UNIVERSITÉ DU QUÉBEC À MONTRÉAL

EXPLORING THE REPRESENTATIONAL POWER OF GRAPH AUTOENCODERS

DISSERTATION

PRESENTED

AS PARTIAL REQUIREMENT

TO THE MASTERS IN COMPUTER SCIENCE

BY

MAROUN HADDAD

OCTOBER 2021

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ÉXPLORATION DU POUVOIR DE REPRÉSENTATION DES GRAPHES AUTOENCODEURS

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
MAROUN HADDAD

OCTOBRE 2021

ACKNOWLEDGEMENT

I would like to dedicate this thesis to my parents and thank them for their continuous encouragement and support throughout the years.

I would like to also say a special thank you to my supervisor, Mr. Mohamed Bouguessa. His advice and feedback helped guide my research and greatly facilitated the writing of this thesis.

CONTENTS

LIST OF TABLES v
LIST OF FIGURES vi
RÉSUMÉ ix
ABSTRACT x
CHAPTER I INTRODUCTION 1
1.1 Context 1
1.2 Motivations 4
1.3 Contributions 6
1.4 Thesis Plan 8
CHAPTER II UNSUPERVISED GRAPH EMBEDDING 9
2.1 Background Information 9
2.2 Node Embedding 13
2.3 Matrix Factorization 16
2.4 Random Walk Techniques 20
2.5 Graph Autoencoders 24
CHAPTER III EXPLORING THE POWER OF GRAPH AUTOENCODERS 32
3.1 Scope of the Experiments 32
3.2 Datasets and Compared Baselines 33
3.3 Topological Features Prediction 37
3.4 Visualisation 45
3.5 **Task 1**: Embedding Clusters Homogeneity 48
3.6 **Task 2**: Node Clustering 50
3.7 **Task 3**: Node Classification 53
3.8 Analysis and Recommendations 54

CHAPTER IV CASE STUDY 56
4.1 DeepInf Framework Summary 57
4.2 Datasets and Compared Baselines 59
4.3 Prediction Results and Discussion 61
CHAPTER V CONCLUSION 65
BIBLIOGRAPHY 67

LIST OF TABLES

Table		Page
3.1	Graph Datasets.	33
3.2	Cora - Degree.	41
3.3	fly-drosophila-medulla-1 - Local Clustering Score.	41
3.4	email-Eu-core - Eigenvector Centrality.	41
3.5	soc-sign-bitcoin-otc - Betweeness Centrality.	41
3.6	Brazil Air-Traffic - Triangle Count.	41
3.7	Embedding clusters homogeneity.	49
3.8	Clustering node embeddings - using K-means.	51
3.9	Clustering node embeddings - using FINCH.	52
3.10	Node Classification - using Logistic Regression.	53
4.1	Graph Datasets for the DeepInf experiments - The field *Observations* indicates the number of sampled ego-graphs. The figures are retrieved from (Qiu et al., 2018).	61
4.2	Prediction results on the four datasets for the five variations of DeepInf-GCN.	62
4.3	Relative gain of DeepInf-CONCAT-64 in terms of AUC against DeepInf-GCN-128 and DeepInf-GCN-128-HF (With Handcrafted Features).	62
4.4	Comparison between the results of DeepInf-CONCAT with hidden layers of size 64 and hidden layers of size 128.	63

LIST OF FIGURES

Figure Page

1.1 A chart summarizing the consecutive steps of our empirical study. 2

2.1 A_1 is an adjacency matrix of the undirected graph G_1. A_2 is an adjacency matrix of the directed graph G_2. A_3 is an adjacency matrix of the weighted directed graph G_3. $X \in \mathbb{R}^{|V| \times k}$ is an attributes matrix where the i^{th} row represents the attributes vector of the i^{th} node. 11

2.2 A weighted graph, where the weights on the edges represent the force of the link between the nodes. 12

2.3 An image representing a conceptual node embedding architecture in the form of an Encoder-Decoder (Hamilton et al., 2017a). For two nodes v_i and v_j, the Encoder (blue) projects the nodes into their respective embedding vectors z_i and z_j. The Decoder (green) reconstructs a proximity measure $s'_{i,j}$ between v_i and v_j out of their representation vectors. The loss l between the reconstructed proximity s' and the groundtruth proximity s is calculated and the error is used to update the Encoder part of the model. 14

2.4 A chart showing both steps in a DeepWalk model: 1) Data preparation and 2) Model Training. For the model training step, we show both the general architecture of the Skip-Gram network and a mock-up example in a matrix format. 22

2.5 An example of message (attributes) passing from layer to layer in the GNN from the perspective of one node (central node: orange). We can observe that the Layer 1 allows the passing of the first-level neighborhood messages (K^1: green) to the central node. While Layer 2 allows the passing of the second-level neighborhood messages (K^2: blue) to the central node. 24

2.6 A layout of a GNN architecture with N layers using the SUM aggregation rule, used in a classification task. The output of each layer (green arrows) is used as input for the next layer. The input of the first layer is X (the attributes matrix). The loss is calculated between the output of the last layer (n) and the groundtruth (real node labels). Backpropagation (red arrows) is used to update the weights in all layers. 26

2.7 A layout of a GAE architecture using the spectral rule. The encoder part is made of a sequence of GCN layers. While the decoder is made of a *Sigmoid* applied on the latent space (Z) multiplied by its transpose (Z^T). 27

2.8 An example of curve smoohting, where each point on the line is adjusted according to the position of its neighbors. 28

2.9 A 2D-tSNE projection of embeddings generated by a GCN of five layers on the Zachary karate club dataset. 30

3.1 Seven variations of Graph Autoencoders. 34

3.2 The number of times each model outperformed the others on the ten evaluation metrics over the eleven datasets per feature. 44

3.3 2D t-SNE projection of the embeddings of Cora. The embeddings are colored according to the seven different ground-truth labels of the dataset. 46

3.4 The embeddings of 5 models projected in 2D for the Cora dataset and colored according to the the topological classes of the vertices (Degree Class, Betweenness Class, Local Clustering Score Class, Eigenvector Centrality , Triangle Count). The colors listed in ascending order are: Blue, Red, Orange, Green, Yellow and Cyan. The figure clearly shows that the embeddings of GAE_FIRST and GAE_CONCAT (highlighted in bold) are organized according to the topological classes of the vertices. 46

3.5 An illustration of the hierarchical arrangement of the node Degree classes for the embeddings generated by GAE_FIRST. Every color represents a range of Degrees as defined in the binning phase of the experiments. The plots show that the degrees in the embedding are arranged in a hierarchy from the smallest (Blue) to the largest (Cyan). 47

3.6 2D t-SNE projection of the GAE_FIRST embeddings for USA Air-Traffic dataset - The colors show that, to a large extent, the vertices that belong to the same ground-truth class also belong to the same Degree class. 50

4.1 A summary of the DeepInf framework, slightly modified for this work. The node v (blue) is the ego-node while the node u (orange) is a neighbor. For the original architecture, please refer to (Qiu et al., 2018). 57

RÉSUMÉ

L'apprentissage des représentations s'est établi comme une méthode efficace pour effectuer de diverses tâches d'apprentissage sur les graphes. Cette approche vise à projeter le graphe dans un espace vectoriel qui encode à la fois sa structure et son contenu. Bien que l'apprentissage des représentations ait eu un grand succès sur de nombreuses tâches d'apprentissage sur les graphes, la compréhension des structures qui sont encodées par les représentations vectorielles reste limitée. Par exemple, nous nous demandons si les caractéristiques topologiques, telles que le nombre de triangles, le degré du nœud, et autres mesures de centralité sont encodés concrètement dans les représentations. De plus, nous nous demandons si la présence de ces structures dans les représentations est nécessaire pour une meilleure performance sur les tâches descriptives et prédictives comme le clustering et la classification. Pour adresser ces questions de recherche, nous menons une étude empirique approfondie sur trois classes de modèles d'apprentissage de représentations non-supervisés et sept variations de Graphe Autoencodeurs. Nos résultats révèlent que cinq caractéristiques topologiques, en particulier: le degré d'un nœud, le score de regroupement local (*Local Clustering Score*), la centralité d'interdépendance (*Betweenness Centrality*), la centralité des vecteurs propres (*Eigenvector Centrality*) ainsi que le nombre de triangles (*Triangle Count*) sont concrètement préservés dans la première couche du Autoencodeur qui utilise la règle d'agrégation SUM, à condition que le modèle préserve la proximité du deuxième ordre. Nous présentons d'autres preuves de l'encodage de ces caractéristiques en révélant une hiérarchie dans la distribution des caractéristiques topologiques dans les représentations du modèle susmentionné, particulièrement pour: le degré du nœud, le score de regroupement local, la centralité d'interdépendance et le nombre de triangles. Nous montrons également qu'un modèle avec de telles propriétés peut surpasser d'autres modèles sur certaines tâches en aval (p. ex. clustering ou classification), en particulier lorsque les caractéristiques topologiques préservées sont pertinentes pour la tâche à accomplir. Enfin, nous évaluons l'impact de nos résultats en effectuant une étude de cas qui se rattache à la prédiction de l'influence sociale.

Mots-clés Intelligence artificielle, Apprentissage profond, Réseaux de neurones pour les graphes, Apprentissage des représentations, Characteristiques topologiques.

ABSTRACT

Representation learning has proven to be an effective method for performing various learning tasks in the domain of graphs. This approach aims to project the graph into an embedding space that captures both its structure and content. While representation learning has yielded a great success on many graph learning tasks, there is little understanding behind the structures that are being captured by these embeddings. For example, we wonder if the topological features, such as the Triangle Count, the Degree of the node, and other centrality measures are concretely encoded in the embeddings. Furthermore, we ask if the presence of these structures in the embeddings is necessary for a better performance on the downstream tasks, such as clustering and classification. To address these questions, we conduct an extensive empirical study over three classes of unsupervised graph embedding models and seven different variants of Graph Autoencoders. Our results show that five topological features: the Degree, the Local Clustering Score, the Betweenness Centrality, the Eigenvector Centrality, and Triangle Count are concretely preserved in the first layer of the Graph Autoencoder that employs the SUM aggregation rule, under the condition that the model preserves the second-order proximity. We supplement further evidence for the presence of these features by revealing a hierarchy in the distribution of the topological features in the embeddings of the aforementioned model, in particular for the Degree, Betweenness, Clustering Score and Triangle Count. We also show that a model with such properties can outperform other models on certain downstream tasks, especially when the preserved features are relevant to the task at hand. Finally, we evaluate the suitability of our findings through a test case study related to social influence prediction.

Keywords Artificial Intelligence, Deep Learning, Graph Neural Networks, Representation Learning, Topological Features.

CHAPTER I

INTRODUCTION

1.1 Context

Graph embedding approaches have emerged in recent decades as a powerful set of tools for learning representations on graphs, as well as for improving the performance on downstream tasks such as node clustering (Wang et al., 2019a), classification (Wu et al., 2019) and link prediction (Ma et al., 2019). While these embedding techniques do bring a lot of advantages over classical methods, such as learning with handcrafted features or direct learning on graphs, they do face a set of challenges that affect their quality and performance. Among these challenges, the greatest contributor to a "good" graph embedding is a vector representation that preserves the structure of the graph (Goyal & Ferrara, 2018). However, while many studies cite "the preservation of the graph structure by the embedding" as a requirement (Goyal & Ferrara, 2018; Cai et al., 2018; Hamilton et al., 2017a), few works have attempted to investigate concretely this assumption and study its effect on the downstream learning tasks. To this end, this work aims at answering the following important questions: (1) Are relevant topological structures being captured in the graph embeddings? (2) If yes, which embedding models best preserve these structures? and (3) What is the effect of preserving these structures on the downstream learning tasks? We believe that answering these questions will

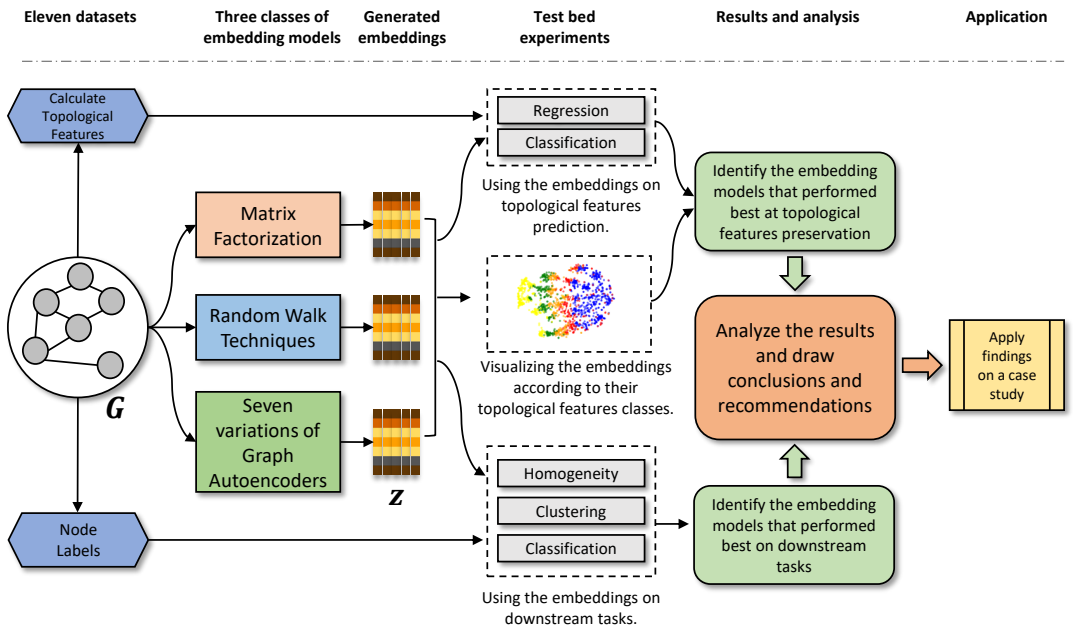

Figure 1.1: A chart summarizing the consecutive steps of our empirical study.

help us better understand and explain the content of the graph embeddings and the source of their representational power.

It is important to note that the preservation of certain structural characteristics such as the orders of proximity is trivial, since the models that generate these embeddings are generally optimized to preserve a certain order of proximity between the embedded components (Goyal & Ferrara, 2018; Cai et al., 2018). This means that two components having a high value for a certain order of proximity between them, will have similar vector representations. However, the preservation of other important topological features such as the Degree of the node, its Local Clustering Score, and other centrality measures is yet to be studied in depth.

To address the questions put forward by this work, we perform a detailed empirical study. First, we position the problem of graph embedding in the context of *Unsupervised Learning* on *attributed* graphs. Accordingly, we study the embeddings generated by three different classes of unsupervised graph embedding

models: (1) Matrix Factorization, (2) Random Walk techniques, and (3) Graph Autoencoders (GAE). We further focus on Graph Autoencoders and study the embeddings generated by seven different variations of GAE.

Second, we investigate the preservation of the topological features in the embeddings and pinpoint the models that best preserve these structures. We hypothesize, that if a certain topological feature can be approximated using the embeddings, then, the embeddings do encode certain information about the approximated feature. To this end, we adopt two strategies: (1) We directly predict the topological features with Linear Regression, using the embeddings as attributes (Rizi & Granitzer, 2017); (2) We transform the problem of preservation of the topological features into a classification problem (Bonner et al., 2019). Furthermore, to better understand the content of the embeddings, we visualize them in 2D space. This visualization will serve as an additional proof for the success of some models over others in preserving the topological features.

Third, we study the effect of the preservation of the topological features on the downstream tasks such as node clustering and node classification. We believe finding a positive correlation between the preservation of the topological features and a good performance on the downstream tasks will be a strong argument for the necessity of having these features encoded in the embeddings. To this end, we evaluate the performance when the embeddings are used on three separate tasks: (1) We cluster the embeddings according to the ground truth of each dataset and measure the homogeneity of each cluster, (2) We evaluate the suitability of the embeddings on the task of node clustering, and (3) We evaluate the effect of the embeddings on the task of node classification. Finally, we evaluate the suitability of our findings through a test case study related to social influence prediction. The performance gains demonstrated in the case study would be an argument for the importance of adopting a "Topological Features Preservation" strategy when

building a GNN architecture. Figure 1.1 illustrates the sequence of steps to be undertaken in our empirical study.

1.2 Motivations

While graph embedding as a whole has received a great deal of interest in recent years, few studies have focused on the representational power of the embeddings or the structures that are being encoded in the vectors. Two closely related works to our study are (Rizi & Granitzer, 2017) and, (Bonner et al., 2019). In (Rizi & Granitzer, 2017), the authors only focused on the Random Walk based techniques, where they attempted to use the embeddings as attributes to reconstruct the centrality measures of the nodes using linear regression. While the authors did find that the Closeness Centrality was being approximated to an extent by the embeddings, they had poor results for the preservation of the other topological features such as the Degree, Betweenness Centrality, and Eigenvector Centrality.

The authors in (Bonner et al., 2019) hypothesized that if a mapping can be found between the embedding space and the topological features, then, these features are approximately captured in the embedding space. To this end, they transformed the problem of the preservation of topological features into a classification problem, where the centrality measures were divided into six classes using histogram binning. Subsequently, multiple classification models such as SVM, Logistic Regression, and Multi-Layer Neural Networks were used to predict the topological class of each node using the embeddings as attributes. However, their results were inconclusive on the majority of the tested benchmarks, except for the Eigenvector Centrality on the ego-Facebook dataset. Furthermore, the paper (Bonner et al., 2019) missed studying the embeddings of the Graph Neural Network (GNN) models under the assumption that the GNN models only cover

supervised learning. This assumption is, however, not necessarily true. Graph Autoencoders (Kipf & Welling, 2016) have been widely used as an unsupervised learning variation of GNN for generating embeddings that hold state-of-the-art results on many unsupervised graph learning tasks (Wang et al., 2019a; Schlichtkrull et al., 2018; Hasanzadeh et al., 2019).

In the domain of Graph Neural Networks, the Graph Isomorphism Network (GIN) (Xu et al., 2019) was one of the first articles to study the expressive power of the SUM aggregation rule. In their seminal work, the authors in (Xu et al., 2019) studied the capacity of several variants of GNN at distinguishing different graph structures. By representing the features of the node neighbors as multisets, the authors argued that a maximally powerful GNN would always aggregate different multisets into different representations. Subsequently, they proposed GIN, a simple GNN that is as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test (Weisfeiler & Leman, 1968). The WL test is broadly used to distinguish different graph structures. WL is similar to the Graph Neural Networks in that it iteratively updates the node features by aggregating the features of the node neighbors (Xu et al., 2019). On the other hand, GIN uses the SUM rule to aggregate the features of the node neighbors and an MLP to map the aggregated features into a representation. Unlike the MEAN and MAX aggregators, the SUM aggregator can represent an injective multiset function, making it maximally powerful at distinguishing different multisets. Similarly, unlike the 1-layer perceptron, an MLP is a universal approximator of multiset functions, allowing it to map different multisets to different representations. However, in our work, we go beyond the capacity of the SUM aggregator at distinguishing general graph structures and study its ability to capture specific graph properties. Furthermore, we investigate the graph structures captured by each layer of the Graph Autoencoder, contributing to a better explainability and understanding of the model functionality.

Lastly, the authors in (Wu et al., 2019) developed a Degree-specific GNN model that explicitly preserves the Degree of the nodes in the embedding using multi-task learning. They found that the presence of the Degree in the embedding did improve the performance of the model. However, the datasets used in (Wu et al., 2019) to evaluate the model are generally skewed in favor of having the Degree in the embedding, either by having the Degree of the node as the ground-truth to be predicted, or by having the ground-truth labels heavily correlated with the Degree of the node. In our study, we show that the Degree and other topological features are naturally preserved in the first layer of the GNN that utilizes the SUM aggregation rule, without the need for explicitly preserving them. On the other hand, our results also suggest that the effect of having the topological features preserved in the embeddings is task-dependent, and may not always lead to a better performance on the downstream tasks.

