University of Amsterdam

MSc Artificial Intelligence
Master Thesis

A Modular Framework for Unsupervised Graph Representation Learning

by

Daniel Fernando Daza Cruz

11660201

July 16, 2019

36 EC
March 2019 - July 2019

Supervisor:
Thomas Kipf, MSc

Assessor:
Prof. Dr. Max Welling

Abstract

Graphs are data structures well suited to model many sources of information in the real world, including social networks, domain knowledge, and molecules. The availability of large datasets motivates the application of machine learning methods in tasks such as detecting clusters, predicting links between entities, and assigning a label to an entity. Several techniques have been proposed in the literature towards solving these problems, some of which involve carefully designing an appropriate representation of the data with enough predictive power that can be used to train a machine learning model.

Instead of specifying a representation in advance, more recent approaches seek to learn the representation from the data, so that it preserves the structure of the graph while being useful to solve related tasks, in the absence of a ground truth. This is the problem of unsupervised graph representation learning. We review recently proposed methods, and we identify across them a design pattern composed by a series of components. This turns into a modular framework that can be used to study existing methods and devise novel ones. We validate our framework with experiments on real world graphs, and we find that some changes in the components of existing methods can yield significant improvements. A hyperparameter study allows us to identify a particularly strong method of representation learning for the tasks of link prediction and node classification.

Following recent advances in optimal transport for machine learning, we experiment with a method to learn node representations using Wasserstein spaces. Under several conditions, we find that these spaces preserve the structure of the graph well, but in spite of regularization strategies, they do not generalize well.

Acknowledgements

I would like to thank Thomas Kipf for his great guidance during the development of this work. He helped to make it a fruitful learning experience, and I am grateful for his time, commitment, and for every advice regarding my work and career.

Many thanks to Max Welling for agreeing to be the assessor of this thesis.

I also thank my parents for their never-ending support, and Daniela, for always being there and encouraging me.

Contents

Chapter 1

Introduction

A wide variety of data in the world can be described as a set of entities that interact between each other: people in a social network, objects in an image, atoms in a molecule, and documents on the Internet are some examples. These sources of information are naturally represented by a *graph*, a data structure that represents entities as *nodes*, and the existence of a relationship between them with *edges* or *links* that connect two nodes.

The theoretical analysis of graphs forms a field of study itself (West et al., 1996) that has been used to model problems in the social sciences (Wasserman et al., 1994), biology, and computer science (Dorogovstev & Mendes, 2003). Graphs have also been proposed as a powerful tool that machines can use to reason about the world (Battaglia et al., 2018). Just as humans observe and interact with the world through the composition of discrete entities in observations (Biederman, 1987; Marr & Nishihara, 1978) and the use of language (Osherson & Smith, 1981; Clark et al., 1985), endowing machines with the ability to process graphs could improve their understanding of the world, by the allowing them to combine knowledge in novel situations (Lake et al., 2017; Marcus, 2018).

The arrival of new technologies has brought applications that benefit from the collection of large amounts of graph-structured data. The availability of the resulting graph datasets enables the use of machine learning algorithms, which learn from observations to solve related tasks such as node classification (Sen et al., 2008; Bhagat et al., 2011a), the design of recommender systems (Fouss et al., 2007; Backstrom & Leskovec, 2011), knowledge base completion (Nickel et al., 2011; Yang et al., 2015a; Schlichtkrull et al., 2018), and scene understanding (Xu et al., 2017; Herzig et al., 2018; Liang et al., 2018).

Some of the methods that have been proposed in the machine learning literature require the specification of hand-engineered functions, or *kernels*, designed to capture similarity between nodes (Vishwanathan et al., 2010; Kriege et al., 2019). Others rely on features containing graph statistics, that are manually extracted (Bhagat et al., 2011b; Liben-Nowell & Kleinberg, 2003; Barabási & Albert, 1999; Zhou et al., 2009). These approaches rely on specific assumptions about the graph and the task at hand, and prescribe machine learning models with a structure that might not generalize to other graphs (Zhang & Chen, 2018).

