests). However, we replace the operation from Definition 6.1 of Chapter 6 (i.e., $Z = \tilde{A}XW$) by[6]:

$$Z = \tilde{A}_{\text{rel}}XW_{\text{rel}} + \tilde{A}_{\text{equi}}XW_{\text{equi}}, \tag{10.8}$$

for some weight matrices $W_{\text{rel}} \in \mathbb{R}^{d \times d}$ and $W_{\text{equi}} \in \mathbb{R}^{d \times d}$, tuned by gradient descent/ascent to optimize the loss/objective presented in the next paragraph. Consistently with the notation

---

[6]Here, for clarity of exposition, we discuss the deterministic GAE framework. However, the changes are equally applicable to the VGAE framework, for which Z has to be replaced by μ and $\log \sigma$ as in Equation (6.6).

adopted throughout this thesis, $X \in \mathbb{R}^{n \times d}$ is the node feature matrix, stacking up all x_i (i.e., $\hat{q}_i$) vectors, and $Z \in \mathbb{R}^{n \times d}$ is the node embedding matrix, stacking up all z_i (i.e., q_i) vectors. We adopt an out-degree normalization (see Definition 5.1) for the two adjacency matrices.

Decoder   In this chapter, our main goal is *not* to precisely reconstruct our graph ontologies from a music genre embedding space. In particular, consistently with Equation (10.1), we would want the music genres of a given music item in different languages to have large inner products in the embedding space. As these genres are not connected in $\mathcal{G}$, a standard GAE/VGAE with an inner product decoder and a reconstruction loss/objective based on the observed edges in $\mathcal{G}$ is unlikely to achieve such a result. As a consequence, we also adapt our decoding schemes, by instead incorporating the retrofitting loss function presented in Section 10.3.4.

Specifically, we propose a modified GAE-inspired model[7]. After encoding nodes as in Equation (10.8), the "decoder" of such a model does not reconstruct the entire graph. It only evaluates $\Phi(Z)$, as defined in Equation (10.5), from the node embedding matrix Z. Then, during training, we tune W_{rel} and W_{equi} by iteratively minimizing $\Phi(Z)$, by gradient descent.

In a similar fashion, we also propose a modified VGAE-inspired model. After encoding nodes as above (and sampling the node embedding matrix Z, in this probabilistic setting), we tune weight matrices by iteratively maximizing, by gradient ascent, the following modified objective, that replaces the standard ELBO from Equation (2.40) of Chapter 2:

$$\mathcal{L}_{\text{VGAE-insp}} = -\Phi(Z) - \lambda \mathcal{D}_{KL}\Big(q(Z|A, X) || \tilde{p}(Z) \Big). \tag{10.9}$$

In Equation (10.9), the $-\Phi(Z)$ term replaces the "reconstruction-based" expectation of the standard ELBO from Equation (2.40). We also modify the Kullback-Leibler divergence term. In Equation (10.9), $\tilde{p}$ corresponds to $\mathcal{N}(x_i, I_d)$ priors on the distribution of each z_i vector. They replace the unit Gaussian priors, and aim to ensure that the z_i embedding vectors remain relatively close to the initial vectors x_i (i.e., $\tilde{q}_i$), consistently with the retrofitting loss. The scaling hyperparameter $\lambda \geq 0$ permits ensuring that both terms are comparable[8].

10.4   Experimental Analysis

In this section, we present and discuss our experimental analysis of the proposed methods, using the evaluation corpus scraped from DBpedia and presented in the previous Section 10.2.3.

---

[7]One could wonder whether the models proposed in this section should be referred to as graph *auto*encoders. On the one hand, they do not aim to reconstruct an original graph, which was the key objective of standard autoencoders. On the other hand, they remain conceptually very close to GAE and VGAE models [187]. For these reasons, we chose to refer to them as "GAE-inspired" and "VGAE-inspired" models, throughout this chapter.

[8]Is anyone reading this? Send me a message! I would be glad to hear that someone has read my thesis so meticulously.

10.4.1   Experimental Setting

Cross-lingual music genre annotation, as formalized in Section 10.2.2, is a multi-label prediction task. For evaluation, we report the mean and standard deviations of AUC scores, macro-averaged [36], using 3-fold cross-validation. For each language, we apply an iterative split [322] of the test corpus that balances the number of samples and the tag distributions across the folds. We pre-process the music genres by either replacing special characters with space (_-/,) or removing them (()':.!$). For Japanese, we introduce spaces between tokens obtained with Mecab[9]. Embedding vectors are then computed from pre-processed tags.

We test two translation baselines, one based on Google Translate[10] and one directly leveraging the aforementioned DBpedia's *SameAs* relations for genre translation. In this case, the source music genres are mapped to a single or no target music genre. Furthermore, we consider both *avg*-based and *sif*-based variants of our fastText (FT) word embeddings, together with their corresponding retrofitted versions, adopting the hyperparameters described in the above sections. We also train GAE-inspired and VGAE-inspired models (with the modified linear encoders and loss/objective of Section 10.3.5) on both variants, using the Adam optimizer, without dropout, with a learning rate of 0.01 and for 200 training iterations. We released our source code on GitHub, along with datasets including our ontologies, embedding vectors, and evaluation corpus[11].

Disclaimer: the data pre-processing operations and retrofitting experiments were conducted by Elena V. Epure, lead author of the paper associated with this work. The GAE/VGAE-inspired experiments, absent from this paper, were subsequently designed and added for this thesis.

10.4.2   Results and Discussion

Results   Table 10.3 reports our results for cross-lingual music genre annotation, for all possible "source → target" language pairs. We observe that a standard translation via Google Translate leads to the lowest results, being outperformed by a knowledge-based translation using DBpedia's *SameAs* links, more adapted to this music domain. The results confirm that a naive translation fails to capture the dissimilar music genre perception across cultures.

On the contrary, the *sif*-based fastText initial word embedding representations (FT$_{\text{sif}}$) can model quite well the varying music genre annotation across languages, w.r.t. the aforementioned baselines. FT$_{\text{sif}}$ embeddings also consistently outperform their FT$_{\text{avg}}$ counterparts for all language pairs, which confirms the relevance of smooth inverse frequency averaging for this application. In the experiments reported in [83], we reached consistent conclusions when considering vectors obtained from the BERT, LASER, and XLM models, instead of fastText.

---

[9]https://taku910.github.io/mecab/
[10]https://translate.google.com
[11]https://github.com/deezer/CrossCulturalMusicGenrePerception

Table 10.3: Macro-AUC scores (in %) for cross-lingual music genre annotation on our DBpedia-based evaluation corpus, presented in Section 10.2.3. The first part corresponds to baselines. The second part corresponds to (initial) fastText embedding vectors, with *avg* or *sif* averaging. The third part corresponds to methods combining graph ontologies and *sif*-based fastText embedding vectors. **Bold** numbers correspond to the best scores.

Language Pair Source → Target	Google Translate	DBpedia's "SameAs"	FT_{avg}	FT_{sif}	VGAE $+ FT_{sif}$	Retrofitting $+ FT_{sif}$
English → Dutch	59.9 ± 0.3	72.2 ± 0.2	75.2 ± 0.2	86.5 ± 0.1	89.4 ± 0.2	**90.0 ± 0.1**
English → French	58.4 ± 0.1	70.0 ± 0.2	78.6 ± 0.2	87.4 ± 0.3	**90.8 ± 0.2**	**90.8 ± 0.2**
English → Spanish	56.9 ± 0.0	65.4 ± 0.2	77.5 ± 0.1	86.9 ± 0.2	**90.0 ± 0.2**	89.9 ± 0.1
English → Czech	60.6 ± 0.6	78.4 ± 0.6	74.5 ± 0.6	88.6 ± 0.4	90.0 ± 0.4	**90.4 ± 0.3**
English → Japanese	60.9 ± 0.1	70.4 ± 0.2	69.6 ± 0.5	80.8 ± 0.3	86.1 ± 0.3	**86.7 ± 0.3**
Dutch → English	53.5 ± 0.1	56.7 ± 0.3	73.8 ± 0.5	79.8 ± 0.4	**84.3 ± 0.3**	**84.3 ± 0.1**
Dutch → French	54.4 ± 0.2	60.0 ± 0.4	63.9 ± 0.5	79.3 ± 0.8	81.0 ± 0.8	**81.5 ± 0.7**
Dutch → Spanish	53.1 ± 0.2	56.8 ± 0.2	63.7 ± 0.4	77.7 ± 0.5	79.9 ± 0.5	**80.5 ± 0.4**
Dutch → Czech	57.8 ± 0.1	50.0 ± 0.0	65.1 ± 0.3	80.6 ± 0.2	82.6 ± 0.6	**83.4 ± 0.5**
Dutch → Japanese	57.5 ± 0.4	62.7 ± 0.4	64.8 ± 0.2	74.9 ± 1.0	79.1 ± 0.9	**80.0 ± 0.7**
French → Dutch	58.6 ± 0.1	65.3 ± 0.4	67.7 ± 1.0	81.9 ± 0.4	83.7 ± 0.4	**84.7 ± 0.3**
French → English	55.3 ± 0.0	59.7 ± 0.2	76.2 ± 0.2	83.0 ± 0.2	87.4 ± 0.2	**87.7 ± 0.1**
French → Spanish	54.1 ± 0.1	59.0 ± 0.1	71.0 ± 0.2	81.8 ± 0.3	84.9 ± 0.3	**85.3 ± 0.2**
French → Czech	59.1 ± 0.3	70.0 ± 0.6	70.4 ± 0.8	83.9 ± 0.4	86.5 ± 0.5	**87.2 ± 0.3**
French → Japanese	59.1 ± 0.2	64.7 ± 0.5	71.1 ± 0.3	77.9 ± 0.1	80.5 ± 0.5	**81.4 ± 0.3**
Spanish → Dutch	59.8 ± 0.3	67.2 ± 0.2	68.5 ± 0.5	82.8 ± 0.9	85.2 ± 0.6	**85.9 ± 0.6**
Spanish → French	57.4 ± 0.2	64.8 ± 0.3	70.8 ± 0.4	85.0 ± 0.3	**87.6 ± 0.5**	87.5 ± 0.3
Spanish → English	57.0 ± 0.1	61.7 ± 0.0	75.3 ± 0.1	84.7 ± 0.2	**88.8 ± 0.3**	**88.8 ± 0.3**
Spanish → Czech	60.3 ± 0.2	72.2 ± 0.4	68.9 ± 0.8	85.6 ± 0.6	87.1 ± 0.6	**88.0 ± 0.4**
Spanish → Japanese	60.9 ± 0.1	67.0 ± 0.5	65.2 ± 0.4	78.3 ± 0.6	82.5 ± 0.6	**83.1 ± 0.6**
Czech → Dutch	57.6 ± 0.6	50.0 ± 0.0	68.5 ± 1.0	78.3 ± 0.9	79.9 ± 1.0	**81.1 ± 1.2**
Czech → French	54.2 ± 0.2	60.0 ± 0.3	64.5 ± 0.9	78.5 ± 0.2	80.4 ± 0.7	**81.4 ± 0.3**
Czech → Spanish	53.7 ± 0.4	56.9 ± 0.3	65.3 ± 0.9	77.7 ± 0.8	80.7 ± 0.8	**81.6 ± 0.9**
Czech → English	54.2 ± 0.2	57.1 ± 0.1	70.3 ± 0.5	78.9 ± 0.1	83.4 ± 0.6	**84.5 ± 0.4**
Czech → Japanese	58.6 ± 0.2	64.0 ± 0.3	67.1 ± 1.1	76.9 ± 0.1	79.3 ± 0.6	**80.5 ± 0.5**
Japanese → Dutch	54.8 ± 0.4	61.6 ± 1.1	62.0 ± 0.5	72.8 ± 1.0	74.0 ± 0.7	**76.9 ± 0.3**
Japanese → French	53.3 ± 0.2	58.4 ± 0.2	66.0 ± 1.1	73.7 ± 0.6	76.4 ± 0.4	**77.8 ± 0.1**
Japanese → Spanish	52.7 ± 0.1	55.9 ± 0.4	63.2 ± 0.3	73.9 ± 0.4	77.5 ± 0.6	**78.8 ± 0.5**
Japanese → Czech	56.1 ± 0.5	65.7 ± 0.7	61.7 ± 1.3	77.5 ± 0.2	78.7 ± 0.7	**80.7 ± 0.4**
Japanese → English	52.5 ± 0.1	55.8 ± 0.1	72.1 ± 1.0	75.6 ± 0.3	80.2 ± 0.8	**81.6 ± 0.8**

The last two columns of Table 10.3 report the results of cross-lingual annotation when using the FT_{sif} vectors retrofitted to monolingual music genre ontologies, and when combining FT_{sif} vectors and ontologies using the VGAE-inspired approach. The latter reaches comparable results w.r.t. its GAE-inspired counterpart, which is therefore not reported in Table 10.3 for brevity. In a nutshell, for all three methods (GAE-inspired, VGAE-inspired, and retrofitting), the domain adaptation of initial word embeddings improves our scores across all pairs of languages.

Nonetheless, in this application, retrofitting tends to outperform our GAE/VGAE-inspired models. Such a result emphasizes the effectiveness of the (arguably simpler) retrofitting approach. We also postulate that some of our current design choices might hurt the performances of GAE/VGAE-inspired models. In particular, in future experiments, one might try to improve the results of Table 10.3 by including DBpedia's *sameAs* cross-lingual links, instead of training GAE/VGAE-inspired models on a graph consisting of six monolingual isolated subgraphs.

Also, while we only separated "equivalence" edges from "relatedness" edges in our encoders, one might instead optimize different weight matrices for each of our five (or six, with *sameAs*) relation types.

Modeling the Music Genre Perception: Discussion   Our results confirm that using translation to produce cross-lingual music genre annotations is limited, as it does not consider the culturally divergent perception of music genres. Instead, by leveraging language-specific semantic representations, combined with GAE/VGAE or retrofitting, one can model this phenomenon rather well. For instance, from *Milton Cardona*'s music genres in Spanish, *salsa* and *jazz*, our Retrofitting+FT$_\text{sif}$ model correctly predicts the equivalent of *fusion* (フュージョン) in Japanese. Yet, while a thorough qualitative analysis requires more work, preliminary exploration suggests that larger gaps in perception might still be inadequately modeled. For instance, for *Santana*'s album *Welcome* tagged with *jazz* in Spanish, it does not predict *pop* in French.

Regarding the scores per language, we obtained the lowest ones for Japanese as the source. We could explain this by either a more challenging test corpus or still incompatible embeddings in Japanese, possibly because of the quality of the individual embedding models for this language and the completeness of the Japanese music genre ontology. Also, we did not notice improvements for pairs of languages from the same family, e.g., French and Spanish. However, we would need a sufficiently large parallel corpus exhaustively annotated in all languages to reliably compare the performance for pairs of languages from the same family or different ones.

Finally, we noticed that given two languages L_1 and L_2, with more music genre embeddings in L_1 than in L_2 (from both the ontologies and the corpus), the results of mapping annotations from L_1 to L_2 seem always better than the results from L_2 to L_1. This observation explains two trends in Table 10.3. Firstly, the scores achieved for English or Spanish as the source, the languages with the largest number of music genres, are the best. Secondly, the results for the same pair of languages could vary a lot, depending on the role each language plays, i.e., source or target. One possible explanation is that the prediction from languages with fewer music genre tags such as L_2 towards languages with more music genre tags such as L_1 is more challenging because the target language contains more specific or rare annotations. For instance, when checking the results per tag from Dutch to English, we observed that among the tags with the lowest scores, we found *moombahton*, *zeuhl*, or *candombe*. However, other common music genres, such as *latin music* or *hard rock*, were also poorly predicted, showing that other causes exist too. Is the unbalanced number of music genres used in annotations a cultural consequence? Related work [95] seems to support this hypothesis. Then could we design a better mapping function that leverages the unbalanced numbers of music genres in cross-cultural annotations? We will dedicate a thorough investigation of these questions as future work.
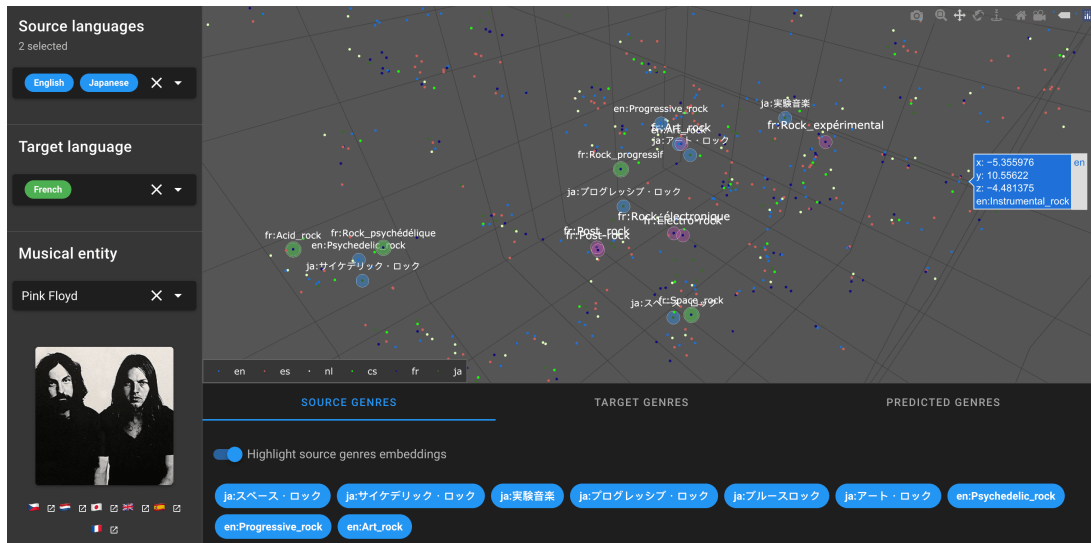
Figure 10.2: The Muzeeglot web interface. Visualization of the music genres associated with the English band Pink Floyd on DBpedia, in English and in Japanese (two pre-selected source languages), together with their Retrofitting+FT$_{sif}$ embedding representations. By clicking on the "Predicting genres" tab, one would display the predicted music genres in French (the pre-selected target language).

10.4.3 The Muzeeglot Web Interface

To support the research presented in this chapter, and in addition to our source code and datasets, our team publicly released the Muzeeglot[12] prototype [84]. As illustrated in Figure 10.2, Muzeeglot is a web interface. Based on a REST API developed in Python with FastAPI and on a frontend developed with VueJS [84] by our research engineer Felix Voituret, and incorporating our models from this research, Muzeeglot permits visualizing multilingual music genre embeddings representations. For some pre-selected music entities from our corpus, and for some pre-selected source and target languages, Muzeeglot displays the corresponding "ground truth" music genres according to DBpedia, along with our predicted genres in the target language(s).

10.5 Conclusion

In this chapter, we aimed to model the music genre perception across language-bound cultures, to address a cross-lingual music genre annotation problem. Our study focused on six languages from four language families, and two common approaches to semantically represent concepts: music genre graph ontologies, and distributed word embeddings. We empirically showed that unsupervised cross-lingual music genre annotation is feasible with high accuracy when combining both types of semantic representations, using either a GAE/VGAE-inspired approach, or an alternative simple yet effective method based on retrofitting.

---

[12]https://github.com/deezer/muzeeglot

This is an important result for music streaming services such as Deezer. Indeed, in future work, our embedding representations could permit tagging music items with genre representations that capture cultural differences, which could improve several genre-related music information retrieval and music recommendation tasks, including the ones mentioned in Section 10.1. Besides, our work also provides a methodological framework to study the annotation behavior across language-bound cultures in other domains. Hence, the effectiveness of language-specific concept representations to model the culturally diverse perception could be further probed.

10.6 Appendices

In this supplementary section, we prove the strict convexity of the retrofitting loss function, as claimed at the end of Section 10.3.4, This result was placed out of the main content of Chapter 10 for the sake of brevity and readability.

Proposition 10.1. Let $\mathcal{V}$ be a finite vocabulary with $|\mathcal{V}| = n$. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an ontology represented as a directed graph which encodes semantic relationships between vocabulary words. Further, let $\hat{\mathcal{V}} \subseteq \mathcal{V}$ be the subset of words which have non-zero initial distributed representations, $\hat{q}_i$. The goal of retrofitting is to learn the matrix $Q \in \mathbb{R}^d$, stacking up the new embeddings $q_i \in \mathbb{R}^d$ for each $i \in \mathcal{V}$. The loss function to be minimized is:

$$\Phi(Q) = \sum_{i \in \hat{\mathcal{V}}} \alpha_i ||q_i - \hat{q}_i||_2^2 + \sum_{i=1}^{n} \sum_{(i,j) \in \mathcal{E}} \beta_{ij} ||q_i - q_j||_2^2, \tag{10.10}$$

where the α_i and β_{ij} are positive scalars. Assuming that each connected component of $\mathcal{G}$ includes at least one word from $\hat{\mathcal{V}}$, the loss function Φ is strictly convex w.r.t. Q.

Proof. First of all, let $\hat{Q}$ denote the $n \times d$ matrix whose i-th row corresponds to $\hat{q}_i$ if $i \in \hat{\mathcal{V}}$, and to the d-dimensional null vector 0_d otherwise. Let A denote the $n \times n$ diagonal matrix verifying $A_{ii} = \alpha_i$ if $i \in \hat{\mathcal{V}}$ and $A_{ii} = 0$ otherwise. Let B denote the $n \times n$ symmetric matrix such as, for all $(i,j) \in \{1, ..., n\}^2$ with $i \neq j$, $B_{ij} = B_{ji} = -\frac{1}{2}(\beta_{ij} + \beta_{ji})$ and $B_{ii} = \sum_{j=1, j \neq i}^{n} |B_{ij}|$. With such a notation, and with $\text{Tr}(\cdot)$ denoting the trace operator for square matrices, we have:

$$\sum_{i \in \hat{\mathcal{V}}} \alpha_i ||q_i - \hat{q}_i||_2^2 = \text{Tr}\Big((Q - \hat{Q})^T A (Q - \hat{Q}) \Big)$$

$$= \text{Tr}\Big(Q^T AQ - \hat{Q}^T AQ - Q^T A\hat{Q} + \hat{Q}^T A\hat{Q} \Big).$$

Also:

$$\sum_{i=1}^{n} \sum_{(i,j) \in \mathcal{E}} \beta_{ij} ||q_i - q_j||_2^2 = \text{Tr}\Big(Q^T BQ \Big).$$

Therefore, as the trace is a linear mapping, we have:

$$\Phi(Q) = \text{Tr}\Big(Q^T(A+B)Q\Big) + \text{Tr}\Big(\hat{Q}^T A\hat{Q} - \hat{Q}^T AQ - Q^T A\hat{Q}\Big).$$

Then, we note that $A+B$ is a weakly diagonally dominant matrix (WDD) as, by construction, $\forall i \in \{1,...,n\}, |(A+B)_{ii}| \geq \sum_{j\neq i} |(A+B)_{ij}|$. Also, for all $i \in \hat{\mathcal{V}}$, the inequality is strict, as $|(A+B)_{ii}| = \alpha_i + \sum_{j\neq i} |B_{ij}| > \sum_{j\neq i} |(A+B)_{ij}| = \sum_{j\neq i} |B_{ij}|$, which means that, for all $i \in \hat{\mathcal{V}}$, row i of $A+B$ is strictly diagonally dominant (SSD). Assuming that each connected component of the graph $\mathcal{G}$ includes at least one node from $\hat{\mathcal{V}}$, we conclude that $A+B$ is a weakly chained diagonally dominant matrix [20], i.e., that:

- $A+B$ is WDD;

- for each $i \in \mathcal{V}$ such that row i is not SSD, there exists a *walk* in the graph whose adjacency matrix is $A+B$ (two nodes i and j are connected if $(A+B)_{ij} = (A+B)_{ji} \neq 0$), starting from i and ending at a node associated with a SSD row.