1.3 Contributions

Motivated by the inconclusive results of the previous studies (Rizi & Granitzer, 2017; Bonner et al., 2019) and their limited scope in covering the most recent graph embedding models such as the GNN, we perform, in this work, an extensive empirical study that covers the Graph Autoencoder models. Our experimental results reveal that the topological features are concretely preserved in the first layer of the Graph Autoencoders that employs the SUM aggregation rule. We further accentuate these findings by revealing an organized hierarchical structure in the distribution of the node Degree, Betweenness, Clustering Score and Triangle Count in the embeddings of this particular model. The visualized embeddings, depicted in the Experiments section, corroborate our claim.

To the extent of our knowledge, we are the first study to investigate the preservation of the topological features for the Graph Autoencoders and their effect on the downstream tasks, and the first study to show substantial and conclusive results in favor of one type of model, across several benchmarks.

We summarize the significance of our work[1] as follows:

- We investigate the representational power of graph embeddings in an extensive set of four test bed experiments that cover three different types of embedding models and seven different variants of Graph Autoencoders over eleven datasets.
- We empirically demonstrate that the topological features are best preserved in the first layer of the Graph Autoencoder that employs the SUM aggregation rule under the condition that the model preserves the second-order proximity.
- We experimentally show that the presence of the topological features in the embeddings can generally improve the performance on the downstream task when the preserved features are relevant to the task.
- We show significant performance gains in a social influence prediction case study that leverages the findings of our experiments and demonstrates the importance of having a "Topological Features Preservation" strategy when building a GNN architecture.

---

[1]Published in Neurocomputing: `https://doi.org/10.1016/j.neucom.2021.06.034`. The related framework and code have been accepted for publication by Software Impacts: Maroun Haddad, Mohamed Bouguessa (2021). TopoDetect: "Framework for topological features detection in graph embeddings".

Finally, for reputability and reproducibility, we draw the attention of the reader that the code, the datasets used in this work as well as more experimental results are also available at: `https://github.com/MH-0/RPGAE`

1.4 Thesis Plan

The rest of this thesis is organized as follows:

- Chapter 2, introduces graph structures, node embedding, as well as the unsupervised embedding models used in this study. We elaborate on three categories of models (1) Matrix Factorization, (2) Random Walk techniques, and (3) Graph Autoencoders.
- Chapter 3, delineates our extensive test bed experiments, their setup, parameters, and results. The chapter is further enriched with visualizations, an in-depth analysis for the performance of the different models, and concludes with our recommendations and suggestions.
- Chapter 4, is devoted for a case study on social influence prediction, where we put the findings of our study and our recommendations into application.
- Chapter 5, concludes the thesis with an overall discussion of our work, its importance and impact in regards to graph representation learning, and discusses future lines of research.

CHAPTER II

UNSUPERVISED GRAPH EMBEDDING

In this chapter, we start by introducing important background information related to graph-structured data. Next, we discuss unsupervised node embedding in the context of plain and attributed graphs and formalize the problem according to a general encoder-decoder architecture. Finally, we elaborate on three categories of representation learning models that would be employed in the subsequent test bed experiments. The models are laid out according to their chronological appearance in literature: (1) Matrix Factorization techniques and one of their earliest algorithms Laplacian Eigenmaps. (2) Random Walk techniques and two of their most known variants, DeepWalk and Nod2Vec. (3) Graph Autoencoders (GAE), which represent the core of our study and which we introduce in the context of deep learning on graphs and Graph Neural Networks.

2.1 Background Information

A graph is defined as a set of objects connected in pairs via links. In the literature, the objects are referred to as nodes (or vertices) and the links as edges. This type of data structure lends itself naturally to many systems in the real world where we are interested in modeling the relationships or interactions of the internal components of the system. For example, when analyzing the connections between

users on social networks (e.g., friends on Facebook or followers on Twitter), product reviews by customers on shopping sites (e.g., recommendation systems), and information research and semantic reasoning on knowledge graphs (e.g., Google Brain determining the context of a search query). Given the diversity and complexity of real-world systems, several types of graph structures have been adapted to fit each type of system. In this study, we focus on directed/undirected and attributed/unattributed graphs.

We start by defining the graph in its simplest form as a *plain undirected* graph. Let $G(V, E)$ be a plain undirected graph. G is defined as a set of nodes $V = \{v_1, v_2, ..., v_n\}$ and a set of edges $E = \{e_1, e_2, ..., e_m\}$ such as $e_k = (v_i, v_j) = (v_j, v_i)$ with $v_i, v_j \in V$ and $e_k \in E$. The structure of a graph G can be expressed as an adjacency matrix that we denote by A with a size $|V| \times |V|$. Every element A_{ij} in A represents a link between two nodes v_i and v_j. For plain undirected graphs, A is binary and symmetrical, and we have $A_{ij} = A_{ji} \in \{0, 1\}$; where 1 indicates the presence of an edge between two nodes v_i and v_j and 0 indicates the absence of the edge. The adjacency matrix plays an important role in the analysis and treatment of graphs and serves the role of input or reconstruction target for several representation learning models. The graph G_1 in Figure 2.1 illustrates a plain undirected graph and A_1 its corresponding adjacency matrix.

On the other hand, the edges in a graph can also have one specific direction. In this case, we refer to the graph as *directed*. $G(V, E)$ is a directed graph if $\forall e_k \in E$, e_k has a direction from a source node v_i to a destination node v_j such as $e_k = (v_i, v_j) \neq (v_j, v_i)$. When illustrating a directed graph, the edges are represented as arrows from the source node to the destination node. For example, G_2 and G_3 in Figure 2.1 depict a directed graph. Furthermore, the adjacency matrix of an undirected graph can be asymmetrical (see for example A_2 and A_3 in Figure 2.1). Note that an undirected graph can be considered as a directed graph

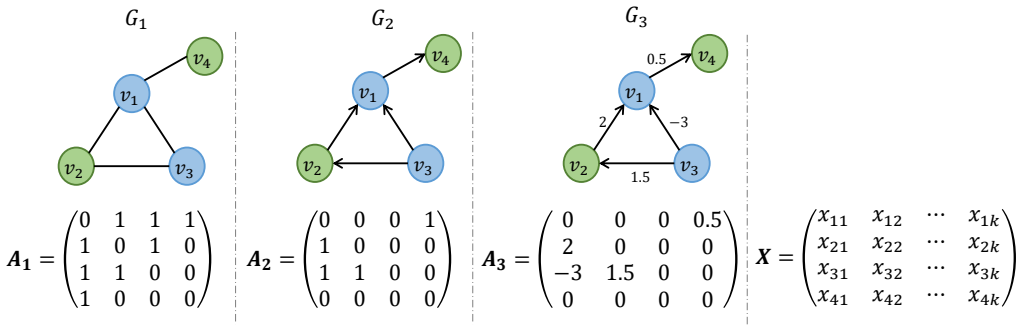

Figure 2.1: A_1 is an adjacency matrix of the undirected graph G_1. A_2 is an adjacency matrix of the directed graph G_2. A_3 is an adjacency matrix of the weighted directed graph G_3. $X \in \mathbb{R}^{|V| \times k}$ is an attributes matrix where the i^{th} row represents the attributes vector of the i^{th} node.

where all the edges are bidirectional. Directed graphs are an important structure when representing systems where the relationships that connect the entities only go in one direction. For example, in Twitter, a user can follow another user, without the latter following him back.

Moreover, the strength of the link between two nodes in a graph can be modeled as a *weight* on the edge connecting them. We refer to this type of graph in literature as a *weighted graph*[1]. $G(V, E, W)$ is a weighted graph, if every edge $e_k \in E$ has a weight $w_k \in W$ such as $w_k \in \mathbb{R}$ and $e_k = (v_i, v_j, w_k)$. In this case, every element in the adjacency matrix $A_{ij} = w_k$, that is, A_{ij} holds the weight on the edge e_k. For example, in Figure 2.1, A_3 is an adjacency matrix corresponding to the weighted graph G_3. Weighted graphs are important for modeling systems where the force of the relationships in the system may vary. For example, in recommender systems, the number of stars that a customer grants to a product in a review could be modeled as a weight on the edge.

[1]We have explored weighted graphs in this section to facilitate the comprehension for the reader when we mention this type of graph in the context of node proximity and Laplacian Eigenmaps. However, in our experimental datasets, we do not include weighted graphs and leave the study of the preservation of topological features in the context of weighted graphs for future work.

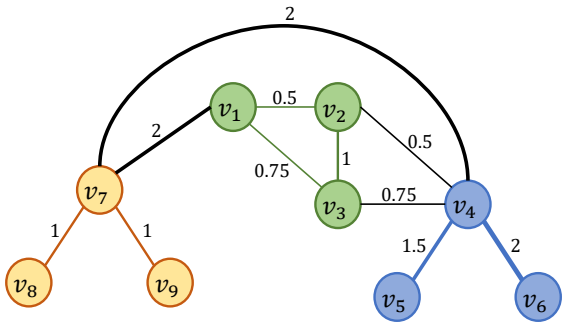

Figure 2.2: A weighted graph, where the weights on the edges represent the force of the link between the nodes.

Unlike plain graphs, *attributed graphs* are associated with vectors that incorporate additional information about the nodes. A graph $G(V, E, X)$ is attributed if every node v_i is associated to an attributes vector $x_i \in X$, such as $X \in \mathbb{R}^{|V| \times k}$, where k represents the dimensionality of the attribute vectors. The matrix X in Figure 2.1 designates the attributes associated to the nodes of the graphs G_1, G_2 and G_3. Attributed graphs are important for modeling systems where the topological structure alone is not sufficient to represent the entirety of the system.

Now that we have introduced four different categories of graphs that are of interest to this study, we tackle the concept of "node proximity" which lies at the heart of numerous node embedding models. The proximity between two nodes in a graph is studied on several orders or levels, where each order corresponds to a neighborhood level of a node. In the context of weighted graphs, the **first-order proximity** ($k = 1$) between the nodes is measured as the force of the direct link between them. The higher the force of the link, the higher the proximity. We denote the first-order proximity between two nodes v_i and v_j by $s_{ij}^{(1)}$. As we can observe in Figure 2.2, for nodes v_1 and v_2, we have $s_{12}^{(1)} = w_{12} = 0.5$, whereas $s_{19}^{(1)} = 0$ for v_1 and v_9. We denote the proximity vector of order $k = 1$ between v_1 and all other nodes as $s_1^{(1)} = [s_{11}^{(1)}, s_{12}^{(1)}, ..., s_{1|V|}^{(1)}] = [0,\ 0.5,\ 0.75,\ 0,\ 0,\ 0,\ 2,\ 0,\ 0]$.

While the first-order proximity is defined in regards to the direct link between two nodes, their **second-order of proximity ($k = 2$)** is measured in reference to the similarity between their first-level neighborhoods. We denote the second-order of proximity between two nodes v_i and v_j by $s_{ij}^{(2)}$. Here, $s_{ij}^{(2)}$ can be estimated by calculating a similarity measure (e.g., Cosine similarity) between $s_i^{(1)}$ and $s_j^{(1)}$ (first-order proximity vectors of v_i and v_j, respectively). For example, in Figure 2.2, if we take v_1 and v_4, we have $s_1^{(1)} = [0, 0.5, 0.75, 0, 0, 0, 2, 0, 0]$ and $s_4^{(1)} = [0, 0.5, 0.75, 0, 1.5, 2, 2, 0, 0]$. Accordingly, $s_{14}^{(2)} = cosine(s_1^{(1)}, s_4^{(1)}) = 0.66$. Here, we can remark that, even though v_1 and v_4 are not linked directly (that is, their first-order proximity $s_{14}^{(1)} = 0$), they share a large portion of their neighborhood. On the other hand, v_1 and v_7 have a relatively high first-order proximity ($s_{17}^{(1)} = 2$), whereas $s_{17}^{(2)} = cosine(s_1^{(1)}, s_7^{(1)}) = 0$, since their neighborhoods do not intersect.

Similarly, the proximity can be generalized to higher orders, where the **$\mathbf{K^{th}}$-order of proximity ($k = K$)** between two nodes v_i and v_j is measured in reference to the similarity between their $K-1$ level neighborhoods ($s_i^{(K-1)}$ and $s_j^{(K-1)}$). Finally, it is important to note that the orders of proximity are not limited to the weights on the edges but can be calculated for other measures of similarity between the nodes. For example, they can be measured by the presence or the absence of an edge between two nodes in plain graphs, or the similarity between the nodes attributes in attributed graphs. The orders of proximity encode vital information about the position of the node in the graph and its relationship with other nodes and are one of the main characteristics to be preserved by the embedding models.

2.2 Node Embedding

Virtually, the problem of node embedding aims to encode the nodes of the graph in a vector representation form while preserving their important characteristics.

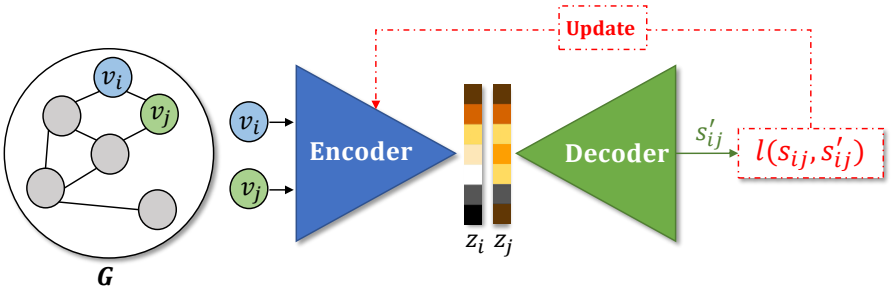

Figure 2.3: An image representing a conceptual node embedding architecture in the form of an Encoder-Decoder (Hamilton et al., 2017a). For two nodes v_i and v_j, the Encoder (blue) projects the nodes into their respective embedding vectors z_i and z_j. The Decoder (green) reconstructs a proximity measure $s'_{i,j}$ between v_i and v_j out of their representation vectors. The loss l between the reconstructed proximity s' and the groundtruth proximity s is calculated and the error is used to update the Encoder part of the model.

While many different types of embeddings exist, including node (Song et al., 2018), edge (Li et al., 2019), cluster and entire graph embedding (Dong et al., 2020), in this study, we focus on node embedding, as it is among the most widely used. Subsequently, for the rest of the thesis, node embedding and graph embedding are used interchangeably.

Definition: Let $G = (E, V)$ be a graph with $V = \{v_1, v_2, ..., v_n\}$ a set of nodes of G and $E = \{e_1, e_2, ..., e_m\}$ its set of edges. f is an embedding model such as $\forall\ i \in n,\ f(v_i) \rightarrow z_i \in \mathbb{R}^d$, where z_i is the representation vector of v_i with $d \ll n$ (d is the dimension of the embedding space while n denote the dimension of the adjacency matrix of G which in turn corresponds to the number nodes in G). The embedding model aims at projecting the nodes into an embedding (or latent) space such as the nodes that are "close" in the graph have similar vector representations (Goyal & Ferrara, 2018; Hamilton et al., 2017a; Cai et al., 2018). Therefore, the main challenge of any embedding model would be to define the "closeness" or proximity measure to be preserved in the embedding space between the nodes (Goyal & Ferrara, 2018).

Before exploring three categories of node embedding models, that are: Matrix factorization based techniques, Random Walk based algorithms and Graph Autoencoders, we review a unified framework that can be used to describe these various approaches. The authors in (Hamilton et al., 2017a) organized the different node embedding models under one unifying architecture in the form of an encoder-decoder. In this conceptual framework, depicted in Figure 2.3, we have $f = ENC$ an encoder that projects the nodes V of G into a latent space, and DEC a decoder that reconstructs a pairwise proximity between the nodes from their learned low-dimensional embeddings. If the decoder is successful at reconstructing the pairwise proximity between the nodes, then in principle, the embeddings do encode structural information about the position of the node in the graph and contain the information necessary for downstream machine learning tasks. The goal for the embedding model is then to optimize the encoder and decoder by minimizing the proximity reconstruction error, such as:

$$DEC(ENC(v_i), ENC(v_j)) = DEC(z_i, z_j) \approx \mathcal{S}(v_i, v_j) \tag{2.1}$$

Here $\mathcal{S}$ is a pairwise proximity function that measures how closely v_i and v_j are connected in the graph G. The reconstructed pairwise proximity could be of the first-order, for example, whether two nodes are connected with an edge, or the weight on the edge that connects them, such as: $\mathcal{S}(v_i, v_j) = A_{i,j}$. The reconstructed proximity could also be of higher order, for example, whether two nodes share a large portion of their neighborhood, or the probability that a node appears during a random walk in the neighborhood of another node.

The objective of the model then becomes to minimize the pairwise loss l between the proximity reconstructed from the representation space and the actual prox-

imity of the nodes. The general loss function $\mathcal{L}$ of the model is defined as:

$$\mathcal{L} = \sum_{v_i, v_j \in V} l(DEC(z_i, z_j), \mathcal{S}(v_i, v_j)) \tag{2.2}$$

This conceptual framework allows us to view the various node embedding architectures, with all their diversity and complexity, through a simple unifying lens that emphasizes the importance of the reconstruction and preservation of a proximity measure in the embeddings. However, as we have previously stated, in this work, we will go beyond the standard proximity measures that are explicitly preserved by the model and look into the indirect preservation of the topological and centrality features in the embeddings. Now that we have introduced important background information related to the graphs and defined the problem of node embedding, we start exploring three mainstream categories of graph representation models.

2.3 Matrix Factorization

Some of the earliest methods for generating vector representations on graphs relied on Matrix Factorization. Laplacian Eigenmaps (LE) (Belkin & Niyogi, 2001) is one of the first algorithms to apply this technique. LE aims to project nodes having a high first-order proximity between them into similar vectors in the embedding space. This method takes advantage of the relationship between the eigenvectors and eigenvalues of the graph Laplacian and the connectivity of the graph components. The spectral decomposition of the Laplacian of a graph G is given as:

$$Ly = \lambda y$$
$$\lambda = y^T L y$$

Where L is the Laplacian of G, y is an eigenvector and λ is its associated eigenvalue. Since L is symmetrical and semi-definite, then all its eigenvalues are positive and real. Subsequently, the problem of embedding G in d dimensions rests on finding the d smallest eigenvectors of L (excluding the first eigenvector).

We summarize the algorithm of Laplacian Eigenmaps in the following four steps. For a set of n points $x_1, x_2, ..., x_n \in \mathbb{R}^k$, that we want to project in $\mathbb{R}^d$ with $d \ll k$:

- Step 1 - Build a weighted graph (represented by an adjacency matrix) with n nodes, one for each point. We connect the nodes in the graph using one of the following methods:
  1. Add an edge between 2 points, if the euclidean distance between them is small($||x_i - x_j||^2 < \epsilon$), with ϵ a small integer defined by the user.
  2. Use the algorithm K-Nearest Neighbor (KNN). Link every point to its K-nearest neighbors.
- Step 2 - Add weights on the edges of the graph using one of the following methods:
  1. Simply add 1 to the connected edges.
  2. Calculate the heat-kernel such as: $w_{ij} = e^{\frac{||x_i - x_j||^2}{t}}$
- Step 3 - Calculate the Laplacian of the graph and derive its eigenvectors and eigenvalues.
- Step 4 - Take the first d eigenvectors $Y = y_1, y_2, ..., y_d$ corresponding to the smallest d eigenvalues. For a point x_i, the embedding $z_i = Y_i^T$(the i^{th} line of Y).

Evidently, if we want to embed the nodes of an existing graph, we skip the first

two steps of the algorithm. In what follows we detail the theoretical analysis for the success of the LE method:

For a graph $G = (V, E)$ with n nodes, where A is the adjacency matrix of G. We want to project the nodes into an embedding space, in a way that two nodes that have a high first-order proximity (that are connected and/or have a high weight on their edge) to have similar vector representations in the embedding space. In other words, for two nodes v_i and v_j with a high weight on their edge (A_{ij}) the distance between their embedding ($distance(z_i, z_j)$) should be small. Therefore, for an embedding of size ($d = 1$), we want to minimize the distance:

$$\sum_{i,j}(z_i - z_j)^2 A_{ij} \tag{2.3}$$

Under the constraint that z is a unit vector (i.e., $\sum_i z_i^2 = 1$), the sum in (2.3) can be written as:

$$\frac{1}{2}\sum_{i,j}(z_i - z_j)^2 A_{ij} = z^T L z \tag{2.4}$$

Where L is the graph Laplacian defined as $L = D - A$, with D the diagonal degree matrix, defined as $D_{ii} = \sum_j A_{ij}$. Therefore, the sum in (2.3) can be written as a spectral decomposition of the Laplacian of G, where the embedding[2] z is the eigenvector of L.