Such matrices are nonsingular [20], which implies that $Q \to Q^T(A+B)Q$ is a positive-definite quadratic form. As $A+B$ is a symmetric positive-definite matrix, there exists a matrix M such that $A+B = M^T M$. Therefore, denoting $||\cdot||_F^2$ the squared Frobenius matrix norm:

$$\text{Tr}\Big(Q^T(A+B)Q\Big) = \text{Tr}\Big(Q^T M^T MQ\Big) = ||QM||_F^2$$

which is strictly convex w.r.t. Q due to the strict convexity of the squared Frobenius norm [67]. Since the sum of strictly convex functions of Q (first trace in $\Phi(Q)$) and linear functions of Q (second trace in $\Phi(Q)$) is still strictly convex w.r.t. Q, we conclude that the loss function Φ is strictly convex w.r.t. Q. □

Corollary 10.2. The retrofitting update procedure is insensitive to the order in which nodes are updated.

Proof. The aforementioned update procedure for Q [87] is derived from the Jacobi iteration procedure [27, 306] and converges for any initialization. Such a convergence result is discussed in Bengio et al. [27]. It can also be directly verified in our specific setting by checking that each irreducible element of $A+B$, i.e., each connected component of the underlying graph constructed from this matrix, is irreducibly diagonally dominant (see Section 4.2.3 in Saad [306]) and then by applying Theorem 4.9 from Saad [306] on each of these components. Besides, due to its strict convexity w.r.t. Q, the loss function Φ admits a unique global minimum. Consequently, the retrofitting update procedure will converge to the same embedding matrix regardless of the order in which nodes are updated. □

<div align="right">

11

</div>

Carousel Personalization with Contextual Bandits

This chapter presents research conducted with Walid Bendada and Théo Bontempelli, and published as a short paper in the proceedings of the 14th ACM Conference on Recommender Systems (RecSys 2020) [26] where it received a "best short paper" honorable mention.

11.1 Introduction

As explained in the introduction, several other projects less related to GAE and VGAE models were also carried out during the last three years. We chose to present two of them in these last two chapters, as they directly relate to music recommendation, and as they led to online A/B tests and subsequently to model deployment on the Deezer service. They complement the previous chapters by providing a larger overview of some of Deezer's strategies and production-facing algorithms to recommend music. Specifically, in this Chapter 11, we present some research on *carousel personalization* at Deezer.

Many mobile apps and websites, notably from the music streaming industry, leverage *swipeable carousels* to display recommended content on their homepages. These carousels, also referred to as *sliders* or *shelves* [252], consist in ranked lists of items or *cards* (albums, artists, playlists...). A few cards are initially displayed to the users, who can click on them or swipe on the screen to see some of the additional cards from the carousel. Selecting and ranking the most relevant cards to display is a challenging task [125, 155, 242, 252], as the catalog size is usually significantly larger than the number of available slots in a carousel, and as users have different preferences. While being close to *slate recommendation* [160, 173, 350] and to *learning to rank* settings [234, 286, 295], carousel personalization also requires dealing with user feedback to adaptively

improve the recommended content via *online learning* strategies [12, 52, 144], and integrating that some cards from the carousel might not be seen by users due to the swipeable structure. Figure 11.1 provides an illustration of a swipeable carousel on Deezer.

In this chapter, we model carousel personalization as a *multi-armed bandit with multiple plays* [12] learning problem. Within our framework, we account for important characteristics of real-world carousels, notably by considering that services have access to contextual information on user preferences, that they might not know which cards are actually seen by users, and that feedback data from carousels might not be available in real time. We show the effectiveness of our approach by addressing a large-scale carousel-based playlist recommendation task on Deezer. With the paper associated with this work [26], we released industrial data from our experiments and an open-source environment to simulate comparable carousel personalization learning problems.
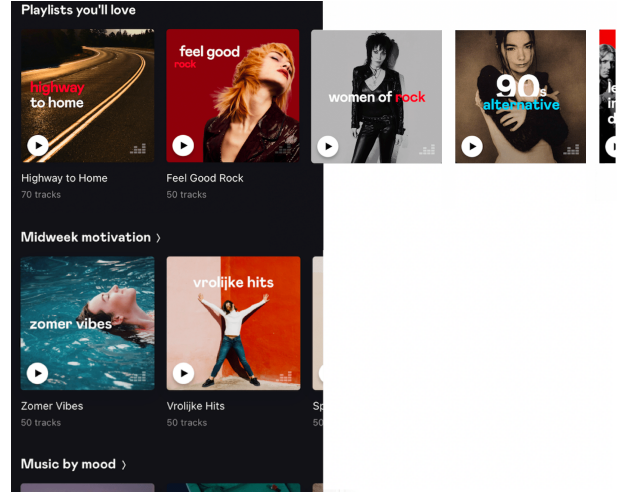

Figure 11.1: An example of a personalized swipeable carousel on the Deezer mobile app, to recommend playlists. These playlists were created by professional curators from Deezer with the purpose of complying with a specific music genre, cultural area, or mood. A few playlists are initially displayed to each user, who can click on them or swipe on the screen to see some of the additional recommended playlists from the carousel.

This chapter is organized as follows. In Section 11.2, we introduce and formalize our multi-armed bandit framework for carousel personalization. We detail our data, our playlist recommendation task and our experimental setting in Section 11.3. We present and discuss our results in Section 11.4, and we conclude in Section 11.5.

11.2 A Contextual Multi-Armed Bandit Framework for Carousel Personalization

In this section, we review key notions on multi-armed bandits and introduce our framework. We note that this Chapter 11, as the upcoming Chapter 12, will introduce quite different concepts w.r.t. those studied in previous chapters on GAE and VGAE models. We will therefore need to introduce a new *notation* for several of these concepts. Nonetheless, when possible, we will try to remain consistent w.r.t. previous chapters (e.g., K will still denote the number of clusters; d will denote the dimension of embedding vectors, etc).

11.2. A Contextual Multi-Armed Bandit Framework for Carousel Personalization

11.2.1 Background on Multi-Armed Bandits with Multiple Plays

Multi-armed bandits are among the most famous instances of sequential decision making problems [338, 349]. Multi-armed bandits *with multiple plays* [12, 194] involve N_{arm} entities called *arms*. At each round $t = 1, 2, ..., T$, a *forecaster* has to select *a set* $S_t \subset \{1, ..., N_{\text{arm}}\}$ of $L < N_{\text{arm}}$ arms (while $L = 1$ in the *single play* version of the problem [338]). The forecaster then receives some *rewards* from the selected arms, that we assume to be binary. The reward associated with an arm $i \in S_t$ is a sample drawn from a Bernoulli(p_i) distribution, with $p_i \in [0, 1]$ being an unknown parameter. Bernoulli distributions of arms $1, ..., N_{\text{arm}}$ are assumed independent, which we later discuss.

The objective of the forecaster is to maximize the sum of rewards received from the selected arms over time. It requires identifying the optimal set:

$$\Omega^*(L) \subset \{1, ..., N_{\text{arm}}\} \tag{11.1}$$

of the L arms associated with the top-L highest Bernoulli parameters, i.e., the L highest expected rewards, as fast as possible.

In such problems, the forecaster faces an *exploration-exploitation dilemma*. As the environment does *not* reveal the rewards of the unselected arms, the forecaster needs to try all arms over time to identify the best ones (*exploration*). However, selecting underperforming arms also leads to lower expected rewards, which encourages the forecaster to repeatedly select the assumed best ones (*exploitation*). Over the past years, several strategies have been proposed and studied, providing efficient trade-offs between these two opposite objectives when sequentially selecting sets S_t. Notable examples include the Upper Confidence Bound (UCB) [18, 52, 203, 379] and Thompson Sampling (TS) [48, 194, 355] algorithms (see Section 11.3). The *expected cumulative regret*:

$$\text{Reg}(T) = \sum_{t=1}^{T} \Big(\sum_{i \in \Omega^*(L)} p_i - \sum_{i \in S_t} p_i \Big), \tag{11.2}$$

which represents the expected total loss endured by the forecaster by selecting non-optimal sets of arms at rounds 1 to T, is a common measure to compare the performances of strategies addressing this top-L best arms identification problem [12, 52, 194, 338, 349, 379].

11.2.2 Carousel Personalization with Multi-Armed Bandits

Throughout this chapter, the N_{arm} arms will correspond to a list of N_{arm} cards/items, such as a catalog of albums or playlists in a music streaming app. They can be recommended to n users through a swipeable carousel containing $L \ll N_{\text{arm}}$ slots. As users have various preferences, different cards can be displayed to different users. The L recommended cards from the carousel

of each user, i.e., the L *selected arms* for each user, are updated at regular intervals or *rounds*, whose frequency depends on the technical constraints of the platform.

We aim to optimize *display-to-stream rates*, i.e., to identify the L cards for which each user is the most likely to click and then to *stream* the underlying content, *at least once* during the round. When a card i is displayed to a user u, such streaming activity, i.e., a reward of 1, occurs during the round with an unknown probability $p_{ui} \in [0, 1]$. Here, we assume that the number of cards, the number of users, and the display-to-stream probabilities p_{ui} are fixed. We later discuss these assumptions.

A naive way to tackle this problem would consist in simultaneously running n standard bandit algorithms, aiming to individually identify the top-L cards with the highest p_{ui} probabilities for each user u. This approach is actually unsuitable and would require a too long training time to reach convergence. Indeed, the number of display-to-stream parameters to estimate would be $N_{\mathrm{arm}} \times n$, which is very large in practice as platforms often have millions of active users. In Section 11.2.3, we describe two strategies to address this problem by leveraging *contextual information* on user preferences.

11.2.3   Leveraging Contextual Information on User Preferences

Semi-Personalization via User Clustering   Firstly, let us assume that we have access to a *clustering of users*, constructed from users' past behaviors on the platform. Each user belongs to one of the K groups $C_1, C_2, ..., C_K$ with $K \ll n$. For instance, on a music streaming app, users from the same group would have homogeneous musical tastes. We propose to assume that users from the same group have identical expected display-to-stream probabilities for each card:

$$\forall c \in \{C_1, ..., C_K\}, \forall u \in c, \forall i \in \{1, ..., N_{\mathrm{arm}}\}, p_{ui} = p_{ci}. \tag{11.3}$$

Then, we run K bandit algorithms, one for each cluster, to identify the top-L best cards to recommend to each group. This strategy reduces the number of parameters to estimate to $N_{\mathrm{arm}} \times K$, which is significantly fewer than $N_{\mathrm{arm}} \times n$ in practice. Moreover, thanks to such users gathering, platforms receive more feedback on each displayed card w.r.t. the previous naive setting. This ensures a faster and more robust identification of optimal sets. However, the empirical performance of this strategy also strongly depends on the quality of the underlying user clustering.

Contextual Multi-Armed Bandits   Instead of relying on clusters, let us now assume that we directly have access to a d-dimensional attribute vector $x_u \in \mathbb{R}^d$ for each user u. These vectors aim to summarize user preferences on the platform, e.g., their musical tastes (in terms of genres, moods, countries...) for a music streaming app. We assume that the expected display-to-stream

probabilities of a user u are functions of his/her attribute vector:

$$\forall i \in \{1, ..., N_{\mathrm{arm}}\}, p_{ui} = \sigma(x_u^T \theta_i), \tag{11.4}$$

where the $\theta_i \in \mathbb{R}^d$ are d-dimensional weight vectors to learn for each of the N_{arm} arms, and where $\sigma(\cdot)$ is the sigmoid function: $\sigma(x) = 1/(1 + e^{-x})$. This corresponds to the *contextual bandit* setting [6, 57, 219], a popular learning paradigm for online recommender systems [125, 219, 220, 252, 293, 352, 377, 410, 412]. Strategies to learn weight vectors are detailed, e.g., in [48, 252]. As $d \ll n$ in practice, such strategy also significantly reduces the number of parameters, to $N_{\mathrm{arm}} \times d$. By design, users with similar preferences will have close expected display-to-stream probabilities. Moreover, all n users can end up with different optimal carousels, contrary to the semi-personalized clustering approach.

11.2.4 Capturing Characteristics of Real-World Carousels

In our framework, we also aim to capture other important characteristics of real-world swipeable carousels. In particular, while standard bandit algorithms usually consider that the forecaster receives rewards (0 or 1) from *each* of the L selected arms at each round, in our setting some selected cards might actually *not* be seen by users. As illustrated in Figure 11.1, only a few cards, say $L_{\mathrm{init}} < L$, are initially displayed on a user's screen. The user needs to swipe right to see additional cards. As we later verify, ignoring this important aspect, and thus returning a reward of 0 for all unclicked cards at each round whatever their rank in the carousel, would lead to underestimating display-to-stream probabilities.

In this chapter, we assume that we do *not* exactly know how many cards were seen by each user. Such an assumption is consistent with Deezer's actual usage data and is realistic. Indeed, on many real-world mobile apps carousels, users do not click on any button to discover additional cards, but, instead, need to continuously swipe left and right on the screen. As a consequence, the card display information is ambiguous, and is technically hard to track with accuracy.

To address this problem, we consider and later evaluate a *cascade-based arm update* model. We draw inspiration from the *cascade model* [64], a popular approach to represent user behaviors when facing ranked lists of recommended items in an interface, with numerous applications and extensions [176, 201, 202, 413]. At each round, we consider that:

- an active user who did not stream any card during the round only saw the L_{init} first ones;

- an active user who streamed the ith card, with $i \in \{1, ..., L\}$, saw all cards from ranks 1 to $\max(L_{\mathrm{init}}, i)$.

For instance, let $L_{\mathrm{init}} = 3$ and $L = 12$. The reward vectors obtained from users who a) did not stream during the round, b) only streamed the 2nd card, and c) streamed the 2nd and 6th cards,

are as follows, with X denoting no reward:

$$\begin{aligned}
a &: \quad [0, 0, 0, X, X, X, X, X, X, X, X, X] \\
b &: \quad [0, 1, 0, X, X, X, X, X, X, X, X, X] \\
c &: \quad [0, 1, 0, 0, 0, 1, X, X, X, X, X, X]
\end{aligned} \tag{11.5}$$

Lastly, to be consistent with real-world constraints, we assume that rewards are not processed on the fly but by *batch*, at the end of each round, e.g., every day. We study the impact of such *delayed batch feedback* in our upcoming experiments.

11.2.5 Related Work

Bandits are very popular models for online recommendation [219, 220, 271, 286, 293, 295, 352, 377, 410]. In particular, McInterney et al. [252] and Gruson et al. [125] also recently studied carousel personalization in mobile apps. McInterney et al. [252] introduced a contextual bandit close to ours. However, their approach focuses more on explainability, they do not model cascade-based displays as we did in Section 11.2.4, and do not integrate semi-personalized strategies. Gruson et al. [125] also considered contextual bandits inspired by [252] for playlist recommendation in carousels, but did not provide details on their models. They instead aimed to predict the online ranking of these models from various offline evaluations. Last, other different sets of ordered items have been studied [160, 168, 173, 350, 413].

11.3 Application to Carousel Personalization on Deezer

In the following, we empirically evaluate and discuss the effectiveness of our framework.

11.3.1 Experimental Setting: Carousel-Based Playlist Recommendation

We study a large-scale carousel-based *playlist recommendation* task on the Deezer mobile app. We consider $N_{\text{arm}} = 862$ playlists, that were created by professional curators from Deezer with the purpose of complying with a specific music genre, cultural area, or mood, and that are among the most popular ones on the service. Playlists' cover images constitute the cards that can be recommended to users on the app homepage in a carousel, updated on a daily basis, with $L = 12$ available slots and $L_{\text{init}} = 3$ cards initially displayed.

To determine which method would best succeed in making users click and stream the displayed playlists, extensive experiments were conducted in two steps. Firstly, *offline* experiments simulating users' responses to carousel-based recommendations were run, on a simulation environ-

ment and on data that we both publicly release[1] with our paper (see Section 11.3.2). We believe that such industrial data and code release will benefit the research community. Then, an *online* large-scale A/B test was run on the Deezer app to validate the findings of offline experiments.

11.3.2 A Simulation Environment and Dataset for Offline Evaluation

For offline experiments, we designed a simulated environment in Python based on 974 960 fully anonymized Deezer users. We release a dataset in which each user u is described by a feature vector x_u of dimension $d = 97$, computed internally by factorizing the interaction matrix between users and songs as described in [152] and then adding a bias term. A k-means clustering with $K = 100$ clusters was also performed to assign each user to a single cluster. In addition, for each user-playlist pair, we release a "ground truth" display-to-stream probability $p_{ui} = \sigma(x_u^T \theta_i)$ where, as in the work of Chapelle and Li [48], the d-dimensional vectors θ_i were estimated by fitting a logistic regression on a click data history from January 2020.

Simulations proceed as follows. At each round, a random subset of users (20 000, in the following) is presented to several sequential algorithms a.k.a. policies to be evaluated. These policies must then recommend an ordered set of $L = 12$ playlists to each user. Streams, i.e., positive binary rewards, are generated according to the aforementioned display-to-stream probabilities and to a configurable cascading browsing model capturing that users explore the carousel from left to right and might not see all recommended playlists. At the end of each round, all policies update their model based on the set of users and on binary rewards received from displayed playlists. Expected cumulative regrets of policies [349] w.r.t. the optimal top-L playlists sets according to p_{ui} probabilities are computed.

11.3.3 List of Multi-Armed Bandit Algorithms

In our experiments, we evaluate semi-personalized versions of several popular sequential decision making algorithms/policies, using the provided $K = 100$ clusters, and compare their performances against fully-personalized methods. As detailed in Section 11.2.3, users within a given cluster share parameters for all semi-personalized policies; they are the ones whose names end with *-seg* in the following list. We consider the following methods:

- *random*: a simple baseline that randomly recommends L playlists to each user.

- *ε-greedy-seg*: recommends playlists randomly with probability ϵ, otherwise recommends the top-L with the highest mean observed rewards. Two versions, *ε-greedy-seg-explore* (ϵ=0.1) and *ε-greedy-seg-exploit* (ϵ=0.01) are evaluated.

---

[1] Data and code are available at: https://github.com/deezer/carousel_bandits

- *etc-seg*: an *explore then commit* strategy, similar to *random* until all arms have been played n times, then recommends the top-L playlists. Two versions, *etc-seg-explore* ($n = 100$) and *etc-seg-exploit* ($n = 20$) are evaluated.

- *kl-ucb-seg*: the Upper Confidence Bound (UCB) strategy [18, 52, 203], that tackles the exploration-exploitation trade-off by computing confidence intervals for the estimation of each arm probability, then selecting the L arms with the highest upper confidence bounds. Here, we use KL-UCB bounds [104], tailored for Bernoulli rewards.

- *ts-seg*: the Thompson Sampling (TS) strategy [48, 355], in which estimated display-to-stream probabilities are samples drawn from Beta distributions [355], whose parameters are updated at each round in a Bayesian fashion, such that variance tends towards zero and expectation converges to empirical mean as more rewards are observed. Two versions, *ts-seg-naive* (prior distributions are Beta$(1, 1)$, i.e., Uniform$(0, 1)$) and *ts-seg-pessimistic* (priors are Beta$(1, 99)$) are evaluated. As the UCB algorithm [52], TS is backed by strong theoretical guarantees [194] on speeds of expected cumulative regrets in the multi-armed bandit with multiple plays setting.

- *ts-lin*: an extension of TS [48] to the linear contextual framework from Section 11.2.3. We follow the method of Chapelle and Li [48] to learn θ_i vectors for each arm i from Gaussian prior distributions. Two versions, *ts-lin-naive* (0 means for all dimensions of the prior) and *ts-lin-pessimistic* (-5 mean for the bias dimension prior) are evaluated.

By default, policies always abide by the cascade model introduced in Section 11.2.4, meaning they do not update the parameters relative to recommended playlists that the cascade model labels as unseen. For comparison, we also implemented versions of these policies that do not abide by this behavior. In the following, they are labeled *no-cascade*.

11.4   Experimental Results

11.4.1   Offline Evaluation

Semi-Personalization vs Personalization   Figure 11.2 provides cumulative regrets over 100 rounds for the different policies, recommending playlists via our offline environment. Both *etc-seg-explore* and *etc-seg-exploit* behave as badly as *random* in the exploration phase, then, shortly after starting to exploit, they both reach competitive performances as illustrated by the brutal flattening of their cumulative regret curves, with *etc-seg-exploit* transitioning 50 rounds earlier. The latter strategy also outperforms *kl-ucb-seg*, which shape suggests slow learning throughout the whole experiment. Moreover, both *ts-lin-pessimistic* and *ts-lin-naive* appear to stabilize to non-flat linear cumulative regret curves after only a few rounds. Pessimistic policies are overall
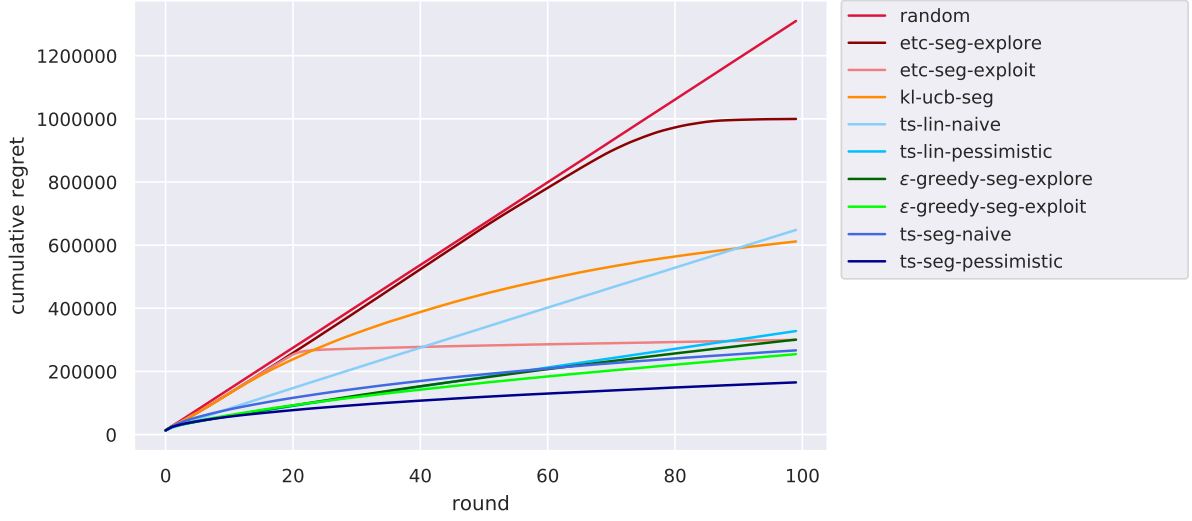
Figure 11.2: Offline evaluation of top-12 playlist recommendation using our simulation environment: expected cumulative regrets of policies over 100 simulated rounds. The empirical gain of *ts-seg-pessimistic* w.r.t. others is statistically significant at the 1% level (p-value < 0.01).

more effective than their naive counterparts, which is due to their lower prior display-to-stream probabilities, which are more realistic. Overall, several semi-personalized policies eventually outclassed fully-personalized alternatives, with *ts-seg-pessimistic* already outperforming them all at the end of the first 25 rounds[2]. This method manages to effectively exploit information and to quickly *rank* playlists, which is an interesting result, as fully-personalized contextual models were actually the only ones able to learn the *exact* display-to-stream probabilities (see generative process in Section 11.3.2), and as both frameworks have comparable numbers of parameters ($N_{arm} \times K$ vs $N_{arm} \times d$). While fully-personalized methods have been the focus of previous works on carousel recommendation [125, 252], our experiments emphasize the empirical benefit of semi-personalization via user clustering that, assuming good underlying clusters, might appear as a suitable alternative for such large-scale real-world applications.

Impact of Delayed Batch Feedback In our experiments, to be consistent with real-world constraints, rewards are not processed on the fly but by batch, at the end of each round. We observe that, for semi-personalization, such setting tends to favor stochastic policies, such as the *ts-seg* or *ϵ-greedy-seg* ones, w.r.t. deterministic ones such as *kl-ucb-seg*. Indeed, as *kl-ucb-seg* selects arms in a deterministic fashion, it always proposes the same playlists to all users of the same cluster until the round is over. On the contrary, stochastic policies propose different playlists sets within the same cluster, ensuring a wider exploration during the round, which might explain why *kl-ucb-seg* underperforms in our experiments.

---

[2]We point out that, in a recent work based on our data and simulation environment, Jeunen and Goethals [167] nonetheless managed to improve the performance of *ts-lin-pessimistic* by adopting different hyperparameters.

Cascade vs No-Cascade   All policies from Figure 11.2 abide by the cascade model introduced in Section 11.2.4. In Figure 11.3, we report results from follow-up experiments. They measure the empirical benefit of taking into account this cascading behavior of users when browsing a sequence of playlists. We compared policies to alternatives that ignored the cascade model, and thus returned a 0 reward for all unstreamed playlists at each round, whatever their rank in the carousel. Only two policy pairs are displayed in Figure 11.3 for brevity.
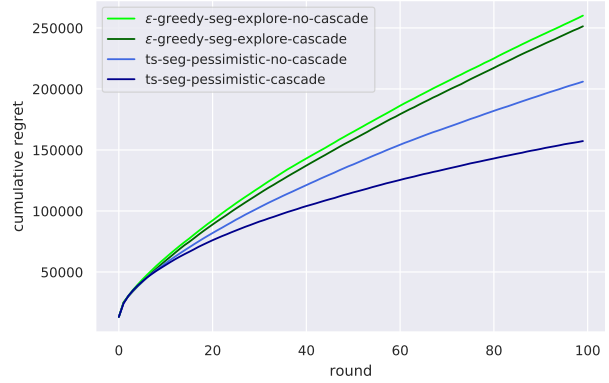
Figure 11.3: Offline evaluation using our simulation environment: comparison of cascade vs no-cascade, over 100 simulated rounds. Differences at final round are statistically significant at the 1% level (p-value < 0.01).

For both of them, the *no-cascade* variant is outperformed by policies integrating our proposed cascade-based update model from Section 11.2.4. This result validates the relevance of capturing such a phenomenon for our carousel-based personalization problem.

11.4.2   Online Evaluation

An industrial-scale A/B test has been run in February 2020 on millions of users, to verify whether results from the simulations would hold on the actual Deezer mobile app. The 12 recommended playlists from each user's carousel were updated on a daily basis on the Deezer app. Due to industrial constraints, only a subset of policies, from (naive) TS, were tested in production. Also, for confidentiality reasons, we do not report the exact number of users involved in each cohort, nor the precise display-to-stream rates. Instead, results are expressed in Figure 11.4 in relative terms.
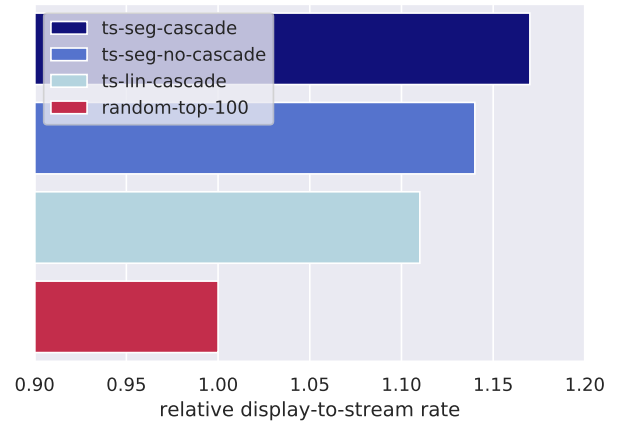
Figure 11.4: Online A/B test evaluation on Deezer: relative display-to-stream gains w.r.t. *random-top-100* baseline (see Section 11.4.2). Differences are statistically significant at the 1% level (p-value < 0.01).

Specifically, results are expressed in terms of relative display-to-stream rates gains w.r.t. *random-top-100*, an internal baseline that randomly recommends 12 playlists from a subset of 100, preselected for each cluster from internal heuristics. Results confirm the superiority of the proposed multi-armed bandit framework for personalization, notably the semi-personalized strategy, and the empirical benefit of integrating a cascade model for arms updates, although users might actually have more complex behaviors on the platform.

11.5 Conclusion

In this chapter, we modeled carousel personalization as a contextual multi-armed bandit problem with multiple plays. By addressing a challenging large-scale playlist recommendation task on Deezer, we highlighted the benefits of our framework, notably the integration of the cascade model and semi-personalization via user clustering.

Along with the paper associated with this work [26], we publicly released a private dataset of user preferences for curated playlists on Deezer, and an open-source environment to recreate comparable learning problems. At the time of writing, this release already benefited research on carousel personalization. Several articles explicitly mentioned our study [90, 91, 94, 236, 390] and/or used our code and data [59, 167] to address various other carousel-based problems.

Despite the promising results presented in this chapter, there is still room for improvement. In particular, we assumed that the number of users and cards was fixed throughout the rounds, which is a limit. In Chapter 12, we will consider the *user cold start* problem and explain how Deezer handles the arrival of new users on the service.

Moreover, future work could also benefit from the advances in GAE and VGAE models presented in previous chapters. For instance, our clustering of users, currently done through a standard k-means, could be improved by applying community detection techniques from Chapters 7 and 9 to a graph of users (e.g., an artificial graph of "similar" users with a comparable listening history, or a social graph of users connected together through "follow" connections on the service). The d-dimensional vector characterizing each user in Section 11.2.3 could also correspond to GAE-based or VGAE-based node embedding representations.

Lastly, in this chapter, we assumed that arms/cards distributions were fixed and independent, which might be unrealistic. A playlist's relative interest might depend on its neighbors in the carousel, and *individually* selecting the top-L playlists does not always lead to the best *set* of L playlists, e.g., in terms of musical diversity. Future research in this direction would definitely lead to the improvement of carousel personalization.

12

A Semi-Personalized System for User Cold Start Recommendation

This chapter presents research conducted with Léa Briand, Walid Bendada, Mathieu Morlon, and Viet-Anh Tran, and published in the proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2021) [39].

12.1 Introduction

A prevalent approach to recommend personalized content on online services such as Deezer is *collaborative filtering* which, broadly, consists in predicting the preferences of a user within a set of items by leveraging the known preferences of some similar users [63, 195, 319, 346]. In particular, several recent works emphasized the empirical effectiveness of *latent models* for collaborative filtering at addressing industrial-level challenges [63, 114, 164, 339]. Analogously to the node embedding methods developed throughout this thesis, these models aim to directly learn vector space representations, i.e., *embeddings* of users and items where proximity should reflect user preferences, typically via the factorization of a user-item interaction matrix [139, 195, 196] or with neural networks architectures processing usage data [63, 260, 372].

However, the performances of these models tend to significantly degrade for new users who only had few interactions with the catalog [42, 227, 319]. They might even become unsuitable for users with no interaction at all, who are absent from user-item matrices in standard algorithms [32, 118, 208]. This is commonly referred to as the *user cold start* problem [32, 208, 227, 319, 367]. Yet, recommending relevant content to these new users is crucial for online services. Indeed, a

new user facing low-quality recommendations might have a bad first impression and decide to stop using the service.

In this last chapter, we present the system recently deployed on Deezer to address this problem. The solution starts from an existing large-scale latent model for collaborative filtering, periodically trained on Deezer's *warm* users (see Section 12.3). It automatically integrates *cold* users into the existing embedding space, by collecting heterogeneous sources of demographic and interaction information on these users at registration day, processed by a deep neural network, and by leveraging a segmentation of warm users to strengthen the final representations and provide semi-personalized recommendations to cold users *by the end of their registration day.*

The proposed system is suitable for an online production use on a large-scale app such as Deezer. Throughout this chapter, we show its practical impact and its empirical effectiveness at predicting the future musical preferences of cold users, through both offline experiments on data extracted from Deezer and an online A/B test on *carousels* from the previous Chapter 11. We also emphasize how this system enables us to provide more interpretable music recommendations. Along with the paper associated with this work [39], we publicly released our source code as well as anonymized usage data of Deezer users from our offline experiments.

This chapter is organized as follows. In Section 12.2, we introduce the user cold start problem more precisely and mention previous research efforts on this topic. In Section 12.3, we present our semi-personalized recommender system. We report and discuss our experimental setting, our data, and our results in Sections 12.4 and 12.5, and we conclude in Section 12.6.

12.2 User Cold Start Recommendation on Music Streaming Apps

In this section, we provide a precise formulation of the problem we aim to address. We also give an overview of the existing related work. Some of the mentioned approaches will constitute relevant baselines to evaluate the effectiveness of our system.

12.2.1 Problem Formulation

Throughout this paper, we consider a catalog of m music tracks available on Deezer. We assume that the catalog remains fixed over time, which we later discuss. At time t, Deezer gathers n_t *warm* users who, according to some criteria internally fixed by our data scientists, had a sufficiently large number of interactions with the catalog, e.g., enough listening sessions, to be used in the training of our recommender systems. We consider a *latent model for collaborative filtering* [63, 195, 196, 260]. From the observed warm user-track interactions and following the processes described in Section 12.3, this model learns a vector space representation of both users and tracks. In this *embedding* space, each user i and track j are represented by d-dimensional

vectors (with $d \ll m$ and $d \ll n_t$), say $u_i \in \mathbb{R}^d$ and $v_j \in \mathbb{R}^d$, capturing musical preferences. To recommend relevant new tracks to the user i, we leverage user-item similarity measures $f(u_i, v_j)$ in this space, encoding user-item affinities. These measures are typically based on an inner-product or a cosine similarity [196]. The model is updated at regular time intervals to take into account the evolution of preferences.

Everyday, new users, referred to as *cold* (analogously to *cold artists* from Chapter 8), will register to the service. They will only have few to no interactions with the catalog during their registration day. As explained in the introduction, a straightforward inclusion of these cold users in the aforementioned latent model is unsuitable [32, 118, 208, 227]. Waiting for them to become warm users, according to internal criteria, is also undesirable: indeed, recommending relevant content as soon as possible is crucial, as new users facing low-quality recommendations might make up their mind on this first impression and quickly stop using the service.

As a consequence, we aim to address the following problem: *given an existing latent model for collaborative filtering learning an embedding space from a set of warm users, how can we effectively include new cold users into this same space, by the end of their registration day on Deezer?* In this chapter, we will evaluate the estimated embedding vectors of cold users by assessing their ability at predicting the future musical preferences of these users on Deezer after their registration day, through the evaluation tasks and metrics presented in Section 12.4.

12.2.2   Related Work

The user cold start problem has initiated significant research efforts over the past decade. In the following, we provide an overview of the most relevant work w.r.t. our approach. We refer the interested reader to some recent surveys [118, 263, 404] for a more exhaustive review of the existing literature, and to [263, 339, 362, 373] as well as Chapter 8 for a presentation of the related *item cold start* problem, which will be out of our scope in most of this Chapter 12.

A prevalent strategy to address the user cold start problem in the total absence of usage data consists in relying on metadata related to new users, and notably on demographic information (such as the age or country of the user) collected during registration [92, 204, 227, 263, 393]. In particular, various approaches aim to cluster warm users, and subsequently assign cold users to existing clusters by leveraging these metadata [42, 89, 227, 263, 346, 393]. Building upon these works, the model we present in Section 12.3 will also leverage demographic information and incorporate a clustering component.

Besides, one can enrich such systems by explicitly asking new users to rate items from the catalog through interview processes, leading to hybrid models based on preferences and side information [118, 263, 331]. On the industry side, Netflix [114] and Spotify [164] are famous examples of services implementing such *onboarding* session for new users. As explained in Section 12.3,

Deezer also adopted this strategy. As the inclusion of an onboarding is not always possible in production, Felício et al. [89] propose to use bandit algorithms to assign cold users to warm user segments, while other studies [229, 325] resort to social media data to connect similar users.

Sometimes, cold users do have a few interactions with the catalog on their registration day. In this case, exploiting such usage signal, in addition to side information, can significantly improve recommendations [32, 138, 319, 346]. In particular, several recent works emphasized the effectiveness of *deep learning* models at dealing with such heterogeneous settings [29, 32, 63, 208, 263, 367, 404]. Notably, Covington et al. [63] explain how a deep neural network, processing various user-item interactions (including watched videos, search queries...) and demographic information to learn embedding vectors, improved the YouTube recommender system. To represent interactions, they average various latent representations of watched/searched items from the same user session, allowing features to have the same dimension for each user. In Section 12.3, we will draw inspiration from their approach to pre-process the user features serving as input to our own model. However, a direct comparison to [63] will be impossible, as no complete description nor implementation of the YouTube recommender system was made publicly available. Several other deep learning approaches were already mentioned in Chapter 8. Here, we provide more details on two of these methods that will be used in our experiments:

- DropoutNet [367] emerged as one of the most powerful latent collaborative filtering models, addressing cold start while preserving performances for warm users. This neural network takes into account usage and content data, and explicitly simulates the cold start situation during training by applying *dropout* [343], alternatively to user and item embedding layers. DropoutNet relies on the assumption that data is *missing at random*, with the risk of introducing biased predictions [231]. Also, it equally considers different types of positive feedback; in Section 12.3, we will furthermore consider negative feedback. Besides, both warm and cold users' embeddings are learned during training, whereas our system will directly incorporate cold users in an existing and fixed embedding space of warm users;

- recently, *meta-learning* methods for cold start have also been proposed [29, 208, 404]. Notably, optimization-based algorithms consider each user as a learning task. From a set of global parameters ensuring an initialization of the recommender system, other local parameters are progressively updated while the user interacts with items to capture his/her preferences. In particular, Lee et al. [208] introduce MeLU (for *Meta-Learned User preference estimator*, already mentioned in Chapter 8), a neural network architecture following such a meta-learning paradigm and learning preferences from the concatenation of user and item information. When new users interact with some items, then *local* parameters of the neural network are updated to refine predictions for these users. In the absence of usage data, users will still be associated with an embedding vector and receive a recommended list of items, thanks to *global* updates of all layers of MeLU.

213
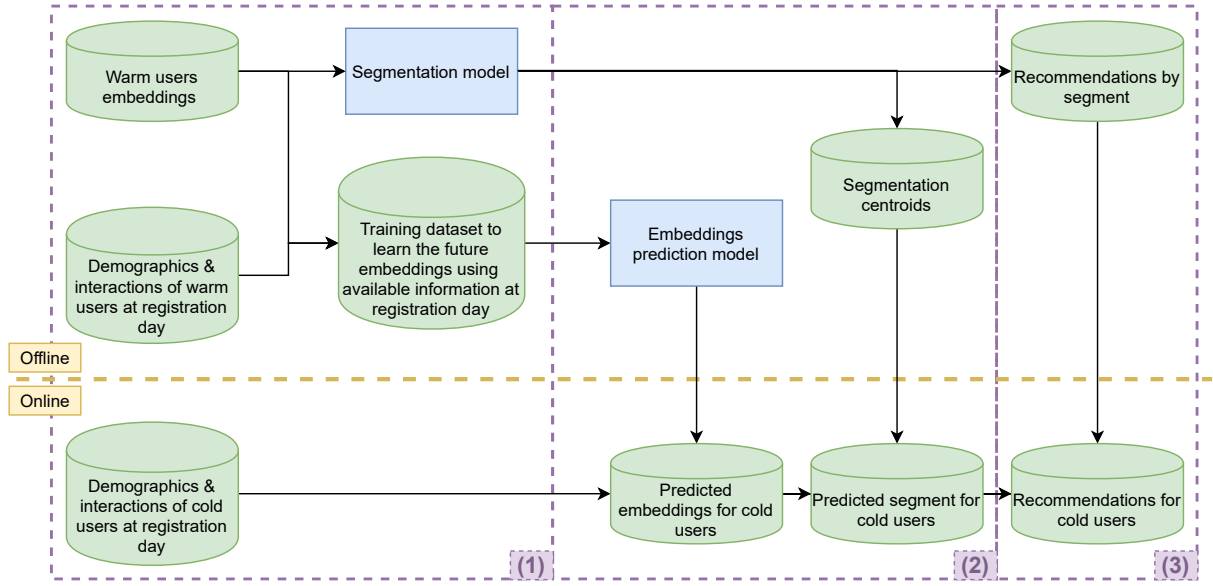
Figure 12.1: The semi-personalized user cold start recommendation framework available for online requests in our production environment, and described throughout Section 12.3. (*1*) Demographics and user-item interactions of a new user are concatenated, as described in Section 12.3.2, to (*2*) predict a user embedding vector from warm user embeddings described in Section 12.3.1. From the estimated user preferences, the new user is assigned to a segment of warm users. (*3*) Combining the online predicted segment with the pre-computed top items by segment, cold users benefit from semi-personalized recommendations.

12.3 A Semi-Personalized System to Address User Cold Start

In this section, we present the system deployed in 2020 on Deezer to address the user cold start problem, as formulated in Section 12.2.1. The architecture of our framework is summarized in Figure 12.1, and discussed thereafter.

12.3.1 Representing Musical Preferences of Warm Users

We recall that our objective is to effectively incorporate, *by the end of their registration day*, a set of cold users in an existing embedding space trained on warm users. In the following, we introduce two different strategies to learn such a space on Deezer's data. Although some technical details on computations are omitted for confidentiality reasons, some embedding vectors from both models will be released with this paper (see Section 12.4). Moreover, experimental results on both spaces will be reported in the next section.

UT-ALS Embeddings   Latent models for collaborative filtering can approximate a preference matrix between users and items from the product of two low-rank matrices, respectively stacking up latent vector representations, a.k.a. embedding vectors, of users and items [195, 346]. At Deezer, we consider a *user-track (UT) interaction matrix* summarizing interactions between

millions of active users and music tracks from the catalog. The *affinity score* between user i and music track j, i.e., the entry (i, j) of the matrix, is computed from various signals, including the number of streams and the potential addition of the music track (or the corresponding album or artist) to a playlist of favorites. The final entry is refined in accordance with internal heuristic rules. Then, we rely on a *weighted matrix factorization*, specifically by using the *alternating least squares* (ALS) method [196], to map both users and music tracks to a joint latent space of dimension $d = 256$. These vector representations will be referred to as *UT-ALS embeddings* in the remainder of this paper.

TT-SVD Embeddings Models inspired by word2vec [256] rely on the *distributional hypothesis* [285] to map items co-occurring in similar contexts to geometrically close embedding vectors. Levy and Goldberg [215] show that word2vec with negative sampling implicitly factorizes a shifted *pointwise mutual information* (PMI) matrix using *singular value decomposition* (SVD) [196]. In this paper, we also consider a PMI matrix, based on the co-occurrences of music tracks in diverse music collections on Deezer, such as music playlists. Then, we factorize this *track-track (TT) matrix* using a distributed implementation of SVD[1], leading to embedding vectors of dimension $d = 128$ for each music track[2]. Finally, we derive embedding vectors for warm users, by averaging music track vectors over their listening history on Deezer. These vector representations, different from *UT-ALS embeddings*, will be referred to as *TT-SVD embeddings* in the remainder of this paper.

Warm User Segmentation On top of UT-ALS or TT-SVD user embeddings, our system also computes a segmentation of warm users, by running a k-means algorithm, with $k = 1\,000$ clusters/segments, in the embedding space. Each user segment is represented by its *centroid*, i.e., by the average of its user embedding vectors. In production, a list of the most popular music items to recommend among each warm user segment is also pre-computed.

12.3.2 Predicting the Preferences of Cold Users

In the following we present our model, illustrated in Figure 12.2, to integrate cold users into these embedding spaces, and subsequently predict their future musical preferences on Deezer.

Overall Strategy Firstly, we gather data from various sources, presented in the next paragraph and referred to as *input features*. They can be collected for warm users and (at least

---

[1]https://github.com/criteo/Spark-RSVD

[2]Embedding dimensions of UT-ALS and TT-SVD have been optimized independently for recommendation, and are therefore different (256 vs 128). The choice of SVD vs ALS factorization is also driven by internal optimizations on Deezer's data. In experiments, we will simply exploit these vectors, independently, for cold start.
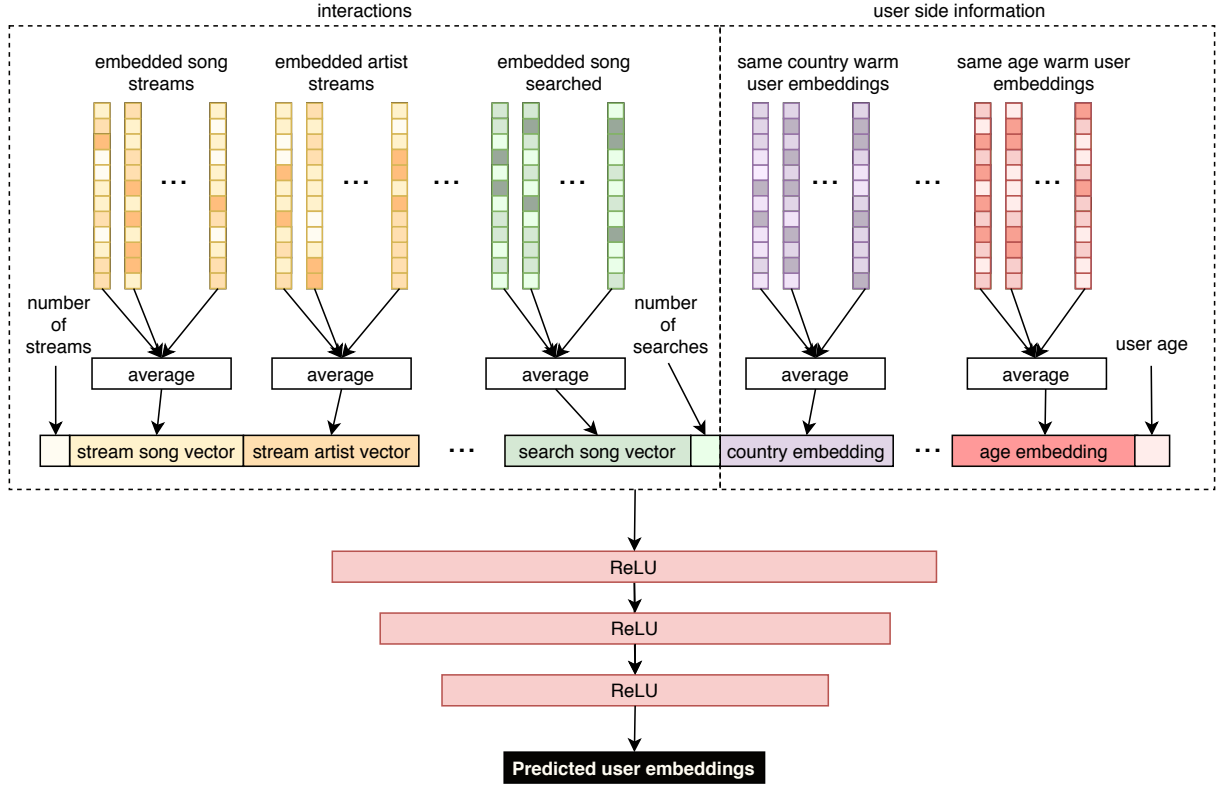
Figure 12.2: Prediction of user embeddings from heterogeneous data. *Top left*: embedding vectors of activated items in usage data, including onboarding (Figure 12.3), are aggregated as described in Section 12.3.2. *Top right*: we enrich representations with demographic information. *Bottom*: after pre-processing the dense input features vector, a deep neural network model, trained as in Section 12.3.2, predicts user embedding vectors in either the UT-ALS embedding space or the TT-SVD embedding space from Section 12.3.1.

partially for) cold users. Then, we train a neural network (see the "Model Training" paragraph thereafter) to map input features of warm users to their (either UT-ALS or TT-SVD) embedding vectors. Last, through a forward pass on this trained neural network, we predict embedding vectors for cold users from their input features. Cold users are therefore integrated into the existing latent space alongside warm users and each track of the catalog. This will permit computing cold user-track similarities, and even similarities between cold and warm users, which we leverage for clustering (see the "Semi-Personalization" paragraph thereafter).