By supposing the graph is undirected and A is symmetrical, the equality in (2.4) can be verified as follows:

---

[2]We have denoted the eigenvector as 'z' instead of the standard 'y' notation, in order to facilitate the association with the embedding notation which is usually 'z'.

$$
\begin{aligned}
\sum_{i,j}(z_i - z_j)^2 A_{ij} &= \sum_{i,j}(z_i^2 + z_j^2 - 2z_i z_j)A_{ij} \\
&= \sum_{i,j} z_i^2 A_{ij} + \sum_{i,j} z_j^2 A_{ij} - 2\sum_{i,j} z_i z_j A_{ij} \\
&= \sum_{i} z_i^2 D_{ii} + \sum_{j} z_j^2 D_{jj} - 2\sum_{i,j} z_i z_j A_{ij} \\
&= 1.\sum_{i} D_{ii} + 1.\sum_{j} D_{jj} - 2\sum_{i,j} z_i z_j A_{ij} \\
&= 2\sum_{i,j} D_{ij} - 2\sum_{i,j} z_i z_j A_{ij} \\
&= 2\sum_{i,j} z_i z_j (D_{ij} - A_{ij}) \\
&= 2\sum_{i,j} z_i z_j L_{ij} = 2z^T Lz
\end{aligned}
\tag{2.5}
$$

Therefore the problem of minimizing (2.3) can be reduced to minimizing $(z^T Lz)$. However, the solution for this problem is already known. It is the second smallest eigenvalue λ_1 with $z_{optimal} = z_1$ the eigenvector that belongs to λ_1, denoted by the Fiedler Vector. Note that $\lambda_0 = 0$ and $z_0 = \mathbf{1}$ can also be considered a solution, however it is trivial, since it is independent of L, and only depends on whether G is a connected graph. The verification we have provided is for an embedding space of size 1 ($d = 1$), however for an embedding of size d, it suffice to take the first d eigenvectors $Z = z_1, z_2, ..., z_d$ corresponding to the smallest d eigenvalues. Thus, for a node v_i we have the embedding $z_i = Z_i^T$(the i^{th} line of Z).

If we model this algorithm according to the Encoder/Decoder architecture established in section 2.2 of this chapter, we find that the decoder is directly reconstructing the similarity of the first-order proximity, such as:

$$\begin{aligned}\mathcal{L} &= min \sum_{v_i,v_j \in V} DEC(z_i, z_j).\mathcal{S}(v_i, v_j) \\ &= min \sum_{v_i,v_j \in V} (z_i - z_j)^2.\mathcal{S}(v_i, v_j) \quad \text{with } \mathcal{S}(v_i, v_j) = A_{ij}\end{aligned} \tag{2.6}$$

The Laplacian Eigenmaps method is simple and direct and widely used, however it suffers from several limitations:

- It only preserves the first-order proximity. However, there exists some extensions that attempt to preserve the second-order of proximity (Ou et al., 2016).
- It only captures the structure of the graph, since it relies on the adjacency matrix. However, other characteristics of the graph, such as the attributes, can also be important for the embedding.
- It suffers from the problem of scaling since calculating the eigenvectors and eigenvalues of the Laplacian is computationally expensive for large graphs.
- It only operates on undirected graphs since the Laplacian ignores the directions of the edges.

2.4 Random Walk Techniques

The distribution of nodes appearing in random walks follows a power-law, much like the distribution of words in natural language (Perozzi et al., 2014). This observation allowed the authors of DeepWalk (Perozzi et al., 2014) to adapt the same architecture of Word2Vec (Mikolov et al., 2013), that is used for word embedding, and repurpose it for node representation learning.

Similar to Word2Vec, the DeepWalk model is divided into two main steps: 1) Data preparation and 2) Model Training. In the first step, the model prepares the data by performing random walks on the graph in order to generate sequences of nodes. These node sequences are equivalent to the sentences in Word2Vec. Subsequently, a sliding window is glided over these sequences to generate segments of consecutive nodes with a constant size. The number and size of the walks, as well as the size of the sliding window are hyperparameters that could be tuned by the user. As for the input nodes, they are encoded with one-hot-encoding in order to be processed by the learning model.

In the second step, the model is trained on the prepared sequences. Each node in the center of the window is passed through a Skip-Gram network which is optimized to predict the neighbors of the input node. The number of forward passes of each node is equal to the number of neighbors in the sliding window. The Skip-Gram network contains a single hidden layer, without an activation function, which serves as a lookup table for the input node. A Softmax is applied on the output layer, which generates a probability for each neighbor. Backward propagation is used to adjust the weights of the layers based on the loss between the predicted neighbors and the groundtruth neighbors of the input node. At the end of the training, the output layer is discarded and the weights of the hidden layer are extracted as the embeddings of nodes. Figure 2.4 details the steps of a standard DeepWalk model.

While DeepWalk uniformly samples the nodes during the random walk, Node2Vec (Grover & Leskovec, 2016) is another variation of the same architecture that introduces two parameters p and q that control the sampling during the random walk. The parameter p controls the probability of going back to a node v after visiting another node t. The parameter q controls the probability of moving away from node v after visiting it. Node2Vec improves on DeepWalk by allowing a more

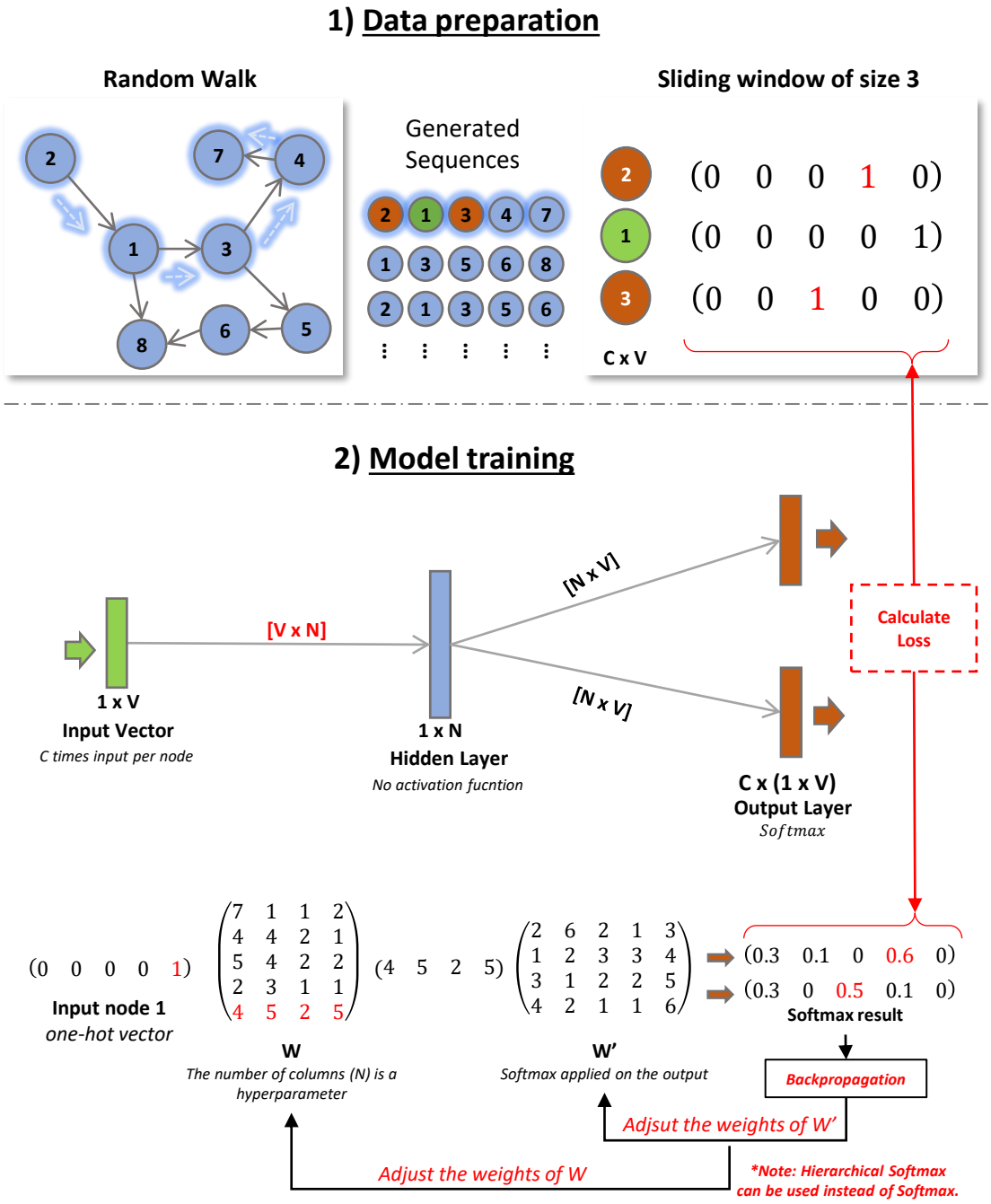

Figure 2.4: A chart showing both steps in a DeepWalk model: 1) Data preparation and 2) Model Training. For the model training step, we show both the general architecture of the Skip-Gram network and a mock-up example in a matrix format.

flexible and controllable exploration of the graph neighborhoods thus enhancing the quality of the sampled sequences in the data preparation step. For these reasons, in our experimentation, we have opted to use Node2Vec instead of the standard DeepWalk model.

It is important to note that in the DeepWalk and Node2Vec implementations, the authors used *Hierarchical Softmax* instead of the standard *Softmax*, which is also inspired by Word2Vec. The reason is that the standard Softmax is not optimal on large-scale data. As a solution for the scaling problem, a tree is built out of binary classifiers, with the nodes of the graph as its leaves. The problem is thus reduced to maximizing the path from the root to a node on the binary tree (i.e., the product of the probabilities given by the classifiers on the tree). Huffman coding is used to quickly access frequent items in the tree. This reduces the model complexity from $O(V)$ to $O(log(V))$.

The Random Walk based techniques have demonstrated adequate results in a wide range of studies. Their architecture is simple and does not require the tuning of many hyperparameters. However, they do present some limitations:

- They only work on weighted and unattributed graphs and do not consider important characteristics of the graph such as the nodes attributes.
- They cannot generalize and lack the weight-sharing capacity during training. This entails that the size of their weights increases linearly with the size of the graph, rendering these models hard to scale on large graphs (Hamilton et al., 2017a).
- They are very slow in comparison with the other models. The data preparation phase which generates the sequences adds a significant overhead to the processing time.

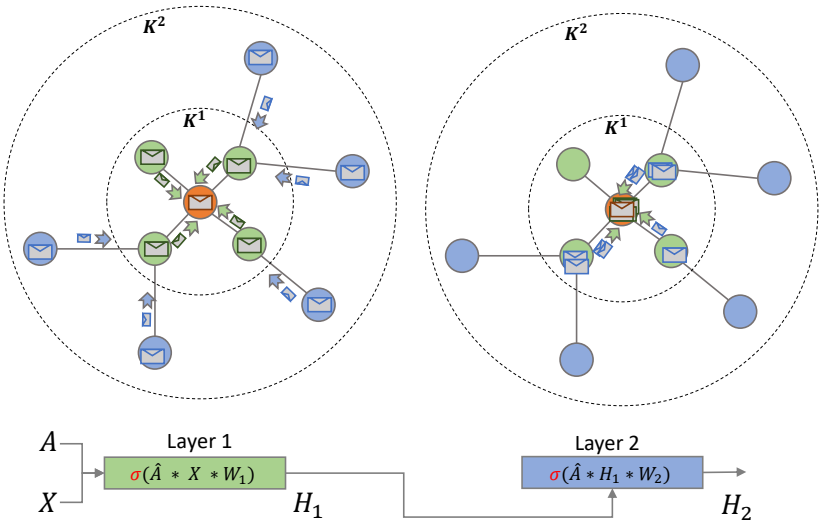

Figure 2.5: An example of message (attributes) passing from layer to layer in the GNN from the perspective of one node (central node: orange). We can observe that the Layer 1 allows the passing of the first-level neighborhood messages (K^1: green) to the central node. While Layer 2 allows the passing of the second-level neighborhood messages (K^2: blue) to the central node.

2.5 Graph Autoencoders

The multi-layer neural networks in the domain of deep learning on graphs are generally referred to in literature as GNN (Graph Neural Networks). The GNN adopts the concept of message passing over the graph. In this technique, each node collects the messages, that is, the attributes of its direct neighborhood, which are then aggregated to form the new message of the node. Figure 2.5 shows an example of message passing from the perspective of one node [3].

Given a graph $G = (V, E, X)$, where V is the set of nodes, E the set of edges and X the set of node attributes. The GNN takes two matrices as entry, the adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$, and the attributes matrix $X \in \mathbb{R}^{|V|\times k}$. In the case of an non-

[3]This operation is shown from the perspective of one node, however in practice, it happens in parallel on all nodes of the graph.

attributed graph, one-hot-encoding is used for messages instead of X. The main component of these networks is the forward propagation rule denoted by FWD. This function dictates the way the messages are being passed and aggregated in the network. FWD takes two arguments as input, the adjancency matrix A and the previous layer's hidden state H_{k-1}:

$$H_k = FWD(H_{k-1}, A) \tag{2.7}$$

FWD is usually a non-linear function such as $ReLU$, $Sigmoid$ or $tanh$. For the first layer we use the attributes of the nodes as input, such as $H_0 = X$. For the subsequent layers, the output of each layer serves as input for the next layer. The messages on the nodes are accumulated from layer to layer, therefore, every layer enables the exploration of a new level of neighborhood. A weight matrix W is added to each layer to be learned and standard backpropagation is used for updating the weights.

The message passing and aggregation operation can be accomplished by multiplying the adjacency matrix with the node attributes matrix: $A \times X$. In order to accumulate the messages from layer to layer, self-loops are added to each node $\hat{A} = A + I$, where I is the identity matrix. Throughout the years, several forward propagation rules were proposed with different aggregation methods. In what follows, we explore three of the main aggregation functions used in the literature.

The following forward propagation rule aggregates the messages via SUM, where σ is the activation function:

$$H_k = \sigma(\hat{A}H_{k-1}W_k) \tag{2.8}$$

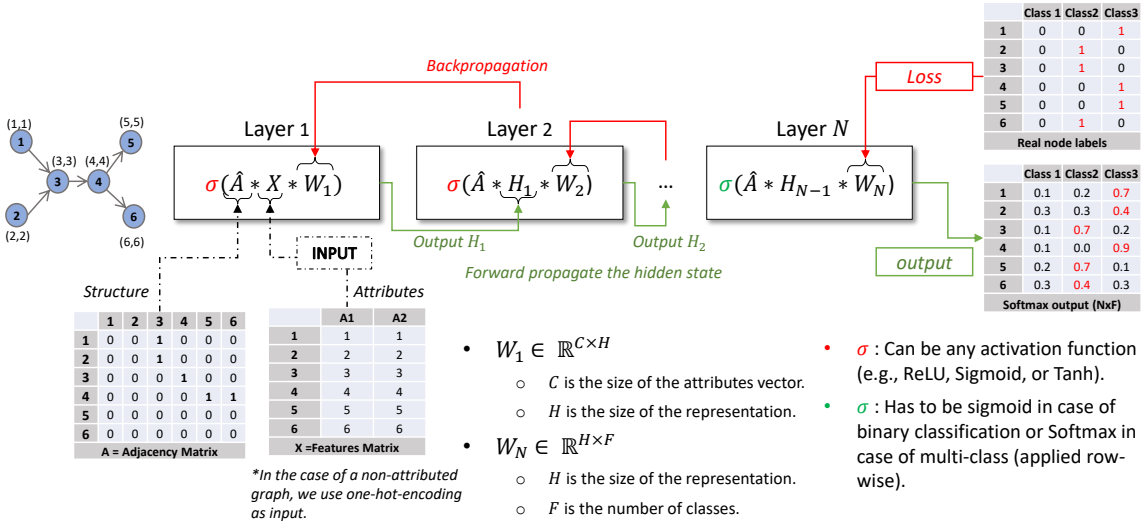

Figure 2.6: A layout of a GNN architecture with N layers using the SUM aggregation rule, used in a classification task. The output of each layer (green arrows) is used as input for the next layer. The input of the first layer is X (the attributes matrix). The loss is calculated between the output of the last layer (n) and the groundtruth (real node labels). Backpropagation (red arrows) is used to update the weights in all layers.

In order to aggregate the MEAN of the messages, a normalized adjacency matrix can be used ($\hat{A}$ multiplied with the inverse degree matrix $\hat{D}^{-1/2}$):

$$H_k = \sigma(\hat{D}^{-1}\hat{A}H_{k-1}W_k) \tag{2.9}$$

Another rule that was developed in (Kipf & Welling, 2017) for the Graph Convolutional Network (GCN), is the SPECTRAL rule that utilizes a renormalized symmetric adjacency matrix:

$$H_k = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H_{k-1}W_k) \tag{2.10}$$

Figure 2.6 displays a complete GNN architecture. We can observe from the figure that the adjacency matrix A is the only component that differentiates a Graph Neural Network architecture from a standard Neural Network.

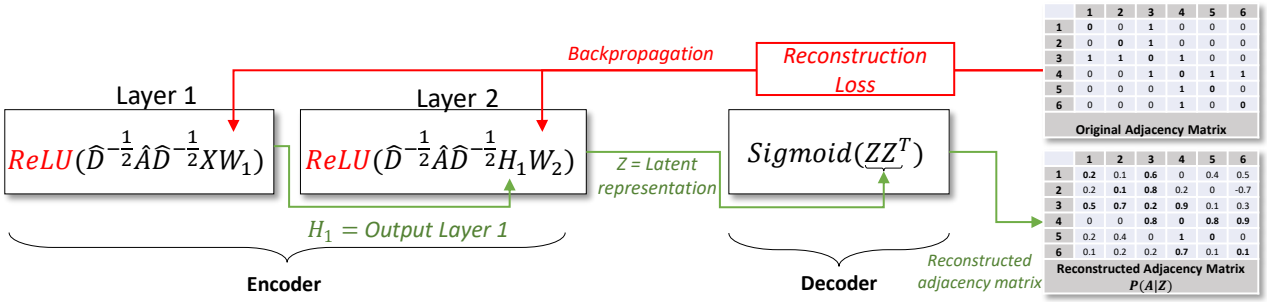

Figure 2.7: A layout of a GAE architecture using the spectral rule. The encoder part is made of a sequence of GCN layers. While the decoder is made of a *Sigmoid* applied on the latent space (Z) multiplied by its transpose (Z^T).

After introducing the GCN model, the authors in (Kipf & Welling, 2016) extended the architecture into a Graph Autoencoder (GAE).

The encoder section of the GAE is built from GNN layers. Here is an example of a two-layer GAE using the SPECTRAL rule:

$$\begin{aligned} H_1 &= ReLU(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}XW_1) \\ H_2 &= ReLU(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H_1W_2) \end{aligned} \tag{2.11}$$

The decoder simply reconstructs the adjacency matrix ($\bar{A}$). In order to reconstruct the matrix, the output of the encoder (latent space $Z = H_2$) is multiplied with its transpose and then a *Sigmoid* is applied element-wise on the product:

$$\bar{A} = Sigmoid(ZZ^T) \tag{2.12}$$

Figure 2.7 is a pictorial representation of a Graph Autoencoder using the SPECTRAL rule for the encoder part.