Input features During registration on Deezer, all users specify their age and country of origin, which we include in input features. This information is enriched with *country embedding* and *age embedding* vectors which, as illustrated in Figure 12.2, are the average of embedding vectors of warm users from respectively the same country and age class. As hybrid models mentioned in Section 12.2.2, we complement this side information with data retrieved from user-item interactions, limiting to interactions at registration day (if any). These interactions include positive or negative) *explicit* and *implicit* signals, including streaming activity, searches,

skips and likes. As the granularity of items is crucial in music [319], we also compute such signals at the album, artist, and playlist levels, deriving embedding vectors for such music entities by averaging the relevant track embeddings (e.g., by averaging the tracks of an album, or part of the discography of an artist).

Then, for each type of interaction and music entity, embedding vectors are averaged, as illustrated in Figure 12.2. We obtain *fixed-size representations* for both demographics and interactions, i.e., independent of the number of modalities or interactions, which is crucial for scalability in a production environment. For some new users, some (or possibly all) types of interactions at registration day may be missing; the corresponding representations are replaced arbitrarily by null vectors. To avoid such a situation, Deezer proposes an *onboarding* process for newly registered users, proposing them to add artists from various music genres to their list of favorites, as illustrated in Figure 12.3.
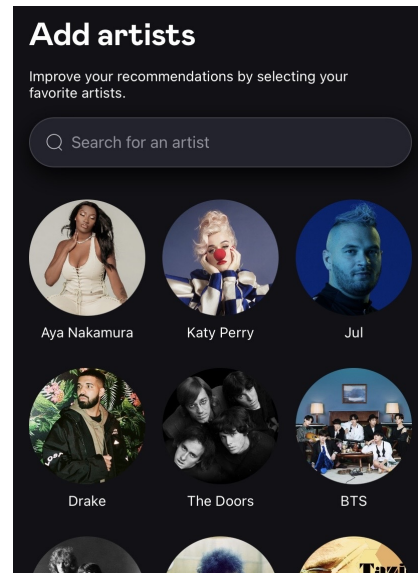

Figure 12.3: Overview of the *onboarding* process, displayed to new users on the Deezer service.

Model Training   Fixed-size representations of demographics and interactions are concatenated to form a unique dense input vector of dimension 5139 (when considering UT-ALS embeddings) or 2579 (TT-SVD embeddings). It constitutes the input layer of a feedforward neural network, with three hidden layers of dimensions 400, 300, and 200, respectively, and an output layer of dimension $d = 256$ (when considering UT-ALS embeddings) or 128 (TT-SVD embedding). We use ReLU activations at each layer except the output, followed by *batch normalization* [117]. We train the model on warm users, by iteratively minimizing the *mean squared error* between the predicted user embeddings and their actual value in UT-ALS or TT-SVD spaces, by *stochastic gradient descent* [117] with a learning rate of 0.001, batch sizes of 512, and 100 (respectively 130) epochs for TT-SVD (resp. UT-ALS).

Semi-Personalization   Our model integrates cold users into the existing embedding space alongside warm users and music tracks. Therefore, one could provide *fully personalized* music recommendation to each of those cold users, by retrieving the most *similar* tracks for each user via an exact or an approximate *nearest neighbors* search and some similarity measure. However, as we will empirically show in our experiments, such a strategy can still lead to noisy results for users with very few to no usage data. As a consequence, our system instead adopts a *semi-personalized* recommendation strategy. On top of our neural network predictions, we include

cold users into the pre-computed *warm user segmentation* previously described. Specifically, each cold user is assigned to the warm cluster whose centroid is the closest w.r.t. the predicted embedding vector of this cold user. We subsequently recommend the pre-computed most popular tracks among warm users from the cluster. Our framework is summarized in Figure 12.1.

Model Deployment   This system is suitable for online production use. At Deezer, the real-time inference service to predict user embeddings is a Golang web server, deployed in a Kubernetes cluster. The web service wraps the onnxruntime library[3], a fast engine for running ONNX machine learning models. It permits fast predictions of cold users' embeddings via forward passes on already trained neural networks. Models are trained offline using PyTorch, on an NVIDIA GTX 1080 GPU and an Intel Xeon Gold 6134 CPU, and then exported to ONNX format and stored on Hadoop. Embeddings of music tracks (from which we also derive embeddings of artists, albums, or playlists), as well as warm segment centroids, are exported weekly in tables in a Cassandra cluster, exposed via a JSON REST service. Embeddings and serialized models are weekly updated to take into account changes in the catalog and preferences, and weekly exported as well.

12.4   Application to Offline Prediction of Listening Data on Deezer

In the remainder of this chapter, we evaluate the performance and impact of our system on our data. Firstly, in this section, we focus on offline experiments.

12.4.1   Experimental Setting: Predicting Future Preferences of Cold Users

Our system permits recommending musical content to cold users. Through experiments on an offline dataset of Deezer active users, described thereafter, we evaluate to which extent the proposed recommendations at registration day would have matched the actual musical preferences of a set of users on their first month on the service. Specifically, we compute the 50 most relevant music tracks for each user of the dataset, from our model and registration day's input features (described in the previous section). We compare them to the tracks listened to by each user during their next 30 days on Deezer, using three standard recommendation metrics: the *Precision*, the *Recall*, as well as the *Normalized Discounted Cumulative Gain* (NDCG) as a measure of ranking quality [319].

---

[3]https://www.onnxruntime.ai/

12.4.2   Dataset and List of Models

Dataset   For offline experiments, we extracted a dataset of 100 000 fully anonymized Deezer users. Among them, 70 000 are *warm* users. They are associated with demographic information (country and self-reported age), as well as their respective UT-ALS and TT-SVD embedding vectors. These vectors correspond to those actually computed by our latent collaborative filtering models on the Deezer production system on November 1st, 2020, from millions of active warm users. Our dataset also includes the UT-ALS and TT-SVD embedding representations of the 50 000 most popular anonymized music tracks on Deezer.

The remaining 30 000 users are *cold* users, who registered on Deezer on the first week of November 2020, and subsequently listened to at least 50 music tracks *on their first month on the service (excluding registration day)*. They are split into a validation set and a test set of respectively 20 000 and 10 000 users. For each cold user, we collected demographic information as well as the list of artists selected during the onboarding, and the lists of available streams, skips, bans, searches, additions to favorites relative to music tracks, artists, albums, and playlists *at registration day only*. 77% of cold users from this dataset streamed at least once at registration day, whereas 95% of cold users fulfilled one of the aforementioned interactions. Thus, for the remaining 5%, only demographic information is available at registration day. Lastly, the dataset includes the tracks listened to by cold users during their next 30 days on Deezer, among the 50 000 tracks.

Along with our paper [39], we publicly released[4] this dataset, as well as the code corresponding to our offline experiments. Besides making our results reproducible, it publicly provides a relevant real-world benchmark dataset to evaluate and compare future cold start models on actual (albeit anonymized) usage data. We, therefore, hope that this open-source release of industrial resources will favor and benefit future research and applications on user cold start problems.

Models   We report results from two versions of our system (one trained on the UT-ALS embeddings of the 70 000 warm users, and one trained on their TT-SVD embeddings) on the task presented in Section 12.3.1. For each embedding space, we simultaneously evaluate:

- the *semi-personalized* recommendations, which are actually used in production at Deezer. In this case, the 50 recommended tracks of each user will correspond to the 50 most popular tracks of his/her *user segment*, as detailed in 12.3.2;

- the *fully-personalized* ones, which directly leverage the predicted embedding vectors of each cold user from the neural network. In this case, we recommend, for each cold user, the 50 *nearest neighbors* music tracks w.r.t. his/her vector in the embedding space, according to a cosine similarity.

---

[4]Data and code are available on: https://github.com/deezer/semi_perso_user_cold_start

Moreover, although the main objective of the applied paper associated with this work [39] was to show the very practical impact of a deployed system and not to chase the state of the art, we also report the performances of several baseline methods as a point of comparison. Foremost, we consider a *Popularity* baseline. This chart-based method recommends the most popular songs to cold users. Besides, to motivate the need for a careful modeling of user cold start, we consider the *Registration Day Streams* method, which includes cold users in the embedding in a more straightforward way. It estimates a cold user's embedding vector by averaging embedding vectors of music tracks listened to at registration day (and relying on popularity in the absence of any stream), then recommends 50 tracks to each cold user via a nearest neighbors search. Moreover, a third ablation baseline, denoted *Input Features Clustering* in the following, will consist in getting rid of our neural network model and directly rely on the *input features* of Section 12.3.2. This method will also cluster all users into segments via a *k*-means, but from their stacked input features, i.e., the large vector illustrated in Figure 12.2. Recommended tracks will correspond to the 50 most popular among warm users of each cluster; as users are no longer in the same space as tracks, we do not evaluate any fully-personalized version of this baseline.

Lastly, we evaluate DropoutNet [367] and MeLU [208], two deep learning models described in Section 12.2.2. We selected these two methods as they are simultaneously *1)* among the most promising approaches, to the best of our knowledge, *2)* scalable to large datasets, and *3)* publicly available online[5]. They process the same input features as our model, with the notable exception that DropoutNet only processes positive user-track interactions (i.e., not skips nor bans, representing approximately 12% of all interactions in our dataset). We carefully tuned each model using the validation set. For each model, we simultaneously evaluate *fully-personalized* recommendations where, as for our system, we recommend to each cold user his/her 50 most similar music tracks, as well as *semi-personalized* recommendations leveraging a warm user segmentation similar to ours. We consider two variants of each model from the last two paragraphs, respectively trained on UT-ALS and TT-SVD embeddings.

12.4.3 Offline Evaluation

Table 12.1 reports performance scores of all models on the UT-ALS and TT-SVD embeddings, along with standard deviations over ten iterations. As the *Popularity* baseline is independent of embedding vectors, it obtains the same scores for the two embeddings. In both settings, *Popularity* is the worst method, although performances are still fairly good for such a simple strategy.

We first focus on TT-SVD embeddings. The *Registration Day Streams* baseline hardly beats *Popularity* for these embeddings, which emphasizes the limits of a direct use of sparse usage data in cold start settings. On the contrary, our proposed semi-personalized system provides

---

[5]Public implementations are available on https://github.com/layer6ai-labs/DropoutNet and https://github.com/hoyeoplee/MeLU respectively.

12.4. Application to Offline Prediction of Listening Data on Deezer

Table 12.1: Offline prediction of the future musical preferences of cold users on Deezer.

Model	TT-SVD Embeddings			UT-ALS Embeddings		
	Precision@50 (in %)	Recall@50 (in %)	NDCG@50 (in %)	Precision@50 (in %)	Recall@50 (in %)	NDCG@50 (in %)
Popularity	8.92 ± 0.21	3.01 ± 0.08	9.72 ± 0.20	8.92 ± 0.21	3.01 ± 0.08	9.72 ± 0.20
Registration Day Streams	9.30 ± 0.22	3.48 ± 0.06	9.73 ± 0.23	16.88 ± 0.46	5.99 ± 0.11	17.72 ± 0.43
Input Features Clustering	8.84 ± 0.22	2.97 ± 0.08	9.75 ± 0.23	8.85 ± 0.22	2.98 ± 0.08	9.75 ± 0.22
DropoutNet Full-Pers.	10.04 ± 0.27	3.75 ± 0.11	10.46 ± 0.29	16.30 ± 0.50	5.77 ± 0.50	17.62 ± 0.54
DropoutNet Semi-Pers.	20.85 ± 0.35	7.55 ± 0.12	22.61 ± 0.36	**19.83 ± 0.32**	6.93 ± 0.16	**21.55 ± 0.44**
MeLU Full-Pers.	15.00 ± 0.40	5.12 ± 0.17	16.79 ± 0.45	13.92 ± 0.36	4.71 ± 0.12	15.49 ± 0.39
MeLU Semi-Pers.	19.66 ± 0.36	6.87 ± 0.15	21.63 ± 0.40	19.35 ± 0.43	6.71 ± 0.14	21.33 ± 0.45
Deezer Full-Pers. (ours)	9.58 ± 0.18	3.53 ± 0.03	9.77 ± 0.17	18.50 ± 0.43	6.63 ± 0.10	20.22 ± 0.41
Deezer Semi-Pers. (ours)	**22.75 ± 0.32**	**8.26 ± 0.15**	**24.59 ± 0.30**	19.00 ± 0.42	**6.93 ± 0.10**	20.38 ± 0.45

significant improvements (e.g., a +14.86 NDCG points increase w.r.t. *Registration Day Streams*), and even reaches competitive results w.r.t. DropoutNet and MeLU. Overall, semi-personalized methods outperform their fully-personalized variants (e.g., a +13.17 precision points increase for *Deezer Semi-Pers.* vs *Deezer Full-Pers.*). This confirms the empirical relevance of our user segmentation strategy, and that, while the studied methods *could* provide fully personalized recommendations to cold users, this strategy can lead to noisier results on real-world applications such as ours. Lastly, our system provides better results than a direct use of the *input features* vector (*Input Features Clustering* baseline), which confirms the relevance of our modeling step on top of this vector. We considered replacing our 3-layer neural network with alternative architectures between *Input Features Clustering* and *Deezer Semi-Pers.*, notably with a 1-layer neural network, i.e., a simpler linear regression model, but did not reach comparable results.

Regarding the UT-ALS embeddings, most conclusions are consistent w.r.t. TT-SVD. *Deezer Semi-Pers.* reaches quite comparable or better results w.r.t. alternatives. Among the key differences, we highlight that, while *Deezer, DropoutNet and MeLU Semi-Pers.* are overperforming, the fully-personalized *Registration Day Streams*, *DropoutNet Full-Pers.* and *Deezer Full-Pers.* obtain significantly stronger results than on TT-SVD embeddings (e.g., a 18.50% precision score for *Deezer Full-Pers.* on UT-ALS, vs 9.58% on TT-SVD). UT-ALS embeddings are constructed from a user-item interaction matrix, an approach appearing as better suited for nearest neighbors search.

To conclude on Table 12.1, we emphasize that, while all scores might seem relatively low, they are actually encouraging considering the intrinsic complexity of the evaluation task (predicting a few listened tracks, among 50 000). Overall, the TT-SVD version of *Deezer Semi-Pers.* reaches the best results. We note that the choice of @50 metrics is not restrictive. We reached consistent model rankings for @25 and @100 scores; the number of items to recommend is a selectable parameter in our public implementation.

To go further, Figure 12.4 reports the mean precision@50 scores obtained by our semi-personalized system trained on TT-SVD embeddings, depending on available interactions at registration day.

We observe that our model returns better results for users who streamed (positive signal) or skipped (negative signal) more tracks at registration day. Liking at least three artists during the onboarding session also improves recommendations, which tends to confirm the relevance of this strategy. On the contrary, our experiments show that users without *any* interaction (5% of them, for which only demographics are available) get a lower average precision@50 score of 17.74%, 5.01 points below the global average.
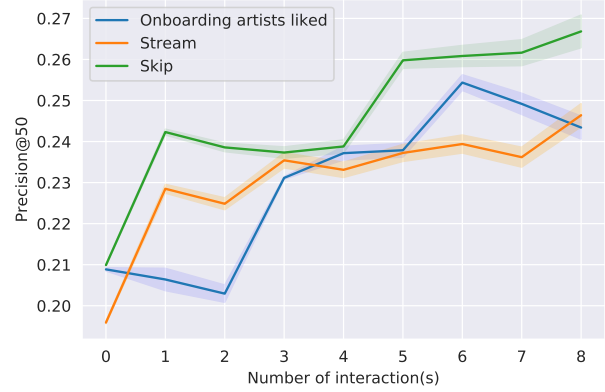
Figure 12.4: Precision@50 scores of Deezer Semi-Pers. depending on the number of artists liked during the onboarding, of streams, and of skips *at registration day.*

Lastly, Figure 12.5 reports the popularity distribution of recommended tracks from our system, trained on TT-SVD embeddings and for users from the top 5 countries on Deezer. We aim to assess whether better performance inevitably means recommending popular tracks, which is referred to as the *popularity bias* [283]. Our semi-personalized system stands out from the popularity baseline, and mainly recommends tracks among the 5 000 most popular in the dataset, out of 50 000.
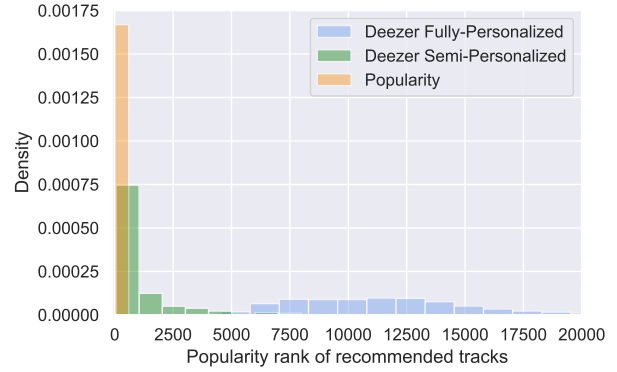

Figure 12.5: Distribution of music tracks recommended to cold users, by popularity rank on Deezer.

The fully-personalized variant of our system permits recommending even less *mainstream* tracks, but, as we showed, this might come at the price of noisier recommendations.

12.5 Application to Online Recommendation on Deezer via Personalized Carousels

We now report our online evaluation on the Deezer app, leveraging *carousels* from Chapter 11.

12.5.1 Experimental Setting: Cold Start Carousel Personalization

In addition to these experiments on data extracted from Deezer, online tests were run to check whether our conclusions would hold on the actual Deezer app. On our homepage, we do not directly recommend music tracks, as in our offline experimental setting, but instead *musical collections* such as albums and music playlists. As a consequence, our online tests will rather consist

in *recommending playlists* to cold users, using the *carousels* [26] fully described in Chapter 11.

We recall that carousels are ranked and swipeable lists of playlists cards. Deezer displays 12 recommended playlists to each user through these carousels. They are updated on a daily basis on the app. The embedding vectors of these playlists are computed from averages of the TT-SVD embeddings of music tracks from each playlist and from internal heuristics. They are leveraged in carousels through a TS extension now in production (technical details are voluntarily omitted). All playlists were created by professional curators from Deezer, with the purpose of complying with a specific music genre, cultural area, or mood.

12.5.2   Online Evaluation and Interpretation of Recommendations

A large-scale A/B test has been run for a month on Deezer in 2020, on new cold users registering during this period. Due to industrial constraints, testing all model variants in production was impossible. Also, for confidentiality reasons, we do not report the exact number of users involved in each cohort.

Results are expressed in *relative* terms w.r.t. the performance of *Deezer Default*, a previous production system estimating cold user embeddings by countries and from internal heuristics. We observe in Figure 12.6 that our new (TT-SVD-based) semi-personalized system leads to significant improvements of the relative *display-to-stream* and *display-to-favorite* rates, i.e., it permits selecting playlists on which cold users are more likely to click on and then to *stream* the underlying content or *add it to their list of favorite content*. Such results validate the relevance of our proposed system, and emphasize its practical impact on industrial-scale applications in a global music streaming app such as Deezer. In 2021, the system is still used in production at Deezer.
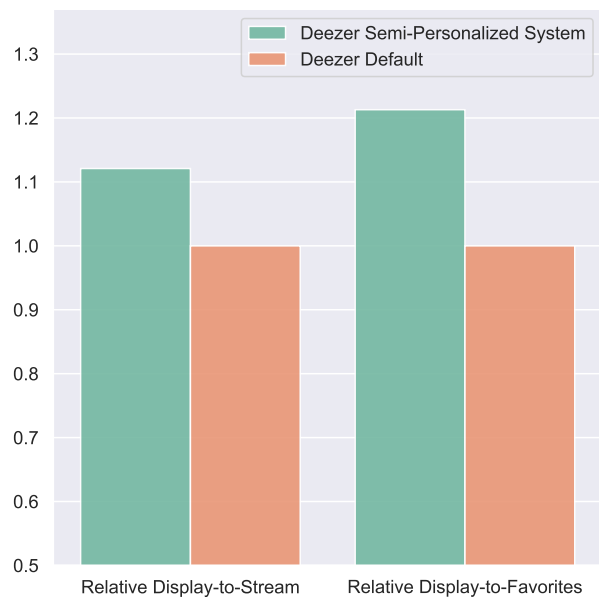
Figure 12.6: Online A/B test: relative *display-to-stream* and *display-to-favorites* rates w.r.t. internal baseline. Differences are statistically significant at the 1% level (p-value < 0.01).

Our semi-personalized system also enables us to provide *interpretable* recommendations on Deezer. Indeed, carousel personalization relies on user embedding vectors that, for cold users, are linked to centroids of warm user segments. Therefore, recommended playlists can be described via the characteristics of these segments, such as the most common country or age class of warm users from each segment, or the most common music genres among their favorite artists.

Table 12.2: Examples of user segments from Deezer's production system, described by most common country, age class and music genres among favorite tracks, together with some playlists recommended on the service.
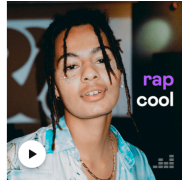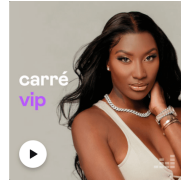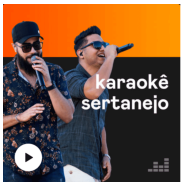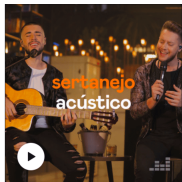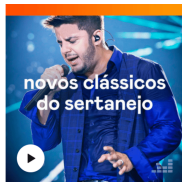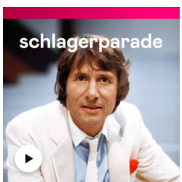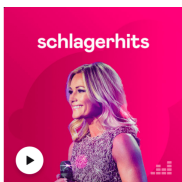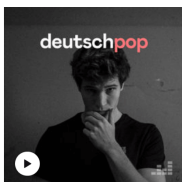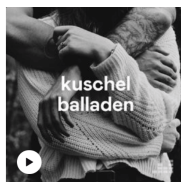
Description of the segment	Playlists recommended on Deezer			
France 18-24 y.o. Rap, Hip-Hop				
Brazil 25-34 y.o. Sertanejo				
Germany 35-49 y.o. Schlager, Pop				

Table 12.2 reports a few illustrative examples of descriptions of user segments, along with music playlists that were actually recommended to cold users from these segments on the Deezer app. Providing interpretable recommendations is often desirable for industrial applications, both for data scientists dealing with opaque model predictions, and for users as a way to improve their satisfaction and trust in the system [5].

12.6 Conclusion

In this chapter, we presented the semi-personalized system recently deployed in production at Deezer to address the challenging user cold start problem. We demonstrated the tangible impact of this system, through both offline and online experiments on music recommendation tasks. Moreover, although the main focus of our work [39] was on practical impact and not on chasing the state of the art, we also showed that our approach is competitive w.r.t. powerful user cold start models from the recent scientific literature.

Along with the paper associated with this work [39], we publicly released our code, as well as the dataset used in our offline experiments, providing information on 100 000 anonymized users and their interactions with the Deezer catalog. We hope that this open-source release of industrial resources will enable future research on user cold start recommendation.

For instance, in this paper, we assumed that, while the number of users increases over time, the musical catalog remains fixed, which is a limit, currently addressed at Deezer via internal

heuristics extending our system. Future studies on a more effective incorporation of new releases in our semi-personalized system would decidedly improve its empirical effectiveness. In this direction, leveraging the GAE and VGAE models from Chapter 8, combined with the FastGAE method from Chapter 4 for scalability, seems to be a promising approach, especially considering the recent successes of graph-based recommender systems on large-scale services such as Alibaba [373] or Pinterest [394], including in cold start settings.

Besides, future studies could also consider different embedding sizes for users and items [138], or the inclusion of temporal and contextual information in our neural network. Lastly, as we rely on user segments, we also believe that future work on a more effective user clustering, e.g., by leveraging graph-based methods, could strengthen our system and permit providing even more refined recommendations to new users.

Conclusion

Graph autoencoders (GAEs) and variational graph autoencoders (VGAEs) emerged as two powerful families of node embedding methods. Nonetheless, at the beginning of this PhD project, i.e., in 2018, existing variants of GAEs and VGAEs were still suffering from several fundamental limitations. As a consequence, leveraging these models for industrial-level applications, e.g., for music recommendation at Deezer, was still a challenging task. The primary objective of this thesis was to address these limitations, with the general aim of improving GAEs and VGAEs, and of facilitating their use for real-world problems involving node embedding representations.

Specifically, we firstly aimed to improve their scalability. While standard GAE and VGAE models were limited to medium-size graphs with a few thousand nodes and edges, due to their quadratic computational complexity, we proposed two effective methods to overcome this issue. They are based on graph degeneracy and stochastic subgraph decoding, respectively. We provided the first application of these models to graphs with up to millions of nodes and edges. Besides, while standard variants of GAEs and VGAEs were limited to undirected graphs, we introduced Gravity-Inspired GAEs and VGAEs, an effective extension to process directed graphs that are ubiquitous in real-world problems. Our models achieved competitive empirical results w.r.t. popular alternatives on several directed link prediction tasks.

Throughout this thesis, we also considered extensions of GAEs and VGAEs for dynamic graphs and graphs with edges of different natures. Furthermore, by introducing Linear GAEs and VGAEs, we emphasized that multi-layer GCN-based GAEs and VGAEs are often unnecessarily complex, and we subsequently proposed to simplify them. This is an important aspect in the context of an industrial thesis, as simpler models are often preferred in production environments. Simultaneously, we provided some recommendations to improve the evaluation of complex GAE and VGAE models. Lastly, we introduced Modularity-Aware GAEs and VGAEs to improve community detection using these models, while jointly preserving their initially good performances on link prediction.

The second objective of this thesis was to evaluate our proposed approaches on industrial graphs extracted from the Deezer service. We put the emphasis on graph-based music recommendation problems, often formulated as link prediction or community detection tasks. In particular, we explained how, to this day, Deezer leverages similarity graphs constructed from usage data for music recommendation. We highlighted how this music streaming service can benefit from the GAE and VGAE models introduced in this thesis, e.g., to improve the detection of artists/albums communities to recommend to users. We also proposed a graph-based approach to rank similar

artists on Deezer, leveraging our Gravity-Inspired GAEs and VGAEs. Using this approach, we successfully addressed the challenging cold start similar artists ranking problem, with an experimental evaluation on data extracted from Deezer's production system. In another research study, we leveraged music genres graph ontologies to effectively model the music genre perception across language-bound cultures. To finish, in the last two chapters of this thesis, we presented two additional projects, less related to GAEs and VGAEs but providing a larger overview of music recommendation at Deezer. We introduced, described, and analyzed some production-facing algorithms developed during the period covered by this PhD project. To this day, they are still running on the Deezer service and recommend music to millions of active users.

Overall, the majority of our experiments led to promising results. We believe and hope that the research conducted over the last three years will benefit some future projects and, in particular, will ease the application of graph autoencoders to large-scale real-world problems involving the use of node embedding representations. To encourage the usage of our methods, we publicly released our code along with each article published in the context of this thesis. Along with five of these articles, we also publicly released new datasets, that were either directly extracted from Deezer's private resources, or scraped and processed from the internet.

Moreover, at the end of each chapter of this thesis, we aimed to describe the potential limitations of our methods. They open the way for future research on the corresponding topics. For instance, we acknowledged that the FastGAE method could underperform on very sparse graphs, and on directed graphs if the method relies on (undirected) core-based or degree-based sampling probabilities. We explained how our recommender systems from Chapters 11 and 12 could be improved by considering the inclusion of graph-based methods in the learning process, and how the ones from Chapters 8 and 9 still require online A/B tests and deployments, undone at the time of writing. Another aspect that we consistently noticed in our experiments regards the absence of clear empirical differences between GAE and VGAE models. While VGAE models seem to slightly outperform their deterministic counterparts in the majority of our community detection experiments, scores of GAE models are nonetheless fairly close. As we conclude this PhD thesis, the question of when one should favor VGAEs over GAEs remains quite open and deserves future research. Lastly, we recall that the GAE and VGAE frameworks are very general. Therefore, in the upcoming years, one could most likely improve our empirical results, e.g., by replacing our encoders (mostly, linear or multi-layer GCN models) with novel GNN architectures from the scientific literature, or by investigating better-suited prior latent distributions and generative models in the case of VGAEs.

The above considerations admittedly constitute relevant research directions for future work. Nonetheless, we believe that the most interesting improvements should come, not necessarily from more complex encoders, but rather from better data and better evaluations. Experiments from this thesis tend to suggest that a better initial representation of nodes could improve the resulting GAE/VGAE-based node embeddings at least as much as a more powerful GNN encoder.

Conclusion

As an illustration, the performances of our GAE and VGAE models from Chapters 8 and 9 strongly depend on how artists or albums are connected in the training graphs under consideration. Therefore, we believe that music streaming services, and other media aiming to leverage such graph-based methods, should redouble their efforts to define better "similarity" scores. As argued throughout this thesis, enriching these training graphs, e.g., with culture-specific information or with contextual information, could also decidedly improve our recommender systems. An even more ambitious direction for future research would consist in developing models that directly *learn* such initial graph connections from the massive amounts of usage data and item descriptions collected on online services, i.e., without any arbitrary pre-processing step or any business-driven assumption on what "similarity" should mean. Concurrently, several of our studies could also benefit from more extensive evaluations. For instance, in the aforementioned Chapter 8, we did not assess the impact of the inclusion of cold nodes in embedding spaces on the *"Fans Also Like"* lists of *warm* artists. In a future study, one could aim to assess whether these cold artists emerge on the recommended lists of warm artists, and therefore to which extent GAE and VGAE models could permit recommending more diverse musical content on music streaming services.

At the beginning of this PhD project three years ago, the graph representation learning field was already growing at a fast pace. As we are now concluding this thesis, we observe a gain of interest for graph representation learning methods in the music streaming industry and, overall, in the research communities working on the specific domains of applications covered at Deezer, i.e., recommender systems and music information retrieval. For instance, at the recent 15th ACM Conference on Recommender Systems (RecSys 2021) where our work received a "best student paper" honorable mention [313], the opening session was marked by a keynote presentation from Max Welling, one of the co-authors of the GCN, GAE and VGAE models [187, 188], on graph neural networks for recommendation. A month later, at the 22nd International Society for Music Information Retrieval Conference (ISMIR 2021) which occurred during the writing of this thesis, researchers from the music streaming service Pandora presented a graph-based approach to compute artist similarities, using a GNN model [197]. Their work also received a "best paper" honorable mention. At the same conference, researchers from the music streaming service Spotify presented another multi-task graph-based model, with application to music recommendation [316]. These recent studies complement this thesis, which mainly focused on GAE/VGAE-based node embedding representations and their applications to link prediction and community detection tasks. In agreement with our research, they confirm the impact, the relevance, and the potential of graph representation learning methods to tackle numerous real-world problems emerging on music streaming services, and, more broadly, on large-scale media providing and recommending content to millions of users.

Bibliography

[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., ET AL. TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* (2016), 265–283.

[2] ABBE, E. Community Detection and Stochastic Block Models: Recent Developments. *The Journal of Machine Learning Research 18*, 1 (2017), 6446–6531.

[3] ADAMIC, L. A. The Small World Web. *Proceedings of the International Conference on Theory and Practice of Digital Libraries* (1999), 443–452.

[4] ADAMS, J. *Gathering Social Network Data.* SAGE Publications Incorporated, 2020.

[5] AFCHAR, D., AND HENNEQUIN, R. Making Neural Networks Interpretable with Attribution: Application to Implicit Signals Prediction. *Proceedings of the Fourteenth ACM Conference on Recommender Systems* (2020), 220–229.

[6] AGARWAL, A., HSU, D., KALE, S., LANGFORD, J., LI, L., AND SCHAPIRE, R. Taming the Monster: A Fast and Simple Algorithm for Contextual Bandits. *Proceedings of the 31st International Conference on Machine Learning* (2014), 1638–1646.

[7] AGGARWAL, C., HE, G., AND ZHAO, P. Edge Classification in Networks. *Proceedings of 32nd IEEE International Conference on Data Engineering (ICDE)* (2016), 1038–1049.

[8] AHMED, A., SHERVASHIDZE, N., NARAYANAMURTHY, S., JOSIFOVSKI, V., AND SMOLA, A. J. Distributed Large-Scale Natural Graph Factorization. *Proceedings of the 22nd International Conference on World Wide Web* (2013), 37–48.

[9] AHN, S. J., AND KIM, M. Variational Graph Normalized AutoEncoders. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (2021), 2827–2831.

[10] AIXELÁ, J. F. Culture-Specific Items in Translation. *Translation, Power, Subversion 8* (1996), 52–78.

[11] ALON, U., AND YAHAV, E. On the Bottleneck of Graph Neural Networks and its Practical Implications. *Proceedings of the International Conference on Learning Representations* (2021).

[12] ANANTHARAM, V., VARAIYA, P., AND WALRAND, J. Asymptotically Efficient Allocation Rules for the Multi-Armed Bandit Problem with Multiple Plays-Part I: IID Rewards. *IEEE Transactions on Automatic Control 32*, 11 (1987), 968–976.

[13] APPERT, G. Information k-Means, Fragmentation and Syntax Analysis. A New Approach to Unsupervised Machine Learning. *Ph.D. Thesis, Institut Polytechnique de Paris* (2020).

[14] APPERT, G., AND CATONI, O. New Bounds for k-Means and Information k-Means. *Under review for publication in Annals of Statistics* (2021).

[15] ARORA, S., LIANG, Y., AND MA, T. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. *Proceedings of the International Conference on Learning Representations* (2017).

[16] ARTETXE, M., AND SCHWENK, H. Massively Multilingual Sentence Embeddings for Zero-Shot Cross-Lingual Transfer and Beyond. *Transactions of the Association for Computational Linguistics 7* (2019), 597–610.

[17] ARTHUR, D., AND VASSILVITSKII, S. k-Means++: the Advantages of Careful Seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2007), 1027–1035.

[18] AUER, P., CESA-BIANCHI, N., AND FISCHER, P. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning 47*, 2-3 (2002), 235–256.

[19] AUER, S., BIZER, C., KOBILAROV, G., LEHMANN, J., CYGANIAK, R., AND IVES, Z. Dbpedia: A Nucleus for a Web of Open Data. *The Semantic Web* (2007), 722–735.

[20] AZIMZADEH, P., AND FORSYTH, P. A. Weakly Chained Matrices, Policy Iteration, and Impulse Control. *SIAM Journal on Numerical Analysis 54*, 3 (2016), 1341–1364.

Bibliography

[21] BALCILAR, M., RENTON, G., HÉROUX, P., GAÜZÈRE, B., ADAM, S., AND HONEINE, P. Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective. *Proceedings of the International Conference on Learning Representations* (2021).

[22] BANNISTER, M. J., EPPSTEIN, D., GOODRICH, M. T., AND TROTT, L. Force-Directed Graph Drawing using Social Gravity and Scaling. *Proceedings of the International Symposium on Graph Drawing* (2012), 414–425.

[23] BARKAN, O., KOENIGSTEIN, N., YOGEV, E., AND KATZ, O. CB2CF: A Neural Multiview Content-to-Collaborative Filtering Model for Completely Cold Item Recommendations. *Proceedings of the 13th ACM Conference on Recommender Systems* (2019), 228–236.

[24] BATAGELJ, V., AND ZAVERSNIK, M. An O(m) Algorithm for Cores Decomposition of Networks. *Preprint associated with a talk at the Conference on Recent Trends in Graph Theory, Algebraic Combinatorics, and Graph Algorithms* (2003).

[25] BELKIN, M., AND NIYOGI, P. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation 15*, 6 (2003), 1373–1396.

[26] BENDADA, W., SALHA, G., AND BONTEMPELLI, T. Carousel Personalization in Music Streaming Apps with Contextual Bandits. *Proceedings of the Fourteenth ACM Conference on Recommender Systems* (2020), 420–425.

[27] BENGIO, Y., DELALLEAU, O., AND LE ROUX, N. Label Propagation and Quadratic Criterion. *Semi-Supervised Learning* (2006), 193–216.

[28] BERG, R. V. D., KIPF, T. N., AND WELLING, M. Graph Convolutional Matrix Completion. *KDD Deep Learning Day* (2018).

[29] BHARADHWAJ, H. Meta-Learning for User Cold-Start Recommendation. *Proceedings of the 2019 International Joint Conference on Neural Networks* (2019), 1–8.

[30] BLEI, D. M., KUCUKELBIR, A., AND MCAULIFFE, J. D. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association 112*, 518 (2017), 859–877.

[31] BLONDEL, V. D., GUILLAUME, J.-L., LAMBIOTTE, R., AND LEFEBVRE, E. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiments 2008*, 10 (2008), P10008.

[32] BOBADILLA, J., ORTEGA, F., HERNANDO, A., AND BERNAL, J. A Collaborative Filtering Approach to Mitigate the New User Cold Start Problem. *Knowledge-Based Systems 26* (2012), 225–238.

[33] BOGDANOV, D., PORTER, A., SCHREIBER, H., URBANO, J., AND ORAMAS, S. The AcousticBrainz Genre Dataset: Multi-Source, Multi-Level, Multi-Label, and Large-Scale. *Proceedings of the 20th International Society for Music Information Retrieval Conference* (2019), 360–367.

[34] BORGATTI, S. P., MEHRA, A., BRASS, D. J., AND LABIANCA, G. Network Analysis in the Social Sciences. *Science 323*, 5916 (2009), 892–895.

[35] BOUTSIDIS, C., KAMBADUR, P., AND GITTENS, A. Spectral Clustering via the Power Method - Provably. *Proceedings of the International Conference on Machine Learning* (2015), 40–48.

[36] BRADLEY, A. P. The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition 30*, 7 (1997), 1145–1159.

[37] BRANDES, U. On Variants of Shortest-Path Betweenness Centrality and their Generic Computation. *Social Networks 30*, 2 (2008), 136–145.

[38] BRANDES, U., DELLING, D., GAERTLER, M., GORKE, R., HOEFER, M., NIKOLOSKI, Z., AND WAGNER, D. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering 20*, 2 (2007), 172–188.

[39] BRIAND, L., SALHA-GALVAN, G., BENDADA, W., MORLON, M., AND TRAN, V.-A. A Semi-Personalized System for User Cold Start Recommendation on Music Streaming Apps. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2021), 2601–2609.

[40] BRONSTEIN, M. Graph ML at Twitter. Blog post on *"Twitter Engineering"*: https://blog.twitter.com/engineering/en_us/topics/insights/2020/graph-ml-at-twitter, 2020.

[41] BRUNA, J., ZAREMBA, W., SZLAM, A., AND LECUN, Y. Spectral Networks and Locally Connected Networks on Graphs. *Proceedings of the International Conference on Learning Representations* (2014).

[42] CAO, K. Y., LIU, Y., AND ZHANG, H. X. Improving the Cold Start Problem in Music Recommender Systems. *Journal of Physics: Conference Series 1651*, 1 (2020), 12–67.

Bibliography

[43] Cao, S., Lu, W., and Xu, Q. GraRep: Learning Graph Representations with Global Structural Information. *Proceedings of the 24th ACM International Conference on Information and Knowledge Management* (2015), 891–900.

[44] Cautis, B., Maniu, S., and Tziortziotis, N. Adaptive Influence Maximization. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), 3185–3186.

[45] Cavendish, H. XXI. Experiments to Determine the Density of the Earth. *Philosophical Transactions of the Royal Society of London*, 88 (1798), 469–526.

[46] Cesa-Bianchi, N., Gentile, C., Vitale, F., and Zappella, G. A Correlation Clustering Approach to Link Classification in Signed Networks. *Proceedings of the Conference on Learning Theory* (2012), 34–1.

[47] Chakrabarti, D., and Faloutsos, C. Graph Mining: Laws, Generators, and Algorithms. *ACM Computing Surveys 38*, 1 (2006), 2.

[48] Chapelle, O., and Li, L. An Empirical Evaluation of Thompson Sampling. *Advances in Neural Information Processing Systems* (2011), 2249–2257.

[49] Chen, H.-H., Gou, L., Zhang, X., and Giles, C. L. Discovering Missing Links in Networks using Vertex Similarity Measures. *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (2012), 138–143.

[50] Chen, J., Ma, T., and Xiao, C. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *Proceedings of the International Conference on Learning Representations* (2018).

[51] Chen, J., Zhu, J., and Song, L. Stochastic Training of Graph Convolutional Networks with Variance Reduction. *Proceedings of the International Conference on Machine Learning* (2018), 942–950.

[52] Chen, W., Wang, Y., Yuan, Y., and Wang, Q. Combinatorial Multi-Armed Bandit and its Extension to Probabilistically Triggered Arms. *Journal of Machine Learning Research 17*, 1 (2016), 1746–1778.

[53] Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019), 257–266.

[54] Chiericetti, F., Dasgupta, A., Kumar, R., Lattanzi, S., and Sarlos, T. On Sampling Nodes in a Network. *Proceedings of the 25th International Conference on World Wide Web* (2016), 471–481.

[55] Choong, J. J., Liu, X., and Murata, T. Learning Community Structure with Variational Autoencoder. *Proceedings of the 2018 IEEE International Conference on Data Mining* (2018), 69–78.

[56] Choong, J. J., Liu, X., and Murata, T. Optimizing Variational Graph Autoencoder for Community Detection with Dual Optimization. *Entropy 22*, 2 (2020), 197.

[57] Chu, W., Li, L., Reyzin, L., and Schapire, R. Contextual Bandits with Linear Payoff Functions. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011), 208–214.

[58] Chung, F. R. K. *Spectral Graph Theory*. Providence, R.I.: American Mathematical Society, 1997.

[59] Claffey, A., Ick, C., Ma, J., and Turkel, D. Evaluating Bandit Policies Across Datasets. *Term project at New York University* (2021).

[60] Conneau, A., and Lample, G. Cross-Lingual Language Model Pretraining. *Advances in Neural Information Processing Systems 32* (2019), 7059–7069.

[61] Corbett-Davies, S., and Goel, S. The Measure and Mismeasure of Fairness: A Critical Review of Fair Machine Learning. *arXiv preprint arXiv:1808.00023* (2018).

[62] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press, 2009.

[63] Covington, P., Adams, J., and Sargin, E. Deep Neural Networks for Youtube Recommendations. *Proceedings of the 10th ACM Conference on Recommender Systems* (2016), 191–198.

[64] Craswell, N., Zoeter, O., Taylor, M., and Ramsey, B. An Experimental Comparison of Click Position-Bias Models. *Proceedings of the 2008 International Conference on Web Search and Data Mining* (2008), 87–94.

[65] Dai, H., Dai, B., and Song, L. Discriminative Embeddings of Latent Variable Models for Structured Data. *Proceedings of the International Conference on Machine Learning* (2016), 2702–2711.

Bibliography

[66] DASOULAS, G., LUTZEYER, J. F., AND VAZIRGIANNIS, M. Learning Parametrised Graph Shift Operators. *Proceedings of the International Conference on Learning Representations* (2021).

[67] DATTORRO, J. *Convex Optimization & Euclidean Distance Geometry.* Meboo Publishing, 2005.

[68] DE LARA, N. The Sparse + Low Rank Trick for Matrix Factorization-Based Graph Algorithms. *Proceedings of the 15th International Workshop on Mining and Learning with Graphs* (2019).

[69] DEFFERRARD, M., BRESSON, X., AND VANDERGHEYNST, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *Advances in Neural Information Processing Systems 29* (2016), 3844–3852.

[70] DELBOUYS, R., HENNEQUIN, R., PICCOLI, F., ROYO-LETELIER, J., AND MOUSSALLAM, M. Music Mood Detection Based on Audio and Lyrics with Deep Neural Net. *Proceedings of the 19th International Society for Music Information Retrieval Conference* (2018), 370–375.

[71] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (2019), 4171–4186.

[72] DO, T. H., NGUYEN, D. M., TSILIGIANNI, E., AGUIRRE, A. L., LA MANNA, V. P., PASVEER, F., PHILIPS, W., AND DELIGIANNIS, N. Matrix Completion with Variational Graph Autoencoders: Application in Hyperlocal Air Quality Inference. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (2019), 7535–7539.

[73] DO NASCIMENTO, E. J. F., SOUZA, A. H., AND MESQUITA, D. Improving Graph Variational Autoencoders with Multi-Hop Simple Convolutions. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2021), 105–110.

[74] DOERSCH, C. Tutorial on Variational Autoencoders. *arXiv preprint arXiv:1606.05908* (2016).

[75] DONKER, S. Networking Data. A Network Analysis of Spotify's Socio-Technical Related Artist Network. *International Journal of Music Business Research 8*, 1 (2019), 67–101.