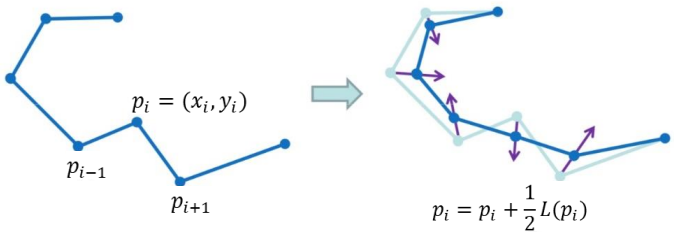

Figure 2.8: An example of curve smoohting, where each point on the line is adjusted according to the position of its neighbors.

After the success of the GCN model, several works have attempted to formalize the model in order to explain its success. The authors in (Li et al., 2018) described the GCN SPECTRAL rule as a form of *Laplacian Smoothing*. In order to smooth a curve, as shown in Figure 2.8, each point on the line is iteratively adjusted according to the positions of its neighbors. Below is the formula for smoothing a curve:

$$\begin{aligned} p_i &= p_i + \frac{1}{2}L(p_i) \\ with \quad L(p_i) &= \frac{p_{i-1} + p_{i+1}}{2} - p_i \end{aligned} \tag{2.13}$$

Where p_i is the position of the point, p_{i-1} and p_{i+1} the position of its two adjacent points. On the other hand, if we want to smooth the points in a graph, we can use the following formula:

$$P_{l+1} = (I - \gamma L_{norm})P_l \tag{2.14}$$

Where L_{norm} is the Normalized Laplacian, which has two forms:

- Random Walk: $L_{rw} = D^{-1}L$
- Symmetrical: $L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$.

In (2.14) γ is a factor that controls the force of the smoothing ($0 < \gamma \leq 1$). If we take $\gamma = 1$ with $P_{l+1} = H$ (the first hidden layer output) and $P_l = X$ (the input attributes), and use L_{rw} for the Normalized Laplacian, we can write (2.14) as:

$$\begin{aligned} H &= (I - \gamma L_{norm})X \\ &= (I - L_{rw})X \\ &= (I - (D^{-1}L)X \\ &= (I - (D^{-1}(D - A))X \\ &= (I - (\frac{D}{D} - D^{-1}A))X \\ &= (I - I + D^{-1}A)X \\ &= D^{-1}AX \end{aligned} \tag{2.15}$$

The result of (2.15) is exactly the *MEAN* rule that we have introduced in the previous section. On the other hand, if we use the Symmetrical Normalized Laplacian, we arrive to the *SPECTRAL* rule ($D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X$). We can conclude that the GNN MEAN and SPECTRAL rules are equivalent to the graph smoothing formulas. Therefore, these rules are *smoothing* the input attributes of the graph so that the strongly connected nodes would be closer to each other, which would help the downstream tasks like clustering and classification.

However, one problem that arises from the smoothing effect is the risk of over-smoothing. In fact, the more layers we add to the GNN (i.e., the deeper the GNN), the more the range of aggregated neighborhoods would increase. A wide range of aggregation would lead to a high overlap in the aggregated attributes thus resulting in similar vector representations for distinct nodes. This effect would eventually reduce the discriminative power of the embeddings. Figure 2.9 displays the node embeddings of the Zachary karate club graph projected into 2D

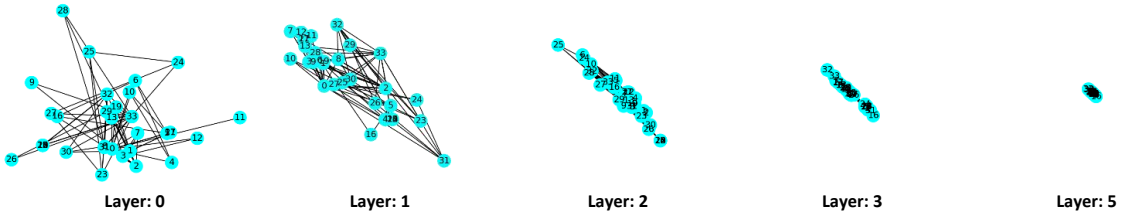

Figure 2.9: A 2D-tSNE projection of embeddings generated by a GCN of five layers on the Zachary karate club dataset.

space. We can observe the effect of oversmoothing with a GCN of five layers. For a small dataset such as Zachary karate club of only 34 nodes, a five layer GCN would be considered too deep. Figure 2.9 clearly illustrates that the embeddings of the nodes have converged into a single point by layer 5. Such convergence of the representations would likely cause a deterioration in the performance of the embeddings on downstream tasks.

The Graph Neural Network models hold state of the art results on many graph learning tasks, specifically classification and clustering, but they also suffer from some problems:

- The risk of oversmoothing with the increase in the number of layers. This fact prevents the GNN from capitalizing on the power of deep networks in deep learning.
- The difficulty of scaling the GNN to large graphs. Some variations of the GNN have attempted to tackle this problem by using sub-sampling and training on mini-batches of nodes (Hamilton et al., 2017b).
- The SPECTRAL rule used in the GCN only works on undirected graphs as it is based on the symmetrical normalized Laplacian. However, this rule can be substituted with other rules such as the MEAN or SUM. The effect of the different aggregation rules on the learning and the structures that they

encode is one of the main focal points of our study.

- The model assigns the same importance to the synthetic self-loops added for message aggregation as to the real neighboring edges. This might introduce some bias to the learning. One possible solution would be to add a weighting factor to the self-loops: $\hat{A} = A + \lambda I$ (Kipf & Welling, 2017).

In the following chapter, we thoroughly explore the representational power of the node embedding models in an extensive set of experiments that cover topological feature preservation and downstream learning tasks. We further highlight our findings with principled analysis and rich visualizations that corroborate our claims. Finally, we conclude the chapter with our recommendations and propose a strategy for node embedding model design.

CHAPTER III

EXPLORING THE POWER OF GRAPH AUTOENCODERS

This chapter aims to deeply investigate the representational power of Graph Autoencoders. To this end, an extensive experimental protocol is conducted to study the characteristics being captured by the embeddings and their effect on downstream tasks such as node clustering and classification. In order to perform this, we need to specify the scope of our experiments and the required experimental configurations.

3.1 Scope of the Experiments

In order to achieve the goals put forward in this thesis, we have laid out the four experiments of this study over three main steps. In the first step, we build multiple variations for each class of embedding model. This is so, in case one of the models is successful, we could pinpoint which elements contributed to its success against the other variations. Next, we perform the first experiment for the purpose of identifying the models that are successful in preserving the topological features, and highlight the reasons for their success. This experiment is further supplemented by a visualization section that serves as additional evidence for the success of some models in preserving the topological features over others. In the final step, we perform three experiments that evaluate the embeddings over three

Table 3.1 Graph Datasets.

Dataset	Nodes	Edges	Classes	Type
Cora	2,708	5,429	7	Citation
Citeseer	3,327	4,732	6	Citation
email-Eu-core	1,005	25,571	42	Email
USA Air-Traffic	1,190	13,599	4	Flight
Europe Air-Traffic	399	5,995	4	Flight
Brazil Air-Traffic	131	1,074	4	Flight
fly-drosophila-medulla-1	1,781	9,016	NA	Biological
ego-Facebook	4,039	88,234	NA	Social
soc-sign-bitcoin-alpha	3,783	24,186	NA	Blockchain
soc-sign-bitcoin-otc	5,881	35,592	NA	Blockchain
ca-GrQc	5,242	14,496	NA	Collaboration

separate tasks (cluster homogeneity, node clustering, and node classification). For every experiment, we compare the performance of the models that preserved the topological features versus the models that did not. This comparison helps in highlighting the importance of the presence of the topological features in the embeddings. We conclude the experiments by an overall analysis of the results and recommendations for embedding model choices.

3.2 Datasets and Compared Baselines

We evaluate the embeddings generated by three types of architecture: (1) Matrix Factorization, (2) Random Walk and (3) Seven variations of Graph Autoencoders. For this purpose, we use eleven real datasets retrieved from the Stanford Large Network Dataset Collection (Leskovec & Krevl, 2014), Deep Graph Library (Wang et al., 2019b) and (Wu et al., 2019). Table 3.1 summarizes the main characteristics of the collected datasets. Note that, Cora and Citeseer are attributed graph datasets. The dimensionality of Cora's attributes is 1,433 while the dimensionality of Citeseer's attributes is 3,703. The rest of the datasets are plain graphs where one-hot-encoding is used for attributes.

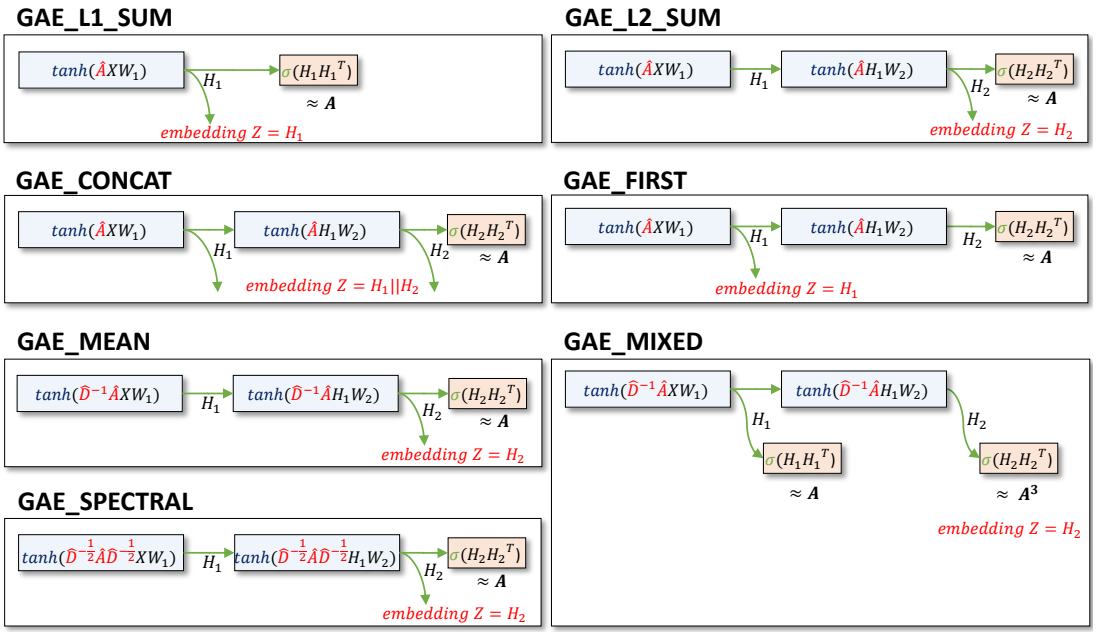

Figure 3.1: Seven variations of Graph Autoencoders.

In what follows, we briefly describe the compared baselines:

Matrix Factorization: We use the implementation for Laplacian Eigenmaps (Belkin & Niyogi, 2001) included in Scikit-learn with the default parameters offered by the library.

Random Walk: We use two different variations of Node2Vec (Grover & Leskovec, 2016) with the same setup presented in (Bonner et al., 2019). Node2Vec-Structural ($p = 0.5, q = 2.0$) which explores global structures in the graph during the random walk and Node2Vec-Homophily ($p = 1.0, q = 0.5$) which explores local structures around the starting node. For both variations we use the parameters of the original paper (walk-length = 80 and number-walks = 10).

Graph Autoencoders: We study seven different variations on the original architecture (see Figure 3.1). We explore multiple aggregation rules: SUM (Xu et al., 2019), MEAN (Hamilton et al., 2017b), SPECTRAL (Kipf & Welling, 2017)).

We also look into different ways for selecting the embedding: Last layer output (Kipf & Welling, 2016), First layer output, or Concatenation of all layers outputs (Xu et al., 2019), as well as, the effect of reconstructing multiple orders of proximity. We adopt a two-layer architecture for all variations except when we mention otherwise.

GAE_L1_SUM is a Graph Autoencoder with a one-layer encoder using the SUM rule, with Z as the embedding:

$$Z = \tanh(\hat{A}XW_1) \tag{3.1}$$

GAE_L2_SUM is a Graph Autoencoder with a two-layer encoder using the SUM rule, with the output of the last layer as embedding:

$$\begin{aligned} H_1 &= \tanh(\hat{A}XW_1) \\ H_2 &= \tanh(\hat{A}H_1W_2) \\ Z &= H_2 \end{aligned} \tag{3.2}$$

GAE_CONCAT is a Graph Autoencoder using the SUM rule. However, the embedding is formed by concatenating the output from all the layers of the encoder, with H_1 and H_2 the output of the first and second layer respectively:

$$\begin{aligned} H_1 &= \tanh(\hat{A}XW_1) \\ H_2 &= \tanh(\hat{A}H_1W_2) \\ Z &= H_1 \parallel H_2 \end{aligned} \tag{3.3}$$

GAE_FIRST is a Graph Autoencoder using the SUM rule. However, the embedding is the output of the first layer of the encoder:

$$
\begin{aligned}
H_1 &= \tanh(\hat{A}XW_1) \\
H_2 &= \tanh(\hat{A}H_1W_2) \\
Z &= H_1
\end{aligned}
\tag{3.4}
$$

GAE_MEAN is a Graph Autoencoder using the MEAN rule:

$$
\begin{aligned}
H_1 &= \tanh(\hat{D}^{-1}\hat{A}XW_1) \\
H_2 &= \tanh(\hat{D}^{-1}\hat{A}H_1W_2) \\
Z &= H_2
\end{aligned}
\tag{3.5}
$$

GAE_SPECTRAL is a Graph Autoencoder that has a GCN as encoder and aggregates the messages using the SPECTRAL rule:

$$
\begin{aligned}
H_1 &= \tanh(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}XW_1) \\
H_2 &= \tanh(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H_1W_2) \\
Z &= H_2
\end{aligned}
\tag{3.6}
$$

GAE_MIXED is a Graph Autoencoder using the MEAN rule that reconstructs two orders of proximity. While the standard GAE reconstructs A. GAE_MIXED reconstructs A and A^3. We use the same encoder as GAE_MEAN. However, the output of the first layer is used to reconstruct A, while the output of the second layer is used to reconstruct A^3. We have tested the reconstruction of multiple orders of proximity and selected A and A^3 as they gave the best results.

For the following models: GAE_L1_SUM, GAE_L2_SUM, GAE_CONCAT, GAE_FIRST, GAE_MEAN and GAE_SPECTRAL the loss is the reconstruction error of the adjacency matrix A calculated from the output of the last layer H_l:

$$
\mathcal{L} = \frac{1}{|V|^2}\sum_{i=1}^{|V|}\sum_{j=1}^{|V|}(\sigma(H_l.H_l^T) - A)_{ij}^2 \tag{3.7}
$$

For GAE_MIXED, the loss is the sum of both reconstruction errors of A and A^3. The output of the first layer H_1 is used to reconstruct A and the output of second layer H_2 is used to reconstruct A^3 with an attenuation factor alpha = 0.5:

$$\mathcal{L} = \frac{1}{|V|^2}\sum_{i=1}^{|V|}\sum_{j=1}^{|V|}(\sigma(H_1.H_1^T) - A)_{ij}^2 + \alpha.\frac{1}{|V|^2}\sum_{i=1}^{|V|}\sum_{j=1}^{|V|}(\sigma(H_2.H_2^T) - A^3)_{ij}^2 \quad (3.8)$$

For the compared models, we use an embedding of size 64. For all Graph Autoencoders, we use a hidden layer of size 64 and an output layer of size 64 with hyperbolic tangent activation function and batch normalization on all layers without any dropout. We use an optimizer Adam with a learning rate of 0.01. We train for 250 epochs with a patience of 10. For the model GAE_CONCAT we use layers of size 32 so the total would be 64. All experiments are run 10 times and we report the mean of the results. We regenerate the embeddings for each run.

3.3 Topological Features Prediction

We start our experiments by evaluating if the embeddings are capturing any of the graph topological features. We focus our study on the following five distinct features:

- **Degree** $DG(v) = K_v = deg(v)$ is the number of edges connected to a node v, counted twice for every self-loop.
- **Triangle Count** $TC(v) = \Delta_v$. The number of triangles of a node v. A tirangle or triplet Δ_v is a pair of neighbours of v that are also connected among each other.
- **Local Clustering Score** $LC(v) = \frac{2\Delta_v}{K_v \times (K_v - 1)}$. It is the number of triangles

or triplets Δ_v over the number of possible edges of v. The Local Clustering Score measures the connectivity of the neighborhood of a node v. For a directed graph (such as email-Eu-core), the number of possible edges is $K_v \times (K_v - 1)$ and $LC(v) = \frac{\Delta_v}{K_v \times (K_v - 1)}$.

- **Eigenvector Centrality** $EC(v) = \frac{1}{\lambda} \sum_{n \in \mathcal{N}(v)} EC(n)$. Where λ is a constant that is the largest eigenvalue. It measures the influence of a node calculated in reference to the influence of its neighborhood. It indicates that if the measure is high for v then it is high for its neighbours.

- **Betweenness Centrality** $BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$. Where σ_{st} is the number of shortest routes between two nodes s and t, and $\sigma_{st}(v)$ is the number of shortest routes between s and t that pass through v. It measures the impact of node v on the data transfer in the graph.

It is worth noting that, in this work, we do not aim at defining new features to reflect the topological structure of a graph. This would be far beyond the scope of our study. Many topological features have already been proposed in the current literature. In our experiments, we utilize existing, well-known features that may characterize the topological structure of a graph, and focus on evaluating if the embeddings are capturing any of the graph topological features. It is also important to note that we have used distinct features that investigate the topological structure of a graph from the perspective of the node, its neighbourhood, its role and importance in the graph.

To address the problem of the preservation of topological features, two approaches are possible: (Rizi & Granitzer, 2017) and (Bonner et al., 2019). In the first approach (Rizi & Granitzer, 2017), the topological features are directly predicted with Linear Regression using the embeddings as attributes. A low prediction error indicates that the predicted feature is indeed encoded in the embeddings. In the second approach (Bonner et al., 2019), the problem is addressed as a classification problem. To this end, the range of values for each topological feature is divided, using histogram binning, into a set of intervals, where each interval of values corresponds to a topological class. This means that, in addition to its ground-truth class, each node will also have a Degree class, a Local Clustering Score class, an Eigenvector Centrality class, a Betweenness Centrality class, and a Triangle Count class. The topological class of the node indicates the interval into which its topological feature value falls. While dividing the topological features into bins, we have tested over multiple number of bins and selected the ones that generated balanced classes. In our study, in order to mitigate the limitations that any of the aforementioned approaches might have, and to further support our claims, we have adopted both approaches, that is, direct prediction and classification.

In order to reconstruct the topological features, we use Linear Regression (LN-R) evaluated with the Mean Square Error (MSE). A low value of MSE suggests a good result. For the classification of the nodes into their topological feature classes, we apply four models: Logistic Regression (LG-R), linear SVM (SVM-L), SVM with RBF Kernel (SVM-RBF) and a MultiLayer Perceptron (MLP). Note that, for the MLP, we use two hidden layers with a ReLU activation function, an Adam optimizer and a 0.001 learning rate. We train for 200 epochs without early stopping. We used the implementation in Scikit-learn for all algorithms with a 5-fold cross validation. To evaluate LG-R, SVM-L, SVM-RBF and MLP, we report two evaluation metrics Macro-F1 and Micro-F1. The higher the values

of these metrics the better the results. Given the large number of experiments for the task of topological features classification (5 topological features times 11 datasets) and in order to avoid encumbering the thesis, we present a representative and diversified subset of the results. Recall that a complete list of all the results can be found at: `https://github.com/MH-0/RPGAE`

Tables 3.2, 3.3, 3.4, 3.5 and 3.6 list the results for a combination of topological features and datasets. The first column of each table holds the results of the linear regression while the rest of the columns display the results of the classification. We found that GAE_FIRST and GAE_CONCAT outperform the other models by a large margin for both approaches (i.e., regression and classification), notably, over 60% for the prediction of the node Degree on the Cora dataset (Table 3.2). Based on the results of our extensive empirical study, we have observed that the success of these two models in preserving the features, especially the Degree of the nodes, can be attributed to their use of the SUM aggregation rule and the inclusion of the first layer output in the embedding.