[76] DONNAT, C., ZITNIK, M., HALLAC, D., AND LESKOVEC, J. Learning Structural Node Embeddings via Diffusion Wavelets. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2018), 1320–1329.

[77] DU, N., WU, B., PEI, X., WANG, B., AND XU, L. Community Detection in Large-Scale Social Networks. *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis* (2007), 16–25.

[78] DUVENAUD, D., MACLAURIN, D., AGUILERA-IPARRAGUIRRE, J., GÓMEZ-BOMBARELLI, R., HIRZEL, T., ASPURU-GUZIK, A., AND ADAMS, R. P. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *Advances in Neural Information Processing Systems 28* (2015), 2224–2232.

[79] DWIVEDI, V. P., AND BRESSON, X. A Generalization of Transformer Networks to Graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications* (2021).

[80] EINSTEIN, A. Erklarung der Perihelionbewegung der Merkur aus der allgemeinen Relativitatstheorie. *Sitzungsber. preuss. Akad. Wiss., vol. 47, No. 2, pp. 831-839, 1915 47* (1915), 831–839.

[81] EPURE, E. V., KHLIF, A., AND HENNEQUIN, R. Leveraging Knowledge Bases And Parallel Annotations For Music Genre Translation. *Proceedings of the 20th International Society for Music Information Retrieval Conference* (2019), 839–846.

[82] EPURE, E. V., SALHA, G., AND HENNEQUIN, R. Multilingual Music Genre Embeddings for Effective Cross-Lingual Music Item Annotation. *Proceedings of the 21st International Society for Music Information Retrieval Conference* (2020), 803–810.

[83] EPURE, E. V., SALHA, G., MOUSSALLAM, M., AND HENNEQUIN, R. Modeling the Music Genre Perception across Language-Bound Cultures. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing* (2020), 4765–4779.

[84] EPURE, E. V., SALHA, G., VOITURET, F., BARANES, M., AND HENNEQUIN, R. Muzeeglot: annotation multilingue et multi-sources d'entités musicales à partir de représentations de genres musicaux. *Actes de la 6e conférence conjointe Journées d'Études sur la Parole (JEP, 33e édition), Traitement Automatique des Langues Naturelles (TALN, 27e édition), Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL, 22e édition). Volume 4: Démonstrations et résumés d'articles internationaux* (2020), 18–21.

Bibliography

[85] Epure, E. V., Salha-Galvan, G., Moussallam, M., and Hennequin, R. Modéliser la perception des genres musicaux à travers différentes cultures à partir de ressources linguistiques (Modeling the Music Genre Perception across Language-Bound Cultures). *Actes de la 28e Conférence sur le Traitement Automatique des Langues Naturelles. Volume 1 : conférence principale* (2021), 270–272.

[86] Fang, L., Luo, Y., Feng, K., Zhao, K., and Hu, A. Knowledge-Enhanced Ensemble Learning for Word Embeddings. *The World Wide Web Conference* (2019), 427–437.

[87] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. Retrofitting Word Vectors to Semantic Lexicons. *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015), 1606–1615.

[88] Fawcett, T. An Introduction to ROC Analysis. *Pattern Recognition Letters 27*, 8 (2006), 861–874.

[89] Felício, C. Z., Paixão, K. V., Barcelos, C. A., and Preux, P. A Multi-Armed Bandit Model Selection for Cold-Start User Recommendation. *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization* (2017), 32–40.

[90] Felicioni, N., Ferrari Dacrema, M., and Cremonesi, P. A Methodology for the Offline Evaluation of Recommender Systems in a User Interface with Multiple Carousels. *Adjunct Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization* (2021), 10–15.

[91] Felicioni, N., Ferrari Dacrema, M., Maurera, F. B. P., and Cremonesi, P. Measuring the Ranking Quality of Recommendations in a Two-Dimensional Carousel Setting. *Proceedings of IIR2021-11th Italian Information Retrieval Workshop* (2021).

[92] Fernández-Tobías, I., Braunhofer, M., Elahi, M., Ricci, F., and Cantador, I. Alleviating the New User Problem in Collaborative Filtering by Exploiting Personality Information. *User Modeling and User-Adapted Interaction 26*, 2 (2016), 221–255.

[93] Ferrari Dacrema, M., Cremonesi, P., and Jannach, D. Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches. *Proceedings of the 13th ACM Conference on Recommender Systems* (2019), 101–109.

[94] Ferrari Dacrema, M., Felicioni, N., and Cremonesi, P. Optimizing the Selection of Recommendation Carousels with Quantum Computing. *Proceedings of the Fifteenth ACM Conference on Recommender Systems* (2021), 691–696.

[95] Ferwerda, B., and Schedl, M. Investigating the Relationship Between Diversity in Music Consumption Behavior and Cultural Dimensions: A Cross-Country Analysis. *24th Conference on User Modeling, Adaptation, and Personalization (UMAP) Extended Proceedings: 1st Workshop on Surprise, Opposition, and Obstruction in Adaptive and Personalized Systems* (2016).

[96] Fey, M., and Lenssen, J. E. Fast Graph Representation Learning with PyTorch Geometric. *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[97] Fortunato, S. Community Detection in Graphs. *Physics Reports 486*, 3-5 (2010), 75–174.

[98] Fowlkes, C., Belongie, S., Chung, F., and Malik, J. Spectral Grouping using the Nystrom Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 2 (2004), 214–225.

[99] Friedman, J. A Proof of Alon's Second Eigenvalue Conjecture. *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* (2003), 720–724.

[100] Fruchterman, T. M., and Reingold, E. M. Graph Drawing by Force-Directed Placement. *Software: Practice and Experience 21*, 11 (1991), 1129–1164.

[101] Fukushima, K., and Miyake, S. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition. *Competition and Cooperation in Neural Nets* (1982), 267–285.

[102] Gama, F., Ribeiro, A., and Bruna, J. Stability of Graph Neural Networks to Relative Perturbations. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (2020), 9070–9074.

[103] Garcia Gasulla, D. Link Prediction in Large Directed Graphs. *Ph.D. Thesis, Universitat Politècnica de Catalunya* (2015).

[104] Garivier, A., and Cappé, O. The KL-UCB Algorithm for Bounded Stochastic Bandits and Beyond. *Proceedings of the 24th Annual Conference on Learning Theory* (2011), 359–376.

Bibliography

[105] GIATSIDIS, C., MALLIAROS, F., THILIKOS, D., AND VAZIRGIANNIS, M. CoreCluster: A Degeneracy Based Graph Clustering Framework. *Proceedings of the AAAI Conference on Artificial Intelligence* (2014), 44–50.

[106] GIATSIDIS, C., THILIKOS, D. M., AND VAZIRGIANNIS, M. D-Cores: Measuring Collaboration of Directed Graphs Based on Degeneracy. *Knowledge and Information Systems 35*, 2 (2013), 311–343.

[107] GILMER, J., SCHOENHOLZ, S. S., RILEY, P. F., VINYALS, O., AND DAHL, G. E. Neural Message Passing for Quantum Chemistry. *Proceedings of the International Conference on Machine Learning* (2017), 1263–1272.

[108] GINOULHAC, R. Dynamic Graph Clustering and Artist Recommendation. *M.Sc. Thesis, Ecole des Ponts ParisTech* (2018).

[109] GIROSI, F., JONES, M., AND POGGIO, T. Regularization Theory and Neural Networks Architectures. *Neural Computation 7*, 2 (1995), 219–269.

[110] GODSIL, C. *Algebraic Combinatorics*, vol. 6. CRC Press, 1993.

[111] GOLDBERG, Y., AND LEVY, O. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. *arXiv preprint arXiv:1402.3722* (2014).

[112] GOLOVIN, D., AND KRAUSE, A. Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization. *Journal of Artificial Intelligence Research 42* (2011), 427–486.

[113] GOLUB, G. H., AND REINSCH, C. Singular Value Decomposition and Least Squares Solutions. *Linear Algebra* (1971), 134–151.

[114] GOMEZ-URIBE, C. A., AND HUNT, N. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems 6*, 4 (2015), 1–19.

[115] GONG, S., BAHRI, M., BRONSTEIN, M. M., AND ZAFEIRIOU, S. Geometrically Principled Connections in Graph Neural Networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), 11415–11424.

[116] GONZALEZ, S., AND MIIKKULAINEN, R. Improved Training Speed, Accuracy, and Data Utilization Through Loss Function Optimization. *Proceedings of the 2020 IEEE Congress on Evolutionary Computation* (2020), 1–8.

[117] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016.

[118] GOPE, J., AND JAIN, S. K. A Survey on Solving Cold Start Problem in Recommender Systems. *Proceedings of the 2017 International Conference on Computing, Communication and Automation* (2017), 133–138.

[119] GORI, M., MONFARDINI, G., AND SCARSELLI, F. A New Model for Learning in Graph Domains. *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, 2005. 2* (2005), 729–734.

[120] GOYAL, P., KAMRA, N., HE, X., AND LIU, Y. DynGEM: Deep Embedding Method for Dynamic Graphs. *IJCAI 2017 Workshop on Representation Learning for Graphs* (2017).

[121] GRAVE, E., BOJANOWSKI, P., GUPTA, P., JOULIN, A., AND MIKOLOV, T. Learning Word Vectors for 157 Languages. *Proceedings of the International Conference on Language Resources and Evaluation* (2018).

[122] GREFF, K., VAN STEENKISTE, S., AND SCHMIDHUBER, J. Neural Expectation Maximization. *Advances in Neural Information Processing Systems* (2017), 6694–6704.

[123] GROVER, A., AND LESKOVEC, J. node2vec: Scalable Feature Learning for Networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), 855–864.

[124] GROVER, A., ZWEIG, A., AND ERMON, S. Graphite: Iterative Generative Modeling of Graphs. *Proceedings of the International Conference on Machine Learning* (2019), 2434–2444.

[125] GRUSON, A., CHANDAR, P., CHARBUILLET, C., MCINERNEY, J., HANSEN, S., TARDIEU, D., AND CARTERETTE, B. Offline Evaluation to Make Decisions about Playlist Recommendation Algorithms. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (2019), 420–428.

[126] GUO, L., AND DAI, Q. End-to-end Variational Graph Clustering with Local Structural Preservation. *Neural Computing and Applications* (2021), 1–16.

[127] HAGBERG, A., SWART, P., AND S CHULT, D. Exploring Network Structure, Dynamics, and Function using NetworkX. *Technical Report, Los Alamos National Lab, US* (2008).

Bibliography

[128] HAMILTON, W. L. Graph Representation Learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning 14*, 3 (2020), 1–159.

[129] HAMILTON, W. L., YING, R., AND LESKOVEC, J. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin* (2017), 52–74.

[130] HAMILTON, W. L., YING, Z., AND LESKOVEC, J. Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems* (2017), 1025–1035.

[131] HAMMOND, D. K., VANDERGHEYNST, P., AND GRIBONVAL, R. Wavelets on Graphs via Spectral Graph Theory. *Applied and Computational Harmonic Analysis 30*, 2 (2011), 129–150.

[132] HANGYA, V., BRAUNE, F., FRASER, A., AND SCHÜTZE, H. Two Methods for Domain Adaptation of Bilingual Tasks: Delightfully Simple and Broadly Applicable. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2018), 810–820.

[133] HAO, Y., CAO, X., FANG, Y., XIE, X., AND WANG, S. Inductive Link Prediction for Nodes Having Only Attribute Information. *Proceedings of the 29th International Joint Conference on Artificial Intelligence* (2020), 1209–1215.

[134] HARRIS, Z. S. Distributional Structure. *Word 10*, 2-3 (1954), 146–162.

[135] HASANZADEH, A., HAJIRAMEZANALI, E., NARAYANAN, K., DUFFIELD, N., ZHOU, M., AND QIAN, X. Semi-Implicit Graph Variational Auto-Encoders. *Advances in Neural Information Processing Systems* (2019), 10712–10723.

[136] HAYES, D. What Just Happened? Evaluating Retrofitted Distributional Word Vectors. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (2019), 1062–1072.

[137] HE, R., AND MCAULEY, J. VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. *Proceedings of the 30th AAAI Confence on Artificial Intelligence* (2016), 144–150.

[138] HE, X., LIAO, L., ZHANG, H., NIE, L., HU, X., AND CHUA, T.-S. Neural Collaborative Filtering. *Proceedings of the Web Conference* (2017), 173–182.

[139] HE, X., ZHANG, H., KAN, M.-Y., AND CHUA, T.-S. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2016), 549–558.

[140] HENAFF, M., BRUNA, J., AND LECUN, Y. Deep Convolutional Networks on Graph-Structured Data. *arXiv preprint arXiv:1506.05163* (2015).

[141] HENNEQUIN, R., ROYO-LETELIER, J., AND MOUSSALLAM, M. Audio Based Disambiguation of Music Genre Tags. *Proceedings of the 19th International Society for Music Information Retrieval Conference* (2018), 645–652.

[142] HIBSHMAN, J. I., GONZALEZ, D., SIKDAR, S., AND WENINGER, T. Joint Subgraph-to-Subgraph Transitions: Generalizing Triadic Closure for Powerful and Interpretable Graph Modeling. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining* (2021), 815–823.

[143] HOEFFDING, W. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association 58*, 301 (1963), 13–30.

[144] HOI, S. C. H., SAHOO, D., LU, J., AND ZHAO, P. Online Learning: A Comprehensive Survey. *Neurocomputing* (2018), 249–289.

[145] HOLLOCOU, A., BONALD, T., AND LELARGE, M. Modularity-Based Sparse Soft Graph Clustering. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics* (2019), 323–332.

[146] HOORY, S., LINIAL, N., AND WIGDERSON, A. Expander Graphs and their Applications. *Bulletin of the American Mathematical Society* (2006), 439–561.

[147] HORN, R. A., AND JOHNSON, C. R. *Matrix Analysis*. Cambridge, UK: Cambridge University Press, 1985.

[148] HOSSIN, M., AND SULAIMAN, M. N. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process 5*, 2 (2015), 1.

[149] HSIEH, C.-K., YANG, L., CUI, Y., LIN, T.-Y., BELONGIE, S., AND ESTRIN, D. Collaborative Metric Learning. *Proceedings of the 26th International Conference on World Wide Web* (2017), 193–201.

[150] HU, W., FEY, M., REN, H., NAKATA, M., DONG, Y., AND LESKOVEC, J. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. *arXiv preprint arXiv:2103.09430* (2021).

Bibliography

[151] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *Advances in Neural Information Processing Systems* (2020).

[152] Hu, Y., Koren, Y., and Volinsky, C. Collaborative Filtering for Implicit Feedback Datasets. *Proceedings of the Eighth IEEE International Conference on Data Mining* (2008), 263–272.

[153] Hu, Y., You, H., Wang, Z., Wang, Z., Zhou, E., and Gao, Y. Graph-MLP: Node Classification without Message Passing in Graph. *arXiv preprint arXiv:2106.04051* (2021).

[154] Huang, J., Zhu, T., and Schuurmans, D. Web Communities Identification from Random Walks. *Proceedings of the 2006 European Conference on Principles of Data Mining and Knowledge Discovery* (2006), 187–198.

[155] Huang, S., Liu, J., Korn, F., Wang, X., Wu, Y., Markowitz, D., and Yu, C. Contextual Fact Ranking and its Applications in Table Synthesis and Compression. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), 285–293.

[156] Huang, T., Pei, Y., Menkovski, V., and Pechenizkiy, M. On Generalization of Graph Autoencoders with Adversarial Training. *arXiv preprint arXiv:2107.02658* (2021).

[157] Huang, W. R. On Edge Reweighting for Link Prediction with Graph Auto-Encoders. *TechRxiv Preprint* (2020).

[158] Huang, Z., Silva, A., and Singh, A. A Broader Picture of Random-Walk Based Graph Embedding. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2021), 685–695.

[159] Hubert, L., and Arabie, P. Comparing Partitions. *Journal of Classification 2*, 1 (1985), 193–218.

[160] Ie, E., Jain, V., Wang, J., Narvekar, S., Agarwal, R., Wu, R., Cheng, H.-T., Chandra, T., and Boutilier, C. SLATEQ: A Tractable Decomposition for Reinforcement Learning with Recommendation Sets. *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (2019), 2592–2599.

[161] Indyk, P., and Motwani, R. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (1998), 604–613.

[162] Ivanov, S. ICLR 2020 Graph Papers. Blog post on *"Medium"*: https://sergei-ivanov.medium.com/iclr-2020-graph-papers-9bc2e90e56b0, 2020.

[163] Ivanov, S. GML Express: Keynotes at ICLR, Topics at ICML 2021, and New GNN Tutorials. Blog post on *"Graph Machine Learning"*: https://graphml.substack.com/p/gml-express-keynotes-at-iclr-topics, 2021.

[164] Jacobson, K., Murali, V., Newett, E., Whitman, B., and Yon, R. Music Personalization at Spotify. *Proceedings of the 10th ACM Conference on Recommender Systems* (2016), 373–373.

[165] Jain, G., Mahara, T., and Tripathi, K. N. A Survey of Similarity Measures for Collaborative Filtering-Based Recommender System. *Soft Computing: Theories and Applications* (2020), 343–352.

[166] Jensen, J. L. W. V. Sur les Fonctions Convexes et les Inégalités entre les Valeurs Moyennes. *Acta Mathematica 30*, 1 (1906), 175–193.

[167] Jeunen, O., and Goethals, B. Top-k Contextual Bandits with Equity of Exposure. *Proceedings of the Fifteenth ACM Conference on Recommender Systems* (2021), 310–320.

[168] Jiang, R., Gowal, S., Mann, T. A., and Rezende, D. J. Beyond Greedy Ranking: Slate Optimization via List-CVAE. *Proceedings of the Eighth International Conference on Learning Representations* (2019).

[169] Jin, W., Barzilay, R., and Jaakkola, T. Junction Tree Variational Autoencoder for Molecular Graph Generation. *Proceedings of the International Conference on Machine Learning* (2018), 2323–2332.

[170] Johnston, M. How "Fans Also Like" Works. Blog post on *"Spotify for Artists"*: https://artists.spotify.com/blog/how-fans-also-like-works, 2019.

[171] Joulin, A., Bojanowski, P., Mikolov, T., Jégou, H., and Grave, E. Loss in Translation: Learning Bilingual Word Mapping with a Retrieval Criterion. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018), 2979–2984.

[172] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bag of Tricks for Efficient Text Classification. *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics* (2017), 427–431.

[173] Kale, S., Reyzin, L., and Schapire, R. E. Non-Stochastic Bandit Slate Problems. *Advances in Neural Information Processing Systems* (2010), 1054–1062.

Bibliography

[174] KALOGA, Y., BORGNAT, P., CHEPURI, S. P., ABRY, P., AND HABRARD, A. Multiview Variational Graph Autoencoders for Canonical Correlation Analysis. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (2021), 5320–5324.

[175] KARATZOGLOU, A., BALTRUNAS, L., AND SHI, Y. Learning to Rank for Recommender Systems. *Proceedings of the ACM Conference on Recommender Systems* (2013), 493–494.

[176] KATARIYA, S., KVETON, B., SZEPESVARI, C., AND WEN, Z. DCM Bandits: Learning to Rank with Multiple Clicks. *Proceedings of the 33rd International Conference on Machine Learning* (2016), 1215–1224.

[177] KATZ, L. A New Status Index Derived from Sociometric Analysis. *Psychometrika 18*, 1 (1953), 39–43.

[178] KAUFMANN, E., BONALD, T., AND LELARGE, M. A Spectral Algorithm with Additive Clustering for the Recovery of Overlapping Communities in Networks. *Proceedings of the International Conference on Algorithmic Learning Theory* (2016), 355–370.

[179] KAZEMI, S. M., GOEL, R., JAIN, K., KOBYZEV, I., SETHI, A., FORSYTH, P., AND POUPART, P. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research 21*, 70 (2020), 1–73.

[180] KEMPE, D., KLEINBERG, J., AND TARDOS, E. Maximizing the Spread of Influence through a Social Network. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2003), 137–146.

[181] KERIVEN, N., AND PEYRÉ, G. Universal Invariant and Equivariant Graph Neural Networks. *Advances in Neural Information Processing Systems 32* (2019), 7092–7101.

[182] KIELA, D., HILL, F., AND CLARK, S. Specializing Word Embeddings for Similarity or Relatedness. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (2015), 2044–2048.

[183] KIM, J.-K., DE MARNEFFE, M.-C., AND FOSLER-LUSSIER, E. Adjusting Word Embeddings with Semantic Intensity Orders. *Proceedings of the 1st Workshop on Representation Learning for NLP* (2016), 62–69.

[184] KINGMA, D. P., AND BA, J. Adam: A Method for Stochastic Optimization. *Proceedings of the International Conference on Learning Representations* (2015).

[185] KINGMA, D. P., AND WELLING, M. Auto-Encoding Variational Bayes. *Proceedings of the International Conference on Learning Representations* (2014).

[186] KIPF, T. N. Deep Learning with Graph-Structured Representations. *Ph.D. Thesis, University of Amsterdam* (2020).

[187] KIPF, T. N., AND WELLING, M. Variational Graph Auto-Encoders. *NeurIPS Workshop on Bayesian Deep Learning* (2016).

[188] KIPF, T. N., AND WELLING, M. Semi-Supervised Classification with Graph Convolutional Networks. *Proceedings of the International Conference on Learning Representations* (2017).

[189] KJUS, Y. Musical Exploration via Streaming Services: The Norwegian Experience. *Popular Communication 14*, 3 (2016), 127–136.

[190] KLEINBERG, B., LI, Y., AND YUAN, Y. An Alternative View: When Does SGD Escape Local Minima? *Proceedings of the International Conference on Machine Learning* (2018), 2698–2707.

[191] KLEMA, V., AND LAUB, A. The Singular Value Decomposition: its Computation and Some Applications. *IEEE Transactions on Automatic Control 25*, 2 (1980), 164–176.

[192] KLICPERA, J., BOJCHEVSKI, A., AND GÜNNEMANN, S. Predict then Propagate: Combining Neural Networks with Personalized PageRank for Classification on Graphs. *Proceedings of the International Conference on Learning Representations* (2019).

[193] KLICPERA, J., WEISSENBERGER, S., AND GÜNNEMANN, S. Diffusion Improves Graph Learning. *Advances in Neural Information Processing Systems 32* (2019), 13354–13366.

[194] KOMIYAMA, J., HONDA, J., AND NAKAGAWA, H. Optimal Regret Analysis of Thompson Sampling in Stochastic Multi-Armed Bandit Problem with Multiple Plays. *Proceedings of the 32nd International Conference on Machine Learning* (2015), 1152–1161.

[195] KOREN, Y., AND BELL, R. Advances in Collaborative Filtering. *Recommender Systems Handbook* (2015), 77–118.

[196] KOREN, Y., BELL, R., AND VOLINSKY, C. Matrix Factorization Techniques for Recommender Systems. *Computer 42*, 8 (2009), 30–37.

Bibliography

[197] KORZENIOWSKI, F., ORAMAS, S., AND GOUYON, F. Artist Similarity with Graph Neural Networks. *Proceedings of the 22nd International Society for Music Information Retrieval Conference* (2021), 350–357.

[198] KRAMSCH, C., AND WIDDOWSON, H. *Language and Culture.* Oxford University Press, 1998.

[199] KULLBACK, S., AND LEIBLER, R. A. On Information and Sufficiency. *The Annals of Mathematical Statistics 22-1* (1951), 79–86.