In order to sustain the aforementioned observations, we provide in the following a principled analysis regarding the preservation of the Degree in the first layer of the GNN when using the SUM rule. In our analysis, we consider two cases: (1) Non-attributed graphs, and (2) Attributed graphs.

Let G be a graph with a set of nodes V. GAE is a Graph Autoencoder applied to G, with a GNN encoder that uses the SUM forward propagation rule (FWD_{SUM}), such as:

$$H_k = FWD_{SUM}(H_{k-1}, \hat{A}) = \sigma(\hat{A}H_{k-1}W_k) \tag{3.9}$$

Table 3.2 Cora - Degree.

	LN-R	LG-R		SVM-L		SVM-RBF		MLP	
Models	MSE	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	**0.107 ±0.023**	**0.865 ±0.026**	**0.899 ±0.020**	**0.789 ±0.046**	**0.837 ±0.037**	**0.869 ±0.031**	**0.904 ±0.025**	**0.768 ±0.03**	**0.850 ±0.018**
GAE_CONCAT	0.152 ±0.035	0.818 ±0.035	0.860 ±0.028	0.682 ±0.043	0.737 ±0.041	0.765 ±0.038	0.796 ±0.039	0.692 ±0.042	0.791 ±0.019
GAE_L1_SUM	4.086 ±0.097	0.126 ±0.012	0.400 ±0.005	0.062 ±0.006	0.151 ±0.003	0.415 ±0.031	0.446 ±0.033	0.155 ±0.015	0.415 ±0.008
GAE_L2_SUM	3.478 ±0.134	0.232 ±0.013	0.401 ±0.010	0.199 ±0.011	0.306 ±0.018	0.238 ±0.011	0.317 ±0.015	0.186 ±0.021	0.408 ±0.011
GAE_MEAN	4.160 ±0.055	0.155 ±0.008	0.375 ±0.006	0.188 ±0.009	0.262 ±0.014	0.156 ±0.008	0.345 ±0.010	0.120 ±0.003	0.391 ±0.002
GAE_MIXED	4.137 ±0.053	0.151 ±0.009	0.374 ±0.004	0.187 ±0.008	0.253 ±0.019	0.158 ±0.008	0.346 ±0.012	0.116 ±0.003	0.392 ±0.001
GAE_SPECTRAL	4.177 ±0.000	0.113 ±0.000	0.394 ±0.000	0.138 ±0.014	0.219 ±0.015	0.340 ±0.010	0.415 ±0.013	0.115 ±0.004	0.394 ±0.002
Matrix Factorization	4.177 ±0.000	0.113 ±0.000	0.394 ±0.000	0.104 ±0.006	0.170 ±0.003	0.134 ±0.006	0.234 ±0.005	0.113 ±0.000	0.392 ±0.001
Node2Vec-S	4.251 ±0.072	0.148 ±0.009	0.358 ±0.005	0.175 ±0.006	0.190 ±0.007	0.173 ±0.008	0.287 ±0.013	0.130 ±0.007	0.381 ±0.006
Node2Vec-H	4.165 ±0.084	0.154 ±0.006	0.364 ±0.005	0.169 ±0.012	0.181 ±0.013	0.173 ±0.009	0.298 ±0.015	0.131 ±0.005	0.381 ±0.008

Table 3.3 fly-drosophila-medulla-1 - Local Clustering Score.

	LN-R	LG-R		SVM-L		SVM-RBF		MLP	
Models	MSE	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	1.574 ±0.130	0.425 ±0.028	0.551 ±0.022	0.292 ±0.043	0.342 ±0.068	**0.52 ±0.014**	**0.642 ±0.01**	0.386 ±0.019	0.557 ±0.017
GAE_CONCAT	**1.416 ±0.112**	**0.435 ±0.027**	**0.562 ±0.018**	**0.367 ±0.018**	**0.465 ±0.031**	0.502 ±0.010	0.634 ±0.008	**0.406 ±0.036**	**0.583 ±0.029**
GAE_L1_SUM	1.496 ±0.074	0.405 ±0.017	0.551 ±0.014	0.292 ±0.028	0.321 ±0.040	0.428 ±0.012	0.606 ±0.006	0.331 ±0.030	0.538 ±0.018
GAE_L2_SUM	1.694 ±0.109	0.346 ±0.020	0.508 ±0.015	0.319 ±0.033	0.453 ±0.044	0.322 ±0.022	0.481 ±0.039	0.307 ±0.022	0.520 ±0.014
GAE_MEAN	1.816 ±0.090	0.317 ±0.017	0.484 ±0.014	0.272 ±0.013	0.392 ±0.026	0.266 ±0.021	0.438 ±0.026	0.273 ±0.021	0.489 ±0.010
GAE_MIXED	1.651 ±0.074	0.343 ±0.014	0.498 ±0.010	0.282 ±0.026	0.395 ±0.027	0.325 ±0.022	0.488 ±0.020	0.283 ±0.021	0.490 ±0.016
GAE_SPECTRAL	2.173 ±0.000	0.167 ±0.000	0.500 ±0.000	0.187 ±0.010	0.348 ±0.010	0.358 ±0.010	0.508 ±0.008	0.167 ±0.000	0.500 ±0.000
Matrix Factorization	2.173 ±0.000	0.167 ±0.000	0.500 ±0.000	0.169 ±0.000	0.294 ±0.000	0.182 ±0.002	0.291 ±0.001	0.167 ±0.000	0.500 ±0.000
Node2Vec-S	2.239 ±0.038	0.208 ±0.005	0.459 ±0.006	0.236 ±0.012	0.279 ±0.017	0.210 ±0.012	0.449 ±0.020	0.187 ±0.009	0.470 ±0.010
Node2Vec-H	2.235 ±0.040	0.206 ±0.008	0.459 ±0.006	0.232 ±0.010	0.280 ±0.008	0.210 ±0.008	0.444 ±0.014	0.187 ±0.006	0.474 ±0.011

Table 3.4 email-Eu-core - Eigenvector Centrality.

	LN-R	LG-R		SVM-L		SVM-RBF		MLP	
Models	MSE	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	0.340 ±0.020	0.680 ±0.016	0.683 ±0.016	0.644 ±0.020	0.646 ±0.020	0.666 ±0.013	0.668 ±0.013	0.354 ±0.021	0.439 ±0.021
GAE_CONCAT	**0.325 ±0.033**	**0.703 ±0.022**	**0.705 ±0.021**	**0.694 ±0.026**	**0.696 ±0.026**	**0.692 ±0.013**	**0.692 ±0.013**	**0.393 ±0.050**	**0.468 ±0.039**
GAE_L1_SUM	0.784 ±0.069	0.522 ±0.010	0.528 ±0.010	0.468 ±0.017	0.472 ±0.017	0.526 ±0.012	0.540 ±0.011	0.307 ±0.024	0.392 ±0.023
GAE_L2_SUM	0.784 ±0.056	0.540 ±0.015	0.542 ±0.015	0.517 ±0.019	0.520 ±0.018	0.498 ±0.017	0.493 ±0.017	0.327 ±0.026	0.401 ±0.023
GAE_MEAN	0.601 ±0.043	0.573 ±0.010	0.574 ±0.009	0.534 ±0.019	0.536 ±0.019	0.507 ±0.017	0.503 ±0.018	0.303 ±0.022	0.373 ±0.020
GAE_MIXED	0.735 ±0.046	0.521 ±0.013	0.526 ±0.013	0.480 ±0.017	0.485 ±0.017	0.491 ±0.013	0.489 ±0.013	0.288 ±0.030	0.361 ±0.031
GAE_SPECTRAL	1.426 ±0.193	0.333 ±0.033	0.415 ±0.019	0.123 ±0.032	0.219 ±0.021	0.491 ±0.051	0.509 ±0.040	0.113 ±0.040	0.226 ±0.042
Matrix Factorization	4.372 ±0.000	0.191 ±0.000	0.236 ±0.000	0.062 ±0.000	0.174 ±0.000	0.380 ±0.000	0.403 ±0.000	0.070 ±0.012	0.173 ±0.010
Node2Vec-S	4.019 ±0.174	0.214 ±0.010	0.237 ±0.010	0.145 ±0.008	0.155 ±0.010	0.212 ±0.010	0.266 ±0.009	0.130 ±0.019	0.200 ±0.014
Node2Vec-H	4.268 ±0.212	0.199 ±0.013	0.226 ±0.012	0.147 ±0.016	0.156 ±0.018	0.213 ±0.014	0.261 ±0.012	0.129 ±0.020	0.189 ±0.019

Table 3.5 soc-sign-bitcoin-otc - Betweeness Centrality.

	LN-R	LG-R		SVM-L		SVM-RBF		MLP	
Models	MSE	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	0.595 ±0.042	0.572 ±0.02	0.689 ±0.011	0.315 ±0.031	0.382 ±0.036	0.524 ±0.016	0.585 ±0.029	**0.576 ±0.014**	0.690 ±0.009
GAE_CONCAT	**0.529 ±0.037**	**0.596 ±0.016**	**0.702 ±0.008**	**0.360 ±0.053**	**0.454 ±0.070**	**0.526 ±0.021**	**0.592 ±0.030**	**0.576 ±0.025**	**0.696 ±0.012**
GAE_L1_SUM	1.975 ±0.086	0.264 ±0.018	0.539 ±0.009	0.067 ±0.003	0.148 ±0.006	0.130 ±0.024	0.137 ±0.015	0.299 ±0.043	0.565 ±0.021
GAE_L2_SUM	1.320 ±0.130	0.407 ±0.020	0.576 ±0.014	0.317 ±0.022	0.423 ±0.025	0.354 ±0.034	0.429 ±0.041	0.397 ±0.022	0.583 ±0.012
GAE_MEAN	1.659 ±0.060	0.370 ±0.010	0.536 ±0.007	0.278 ±0.023	0.363 ±0.028	0.339 ±0.026	0.501 ±0.018	0.333 ±0.016	0.544 ±0.017
GAE_MIXED	1.555 ±0.041	0.381 ±0.011	0.545 ±0.007	0.293 ±0.024	0.359 ±0.041	0.365 ±0.022	0.507 ±0.023	0.353 ±0.013	0.555 ±0.009
GAE_SPECTRAL	2.333 ±0.000	0.167 ±0.000	0.500 ±0.000	0.189 ±0.007	0.315 ±0.017	0.355 ±0.006	0.490 ±0.005	0.167 ±0.000	0.500 ±0.000
Matrix Factorization	2.333 ±0.000	0.167 ±0.000	0.500 ±0.000	0.074 ±0.000	0.158 ±0.000	0.109 ±0.001	0.192 ±0.001	0.166 ±0.001	0.492 ±0.008
Node2Vec-S	2.341 ±0.023	0.188 ±0.004	0.488 ±0.002	0.163 ±0.011	0.192 ±0.010	0.136 ±0.006	0.250 ±0.006	0.185 ±0.004	0.476 ±0.005
Node2Vec-H	2.371 ±0.023	0.186 ±0.005	0.485 ±0.003	0.181 ±0.010	0.205 ±0.009	0.137 ±0.008	0.259 ±0.009	0.186 ±0.007	0.476 ±0.006

Table 3.6 Brazil Air-Traffic - Triangle Count.

	LN-R	LG-R		SVM-L		SVM-RBF		MLP	
Models	MSE	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	**0.082 ±0.014**	**0.917 ±0.015**	**0.918 ±0.014**	**0.910 ±0.017**	**0.911 ±0.017**	**0.892 ±0.012**	**0.893 ±0.011**	0.534 ±0.113	0.619 ±0.097
GAE_CONCAT	0.111 ±0.015	0.888 ±0.015	0.889 ±0.015	0.891 ±0.026	0.892 ±0.025	0.832 ±0.028	0.833 ±0.027	**0.577 ±0.082**	**0.651 ±0.065**
GAE_L1_SUM	0.191 ±0.029	0.809 ±0.022	0.814 ±0.022	0.804 ±0.026	0.808 ±0.025	0.798 ±0.026	0.800 ±0.025	0.508 ±0.085	0.586 ±0.063
GAE_L2_SUM	0.237 ±0.045	0.768 ±0.037	0.773 ±0.037	0.751 ±0.033	0.758 ±0.032	0.693 ±0.032	0.707 ±0.029	0.493 ±0.064	0.563 ±0.060
GAE_MEAN	0.258 ±0.031	0.752 ±0.024	0.763 ±0.024	0.749 ±0.029	0.756 ±0.029	0.739 ±0.023	0.746 ±0.021	0.489 ±0.075	0.545 ±0.069
GAE_MIXED	0.323 ±0.025	0.712 ±0.028	0.723 ±0.024	0.702 ±0.021	0.712 ±0.019	0.770 ±0.023	0.778 ±0.021	0.492 ±0.078	0.559 ±0.065
GAE_SPECTRAL	0.401 ±0.019	0.534 ±0.004	0.655 ±0.004	0.240 ±0.006	0.400 ±0.007	0.780 ±0.021	0.790 ±0.019	0.207 ±0.047	0.365 ±0.036
Matrix Factorization	1.491 ±0.071	0.303 ±0.024	0.393 ±0.018	0.168 ±0.000	0.336 ±0.000	0.818 ±0.000	0.825 ±0.000	0.207 ±0.029	0.346 ±0.023
Node2Vec-S	1.390 ±0.104	0.306 ±0.055	0.332 ±0.052	0.283 ±0.047	0.317 ±0.050	0.308 ±0.035	0.345 ±0.031	0.252 ±0.045	0.336 ±0.048
Node2Vec-H	1.391 ±0.158	0.306 ±0.050	0.326 ±0.050	0.278 ±0.047	0.317 ±0.047	0.255 ±0.043	0.290 ±0.044	0.237 ±0.035	0.330 ±0.030

(1) Non-attributed graphs:

For non-attributed graphs, we use one-hot-encoding as input for the first layer, such as $H_0 = I$, where I is an identity matrix. Therefore, $H_1 = \sigma(\hat{A}H_0W_1) = \sigma(\hat{A}IW_1)$. Since $\hat{A}I = \hat{A}$, we have $H_1 = \sigma(\hat{A}W_1)$. This implies that the hidden state of a node v : $h_{1(v)}$ in layer 1 is formed from an aggregation over a set of weights from that layer. Furthermore, there is a one-to-one correspondence between the indices of the nodes in A and the indices of the weight vectors in W_1. Therefore, ignoring σ, we have:

$$h_{1(v)} = \sum_{n \in \mathcal{N}_i(v)} w_n = \sum_{\bar{w} \in \bar{W}_{\mathcal{N}_i(v)}} \bar{w} \tag{3.10}$$

Where $\mathcal{N}_i(v)$ is the set of indices of the neighbors of v. w_n is a weight vector in W_1 that corresponds to the n^{th} neighbor of v. We further define $\bar{W}_{\mathcal{N}_i(v)}$ as the set of weights in W_1 whose indices correspond to the indices in $\mathcal{N}_i(v)$.

Let $\bar{V} \subseteq V$ be a subset of nodes of G, where the nodes of $\bar{V}$ have high second-order proximity among each other. For example, all the nodes in $\bar{V}$ have the same label. We define $\mathcal{N}_i(\bar{V})$ as the set of all neighbor indices for the nodes in $\bar{V}$. $\bar{W}_{\mathcal{N}_i(\bar{V})}$ is the set of weights whose indices correspond to $\mathcal{N}_i(\bar{V})$. If GAE preserves the second-order proximity between the nodes of $\bar{V}$ in the embeddings, then the distance between the embedding of any pair of nodes in $\bar{V}$ is small, such as: $\forall(v_i, v_j) \in \bar{V}$ we have $dist(h_{1(v_i)}, h_{1(v_j)}) < \epsilon$, where ϵ is a small positive value close to zero. Following Equation (3.10), we have:

$$dist\Big(\sum \bar{W}_{\mathcal{N}_i(v_i)}, \sum \bar{W}_{\mathcal{N}_i(v_j)}\Big) < \epsilon \tag{3.11}$$

Since the nodes in $\bar{V}$ have a high second-order proximity among each other, then $\forall(v_i, v_j) \in \bar{V}$ we have $\mathcal{N}_i(v_i) \simeq \mathcal{N}_i(v_j) \Rightarrow \bar{W}_{\mathcal{N}_i(v_i)} \simeq \bar{W}_{\mathcal{N}_i(v_j)}$. Since this is true

$\forall(v_i, v_j) \in \bar{V}$, then as $\epsilon \to 0$ we observe that the weights in $\bar{W}_{\mathcal{N}_i(\bar{V})}$ converge closer to each other, such as: $\exists \bar{w_c}$, $\forall \bar{w} \in \bar{W}_{\mathcal{N}_i(\bar{V})}$ we have $dist(\bar{w}, \bar{w_c}) < \delta$. For $\delta > 0$ and small. Then, $\forall v \in \bar{V}$ we have:

$$h_{1(v)} = \sum_{n \in \mathcal{N}_i(v)} w_n \approx |\mathcal{N}_i(v)|.\bar{w_c} \tag{3.12}$$

This effect is however lost in the subsequent layers of the encoder, since $\hat{A}H_k \neq \hat{A}$ for $k > 0$. That is why the vanilla GAE_L2_SUM that uses the output of the last layer in the encoder as embedding does not preserve the topological features. On the other hand, when we use the output of the first layer as the embedding or when we concatenate it with the output of the other layers, we preserve the topological features.

(2) Attributed graphs:

For attributed graphs, we consider sparse Bag-of-words attributes, such as the ones used in our experiments with Cora and Citeseer datasets. In this case, we have a binary sparse attribute matrix $X \in \{0, 1\}$. We observe that $\hat{A}X \approx \hat{A}$. Therefore, in case of a GAE with a SUM aggregation rule, we have $H_1 = \sigma(\hat{A}H_0W_1) = \sigma(\hat{A}XW_1) \approx \sigma(\hat{A}W_1)$. This leads to the same effect of aggregating over a converging set of weights, as described in the case of non-attributed graphs.

In both cases, of attributed and non-Attributed graphs, the condition of the preservation of the second-order proximity is crucial for the preservation of the topological features in the embeddings. We observe this fact in our experiments with the under-performance of GAE_L1_SUM (Autoencoder with one layer). Since it uses the SUM rule and the output of the first layer as embedding, it should have preserved the topological features. However, one layer is not sufficient for the

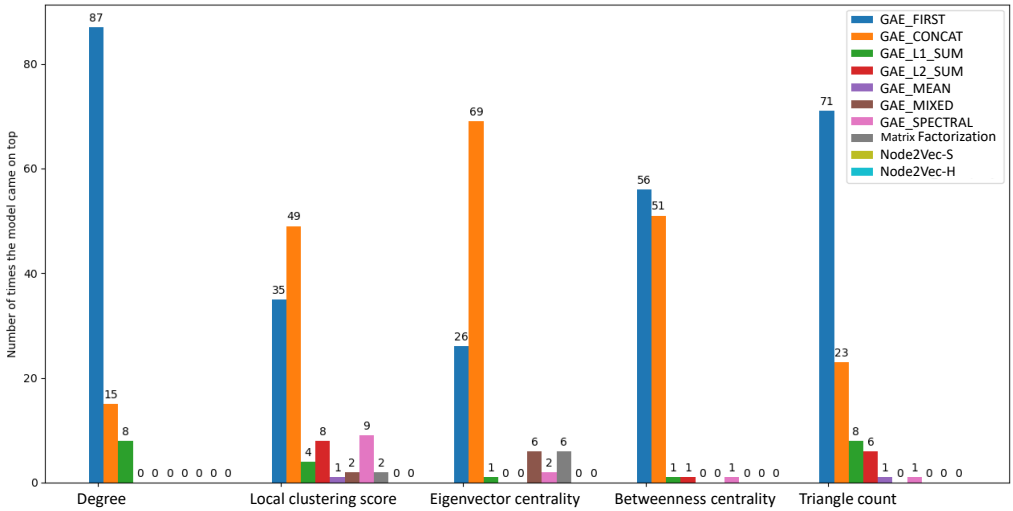

Figure 3.2: The number of times each model outperformed the others on the ten evaluation metrics over the eleven datasets per feature.