[200] KUMAR, A., SINGH, S. S., SINGH, K., AND BISWAS, B. Link Prediction Techniques, Applications, and Performance: A Survey. *Physica A: Statistical Mechanics and its Applications 553* (2020), 124289.

[201] KVETON, B., SZEPESVARI, C., WEN, Z., AND ASHKAN, A. Cascading Bandits: Learning to Rank in the Cascade Model. *Proceedings of the 32nd International Conference on Machine Learning* (2015), 767–776.

[202] LAGRÉE, P., VERNADE, C., AND CAPPE, O. Multiple-Play Bandits in the Position-Based Model. *Advances in Neural Information Processing Systems* (2016), 1597–1605.

[203] LAI, T. L., AND ROBBINS, H. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics 6*, 1 (1985), 4–22.

[204] LAM, X. N., VU, T., LE, T. D., AND DUONG, A. D. Addressing Cold-Start Problem in Recommendation Systems. *Proceedings of the International Conference on Ubiquitous Information Management and Communication* (2008), 208–211.

[205] LECUN, Y., BENGIO, Y., AND HINTON, G. Deep Learning. *Nature 521*, 7553 (2015), 436–444.

[206] LECUN, Y., HAFFNER, P., BOTTOU, L., AND BENGIO, Y. Object Recognition with Gradient-Based Learning. *Shape, Contour and Grouping in Computer Vision* (1999), 319–345.

[207] LEE, G., KANG, S., AND WHANG, J. J. Hyperlink Classification via Structured Graph Embedding. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2019), 1017–1020.

[208] LEE, H., IM, J., JANG, S., CHO, H., AND CHUNG, S. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), 1073–1082.

[209] LEE, J., ABU-EL-HAIJA, S., VARADARAJAN, B., AND NATSEV, A. Collaborative Deep Metric Learning for Video Understanding. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2018), 481–490.

[210] LEE, J. H., CHOI, K., HU, X., AND DOWNIE, J. S. K-Pop Genres: A Cross-Cultural Exploration. *Proceedings of the 14th International Society for Music Information Retrieval Conference* (2013), 529–534.

[211] LESKOVEC, J., AND FALOUTSOS, C. Sampling from Large Graphs. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge discovery and data mining* (2006), 631–636.

[212] LESKOVEC, J., AND KREVL, A. SNAP Datasets: Stanford Large Network Dataset Collection. https://snap.stanford.edu/data, 2014.

[213] LESKOVEC, J., RAJARAMAN, A., AND ULLMAN, J. D. *Mining of Massive Data Sets.* Cambridge University Press, 2020.

[214] LEVIE, R., MONTI, F., BRESSON, X., AND BRONSTEIN, M. M. Cayleynets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters. *IEEE Transactions on Signal Processing 67*, 1 (2019), 97–109.

[215] LEVY, O., AND GOLDBERG, Y. Neural Word Embedding as Implicit Matrix Factorization. *Advances in Neural Information Processing Systems 27* (2014), 2177–2185.

[216] LI, J., LIU, M., ZHANG, H., WANG, P., WEN, Y., PAN, L., AND CHENG, H. Mask-GVAE: Blind Denoising Graphs via Partition. *Proceedings of the Web Conference* (2021), 3688–3698.

[217] LI, J., YU, J., LI, J., ZHANG, H., ZHAO, K., RONG, Y., CHENG, H., AND HUANG, J. Dirichlet Graph Variational Autoencoder. *Advances in Neural Information Processing Systems 33* (2020).

[218] LI, J., YU, T., JUAN, D.-C., GOPALAN, A., CHENG, H., AND TOMKINS, A. Graph Autoencoders with Deconvolutional Networks. *arXiv preprint arXiv:2012.11898* (2020).

[219] LI, L., CHU, W., LANGFORD, J., AND SCHAPIRE, R. E. A Contextual-Bandit Approach to Personalized News Article Recommendation. *Proceedings of the 19th International Conference on World Wide Web* (2010), 661–670.

Bibliography

[220] LI, L., CHU, W., LANGFORD, J., AND WANG, X. Unbiased Offline Evaluation of Contextual-Bandit-Based News Article Recommendation Algorithms. *Proceedings of the 4th ACM International Conference on Web Search and Data Mining* (2011), 297–306.

[221] LI, Q., HAN, Z., AND WU, X.-M. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence* (2018), 3538–3545.

[222] LI, X., AND SHE, J. Collaborative Variational Autoencoder for Recommender Systems. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), 305–314.

[223] LI, Y., FAN, J., WANG, Y., AND TAN, K.-L. Influence Maximization on Social Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering 30*, 10 (2018), 1852–1872.

[224] LIAO, R., LI, Y., SONG, Y., WANG, S., HAMILTON, W. L., DUVENAUD, D., URTASUN, R., AND ZEMEL, R. Efficient Graph Generation with Graph Recurrent Attention Networks. *Advances in Neural Information Processing Systems* (2019), 4255–4265.

[225] LIBEN-NOWELL, D., AND KLEINBERG, J. The Link-Prediction Problem for Social Networks. *Journal of the American society for information science and technology 58*, 7 (2007), 1019–1031.

[226] LICHTENWALTER, R. N., LUSSIER, J. T., AND CHAWLA, N. V. New Perspectives and Methods in Link Prediction. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2010), 243–252.

[227] LIKA, B., KOLOMVATSOS, K., AND HADJIEFTHYMIADES, S. Facing the Cold Start Problem in Recommender Systems. *Expert Systems with Applications 41*, 4 (2014), 2065–2073.

[228] LIN, J. The Neural Hype and Comparisons Against Weak Baselines. *ACM SIGIR Forum 52*, 2 (2019), 40–51.

[229] LIN, J., SUGIYAMA, K., KAN, M.-Y., AND CHUA, T.-S. Addressing Cold-Start in App Recommendation: Latent User Models Constructed from Twitter Followers. *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2013), 283–292.

[230] LISENA, P., TODOROV, K., CECCONI, C., LERESCHE, F., CANNO, I., PUYRENIER, F., VOISIN, M., LE MEUR, T., AND TRONCY, R. Controlled Vocabularies for Music Metadata. *Proceedings of the 19th International Society for Music Information Retrieval Conference* (2018).

[231] LITTLE, R. J., AND RUBIN, D. B. *Statistical Analysis with Missing Data*, vol. 793. John Wiley & Sons, 2019.

[232] LIU, F., XUE, S., WU, J., ZHOU, C., HU, W., PARIS, C., NEPAL, S., YANG, J., AND YU, P. S. Deep Learning for Community Detection: Progress, Challenges and Opportunities. *Proceedings of the 29th International Joint Conference on Artificial Intelligence* (2020), 4981–4987.

[233] LIU, Q., ALLAMANIS, M., BROCKSCHMIDT, M., AND GAUNT, A. Constrained Graph Variational Autoencoders for Molecule Design. *Advances in Neural Information Processing Systems* (2018), 7806–7815.

[234] LIU, T.-Y. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval 3*, 3 (2009), 225–331.

[235] LIU, X., ZHUANG, C., MURATA, T., KIM, K.-S., AND KERTKEIDKACHORN, N. How Much Topological Structure is Preserved by Graph Embeddings? *Computer Science and Information Systems 16*, 2 (2019), 597–614.

[236] LO, C., YU, H., YIN, X., SHETTY, K., HE, C., HU, K., PLATZ, J. M., ILARDI, A., AND MADHVANATH, S. Page-Level Optimization of e-Commerce Item Recommendations. *Proceedings of the Fifteenth ACM Conference on Recommender Systems* (2021), 495–504.

[237] LOBOV, I., AND IVANOV, S. Unsupervised Community Detection with Modularity-Based Attention Model. *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds* (2019).

[238] LOVÁSZ, L. Eigenvalues of Graphs. *Technical report, Eotvos Lorand University* (2007).

[239] LÜ, L., AND ZHOU, T. Link Prediction in Complex Networks: A Survey. *Physica A: Statistical Mechanics and its Applications 390*, 6 (2011), 1150–1170.

[240] LUTZEYER, J. F. Network Representation Matrices and their Eigenproperties: A Comparative Study. Ph.D. Thesis, Imperial College London, 2020.

[241] LUTZEYER, J. F., AND WALDEN, A. T. Extending the Davis-Kahan Theorem for Comparing Eigenvectors of Two Symmetric Matrices I: Theory. *arXiv preprint arXiv:1908.03462* (2019).

Bibliography

[242] MA, H., AND KE, Y. An Introduction to Entity Recommendation and Understanding. *Proceedings of the 24th International Conference on World Wide Web* (2015), 1521–1522.

[243] MA, T., CHEN, J., AND XIAO, C. Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. *Advances in Neural Information Processing Systems 31* (2018), 7113–7124.

[244] MACQUEEN, J. Some Methods for Classification and Analysis of Multivariate Observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability 1*, 14 (1967), 281–297.

[245] MADDALENA, L., MANIPUR, I., MANZO, M., AND GUARRACINO, M. On Whole-Graph Embedding Techniques. *Proceedings of the International Symposium on Mathematical and Computational Biology* (2020), 115–131.

[246] MAHDAVI, S., KHOSHRAFTAR, S., AND AN, A. dynnode2vec: Scalable Dynamic Network Embedding. *Proceedings of the 2018 IEEE International Conference on Big Data* (2018), 3762–3765.

[247] MALLIAROS, F. D., GIATSIDIS, C., PAPADOPOULOS, A. N., AND VAZIRGIANNIS, M. The Core Decomposition of Networks: Theory, Algorithms and Applications. *The VLDB Journal* (2019), 1–32.

[248] MALLIAROS, F. D., AND VAZIRGIANNIS, M. Clustering and Community Detection in Directed Networks: A Survey. *Physics Reports 533*, 4 (2013), 95–142.

[249] MANDELBROT, B. An Informational Theory of the Statistical Structure of Language. *Communication Theory 84* (1953), 486–502.

[250] MARON, H., BEN-HAMU, H., SERVIANSKY, H., AND LIPMAN, Y. Provably Powerful Graph Networks. *Advances in Neural Information Processing Systems* (2019), 2156–2167.

[251] MCCLISH, D. K. Analyzing a Portion of the ROC Curve. *Medical Decision Making 9*, 3 (1989), 190–195.

[252] MCINERNEY, J., LACKER, B., HANSEN, S., HIGLEY, K., BOUCHARD, H., GRUSON, A., AND MEHROTRA, R. Explore, Exploit, and Explain: Personalizing Explainable Recommendations with Bandits. *Proceedings of the 12th ACM Conference on Recommender Systems* (2018), 31–39.

[253] MEHROTRA, R., LALMAS, M., KENNEY, D., LIM-MENG, T., AND HASHEMIAN, G. Jointly Leveraging Intent and Interaction Signals to Predict User Satisfaction with Slate Recommendations. *Proceedings of the World Wide Web Conference* (2019), 1256–1267.

[254] MERKWIRTH, C., AND LENGAUER, T. Automatic Generation of Complementary Descriptors with Molecular Graph Networks. *Journal of Chemical Information and Modeling 45*, 5 (2005), 1159–1168.

[255] MERRIS, R. Laplacian Matrices of Graphs: a Survey. *Linear Algebra and its Applications 197* (1994), 143–176.

[256] MIKOLOV, T., GRAVE, É., BOJANOWSKI, P., PUHRSCH, C., AND JOULIN, A. Advances in Pre-Training Distributed Word Representations. *Proceedings of the International Conference on Language Resources and Evaluation* (2018).

[257] MIKOLOV, T., LE, Q. V., AND SUTSKEVER, I. Exploiting Similarities among Languages for Machine Translation. *arXiv preprint arXiv:1309.4168* (2013).

[258] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems* (2013), 3111–3119.

[259] MILLER, K., JORDAN, M. I., AND GRIFFITHS, T. L. Nonparametric Latent Feature Models for Link Prediction. *Advances in Neural Information Processing Systems 22* (2009), 1276–1284.

[260] MONGIA, A., JHAMB, N., CHOUZENOUX, E., AND MAJUMDAR, A. Deep Latent Factor Model for Collaborative Filtering. *Signal Processing 169* (2020), 107366.

[261] MORRIS, C., RITZERT, M., FEY, M., HAMILTON, W. L., LENSSEN, J. E., RATTAN, G., AND GROHE, M. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence 33-1* (2019), 4602–4609.

[262] MRABAH, N., BOUGUESSA, M., TOUATI, M. F., AND KSANTINI, R. Rethinking Graph Autoencoder Models for Attributed Graph Clustering. *arXiv preprint arXiv:2107.08562* (2021).

[263] MU, R. A Survey of Recommender Systems Based on Deep Learning. *IEEE Access 6* (2018), 69009–69022.

[264] MURPHY, R., SRINIVASAN, B., RAO, V., AND RIBEIRO, B. Relational Pooling for Graph Representations. *Proceedings of the International Conference on Machine Learning* (2019), 4663–4673.

Bibliography

[265] NARAYANAN, A., CHANDRAMOHAN, M., VENKATESAN, R., CHEN, L., LIU, Y., AND JAISWAL, S. graph2vec: Learning Distributed Representations of Graphs. *Proceedings of the 13th International Workshop on Mining and Learning with Graphs* (2017).

[266] NEWMAN, M. *Networks: An Introduction*. UK Oxford University Press, 2010.

[267] NEWMAN, M. E. J. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Sciences 103*, 23 (2006), 8577–8582.

[268] NEWMARK, P. *A Textbook of Translation*, vol. 66. Prentice Hall New York, 1988.

[269] NEWTON, I. *Philosophiae Naturalis Principia Mathematica*. 1687.

[270] NG, A. Y., JORDAN, M. I., AND WEISS, Y. On Spectral Clustering: Analysis and an Algorithm. *Advances in Neural Information Processing Systems* (2002), 849–856.

[271] NGUYEN, T. T., AND LAUW, H. W. Dynamic Clustering of Contextual Multi-Armed Bandits. *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management* (2014), 1959–1962.

[272] NIKOLENTZOS, G., MELADIANOS, P., LIMNIOS, S., AND VAZIRGIANNIS, M. A Degeneracy Framework for Graph Similarity. *Proceedings of the International Joint Conference on Artificial Intelligence* (2018), 2595–2601.

[273] NIKOLENTZOS, G., SIGLIDIS, G., AND VAZIRGIANNIS, M. Graph Kernels: A Survey. *Journal of Artificial Intelligence Research 72* (2021), 943–1027.

[274] NIKOLENTZOS, G., TIXIER, A., AND VAZIRGIANNIS, M. Message Passing Attention Networks for Document Understanding. *Proceedings of the AAAI Conference on Artificial Intelligence 34*, 05 (2020), 8544–8551.

[275] ORTEGA, A., FROSSARD, P., KOVAČEVIĆ, J., MOURA, J. M., AND VANDERGHEYNST, P. Graph Signal Processing: Overview, Challenges, and Applications. *Proceedings of the IEEE 106* (2018), 808–828.

[276] OU, M., CUI, P., PEI, J., ZHANG, Z., AND ZHU, W. Asymmetric Transitivity Preserving Graph Embedding. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), 1105–1114.

[277] OUSSIDI, A., AND ELHASSOUNY, A. Deep Generative Models: Survey. *Proceedings of the 2018 International Conference on Intelligent Systems and Computer Vision* (2018), 1–8.

[278] PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. The PageRank Citation Ranking: Bringing Order to the Web. *Stanford InfoLab* (1999).

[279] PAN, S., HU, R., LONG, G., JIANG, J., YAO, L., AND ZHANG, C. Adversarially Regularized Graph Autoencoder for Graph Embedding. *Proceedings of the International Joint Conference on Artificial Intelligence* (2018), 2609–2615.

[280] PAPADOPOULOS, S., KOMPATSIARIS, Y., VAKALI, A., AND SPYRIDONOS, P. Community Detection in Social Media. *Data Mining and Knowledge Discovery 24*, 3 (2012), 515–554.

[281] PARK, H.-S., AND JUN, C.-H. A Simple and Fast Algorithm for K-Medoids Clustering. *Expert Systems with Applications 36*, 2 (2009), 3336–3341.

[282] PARK, J., LEE, M., CHANG, H. J., LEE, K., AND CHOI, J. Y. Symmetric Graph Convolutional Autoencoder for Unsupervised Graph Representation Learning. *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), 6519–6528.

[283] PARK, Y.-J., AND TUZHILIN, A. The Long Tail of Recommender Systems and How To Leverage It. *Proceedings of the ACM Conference on Recommender Systems* (2008), 11–18.

[284] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[285] PENNINGTON, J., SOCHER, R., AND MANNING, C. D. Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (2014), 1532–1543.

[286] PEREIRA, B. L., UEDA, A., PENHA, G., SANTOS, R. L., AND ZIVIANI, N. Online Learning to Rank for Sequential Music Recommendation. *Proceedings of the 13th ACM Conference on Recommender Systems* (2019), 237–245.

[287] PEROZZI, B., AL-RFOU, R., AND SKIENA, S. Deepwalk: Online Learning of Social Representations. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2014), 701–710.

Bibliography

[288] PEROZZI, B., KULKARNI, V., CHEN, H., AND SKIENA, S. Don't Walk, Skip! Online Learning of Multi-Scale Network Embeddings. *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017* (2017), 258–265.

[289] PFEIL, U., ZAPHIRIS, P., AND ANG, C. S. Cultural Differences in Collaborative Authoring of Wikipedia. *Journal of Computer-Mediated Communication 12*, 1 (2006), 88–113.

[290] PLANTIÉ, M., AND CRAMPES, M. Survey on Social Community Detection. *Social Media Retrieval* (2013), 65–85.

[291] PONTI, E. M., VULIĆ, I., GLAVAŠ, G., REICHART, R., AND KORHONEN, A. Cross-lingual Semantic Specialization via Lexical Relation Induction. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (2019), 2206–2217.

[292] QIAN, T., LIANG, Y., LI, Q., AND XIONG, H. Attribute Graph Neural Networks for Strict Cold Start Recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2020), 1–1.

[293] QIN, L., CHEN, S., AND ZHU, X. Contextual Combinatorial Bandit and its Application on Diversified Online Recommendation. *Proceedings of the 2014 SIAM International Conference on Data Mining* (2014), 461–469.

[294] QIU, J., DONG, Y., MA, H., LI, J., WANG, K., AND TANG, J. Network Embedding as Matrix Factorization: Unifying Deepwalk, LINE, PTE, and node2vec. *Proceedings of the 11th ACM International Conference on Web Search and Data Mining* (2018), 459–467.

[295] RADLINSKI, F., KLEINBERG, R., AND JOACHIMS, T. Learning Diverse Rankings with Multi-Armed Bandits. *Proceedings of the 25th International Conference on Machine Learning* (2008), 784–791.

[296] RAFAILIDIS, D., AND CRESTANI, F. Learning to Rank with Trust and Distrust in Recommender Systems. *Proceedings of the ACM Conference on Recommender Systems* (2017), 5–13.

[297] RAIMOND, Y., ABDALLAH, S. A., SANDLER, M. B., AND GIASSON, F. The Music Ontology. *Proceedings of the 8th International Society for Music Information Retrieval Conference* (2007).

[298] RAY, S., LALL, S., MUKHOPADHYAY, A., BANDYOPADHYAY, S., AND SCHÖNHUTH, A. Predicting Potential Drug Targets and Repurposable Drugs for Covid-19 via a Deep Generative Model for Graphs. *arXiv preprint arXiv:2007.02338* (2020).

[299] RENNARD, V., NIKOLENTZOS, G., AND VAZIRGIANNIS, M. Graph Auto-Encoders for Learning Edge Representations. *Proceedings of the International Conference on Complex Networks and Their Applications* (2020), 117–129.

[300] RIBEIRO, L. F., SAVERESE, P. H., AND FIGUEIREDO, D. R. struc2vec: Learning Node Representations from Structural Identity. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), 385–394.

[301] ROBERTSON, S. A New Interpretation of Average Precision. *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2008), 689–690.

[302] RONG, Y., HUANG, W., XU, T., AND HUANG, J. Dropedge: Towards Deep Graph Convolutional Networks on Node Classification. *Proceedings of the International Conference on Learning Representations* (2020).

[303] ROSENBLATT, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review 65*, 6 (1958), 386.

[304] ROY, I., DE, A., AND CHAKRABARTI, S. Adversarial Permutation Guided Node Representations for Link Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence 35*, 11 (2021), 9445–9453.

[305] RUSSELL, J. A. A Circumplex Model of Affect. *Journal of Personality and Social Psychology 39*, 6 (1980), 1161.

[306] SAAD, Y. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, 2003.

[307] SALHA, G., HENNEQUIN, R., REMY, J.-B., MOUSSALLAM, M., AND VAZIRGIANNIS, M. FastGAE: Scalable Graph Autoencoders with Stochastic Subgraph Decoding. *Neural Networks 142* (2021), 1–19.

[308] SALHA, G., HENNEQUIN, R., TRAN, V. A., AND VAZIRGIANNIS, M. A Degeneracy Framework for Scalable Graph Autoencoders. *Proceedings of the International Joint Conference on Artificial Intelligence* (2019), 3353–3359.

[309] SALHA, G., HENNEQUIN, R., AND VAZIRGIANNIS, M. Keep it Simple: Graph Autoencoders Without Graph Convolutional Networks. *NeurIPS 2019 Workshop on Graph Representation Learning* (2019).

Bibliography

[310] SALHA, G., HENNEQUIN, R., AND VAZIRGIANNIS, M. Simple and Effective Graph Autoencoders with One-Hop Linear Models. *Proceedings of the 2020 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (2020), 319–334.

[311] SALHA, G., LIMNIOS, S., HENNEQUIN, R., TRAN, V. A., AND VAZIRGIANNIS, M. Gravity-Inspired Graph Autoencoders for Directed Link Prediction. *Proceedings of the ACM International Conference on Information and Knowledge Management* (2019), 589–598.

[312] SALHA, G., TZIORTZIOTIS, N., AND VAZIRGIANNIS, M. Adaptive Submodular Influence Maximization with Myopic Feedback. *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (2018), 455–462.

[313] SALHA-GALVAN, G., HENNEQUIN, R., CHAPUS, B., TRAN, V.-A., AND VAZIRGIANNIS, M. Cold Start Similar Artists Ranking with Gravity-Inspired Graph Autoencoders. *Proceedings of the Fifteenth ACM Conference on Recommender Systems* (2021), 443–452.

[314] SALHA-GALVAN, G., LUTZEYER, J. F., DASOULAS, G., HENNEQUIN, R., AND VAZIRGIANNIS, M. Modularity-Aware Graph Autoencoders for Joint Link Prediction and Community Detection. *arXiv preprint arXiv:2202.00961* (2022).

[315] SANDRYHAILA, A., AND MOURA, J. M. Big Data Analysis with Signal Processing on Graphs: Representation and Processing of Massive Data Sets with Irregular Structure. *IEEE Signal Processing Magazine 31*, 5 (2014), 80–90.

[316] SARAVANOU, A., TOMASI, F., MEHROTRA, R., AND LALMAS, M. Multi-Task Learning of Graph-Based Inductive Representations of Music Content. *Proceedings of the 22nd International Society for Music Information Retrieval Conference* (2021), 602–609.

[317] SCARSELLI, F., GORI, M., TSOI, A. C., HAGENBUCHNER, M., AND MONFARDINI, G. The Graph Neural Network Model. *IEEE Transactions on Neural Networks 20*, 1 (2008), 61–80.