GAE to converge on a dataset of the size of either Cora or Citeseer. This means that the weights associated to the neighbors of two nodes that have the same label, might not be similar. Aggregating these weights will not lead to similar embeddings for the two nodes even if they have the same number of neighbors. On the other hand, for relatively small networks such as USA Air-Traffic, or the ego-Facebook, one layer networks are sufficient for convergence and the topological features are preserved in GAE_L1_SUM. This is why the condition for the model convergence (i.e., the preservation of the second-order proximity) is necessary for the preservation of the topological features in the first layer.

Furthermore, in order to have a better understanding of the performance of the models across the large number of experiments conducted in this work, we compile the results in a figure that illustrates the overall trends in the scores for the task of preservation of the topological features. Figure 3.2 shows the number of times each model outperformed the others for the ten evaluation metrics over the eleven datasets (that is, 11 x 10 = 110 evaluation scores per feature). The evaluation metrics are (MSE and MAE) for: Linear Regression. (F1-Macro and F1-Micro)

for each of the four models: Logistic Regression (LG-R), Linear SVM (SVM-L), SVM with RBF Kernel (SVM-RBF), and Multilayer Perceptron (MLP).

Figure 3.2 clearly shows that GAE_FIRST (dark blue) and GAE_CONCAT (orange) outperform all other models by a large margin. In particular, it shows that GAE_FIRST is favorable for capturing the Degree and the Triangle Count, while GAE_CONCAT better captures the Local Clustering Score and the Eigenvector Centrality. As for the Betweeness Centrality, the results are shared between both GAE_FIRST and GAE_CONCAT. We believe that this is due to the nature of each feature. The Degree and Trianle Count of the node are related to its one-hop neighborhood. For this reason, the Degree and Triangle Count are best preserved in the first hidden layer of the encoder, since this layer aggregates the one-hop neighborhood features of the node. However, the Eigenvector Centrality and the Local Clustering Score are two measures that depend on the two-hop neighborhood, that is, the importance and structure of the neighborhood of the node. The Eigenvector Centrality for a certain node is high, if it is also high for its neighbors. As for the Local Clustering Score, it measures the connectivity of the neighborhood. This is why these features are best preserved when we also include the output of the second hidden layer in the embedding, since this layer captures information about the two-hop neighborhood of the node.

3.4 Visualisation

In this section, we aim to visualize the embeddings for the purpose of better understanding the success of GAE_FIRST and GAE_CONCAT in preserving the topological features. To this end, we project the embeddings to two dimensions using t-SNE (t-distributed Stochastic Neighbor Embedding)(Maaten & Hinton, 2008).

Figure 3.3: 2D t-SNE projection of the embeddings of Cora. The embeddings are colored according to the seven different ground-truth labels of the dataset.

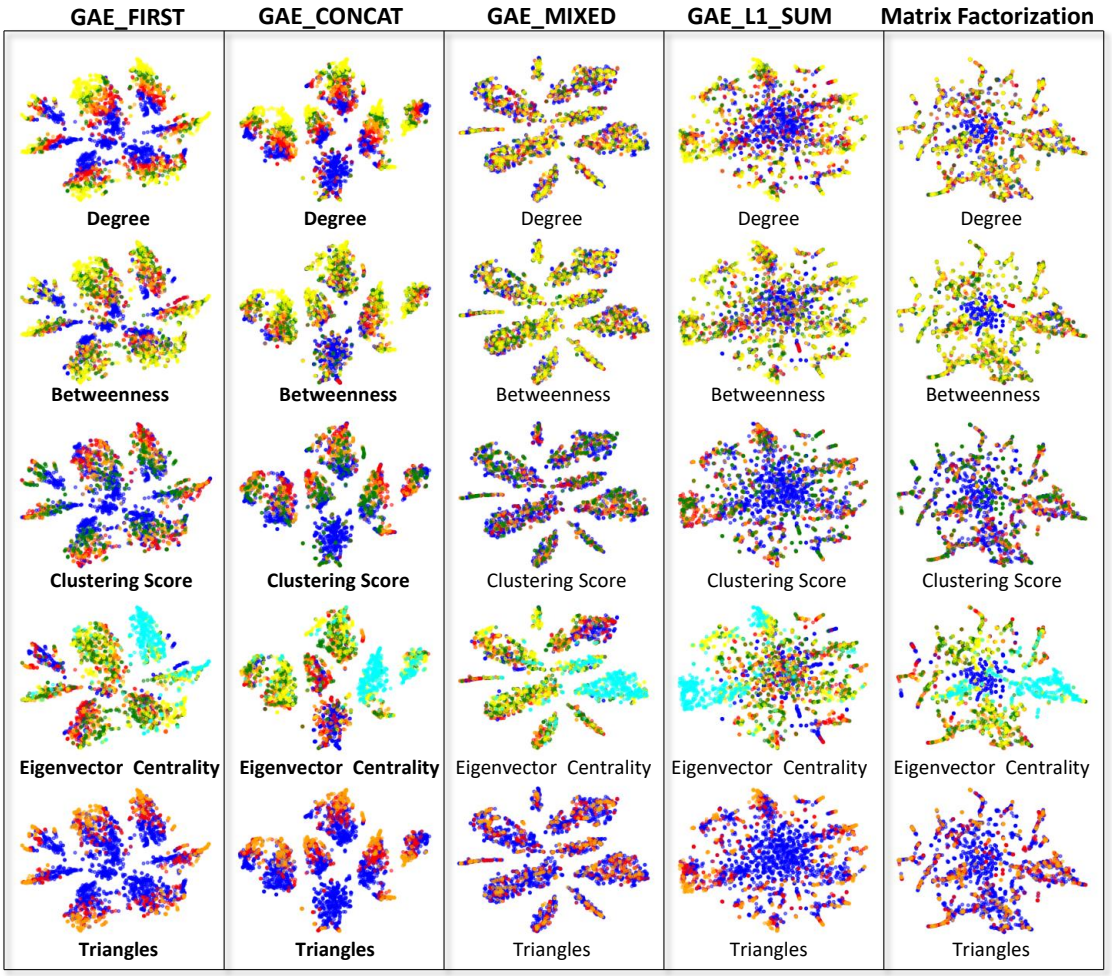

Figure 3.4: The embeddings of 5 models projected in 2D for the Cora dataset and colored according to the the topological classes of the vertices (Degree Class, Betweenness Class, Local Clustering Score Class, Eigenvector Centrality , Triangle Count). The colors listed in ascending order are: Blue, Red, Orange, Green, Yellow and Cyan. The figure clearly shows that the embeddings of GAE_FIRST and GAE_CONCAT (highlighted in bold) are organized according to the topological classes of the vertices.

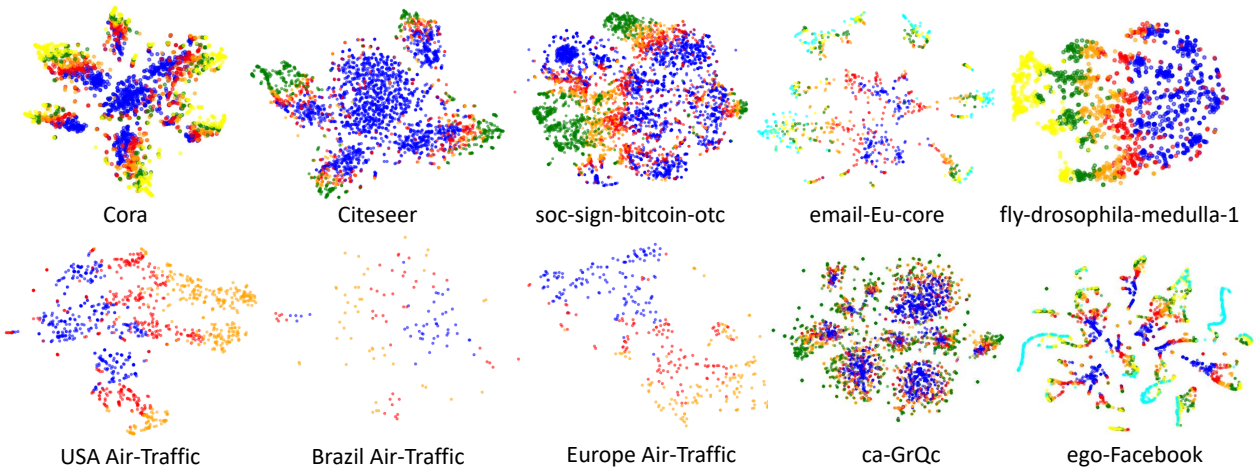

Figure 3.5: An illustration of the hierarchical arrangement of the node Degree classes for the embeddings generated by GAE_FIRST. Every color represents a range of Degrees as defined in the binning phase of the experiments. The plots show that the degrees in the embedding are arranged in a hierarchy from the smallest (Blue) to the largest (Cyan).

Figures 3.3 and 3.4 [1] display the 2D projections of the embeddings generated on Cora dataset. As we can see from Figure 3.3, when we color the nodes according to the ground-truth, we notice that, for GAE_FIRST and GAE_CONCAT, each cluster of points in the plot is dominated by a class. This is due to the fact that the embedding models are mostly optimized to preserve the first-order and second-order proximity which are generally correlated with the ground-truth labels of the nodes. For example, in the case of the Cora dataset, the nodes are articles, the edges are citations, and the labels are the field of publication per article. Two articles (nodes) that have a high first-order and second-order proximity tend to belong to the same field of publication (i.e., they have the same label). A model that well preserves the orders of proximity, will also be preserving the labels of the graph.

However, in Figure 3.4, when we color the nodes according to their topological fea-

[1] All figures in this thesis are best viewed in color/screen.

ture classes (Degree class, Local Clustering class, Betweenness class, Eigenvector Centrality class, and Triangle Count class), we notice that, for GAE_FIRST and GAE_CONCAT, the colors are clearly arranged in an organized manner, in contrast to the other models. In fact, the colors for the Degree in particular, exhibit a hierarchy in their arrangement, where, the nodes that have the lowest Degree are projected close together (colored blue), followed by the nodes with higher Degree (colored red) in an ascending order, all the way to the nodes belonging to the highest Degree range (colored yellow). For GAE_FIRST in particular, this hierarchy in the arrangement of the Degrees is consistent with the majority of the tested datasets as illustrated by Figure 3.5.

This neat arrangement in the topological features is, however, absent from the embeddings of the other models. Consider, for example, the models that use the MEAN aggregation rule such as GAE_MIXED. As we can see from Figure 3.4, the topological features are arbitrarily distributed without any structure in the embeddings. The pictorial illustration of the organization of the topological classes in the embeddings of GAE_FIRST and GAE_CONCAT corroborate our claim on the fact that these two models preserve the topological features in their embeddings. Furthermore, we can confirm the failure of GAE_L1_SUM in capturing the topological features on Cora, as it failed to preserve the second-order proximity (GAE_L1_SUM in Figure 3.3 shows no clear separation of ground-truth labels) and therefore did not preserve the topological features (GAE_L1_SUM in Figure 3.4)

3.5 **Task 1**: Embedding Clusters Homogeneity

Next, we study the homogeneity of the embeddings when clustered according to the ground truth labels of each dataset. We look into three well-known evaluation metrics.

Table 3.7 Embedding clusters homogeneity.

	Cora			**email-Eu-core**		
Models	DB	CH	SC	DB	CH	SC
GAE_FIRST	3.095 ±0.288	144.535 ±15.695	0.066 ±0.015	3.965 ±0.214	15.863 ±1.547	0.026 ±0.006
GAE_CONCAT	3.248 ±0.523	213.227 ±26.267	0.100 ±0.033	4.204 ±0.268	11.018 ±1.077	0.043 ±0.005
GAE_L1_SUM	3.899 ±0.509	92.613 ±33.691	-0.065 ±0.028	4.764 ±0.35	6.472 ±0.774	-0.008 ±0.008
GAE_L2_SUM	2.938 ±0.371	297.779 ±31.627	0.176 ±0.029	4.687 ±0.386	5.095 ±0.446	-0.047 ±0.01
GAE_MEAN	2.158 ±0.518	**392.946 ±59.420**	0.186 ±0.024	6.205 ±0.202	**3.545 ±0.244**	-0.017 ±0.004
GAE_MIXED	**2.051 ±0.401**	390.704 ±48.115	**0.211 ±0.027**	6.043 ±0.142	3.188 ±0.107	**-0.038 ±0.004**
GAE_SPECTRAL	5.000 ±0.429	40.336 ±5.400	-0.044 ±0.010	6.165 ±0.131	3.728 ±0.3	0.01 ±0.005
Matrix Factorization	5.174 ±0.085	10.706 ±0.998	-0.196 ±0.035	**7.353 ±0.13**	0.871 ±0.03	-0.135 ±0.001
Node2Vec-S	29.233 ±1.473	1.026 ±0.127	-0.034 ±0.003	11.506 ±3.325	1.021 ±0.345	-0.059 ±0.01
Node2Vec-H	28.532 ±1.293	1.074 ±0.092	-0.042 ±0.003	11.635 ±2.015	0.764 ±0.155	-0.07 ±0.012

	Citeseer			**USA Air-Traffic**		
Models	DB	CH	SC	DB	CH	SC
GAE_FIRST	4.294 ±0.594	131.461 ±14.800	-0.061 ±0.016	**4.647 ±0.310**	**139.871 ±9.416**	0.019 ±0.007
GAE_CONCAT	4.721 ±0.606	155.863 ±20.255	-0.026 ±0.019	5.906 ±0.419	93.810 ±8.284	**0.044 ±0.009**
GAE_L1_SUM	6.366 ±1.432	34.739 ±4.089	-0.215 ±0.022	4.946 ±0.402	57.761 ±5.151	-0.050 ±0.010
GAE_L2_SUM	5.612 ±1.048	209.234 ±22.836	0.014 ±0.032	8.219 ±1.201	45.644 ±6.358	0.016 ±0.008
GAE_MEAN	4.088 ±0.608	306.785 ±71.098	0.094 ±0.041	7.624 ±0.686	31.904 ±2.373	0.016 ±0.002
GAE_MIXED	**3.151 ±0.378**	**371.070 ±55.394**	**0.172 ±0.029**	8.843 ±0.733	42.489 ±3.717	0.025 ±0.003
GAE_SPECTRAL	8.882 ±0.662	26.218 ±2.863	-0.032 ±0.005	12.555 ±1.235	6.217 ±0.995	-0.088 ±0.003
Matrix Factorization	9.615 ±0.265	10.407 ±0.171	-0.07 ±0.005	9.781 ±0.029	2.705 ±0.017	-0.183 ±0.001
Node2Vec-S	33.581 ±1.514	1.087 ±0.115	-0.01 ±0.002	26.090 ±1.715	0.952 ±0.165	-0.033 ±0.003
Node2Vec-H	34.023 ±1.832	1.042 ±0.119	-0.012 ±0.002	25.775 ±2.798	1.097 ±0.293	-0.028 ±0.003

	Europe Air-Traffic			**Brazil Air-Traffic**		
Models	DB	CH	SC	DB	CH	SC
GAE_FIRST	**5.347 ±0.351**	**34.376 ±3.991**	-0.015 ±0.003	**3.965 ±0.214**	**15.863 ±1.547**	0.026 ±0.006
GAE_CONCAT	6.126 ±0.394	25.105 ±2.491	**-0.010 ±0.004**	4.204 ±0.268	11.018 ±1.077	**0.043 ±0.005**
GAE_L1_SUM	6.938 ±0.208	11.394 ±0.402	-0.061 ±0.003	4.764 ±0.350	6.472 ±0.774	-0.008 ±0.008
GAE_L2_SUM	7.782 ±0.657	9.631 ±1.413	-0.061 ±0.006	4.687 ±0.386	5.095 ±0.446	-0.047 ±0.010
GAE_MEAN	10.252 ±0.563	5.414 ±0.455	-0.056 ±0.004	6.205 ±0.202	3.545 ±0.244	-0.017 ±0.004
GAE_MIXED	10.174 ±0.577	5.699 ±0.365	-0.070 ±0.002	6.043 ±0.142	3.188 ±0.107	-0.038 ±0.004
GAE_SPECTRAL	8.158 ±0.240	12.623 ±1.197	-0.010 ±0.001	6.165 ±0.131	3.728 ±0.300	0.010 ±0.005
Matrix Factorization	11.537 ±0.000	1.140 ±0.000	-0.166 ±0.000	7.353 ±0.13	0.871 ±0.030	-0.135 ±0.001
Node2Vec-S	11.287 ±1.993	4.281 ±0.728	-0.033 ±0.004	11.506 ±3.325	1.021 ±0.345	-0.059 ±0.010
Node2Vec-H	11.871 ±1.319	3.590 ±0.396	-0.020 ±0.003	11.635 ±2.015	0.764 ±0.155	-0.070 ±0.012

Davies-Bouldin Index (DB): The lower the value, the better the results (Davies & Bouldin, 1979).

Silhouette Score (SC): Score between -1 and 1 with 1 being the best score (Rousseeuw, 1987).

Calinski-Harabasz (CH): The higher the value, the better the consistency of the clusters (Caliński & Harabasz, 1974).

Table 3.7 reports the results on the datasets with ground truth labels. We find two trends in the results. On one hand, the models that use the MEAN and SPECTRAL aggregation rules (GAE_MIXED, GAE_MEAN and GAE_SPECTRAL)

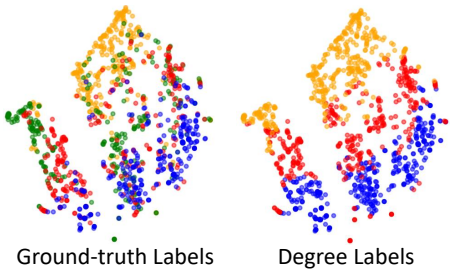

Figure 3.6: 2D t-SNE projection of the GAE_FIRST embeddings for USA Air-Traffic dataset - The colors show that, to a large extent, the vertices that belong to the same ground-truth class also belong to the same Degree class.

perform best on the Cora, Citeseer and email-Eu-core datasets. This is most probably due to the smoothing effect that these two rules have on the attributes (Li et al., 2018), leading to more homogeneous embeddings. We also notice that GAE_MIXED, the model that reconstructs two orders of proximity, gave the best results on two out of the three datasets. On the other hand, the models that preserved the topological features (GAE_FIRST and GAE_CONCAT) dominated the three flight datasets (USA, Europe, and Brazil). The nodes in these datasets are airports, the edges are flights connecting the airports and the ground-truth labels are set according to the amount of activity that each airport receives. Therefore, there is a direct relation between the Degree of each node (the number of flights that connect to it) and its label (the amount of activity the airport receives). The knowledgeable reader can observe this fact from Figure 3.6, where we notice that, to a large extent, the nodes that have the same ground-truth label also belong to the same Degree class. That is why the models that capture the Degree in the embedding best perform on these datasets.

3.6 **Task 2**: Node Clustering

Let us now evaluate the suitability of the embeddings on the task of node clustering. To this end, we apply K-means and a recently proposed algorithm named

Table 3.8 Clustering node embeddings - using K-means.

	Cora			**email-Eu-core**		
Models	ACC	NMI	ARI	ACC	NMI	ARI
GAE_FIRST	0.438 ±0.039	0.324 ±0.030	0.133 ±0.027	0.363 ±0.015	0.564 ±0.01	0.228 ±0.016
GAE_CONCAT	0.586 ±0.059	0.440 ±0.031	0.367 ±0.059	0.344 ±0.018	0.558 ±0.012	0.247 ±0.025
GAE_L1_SUM	0.372 ±0.018	0.194 ±0.029	0.044 ±0.016	0.364 ±0.008	0.531 ±0.008	0.163 ±0.013
GAE_L2_SUM	0.629 ±0.036	0.469 ±0.022	0.403 ±0.028	0.384 ±0.017	0.567 ±0.011	0.327 ±0.028
GAE_MEAN	**0.668 ±0.021**	0.505 ±0.021	**0.430 ±0.031**	0.432 ±0.011	**0.605 ±0.008**	**0.356 ±0.011**
GAE_MIXED	0.659 ±0.019	**0.507 ±0.017**	0.417 ±0.028	**0.438 ±0.035**	0.603 ±0.013	0.350 ±0.038
GAE_SPECTRAL	0.419 ±0.050	0.231 ±0.031	0.158 ±0.038	0.395 ±0.040	0.547 ±0.057	0.172 ±0.013
Matrix Factorization	0.304 ±0.002	0.010 ±0.002	0.000 ±0.001	0.170 ±0.020	0.218 ±0.032	0.006 ±0.007
Node2Vec-S	0.229 ±0.006	0.005 ±0.001	0.002 ±0.002	0.109 ±0.003	0.195 ±0.004	0.000 ±0.001
Node2Vec-H	0.217 ±0.005	0.004 ±0.001	0.001 ±0.003	0.109 ±0.004	0.183 ±0.005	0.000 ±0.001

	Citeseer			**USA Air-Traffic**		
Models	ACC	NMI	ARI	ACC	NMI	ARI
GAE_FIRST	0.370 ±0.027	0.181 ±0.027	0.048 ±0.011	0.468 ±0.009	**0.273 ±0.005**	**0.196 ±0.004**
GAE_CONCAT	0.450 ±0.035	0.217 ±0.024	0.156 ±0.029	**0.484 ±0.01**	0.200 ±0.012	0.184 ±0.016
GAE_L1_SUM	0.231 ±0.007	0.056 ±0.009	-0.001 ±0.001	0.468 ±0.016	0.224 ±0.028	0.169 ±0.024
GAE_L2_SUM	0.460 ±0.037	0.234 ±0.028	0.178 ±0.031	0.452 ±0.021	0.141 ±0.025	0.134 ±0.028
GAE_MEAN	0.565 ±0.041	0.309 ±0.030	0.297 ±0.036	0.432 ±0.038	0.125 ±0.025	0.120 ±0.030
GAE_MIXED	**0.609 ±0.037**	**0.361 ±0.029**	**0.352 ±0.039**	0.469 ±0.007	0.182 ±0.024	0.184 ±0.024
GAE_SPECTRAL	0.353 ±0.038	0.098 ±0.019	0.089 ±0.020	0.351 ±0.019	0.059 ±0.012	0.033 ±0.011
Matrix Factorization	0.224 ±0.018	0.024 ±0.024	0.004 ±0.006	0.254 ±0.004	0.01 ±0.003	0.000 ±0.000
Node2Vec-S	0.204 ±0.003	0.003 ±0.001	0.000 ±0.001	0.282 ±0.007	0.006 ±0.002	0.003 ±0.001
Node2Vec-H	0.204 ±0.004	0.003 ±0.001	-0.000 ±0.001	0.287 ±0.007	0.008 ±0.003	0.005 ±0.003

	Europe Air-Traffic			**Brazil Air-Traffic**		
Models	DB	CH	SC	DB	CH	SC
GAE_FIRST	0.425 ±0.010	0.199 ±0.017	0.175 ±0.015	0.479 ±0.021	**0.304 ±0.009**	0.219 ±0.012
GAE_CONCAT	0.394 ±0.018	0.108 ±0.012	0.098 ±0.017	0.490 ±0.016	0.293 ±0.025	**0.235 ±0.022**
GAE_L1_SUM	0.411 ±0.011	0.133 ±0.006	0.123 ±0.005	**0.496 ±0.042**	0.257 ±0.048	0.187 ±0.046
GAE_L2_SUM	0.362 ±0.023	0.078 ±0.021	0.051 ±0.020	0.420 ±0.028	0.175 ±0.041	0.090 ±0.041
GAE_MEAN	0.336 ±0.010	0.061 ±0.012	0.027 ±0.008	0.351 ±0.014	0.079 ±0.015	0.022 ±0.012
GAE_MIXED	0.329 ±0.011	0.058 ±0.011	0.020 ±0.008	0.354 ±0.023	0.107 ±0.023	0.029 ±0.017
GAE_SPECTRAL	**0.439 ±0.016**	**0.213 ±0.010**	**0.178 ±0.011**	0.482 ±0.031	0.278 ±0.050	0.215 ±0.038
Matrix Factorization	0.261 ±0.011	0.025 ±0.011	0.001 ±0.001	0.257 ±0.005	0.043 ±0.005	-0.001 ±0.000
Node2Vec-S	0.316 ±0.013	0.026 ±0.009	0.017 ±0.008	0.305 ±0.013	0.020 ±0.007	-0.003 ±0.007
Node2Vec-H	0.320 ±0.007	0.029 ±0.005	0.014 ±0.004	0.310 ±0.018	0.022 ±0.009	0.000 ±0.007

FINCH (a hierarchical agglomerative method) (Sarfraz et al., 2019) on the embeddings of each model[2]. For evaluation, we consider three well known metrics: Adjusted Rand Index (ARI), Normalized Mutual Information (NMI) and Clustering Accuracy (ACC) as in (Wang et al., 2017) and (Park et al., 2019). All the metrics are between 0 and 1 with 1 being the best result. We use the node

[2]We draw the attention of the reader to the fact that the goal of this work is not to evaluate clustering algorithms but to evaluate the embeddings generated by various graph learning models for the task of clustering. We elected the use of K-means due to its efficiency and simplicity. We selected FINCH (Sarfraz et al., 2019) because it is a recent efficient parameter-free clustering algorithm that does not require any parameters setting, including the number of clusters.

Table 3.9 Clustering node embeddings - using FINCH.

	Cora			**email-Eu-core**		
Models	ACC	NMI	ARI	ACC	NMI	ARI
GAE_FIRST	0.555 ±0.042	0.439 ±0.030	0.342 ±0.047	0.305 ±0.009	0.443 ±0.012	0.164 ±0.014
GAE_CONCAT	0.580 ±0.064	0.435 ±0.036	0.362 ±0.060	0.330 ±0.018	0.482 ±0.014	0.232 ±0.019
GAE_L1_SUM	0.397 ±0.056	0.234 ±0.092	0.070 ±0.051	0.294 ±0.011	0.428 ±0.006	0.149 ±0.015
GAE_L2_SUM	0.624 ±0.044	0.464 ±0.027	0.397 ±0.037	0.338 ±0.009	0.490 ±0.010	0.243 ±0.007
GAE_MEAN	**0.656 ±0.035**	**0.502 ±0.021**	**0.423 ±0.029**	**0.369 ±0.011**	**0.508 ±0.018**	**0.262 ±0.023**
GAE_MIXED	0.638 ±0.030	0.492 ±0.019	0.402 ±0.029	0.359 ±0.012	0.497 ±0.015	0.215 ±0.023
GAE_SPECTRAL	0.309 ±0.033	0.048 ±0.057	0.006 ±0.032	0.272 ±0.027	0.312 ±0.041	0.042 ±0.025
Matrix Factorization	0.293 ±0.003	0.036 ±0.007	-0.003 ±0.003	0.245 ±0.017	0.387 ±0.010	0.153 ±0.015
Node2Vec-S	0.250 ±0.010	0.004 ±0.001	-0.001 ±0.003	0.100 ±0.004	0.052 ±0.005	-0.000 ±0.001
Node2Vec-H	0.202 ±0.015	0.004 ±0.001	-0.000 ±0.001	0.099 ±0.004	0.052 ±0.005	-0.001 ±0.002

	Citeseer			**USA Air-Traffic**		
Models	ACC	NMI	ARI	ACC	NMI	ARI
GAE_FIRST	0.467 ±0.046	0.240 ±0.038	0.172 ±0.039	**0.458 ±0.056**	0.232 ±0.022	**0.210 ±0.040**
GAE_CONCAT	0.468 ±0.036	0.223 ±0.022	0.176 ±0.029	0.410 ±0.048	0.202 ±0.022	0.162 ±0.035
GAE_L1_SUM	0.259 ±0.035	0.060 ±0.032	0.017 ±0.023	0.413 ±0.024	**0.238 ±0.021**	0.160 ±0.014
GAE_L2_SUM	0.463 ±0.041	0.236 ±0.028	0.182 ±0.029	0.382 ±0.036	0.158 ±0.009	0.122 ±0.018
GAE_MEAN	0.558 ±0.047	0.309 ±0.029	0.295 ±0.040	0.348 ±0.023	0.171 ±0.011	0.109 ±0.013
GAE_MIXED	**0.601 ±0.037**	**0.368 ±0.021**	**0.360 ±0.028**	0.369 ±0.015	0.196 ±0.008	0.118 ±0.009
GAE_SPECTRAL	0.211 ±0.004	0.016 ±0.005	0.000 ±0.001	0.354 ±0.030	0.071 ±0.020	0.062 ±0.029
Matrix Factorization	0.427 ±0.017	0.170 ±0.010	0.158 ±0.011	0.309 ±0.021	0.063 ±0.013	0.046 ±0.013
Node2Vec-S	0.211 ±0.003	0.004 ±0.001	0.001 ±0.001	0.232 ±0.012	0.005 ±0.001	0.000 ±0.001
Node2Vec-H	0.206 ±0.003	0.004 ±0.001	0.002 ±0.002	0.245 ±0.019	0.007 ±0.002	0.001 ±0.001

	Europe Air-Traffic			**Brazil Air-Traffic**		
Models	ACC	NMI	ARI	ACC	NMI	ARI
GAE_FIRST	**0.36 ±0.039**	**0.154 ±0.034**	**0.120 ±0.034**	0.402 ±0.044	**0.262 ±0.03**	0.173 ±0.037
GAE_CONCAT	0.321 ±0.027	0.123 ±0.013	0.094 ±0.018	0.415 ±0.048	0.261 ±0.021	0.182 ±0.032
GAE_L1_SUM	0.343 ±0.014	0.110 ±0.012	0.103 ±0.011	**0.424 ±0.05**	0.245 ±0.054	**0.188 ±0.065**
GAE_L2_SUM	0.332 ±0.035	0.123 ±0.016	0.093 ±0.019	0.391 ±0.034	0.250 ±0.024	0.164 ±0.028
GAE_MEAN	0.267 ±0.015	0.049 ±0.008	0.031 ±0.009	0.330 ±0.020	0.166 ±0.023	0.090 ±0.018
GAE_MIXED	0.266 ±0.016	0.048 ±0.013	0.030 ±0.012	0.337 ±0.021	0.162 ±0.014	0.095 ±0.015
GAE_SPECTRAL	0.280 ±0.013	0.073 ±0.019	0.005 ±0.005	0.283 ±0.006	0.102 ±0.007	0.000 ±0.001
Matrix Factorization	0.238 ±0.000	0.022 ±0.000	0.000 ±0.000	0.297 ±0.018	0.051 ±0.008	0.008 ±0.004
Node2Vec-S	0.233 ±0.012	0.030 ±0.006	0.010 ±0.005	0.255 ±0.016	0.042 ±0.007	-0.002 ±0.007
Node2Vec-H	0.234 ±0.012	0.035 ±0.005	0.012 ±0.004	0.266 ±0.025	0.046 ±0.011	0.002 ±0.009

labels as ground truth. As depicted by Tables 3.8 and 3.9, the results on this task are consistent with those of the embedding homogeneity, with GAE_MEAN and GAE_MIXED performing best on Cora, Citeseer and Email-Eu-Code. On the other hand, GAE_FIRST and GAE_CONCAT performed best on the flight datasets. However, it is important to note that the models that did preserve the topological features under-performed the vanilla model GAE_L2_SUM on three out of the six dataset for the task of clustering.

Table 3.10 Node Classification - using Logistic Regression.

	Cora		**email-Eu-core**	
Models	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	**0.796 ±0.009**	**0.809 ±0.008**	0.516 ±0.018	0.683 ±0.010
GAE_CONCAT	0.758 ±0.015	0.780 ±0.014	0.447 ±0.018	0.639 ±0.012
GAE_L1_SUM	0.132 ±0.024	0.348 ±0.022	0.475 ±0.011	0.629 ±0.006
GAE_L2_SUM	0.733 ±0.025	0.76 ±0.016	0.330 ±0.016	0.544 ±0.009
GAE_MEAN	0.765 ±0.016	0.783 ±0.017	0.509 ±0.016	0.681 ±0.008
GAE_MIXED	0.752 ±0.016	0.775 ±0.015	**0.532 ±0.014**	**0.696 ±0.007**
GAE_SPECTRAL	0.066 ±0.000	0.302 ±0.000	0.025 ±0.001	0.169 ±0.002
Matrix Factorization	0.067 ±0.001	0.303 ±0.001	0.008 ±0.000	0.112 ±0.000
Node2Vec-S	0.107 ±0.005	0.259 ±0.004	0.024 ±0.006	0.077 ±0.007
Node2Vec-H	0.105 ±0.005	0.257 ±0.004	0.020 ±0.005	0.078 ±0.005

	Citeseer		**USA Air-Traffic**	
Models	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	**0.630 ±0.010**	0.682 ±0.011	**0.650 ±0.009**	**0.654 ±0.009**
GAE_CONCAT	0.581 ±0.013	0.634 ±0.016	0.643 ±0.009	0.646 ±0.010
GAE_L1_SUM	0.133 ±0.022	0.264 ±0.019	0.634 ±0.022	0.634 ±0.022
GAE_L2_SUM	0.549 ±0.014	0.614 ±0.017	0.552 ±0.031	0.564 ±0.027
GAE_MEAN	0.584 ±0.017	0.655 ±0.018	0.594 ±0.014	0.597 ±0.014
GAE_MIXED	0.608 ±0.012	**0.689 ±0.009**	0.593 ±0.014	0.597 ±0.012
GAE_SPECTRAL	0.058 ±0.000	0.211 ±0.000	0.414 ±0.021	0.439 ±0.019
Matrix Factorization	0.325 ±0.004	0.433 ±0.003	0.482 ±0.001	0.499 ±0.001
Node2Vec-S	0.153 ±0.009	0.189 ±0.010	0.239 ±0.014	0.248 ±0.011
Node2Vec-H	0.150 ±0.008	0.189 ±0.008	0.240 ±0.016	0.247 ±0.015

	Europe Air-Traffic		**Brazil Air-Traffic**	
Models	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GAE_FIRST	**0.509 ±0.023**	**0.524 ±0.022**	**0.564 ±0.032**	**0.572 ±0.030**
GAE_CONCAT	0.502 ±0.029	0.518 ±0.029	0.554 ±0.029	0.569 ±0.030
GAE_L1_SUM	0.461 ±0.027	0.474 ±0.026	0.522 ±0.018	0.528 ±0.022
GAE_L2_SUM	0.400 ±0.017	0.414 ±0.021	0.488 ±0.023	0.509 ±0.024
GAE_MEAN	0.468 ±0.014	0.478 ±0.011	0.495 ±0.026	0.511 ±0.026
GAE_MIXED	0.453 ±0.013	0.469 ±0.013	0.496 ±0.023	0.514 ±0.022
GAE_SPECTRAL	0.347 ±0.004	0.469 ±0.004	0.290 ±0.0210	0.400 ±0.022
Matrix Factorization	0.149 ±0.000	0.198 ±0.000	0.203 ±0.008	0.247 ±0.009
Node2Vec-S	0.311 ±0.020	0.334 ±0.019	0.238 ±0.035	0.257 ±0.036
Node2Vec-H	0.297 ±0.029	0.323 ±0.029	0.231 ±0.048	0.252 ±0.049

3.7 **Task 3**: Node Classification

Here, we evaluate the effect of the embeddings on the task of node classification. We apply the same models used for the classification of the topological features with the same parameters (LG-R, Linear SVM, SVM with RBF Kernel, and MLP). We use the embeddings as attributes and the dataset node labels as ground truth. We report the results of the Logistic Regression classifier (Table 3.10) evaluated with Micro-F1 and Macro-F1. Note that, as reported in the supplementary material website associated with this thesis (`https://github.com/MH-0/RPGAE`),

the results of the other models, that is, SVM-L, SVM-RBF and MLP, are to a large extent consistent with that of Logistic Regression.

From Table 3.10, we notice that GAE_FIRST outperforms the other models on five out of the six datasets, with GAE_CONCAT in close second, while GAE_MIXED gave the best results on email-Eu-Core. The good performance of GAE_FIRST and GAE_CONCAT on the flight datasets for all three tasks is a reconfirmation for the importance of the encoding of the topological features in the embeddings, especially when the ground-truth labels are related to the structural role of the node. On the other hand, the consistent performance of GAE_MIXED on all three tasks is a sign for the importance of the preservation of multiple orders of proximity in the embeddings.

3.8 Analysis and Recommendations

Following the extensive experiments that we have performed, we conclude that, though it is primarily beneficial to have the topological features encoded in the embeddings of the Graph Autoencoder, it is not always necessary. The choice of model should come down to two main factors: (1) The type of task for which the embeddings will be used, such as node classification or clustering, and (2) The correlation between the ground-truth labels of the dataset and the topological features of the nodes. For example, if the embeddings will be used for clustering, and the ground-truth labels are not related to any topological feature, we recommend using the MEAN or the SPECTRAL rule. This is so the model can harness their smoothing power, even if it will be at the expense of losing some of the topological information in the embeddings. On the other hand, if the ground-truth labels are correlated to the Degree of the node or its Triangle Count, we recommend using a model that employs the aggregation by SUM, and that includes the first

layer of the encoder in the embedding. Furthermore, if the ground-truth labels are correlated to a feature that depends on the two-hop neighborhood such as Eigenvector Centrality or the Betweenness Centrality, we recommend concatenating the output of the first layers with the outputs of deeper layers to form the embedding.

This practice could be standardized whenever designing a new architecture for learning on graphs, in what we denote as "Topological Features Preservation" strategy. First, the developer pinpoints the topological features relevant to the task at hand. Second, the developer adopts an architecture that is suitable for the task and also capable of encoding the relevant topological features. This strategy would help improve the performance and better guide the developer to design his/her architecture. In the upcoming chapter, we put this proposed strategy to work in a case study on social influence prediction.

CHAPTER IV

CASE STUDY

In this chapter, we put our findings to work with an application that highlights the importance of adopting a "Topological Features Preservation" strategy when designing a GNN architecture. The purpose of our case study is to show that by adopting an architecture capable of capturing topological features relevant to the underlying task of the model, we can boost the performance of this model without the need for any hand-crafted features. To illustrate our point, we have elected the use of DeepInf (Qiu et al., 2018), a recent deep learning approach for social influence prediction. In what follows, we lay out the scope of our case study over three main steps. First, we introduce DeepInf framework with a summary of its architecture, main components, and any modifications that we have made for the purpose of our case study. Next, we identify the topological features relevant to the social influence problem and adopt an architecture capable of capturing these features, following our findings and recommendations as suggested in the previous section. Finally, we conduct detailed experiments to assess the suitability of the adopted architecture.

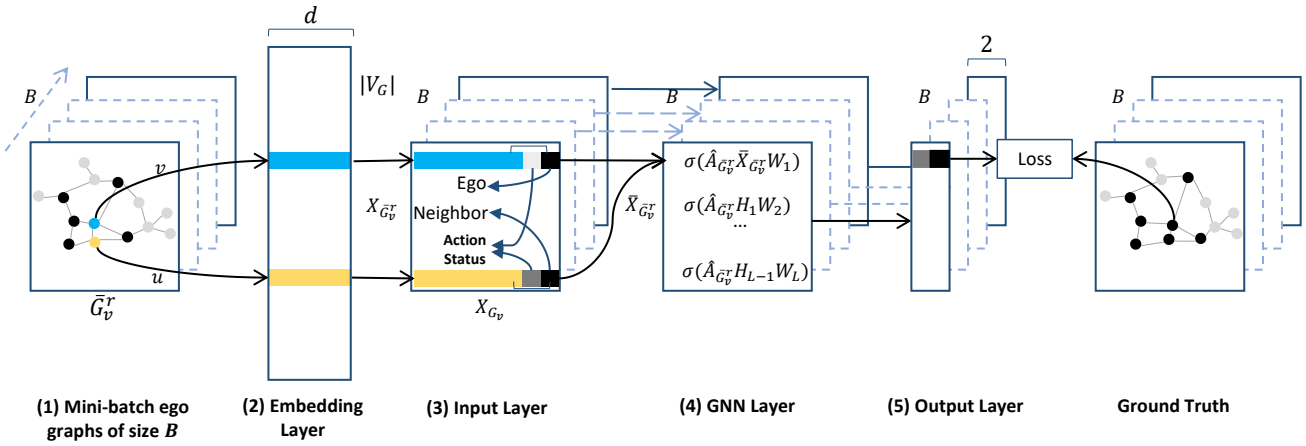

Figure 4.1: A summary of the DeepInf framework, slightly modified for this work. The node v (blue) is the ego-node while the node u (orange) is a neighbor. For the original architecture, please refer to (Qiu et al., 2018).