[318] SCHALL, D. Link Prediction for Directed Graphs. *Social Network-Based Recommender Systems* (2015), 7–31.

[319] SCHEDL, M., ZAMANI, H., CHEN, C.-W., DELDJOO, Y., AND ELAHI, M. Current Challenges and Visions in Music Recommender Systems Research. *Proceedings of the International Journal of Multimedia Information Retrieval 7*, 2 (2018), 95–116.

[320] SCHLICHTKRULL, M., KIPF, T. N., BLOEM, P., VAN DEN BERG, R., TITOV, I., AND WELLING, M. Modeling Relational Data with Graph Convolutional Networks. *Proceedings of the European Semantic Web Conference* (2018), 593–607.

[321] SCHREIBER, H. Genre Ontology Learning: Comparing Curated with Crowd-Sourced Ontologies. *Proceedings of the International Conference on Music Information Retrieval* (2016), 400–406.

[322] SECHIDIS, K., TSOUMAKAS, G., AND VLAHAVAS, I. On the Stratification of Multi-Label Data. *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases* (2011), 145–158.

[323] SEN, P., NAMATA, G., BILGIC, M., GETOOR, L., GALLIGHER, B., AND ELIASSI-RAD, T. Collective Classification in Network Data. *AI Magazine 29*, 3 (2008), 93–93.

[324] SHAKIBIAN, H., AND CHARKARI, N. Mutual Information Model for Link Prediction in Heterogeneous Complex Networks. *Scientific Reports 7*, 1 (2017), 1–16.

[325] SHAPIRA, B., ROKACH, L., AND FREILIKHMAN, S. Facebook Single and Cross Domain Data for Recommendation Systems. *User Modeling and User-Adapted Interaction 23*, 2-3 (2013), 211–247.

[326] SHCHUR, O., MUMME, M., BOJCHEVSKI, A., AND GÜNNEMANN, S. Pitfalls of Graph Neural Network Evaluation. *NeurIPS 2018 Workshop on Relational Representation Learning* (2018).

[327] SHERVASHIDZE, N., SCHWEITZER, P., VAN LEEUWEN, E. J., MEHLHORN, K., AND BORGWARDT, K. M. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research 12*, 9 (2011).

[328] SHI, C., LI, Y., ZHANG, J., SUN, Y., AND PHILIP, S. Y. A Survey of Heterogeneous Information Network Analysis. *IEEE Transactions on Knowledge and Data Engineering 29*, 1 (2016), 17–37.

[329] SHI, H., FAN, H., AND KWOK, J. T. Effective Decoding in Graph Auto-Encoder using Triadic Closure. *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), 906–913.

[330] SHI, J., AND MALIK, J. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8 (2000), 888–905.

[331] SHI, L., ZHAO, W. X., AND SHEN, Y.-D. Local Representative-Based Matrix Factorization for Cold-Start Recommendation. *ACM Transactions on Information Systems (TOIS) 36*, 2 (2017), 1–28.

Bibliography

[332] SHIN, J., KIM, K., PARK, D., KIM, S., AND KANG, J. Bipartite Link Prediction by Intra-Class Connection Based Triadic Closure. *IEEE Access 8* (2020), 140194–140204.

[333] SHIOKAWA, H., FUJIWARA, Y., AND ONIZUKA, M. Fast Algorithm for Modularity-Based Graph Clustering. *Proceedings of the AAAI Conference on Artificial Intelligence 27*, 1 (2013), 1170–1176.

[334] SHUMAN, D. I., NARANG, S. K., FROSSARD, P., ORTEGA, A., AND VANDERGHEYNST, P. The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains. *IEEE Signal Processing Magazine* (2013), 83–98.

[335] SHWARTZ, V., AND DAGAN, I. Still a Pain in the Neck: Evaluating Text Representations on Lexical Composition. *Transactions of the Association for Computational Linguistics 7* (2019), 403–419.

[336] SIMONOVSKY, M., AND KOMODAKIS, N. GraphVAE: Towards Generation of Small Graphs using Variational Autoencoders. *Proceedings of the International Conference on Artificial Neural Networks* (2018), 412–422.

[337] SKOWRON, M., LEMMERICH, F., FERWERDA, B., AND SCHEDL, M. Predicting Genre Preferences from Cultural and Socio-Economic Factors for Music Retrieval. *Advances in Information Retrieval* (2017), 561–567.

[338] SLIVKINS, A. Introduction to Multi-Armed Bandits. *Foundations and Trends in Machine Learning 12*, 1-2 (2019), 1–286.

[339] SMITH, B., AND LINDEN, G. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Computing 21*, 3 (2017), 12–18.

[340] SORDO, M., CELMA, O., BLECH, M., AND GUAUS, E. The Quest for Musical Genres: Do the Experts and the Wisdom of Crowds Agree? *Proceedings of the International Society for Music Information Retrieval Conference* (2008), 255–260.

[341] SPEER, R., AND CHIN, J. An Ensemble Method to Produce High-Quality Word Embeddings. *arXiv preprint arXiv:1604.01692* (2016).

[342] SPIELMAN, D. A. Spectral Graph Theory and its Applications. *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science* (2007), 29–38.

[343] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: a Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[344] STEELE, J. M. *The Cauchy-Schwarz Master Class: an Introduction to the Art of Mathematical Inequalities.* Cambridge University Press, 2004.

[345] STEWART, G. W. On the Early History of the Singular Value Decomposition. *SIAM Review 35*, 4 (1993), 551–566.

[346] SU, X., AND KHOSHGOFTAAR, T. M. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence 2009* (2009).

[347] SUN, D., LI, D., DING, Z., ZHANG, X., AND TANG, J. Dual-Decoder Graph Autoencoder for Unsupervised Graph Representation Learning. *Knowledge-Based Systems* (2021), 107564.

[348] SUN, Y., AND HAN, J. Mining Heterogeneous Information Networks: Principles and Methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery 3*, 2 (2012), 1–159.

[349] SUTTON, R. S., AND BARTO, A. G. *Reinforcement Learning: An Introduction.* MIT Press, 2018.

[350] SWAMINATHAN, A., KRISHNAMURTHY, A., AGARWAL, A., DUDIK, M., LANGFORD, J., JOSE, D., AND ZITOUNI, I. Off-Policy Evaluation for Slate Recommendation. *Advances in Neural Information Processing Systems* (2017), 3632–3642.

[351] TANG, J., QU, M., WANG, M., ZHANG, M., YAN, J., AND MEI, Q. Line: Large-Scale Information Network Embedding. *Proceedings of the International Conference on World Wide Web* (2015), 1067–1077.

[352] TANG, L., JIANG, Y., LI, L., AND LI, T. Ensemble Contextual Bandits for Personalized Recommendation. *Proceedings of the 8th ACM Conference on Recommender Systems* (2014), 73–80.

[353] TAUSSKY, O. A Recurring Theorem on Determinants. *The American Mathematical Monthly 56*, 10P1 (1949), 672–676.

[354] TEKE, O., AND VAIDYANATHAN, P. P. Uncertainty Principles and Sparse Eigenvectors of Graphs. *IEEE Transactions on Signal Processing 65*, 20 (2017), 5406–5420.

[355] THOMPSON, W. R. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika 25*, 3/4 (1933), 285–294.

[356] Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. Learning Deep Representations for Graph Clustering. *Proceedings of the AAAI Conference on Artificial Intelligence* (2014), 1293–1299.

[357] Tixier, A., Malliaros, F., and Vazirgiannis, M. A Graph Degeneracy-Based Approach to Keyword Extraction. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (2016), 1860–1870.

[358] Tonnaer, L. Active Learning in VAE Latent Space. *M.Sc Thesis, Eindhoven University of Technology* (2017).

[359] Tran, P. V. Multi-Task Graph Autoencoders. *NeurIPS 2018 Workshop on Relational Representation Learning* (2018).

[360] Tran, V.-A., Salha-Galvan, G., Hennequin, R., and Moussallam, M. Hierarchical Latent Relation Modeling for Collaborative Metric Learning. *Proceedings of the Fifteenth ACM Conference on Recommender Systems* (2021), 302–309.

[361] Vaibhav, V., Huang, P.-Y., and Frederking, R. RWR-GAE: Random Walk Regularization for Graph Auto Encoders. *arXiv preprint arXiv:1908.04003* (2019).

[362] Van Den Oord, A., Dieleman, S., and Schrauwen, B. Deep Content-Based Music Recommendation. *Advances in Neural Information Processing Systems 26* (2013), 2643–2651.

[363] Van der Maaten, L., and Hinton, G. Visualizing Data using t-SNE. *Journal of Machine Learning Research 9*, 11 (2008).

[364] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph Attention Networks. *Proceedings of the International Conference on Learning Representations* (2018).

[365] Verma, J., Gupta, S., Mukherjee, D., and Chakraborty, T. Heterogeneous Edge Embedding for Friend Recommendation. *Proceedings of the European Conference on Information Retrieval* (2019), 172–179.

[366] Vinh, N. X., Epps, J., and Bailey, J. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *The Journal of Machine Learning Research 11* (2010), 2837–2854.

[367] Volkovs, M., Yu, G., and Poutanen, T. DropoutNet: Addressing Cold Start in Recommender Systems. *Advances in Neural Information Processing Systems* (2017), 4957–4966.

[368] Von Luxburg, U. A Tutorial on Spectral Clustering. *Statistics and computing 17*, 4 (2007), 395–416.

[369] Wahid-Ul-Ashraf, A., Budka, M., and Musial-Gabrys, K. Newton's Gravitational Law for Link Prediction in Social Networks. *Proceedings of the International Conference on Complex Networks and their Applications* (2017), 93–104.

[370] Wang, C., Pan, S., Long, G., Zhu, X., and Jiang, J. MGAE: Marginalized Graph Autoencoder for Graph Clustering. *Proceedings of the ACM International Conference on Information and Knowledge Management* (2017), 889–898.

[371] Wang, D., Cui, P., and Zhu, W. Structural Deep Network Embedding. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), 1225–1234.

[372] Wang, H., Wang, N., and Yeung, D.-Y. Collaborative Deep Learning for Recommender Systems. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), 1235–1244.

[373] Wang, J., Huang, P., Zhao, H., Zhang, Z., Zhao, B., and Lee, D. L. Billion-Scale Commodity Embedding for E-Commerce Recommendation in Alibaba. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), 839–848.

[374] Wang, Q., Mao, Z., Wang, B., and Guo, L. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering 29*, 12 (2017), 2724–2743.

[375] Wang, S., Hu, L., Wang, Y., He, X., Sheng, Q. Z., Orgun, M., Cao, L., Ricci, F., and Yu, P. S. Graph Learning Based Recommender Systems: A Review. *Proceedings of the International Joint Conference on Artificial Intelligence* (2021).

[376] Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W., and Yang, S. Community Preserving Network Embedding. *Proceedings of the 31st AAAI Conference on Artificial Intelligence* (2017), 203–209.

[377] Wang, Y., Ouyang, H., Wang, C., Chen, J., Asamov, T., and Chang, Y. Efficient Ordered Combinatorial Semi-Bandits for Whole-Page Recommendation. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (2017), 2746–2753.

Bibliography

[378] WANG, Y., WANG, L., LI, Y., HE, D., CHEN, W., AND LIU, T.-Y. A Theoretical Analysis of NDCG Ranking Measures. *Proceedings of the 26th Annual Conference on Learning Theory 8* (2013), 6.

[379] WANG, Z., ZHOU, R., AND SHEN, C. Regional Multi-Armed Bandits. *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics* (2018), 510–518.

[380] WEISFEILER, B., AND LEMAN, A. The Reduction of a Graph to Canonical Form and the Algebra which Appears Therein. *NTI, Series 2*, 9 (1968), 12–16.

[381] WEYL, H. Das asymptotische Verteilungsgesetz der Eigenwerte linearer partieller Differentialgleichungen (mit einer Anwendung auf die Theorie der Hohlraumstrahlung). *Mathematische Annalen 71*, 4 (1912), 441–479.

[382] WOLF, T., CHAUMOND, J., DEBUT, L., SANH, V., DELANGUE, C., MOI, A., CISTAC, P., FUNTOWICZ, M., DAVISON, J., SHLEIFER, S., ET AL. Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (2020), 38–45.

[383] WU, F., SOUZA, A., ZHANG, T., FIFTY, C., YU, T., AND WEINBERGER, K. Simplifying Graph Convolutional Networks. *Proceedings of the International Conference on Machine Learning* (2019), 6861–6871.

[384] WU, S., SUN, F., ZHANG, W., AND CUI, B. Graph Neural Networks in Recommender Systems: A Survey. *Journal of the ACM 37, 4, 111* (2020), 1–34.

[385] WU, X., AND CHENG, Q. Deepened Graph Auto-Encoders Help Stabilize and Enhance Link Prediction. *arXiv preprint arXiv:2103.11414* (2021).

[386] WU, Z., PAN, S., CHEN, F., LONG, G., ZHANG, C., AND PHILIP, S. Y. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems 32*, 1 (2020), 4–24.

[387] XIANG, S., WEN, D., CHENG, D., ZHANG, Y., QIN, L., QIAN, Z., AND LIN, X. General Graph Generators: Experiments, Analyses, and Improvements. *The VLDB Journal* (2021), 1–29.

[388] XU, K., HU, W., LESKOVEC, J., AND JEGELKA, S. How Powerful are Graph Neural Networks? *Proceedings of the International Conference on Learning Representations* (2019).

[389] XU, K., LI, C., TIAN, Y., SONOBE, T., KAWARABAYASHI, K.-I., AND JEGELKA, S. Representation Learning on Graphs with Jumping Knowledge Networks. *Proceedings of the International Conference on Machine Learning* (2018), 5453–5462.

[390] YAN, C., HAN, H., WANG, Z., AND ZHANG, Y. Two-Phase Multi-armed Bandit for Online Recommendation. *2021 IEEE 8th International Conference on Data Science and Advanced Analytics* (2021), 1–8.

[391] YANG, L., CAO, X., HE, D., WANG, C., WANG, X., AND ZHANG, W. Modularity Based Community Detection with Deep Learning. *Proceedings of the International Joint Conference on Artificial Intelligence* (2016), 2252–2258.

[392] YANG, Y., LICHTENWALTER, R. N., AND CHAWLA, N. V. Evaluating Link Prediction Methods. *Knowledge and Information Systems 45*, 3 (2015), 751–782.

[393] YANXIANG, L., DEKE, G., FEI, C., AND HONGHUI, C. User-Based Clustering with Top-n Recommendation on Cold-Start Problem. *Proceedings of the Third International Conference on Intelligent System Design and Engineering Applications* (2013), 1585–1589.

[394] YING, R., HE, R., CHEN, K., EKSOMBATCHAI, P., HAMILTON, W. L., AND LESKOVEC, J. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), 974–983.

[395] YOO, H., LEE, Y.-C., AND KIM, S.-W. Evaluation of Directed Network Embedding Methods. *Proceedings of the Korea Information Processing Society Conference* (2021), 377–378.

[396] YU, Y., AND WANG, X. Link Prediction in Directed Network and its Application in Microblog. *Mathematical Problems in Engineering* (2014).

[397] ZENG, H., ZHOU, H., SRIVASTAVA, A., KANNAN, R., AND PRASANNA, V. GraphSAINT: Graph Sampling Based Inductive Learning Method. *Proceedings of the International Conference on Learning Representations* (2020).

[398] ZHANG, D., YIN, J., ZHU, X., AND ZHANG, C. Network Representation Learning: A Survey. *IEEE Transactions on Big Data 6*, 1 (2018), 3–28.

[399] ZHANG, J., SHI, X., ZHAO, S., AND KING, I. STAR-GCN: Stacked and Reconstructed Graph Convolutional Networks for Recommender Systems. *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (2019), 4264–4270.

Bibliography

[400] ZHANG, J., XIA, J., LI, L., SHEN, B., WANG, J., AND XUAN, Q. Find Your Meal Pal: A Case Study on Yelp Network. *Graph Data Mining* (2021), 177–188.

[401] ZHANG, J., ZHENG, J., CHEN, J., AND XUAN, Q. Hyper-Substructure Enhanced Link Predictor. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (2020), 2305–2308.

[402] ZHANG, M., AND CHEN, Y. Link Prediction Based on Graph Neural Networks. *Advances in Neural Information Processing Systems 31* (2018), 5165–5175.

[403] ZHANG, M., JIANG, S., CUI, Z., GARNETT, R., AND CHEN, Y. D-VAE: A Variational Autoencoder for Directed Acyclic Graphs. *Advances in Neural Information Processing Systems 32* (2019), 1588–1600.

[404] ZHANG, S., YAO, L., SUN, A., AND TAY, Y. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys (CSUR) 52*, 1 (2019), 1–38.

[405] ZHANG, Y., LEVINA, E., AND ZHU, J. Detecting Overlapping Communities in Networks using Spectral Methods. *SIAM Journal on Mathematics of Data Science 2*, 2 (2020), 265–283.

[406] ZHANG, Z., CUI, P., AND ZHU, W. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2020), 1–1.

[407] ZHAO, L., AND AKOGLU, L. PairNorm: Tackling Oversmoothing in GNNs. *Proceedings of the International Conference on Learning Representations* (2020).

[408] ZHOU, C., LIU, Y., LIU, X., LIU, Z., AND GAO, J. Scalable Graph Embedding for Asymmetric Proximity. *Proceedings of the AAAI Conference on Artificial Intelligence* (2017), 2942–2948.

[409] ZHOU, J., CUI, G., HU, S., ZHANG, Z., YANG, C., LIU, Z., WANG, L., LI, C., AND SUN, M. Graph Neural Networks: A Review of Methods and Applications. *AI Open 1* (2020), 57–81.

[410] ZHOU, L., AND BRUNSKILL, E. Latent Contextual Bandits and their Application to Personalized Recommendations for New Users. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (2016), 3646–3653.

[411] ZIPF, G. K. *Human Behavior and the Principle of Least Effort.* Cambridge, MA, Addison-Wesle, 1949.

[412] ZOGHI, M., TUNYS, T., GHAVAMZADEH, M., KVETON, B., SZEPESVARI, C., AND WEN, Z. Online Learning to Rank in Stochastic Click Models. *Proceedings of the 34th International Conference on Machine Learning* (2017), 4199–4208.

[413] ZONG, S., NI, H., SUNG, K., KE, N. R., WEN, Z., AND KVETON, B. Cascading Bandits for Large-Scale Recommendation Problems. *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (2016), 835–844.

Eta nik, txoria nuen maite...

Titre : Contributions à l'apprentissage de représentations à partir d'autoencodeurs de graphes, et applications à la recommandation musicale

Mots clés : Autoencodeurs de graphes, Réseaux de neurones de graphes, Représentations vectorielles de nœuds, Prédiction de liens manquants, Détection de communautés, Recommandation musicale

Résumé : Les autoencodeurs de graphes (GAE) et les autoencodeurs variationnels de graphes (VGAE) se sont imposés comme deux puissants groupes de méthodes permettant de construire des représentations vectorielles des nœuds d'un graphe de manière non supervisée, avec des applications à divers problèmes d'apprentissage tels que la prédiction de liens manquants et la détection de communautés de nœuds. Néanmoins, au début de ce projet de thèse, les GAE et VGAE souffraient de limitations majeures. Ces dernières entravaient l'utilisation de ces modèles dans le cadre d'applications industrielles. Dans cette thèse, nous présentons plusieurs contributions permettant d'améliorer les GAE et VGAE afin de faciliter de telles utilisations.

Tout d'abord, nous proposons deux stratégies permettant de surmonter les problèmes de passage à l'échelle des GAE et VGAE, et d'entraîner ces modèles sur des graphes ayant des millions de nœuds et d'arêtes. Ces stratégies exploitent respectivement des techniques de dégénérescence de graphes et de décodage stochastique de sous-graphes. Par ailleurs, nous présentons nos GAE et VGAE inspirés de la gravité (de l'anglais *Gravity-Inspired GAE and VGAE*), qui constituent les premières extensions de ces modèles destinées aux graphes dirigés, qui sont omniprésents dans les applications industrielles. Nous étudions également des extensions destinées aux graphes dynamiques. En outre, nous démontrons que les GAE et VGAE existants sont souvent inutilement complexes, et nous proposons donc de les simplifier en ayant recours à des encodeurs linéaires. Enfin, nous présentons nos GAE et VGAE informés par la modularité (de l'anglais *Modularity-Aware GAE and VGAE*), qui permettent d'améliorer la détection de communautés de nœuds, tout en préservant de bonnes performances pour la prédiction de liens manquants.

Dans la dernière partie de cette thèse, nous évaluons nos méthodes sur plusieurs graphes extraits du service de *streaming* musical Deezer. Nous nous concentrons sur des problèmes de recommandation musicale à partir de graphes. En particulier, nous montrons que nos méthodes permettent d'améliorer la détection de communautés d'entités musicales à recommander aux mêmes utilisateurs, mais aussi de mieux classer des artistes similaires dans un contexte de démarrage à froid (de l'anglais *cold start*), et enfin de mieux modéliser la perception des genres musicaux à travers différentes cultures. Pour terminer, nous présentons également deux autres modèles, récemment déployés en production chez Deezer afin de recommander de la musique à des millions d'utilisateurs. Bien qu'étant moins directement liés aux GAE et VGAE, ils fournissent un point de vue complémentaire sur des sujets de recommandation musicale connexes à ceux étudiés précédemment.

Title : Contributions to Representation Learning with Graph Autoencoders and Applications to Music Recommendation

Keywords : Graph Autoencoders, Graph Neural Networks, Node Embedding Representations, Missing Link Prediction, Community Detection, Music Recommendation

Abstract : Graph autoencoders (GAE) and variational graph autoencoders (VGAE) emerged as two powerful groups of unsupervised node embedding methods, with various applications to graph-based machine learning problems such as link prediction and community detection. Nonetheless, at the beginning of this PhD project, GAE and VGAE models were also suffering from key limitations, preventing them from being adopted in the industry. In this thesis, we present several contributions to improve these models, with the general aim of facilitating their use to address industrial-level problems involving graph representations.

Firstly, we propose two strategies to overcome the scalability issues of previous GAE and VGAE models, permitting to effectively train these models on large graphs with millions of nodes and edges. These strategies leverage graph degeneracy and stochastic subgraph decoding techniques, respectively. Besides, we introduce Gravity-Inspired GAE and VGAE, providing the first extensions of these models for directed graphs, that are ubiquitous in industrial applications. We also consider extensions of GAE and VGAE models for dynamic graphs. Furthermore, we argue that GAE and VGAE models are often unnecessarily complex, and we propose to simplify them by leveraging linear encoders. Lastly, we introduce Modularity-Aware GAE and VGAE to improve community detection on graphs, while jointly preserving good performances on link prediction.

In the last part of this thesis, we evaluate our methods on several graphs extracted from the music streaming service Deezer. We put the emphasis on graph-based music recommendation problems. In particular, we show that our methods can improve the detection of communities of similar musical items to recommend to users, that they can effectively rank similar artists in a cold start setting, and that they permit modeling the music genre perception across cultures. At the end of this thesis, we present two additional models, recently deployed in production on the Deezer service to recommend music to millions of users. While being less directly linked to GAE and VGAE models, they provide a complementary perspective on music recommendation topics related to the ones we previously studied.