4.1 DeepInf Framework Summary

DeepInf (Qiu et al., 2018) is an end-to-end Graph Neural Network framework that predicts the user-level social influence in a graph. In other words, DeepInf tries to predict whether a user will perform a certain action (or be *activated*) after being influenced by his surrounding or local network. For example, whether a user will buy a product that his friends bought or will share a story about a subject that his friends are sharing. When this problem is modeled as a graph, the status of a node (i.e., whether activated or not) is referred to as *Action Status* $s_v^t \in \{0, 1\}$ (1 for node v activated at time t, 0 for node v not activated at time t). DeepInf accomplishes the task of social influence prediction, by taking the structure and Action Status of the local neighborhood of a node as input and employing it to learn a latent feature representation of that node. The learned representation is subsequently used to predict whether that node will be activated or not in the future.

Below is a summary that enumerates the steps of the DeepInf framework:

(1) In the first step, DeepInf samples nodes from the graph and extracts their local neighborhoods (i.e., ego-graphs) by using the technique Random Walk with Restart (RWR) (Tong et al., 2006). The ego-graphs are then sampled to have an equal number of nodes and balanced Action Statuses. Subsequently, the ego-graphs are fed into the model in random batches, which accelerates the training and introduces stochasticity to the learning.

(2) Next, an *Embedding Layer* generates feature representations for the nodes. In our work, we use the same pre-trained features provided by (Qiu et al., 2018)

(3) In the *Input Layer*, the features of each node are concatenated with two extra fields. One field determines the Action Status of the node, and the other determines the type of the node, i.e., whether this is the central node (ego-node) or a neighbor node.

(4) In this step, a *GNN Layer* builds a representation of the nodes using both the structure and embedding features of the ego-graph as input. In (Qiu et al., 2018), two variations of the GNN Layer were proposed, DeepInf-GCN, a GNN that employs the SPECTRAL rule (Kipf & Welling, 2017) and DeepInf-GAT, a GNN that uses multi-head attention when aggregating the features of the node (Velickovic et al., 2018). In this work, we focus on improving the performance of DeepInf-GCN.

(5) Finally, the *Output Layer* predicts the Action Status of the central node, minimizing the following loss function:

$$\mathcal{L}(\Theta) = -\sum_{i=1}^{N} log(P_{\Theta}(s_{v_i}^{t+\Delta t} | \bar{G}_{v_i}^{r}, \bar{S}_{v_i}^{t})) \tag{4.1}$$

Therefore, the problem of predicting the user-level influence is reduced to a binary classification problem that is solved by minimizing the negative log-likelihood for

the Action Status of a node at time $t + \Delta t$ w.r.t to the model parameters Θ. Where $\bar{G}^r_{v_i}$ is the graph structure of the sampled neighborhood of node v_i at time t, and $\bar{S}^t_{v_i}$ is the Action Status of the neighborhood of node v_i at time t.

Figure 4.1 represents a summary of the DeepInf framework. Note that in our work, two steps were removed from the original framework: The first is *Instance Normalization*. This step normalizes the embeddings with learnable parameters to avoid overfitting, as described in (Ulyanov et al., 2016). The effect of such parametrized normalization on the preservation of the topological features is beyond the scope of this study, therefore this step was omitted. The second is concatenating *Hand-crafted Centrality Features* to the pre-trained embeddings. We forgo this step since we want to test the ability of the models to predict the influence, by leveraging their own capacity at automatically encoding the centrality features, rather than using manually defined ones. More details about the original framework can be found in (Qiu et al., 2018).

4.2 Datasets and Compared Baselines

Motivated by the fact that the social influence of a node is highly connected to some of its topological features, such as Degree, Eigenvector, and Betweenness Centrality (Li et al., 2016), we leverage the finding of our work — that GAE_CONCAT is capable of capturing these important centrality measures in its embedding — and propose DeepInf-CONCAT, a simple variation to the DeepInf-GCN model. As we will show, our experimental results illustrate that DeepInf-CONCAT significantly improves the prediction capacity of the original model, without the need for explicitly introducing hand-crafted features in the training. In the following, we describe DeepInf variations that we have considered in our comparative analysis.

- **DeepInf-CONCAT-64*** is a variation of with two layers that uses the SUM aggregation rule and concatenates the output of the first and second layer as embedding, equivalent to GAE_CONCAT.
- **DeepInf-L1-SUM-128** is a variation with one layer that uses the SUM aggregation rule, equivalent to GAE_L1_SUM.
- **DeepInf-L2-SUM-128** is a variation with two layers that uses the SUM aggregation rule and the output of the second layer as embedding, equivalent to GAE_L2_SUM.
- **DeepInf-MEAN-128** is a variation with two layers that uses the MEAN aggregation rule with the output of the second layer as embedding, equivalent to GAE_MEAN.
- **DeepInf-GCN-128** is the same model presented in (Qiu et al., 2018). It aggregates the messages using the SPECTRAL rule and uses the output of the last layer as embedding, equivalent to GAE_SPECTRAL.

For the hidden layers of the models, we use the hyperbolic tangent activation function for the new variations, and a hidden layer of size 128, except for DeepInf-CONCAT-64 where we use a hidden layer of size 64. It is important to note that given the nature of the DeepInf model as a GNN rather than GAE, implementing an equivalent to GAE_FIRST or GAE_MIXED is not possible. Furthermore, since the effect of multi-head attention on the preservation of topological features is beyond the scope of this work, and for a fairer and clearer comparison between the other models that do not employ attention, we do not include the results of DeepInf-GAT.

To properly measure the effect of the proposed variations on the DeepInf-GCN model, we adopt the same benchmarks in (Qiu et al., 2018): Digg, Twitter, OAG,

Table 4.1 Graph Datasets for the DeepInf experiments - The field *Observations* indicates the number of sampled ego-graphs. The figures are retrieved from (Qiu et al., 2018).

Dataset	**Nodes**	**Edges**	**Observations**
Digg	279,630	1,548,126	24,428
Twitter	456,626	12,508,413	499,160
OAG	953,675	4,151,463	499,848
Weibo	1,776,950	308,489,739	779,164

and Weibo, with the same experimental and training setup. Table 4.1 summarizes the details of the four datasets.

4.3 Prediction Results and Discussion

We report four metrics for evaluating the performance of the models: Area Under Curve (AUC), Precision (REC), Recall (REC), and F1-Measure (F1). Table 4.2 lists the performance of the compared models on the four datasets. Note that we have also included the results of DeepInf-GCN as reported in the original paper (DeepInf-GCN-128-HF). That is DeepInf-GCN trained with the inclusion of the hand-crafted features and the instance normalization trick. The results clearly show that DeepInf-CONCAT-64 outperforms the other models for both AUC and F1 on all four benchmarks. Furthermore, as shown in Table 4.3, DeepInf-CONCAT-64 significantly outperforms DeepInf-GCN-128 with 10.1% and 7.6% gains on both OAG and Weibo datasets respectively.

This boost in performance can be attributed in part to the ability of DeepInf-CONCAT to automatically encode topological features that are relevant to the node-level influence in the graph. This observation can be further backed by the fact that the other variations, which also use the SUM aggregation rule, underperformed DeepInf-CONCAT. In other words, the gains of DeepInf-CONCAT are

Table 4.2 Prediction results on the four datasets for the five variations of DeepInf-GCN.

Dataset	Model	AUC	PREC	REC	F1
OAG	DeepInf-CONCAT-64	**0.675**	0.338	0.672	**0.450**
	DeepInf-L1-SUM-128	0.651	0.327	0.660	0.438
	DeepInf-L2-SUM-128	0.671	**0.348**	0.632	0.449
	DeepInf-MEAN-128	0.652	0.331	0.647	0.438
	DeepInf-GCN-128	0.613	0.289	**0.746**	0.417
	DeepInf-GCN-128-HF	0.635	0. 302	0.743	0.430
Digg	DeepInf-CONCAT-64	**0.881**	**0.722**	0.718	**0.720**
	DeepInf-L1-SUM-128	0.874	0.686	0.716	0.701
	DeepInf-L2-SUM-128	0.874	0.718	0.692	0.705
	DeepInf-MEAN-128	0.804	0.606	0.675	0.639
	DeepInf-GCN-128	0.866	0.680	**0.720**	0.699
	DeepInf-GCN-128-HF	0.841	0.587	0.676	0.628
Twitter	DeepInf-CONCAT-64	**0.783**	**0.473**	0.665	**0.553**
	DeepInf-L1-SUM-128	0.762	0.442	0.671	0.533
	DeepInf-L2-SUM-128	0.772	0.448	**0.675**	0.539
	DeepInf-MEAN-128	0.727	0.408	0.624	0.494
	DeepInf-GCN-128	0.768	0.459	0.633	0.532
	DeepInf-GCN-128-HF	0.766	0.443	0.667	0.532
Weibo	DeepInf-CONCAT-64	**0.810**	**0.472**	0.732	**0.574**
	DeepInf-L1-SUM-128	0.781	0.445	0.708	0.547
	DeepInf-L2-SUM-128	0.796	0.444	**0.742**	0.556
	DeepInf-MEAN-128	0.774	0.433	0.723	0.542
	DeepInf-GCN-128	0.753	0.410	0.707	0.519
	DeepInf-GCN-128-HF	0.768	0.424	0.713	0.532

Table 4.3 Relative gain of DeepInf-CONCAT-64 in terms of AUC against DeepInf-GCN-128 and DeepInf-GCN-128-HF (With Handcrafted Features).

Model	OAG	Digg	Twitter	Weibo
DeepInf-CONCAT-64	0.675	0.881	0.783	0.810
DeepInf-GCN-128	0.613	0.866	0.768	0.753
Relative Gain	**10.1%**	**1.7%**	**2.0%**	**7.6%**
DeepInf-GCN-128-HF	0.635	0.841	0.766	0.768
Relative Gain	**6.3%**	**4.8%**	**2.2%**	**5.5%**

Table 4.4 Comparison between the results of DeepInf-CONCAT with hidden layers of size 64 and hidden layers of size 128.

Dataset	Model	AUC	PREC	REC	F1
OAG	DeepInf-CONCAT-64	**0.675**	0.338	**0.672**	0.450
	DeepInf-CONCAT-128	0.674	**0.342**	0.663	**0.451**
Digg	DeepInf-CONCAT-64	0.881	0.722	0.718	0.720
	DeepInf-CONCAT-128	**0.885**	**0.738**	**0.723**	**0.730**
Twitter	DeepInf-CONCAT-64	0.783	**0.473**	**0.665**	**0.553**
	DeepInf-CONCAT-128	**0.785**	0.470	0.659	0.549
Weibo	DeepInf-CONCAT-64	0.810	0.472	**0.732**	0.574
	DeepInf-CONCAT-128	**0.811**	**0.474**	0.728	**0.575**

partly due to the concatenation of the first and second layers to form the embedding and the relevant topological information that they encode. As shown in table 4.3, DeepInf-CONCAT-64 also outperforms DeepInf-GCN-128-HF on all four benchmarks, highlighting the importance of allowing an automatic encoding of the relevant topological features vs manually defining them.

It is important to note that DeepInf-CONCAT-64 outperforms the other models while having half their hidden layer size. To study the effect of the hidden layer size and eliminate *overparameterization* as a potential cause for the underperformance of the other models, we compare DeepInf-CONCAT-64 to DeepInf-CONCAT-128, which constitutes the same model with a hidden layer size of 128. As shown in table 4.4, with DeepInf-CONCAT-128, we notice an even higher boost in performance on most of the datasets, further demonstrating the effectiveness of the DeepInf-CONCAT model.

To summarize, in this section, we have demonstrated how we can put to practice the findings of our work concerning the representational power of graph embeddings. We have shown that by carefully choosing an architecture that has been experimentally shown to preserve task-relevant topological features, we are able to significantly improve the model's performance. We believe that such an un-

derstanding of the representational power of the GNN and the specialization of each layer in preserving the topological features can greatly reduce the time of the model-search step in the development process and architecture design.

CHAPTER V

CONCLUSION

In this study, we investigated the representational power of the graph embeddings at preserving important topological structures in the graph. For this purpose, we conducted an extensive empirical study on three unsupervised classes of embedding models, and seven different variants of Graph Autoencoders. Our results show that five topological features: The Degree, the Local Clustering Score, the Betweenness Centrality, the Eigenvector Centrality, and Triangle Count are concretely preserved in the first layer of the Graph Autoencoder that employs the SUM aggregation rule, under the condition that the model preserves the second-order proximity. We further support our claims with 2D visualizations that reveal a well-organized hierarchical distribution of the topological features in the embeddings of the aforementioned model. Furthermore, we studied the effect of topological features preservation on downstream tasks. Our results show that the presence of the topological features in the embeddings can significantly enhance the performance of the model on downstream tasks, especially when the preserved topological features are relevant to the target task. Finally, we put our findings to practice in a case study that involved applying our recommendations to a social influence prediction framework. Our results show a great boost in performance, highlighting the importance of having a "Topological Features Preservation" strategy when designing a GNN architecture.

We believe that having a better understanding of embeddings composition would help the interpretability and explainability of the graph learning results. Subsequently, this would help the developers of the Graph Neural Network technology better design their architectures, by having a deeper understanding of the role of each layer in the network and the specialization of the different aggregation functions. This knowledge would guide them in devising robust models that are better suited for the downstream tasks and greatly reduce the model search-time step in the development process. Furthermore, we emphasize that few works have explored the interpretability aspect of graph embeddings. Therefore, we are confident that our work would constitute a reference for future studies to expand the investigation into other important structures that might be encoded in the embeddings and their role and effect on graph learning.

For future studies, it would be interesting to explore the representational power of the embedding models in the context of other types of graph structures and their unique characteristics. For example, investigating how the temporal aspects of the dynamic graphs are encoded in the embeddings. Or, how different dimensions could be encoded in multidimensional graphs. It would be equally interesting to investigate the effect of the preservation of topological features on other types of downstream tasks. For example, the task of link prediction in heterogeneous graphs when encoded with relational Graph Neural Networks (rGNN) (Schlichtkrull et al., 2018). Another important venue would be the study of the effect of attention (GAT) (Velickovic et al., 2018) on the preservation of the topological features and its subsequent effect on downstream tasks. We believe that all these directions would help further our understanding of the composition of the embeddings and would lead to substantial improvements in graph learning.

BIBLIOGRAPHY

Belkin, M. & Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of Neural Information Processing Systems (NIPS)*, pp. 585–591.

Bonner, S., Kureshi, I., Brennan, J., Theodoropoulos, G., McGough, A. S. & Obara, B. (2019). Exploring the semantic content of unsupervised graph embeddings: An empirical study. *Data Science and Engineering*, *4*(3), 269–289.

Cai, H., Zheng, V. W. & Chang, K. C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, *30*(9), 1616–1637.

Caliński, T. & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, *3*(1), 1–27.

Davies, D. L. & Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *1*(2), 224–227.

Dong, W., Wu, J., Bai, Z., Li, W. & Qiao, W. (2020). Design of affinity-aware encoding by embedding graph centrality for graph classification. *Neurocomputing*, *387*, 321–333.

Goyal, P. & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, *151*, 78–94.

Grover, A. & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining*, pp. 855–864.

Hamilton, W. L., Ying, R. & Leskovec, J. (2017a). Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, *40*(3), 52–74.

Hamilton, W. L., Ying, Z. & Leskovec, J. (2017b). Inductive representation learning on large graphs. In *Proceedings of Neural Information Processing Systems (NIPS)*, pp. 1024–1034.

Hasanzadeh, A., Hajiramezanali, E., Narayanan, K. R., Duffield, N., Zhou, M. & Qian, X. (2019). Semi-implicit graph variational auto-encoders. In *Proceedings of Neural Information Processing Systems (NIPS)*, pp. 10711–10722.

Kipf, T. N. & Welling, M. (2016). Variational graph auto-encoders. In *Proceedings of NIPS Workshop on Bayesian Deep Learning*, pp. 1–3.

Kipf, T. N. & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations, (ICLR)*, pp. 1–14.

Leskovec, J. & Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Li, Q., Cao, Z., Zhong, J. & Li, Q. (2019). Graph representation learning with encoding edges. *Neurocomputing*, *361*, 29–39.

Li, Q., Han, Z. & Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 32nd Conference on Artificial Intelligence (AAAI)*, pp. 1–8.

Li, X., Liu, Y., Jiang, Y. & Liu, X. (2016). Identifying social influence in complex networks: A novel conductance eigenvector centrality model. *Neurocomputing*, *210*, 141–154.

Ma, Y., Wang, S., Aggarwal, C. C., Yin, D. & Tang, J. (2019). Multi-dimensional graph convolutional networks. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pp. 657–665.

Maaten, L. V. D. & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, *9*, 2579–2605.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of the 1st International Conference on Learning Representations (ICLR) Workshop*.

Ou, M., Cui, P., Pei, J., Zhang, Z. & Zhu, W. (2016). Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining*, pp. 1105–1114.

Park, J., Lee, M., Chang, H. J., Lee, K. & Choi, J. Y. (2019). Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6519–6528.

Perozzi, B., Al-Rfou, R. & Skiena, S. (2014). Deepwalk: Online learning of social

representations. In *Proceedings of the 20th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 701–710.

Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K. & Tang, J. (2018). Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 2110–2119.

Rizi, F. S. & Granitzer, M. (2017). Properties of vector embeddings in social networks. *Algorithms*, *10*(4), 109.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65.

Sarfraz, M. S., Sharma, V. & Stiefelhagen, R. (2019). Efficient parameter-free clustering using first neighbor relations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8934–8943.

Schlichtkrull, M. S., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I. & Welling, M. (2018). Modeling relational data with graph convolutional networks. In *Proceedings of The 15th International Semantic Web Conference (ISWC)*, pp. 593–607.

Song, W., Wang, S., Yang, B., Lu, Y., Zhao, X. & Liu, X. (2018). Learning node and edge embeddings for signed networks. *Neurocomputing*, *319*, 42–54.

Tong, H., Faloutsos, C. & Pan, J. (2006). Fast random walk with restart and its applications. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM)*, pp. 613–622.

Ulyanov, D., Vedaldi, A. & Lempitsky, V. S. (2016). Instance normalization: The missing ingredient for fast stylization. *CoRR abs/1607.08022*.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. & Bengio, Y. (2018). Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, pp. 1–12.

Wang, C., Pan, S., Hu, R., Long, G., Jiang, J. & Zhang, C. (2019a). Attributed graph clustering: A deep attentional embedding approach. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3670–3676.

Wang, C., Pan, S., Long, G., Zhu, X. & Jiang, J. (2017). Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pp. 889—-898.

Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., Huang, Z., Guo, Q., Zhang, H., Lin, H., Zhao, J., Li, J., Smola, A. J. &

Zhang, Z. (2019b). Deep graph library: Towards efficient and scalable deep learning on graphs. In *Proceedings of ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Weisfeiler, B. & Leman, A. (1968). The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, *2*(9), 12–16.

Wu, J., He, J. & Xu, J. (2019). Demo-net: Degree-specific graph neural networks for node and graph classification. In *Proceedings of the 25th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 406–415.

Xu, K., Hu, W., Leskovec, J. & Jegelka, S. (2019). How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, pp. 1–17.