A more promising approach is to use the data to learn a representation that is flexible enough to capture enough information from the observation, while discarding less revelant aspects. This is the problem of *representation learning* (Bengio et al., 2013). In the context of graphs, a representation commonly consists of a real-valued vector, or an *embedding*, that is assigned to each node, edge, or to the graph itself. Embeddings are the realization of distributed representations of entities, which have long been identified as computationally efficient (Hinton et al., 1984) and are widely applied in deep learning. In this work, we are particularly interested in learning representations of nodes.

In semi-supervised approaches to machine learning on graphs, where partially labeled data for a given task is available, previous works have used Laplacian regularization to enforce the graph structure, but they do not learn embeddings (Zhu et al., 2003; Belkin et al., 2006; Weston et al., 2012). Following the application of neural networks to graph-structured data (Gori et al., 2005; Scarselli et al., 2009), more recent works consider networks as a general case of data structured in sequences or grids (such as audio or image signals) and propose a convolutional operator on graphs (Bruna et al., 2013; Duvenaud et al., 2015; Defferrard et al., 2016; Kipf & Welling, 2016a). This has sparked interest in architectures for deep learning that process graph data, such as message passing networks (Gilmer et al., 2017), attention mechanisms (Veličković et al., 2017), and other variants (Zhou et al., 2018; Wu et al., 2019b). These architectures provide an inductive bias to train models that learn from graphs of different types, including directed and undirected graphs, and in the presence of node, edge, and graph attributes (Battaglia et al., 2018).

In this thesis, we address the problem of learning representations of nodes that capture the structure of the graph, while not being limited to a specific application. This corresponds to the *unsupervised learning* problem, where we assume that no target values are given during the learning process, although we seek for representations that have general applicability in graph-related problems. Our aim is thus to

study and improve upon existing methods for unsupervised graph representation learning.

1.1 Contributions

The problem of unsupervised graph representation learning has a diverse background that has been influenced by related research in dimensionality reduction, variational inference, natural language processing and others. In addition to this, as the field of machine learning advances, new methods become applicable. A review of the literature shows that several methods have been proposed that when compared, reveal a design pattern. We make this pattern concrete by establishing a framework of five components for representation learning on graphs.

Our modular framework allows us to analyze algorithms for representation learning on graphs, by studying their behavior under changes in their components. This results in insights about their design and the methodology and datasets used to evaluate them. We also question the utility of certain design choices, and we find that in some cases, methods can be simplified significantly while retaining competitive performance.

We additionally leverage the modular framework to devise novel variants. We carry out a hyperparameter study that aims to find a combination of components with improved generalization, which results in a method that outperforms all others in the tasks of link prediction and node classification. Our analysis of this method shows that a linear model suffices to learn embeddings that are competitive when tested in real-world networks.

Motivated by recent results in optimal transport for machine learning (Frogner et al., 2019), we extend our framework by considering embeddings on Wasserstein spaces, which have been shown to be powerful spaces that preserve well the geometry of the space on which the objects being embedded lie. While Frogner et al. (2019) evaluate the distortion caused by the embedding on the training data using small networks, we further expand on their results and evaluate the method on real-world networks, in combination with different components of our modular framework. We find that Wasserstein spaces successfully preserve the structure of the graph, but the learned embeddings do not present a clear advantage in generalization when compared to other methods.

Chapter 2

Unsupervised Graph Representation Learning

Graphs provide a way to represent information about entities and the relations between them. They are fundamentally defined by a set of links, or *edges*, between entities. For attributed graphs, every node can be further associated with a set of features, for example demographic user features in social networks, or a bag-of-words vector for a document in a publication network. In the absence of features, an initial representation can be given by a one-hot-vector that uniquely identifies each node. These features form a representation that is usually high-dimensional and sparse. We are thus concerned with learning low-dimensional node representations or *embeddings* in an unsupervised way, that capture node features and the structure of the graph, so that they can be used by machine learning models without having to refer to the original graph. These models could then be applied in tasks such as link prediction and node classification.

In this chapter, we provide an overview of the problem, and we summarize the details of methods proposed in the literature to address it. We find that these methods can be unified under a framework that motivates extensions and multiple questions that are treated in our work.

2.1 Problem description

We consider an undirected, unweighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges of the form (v_i, v_j), with $v_i, v_j \in \mathcal{V}$. Let $|\mathcal{V}| = N$ be the number of nodes in the graph. We define $\mathbf{A} \in \mathbb{R}^{N \times N}$ to be the binary adjacency

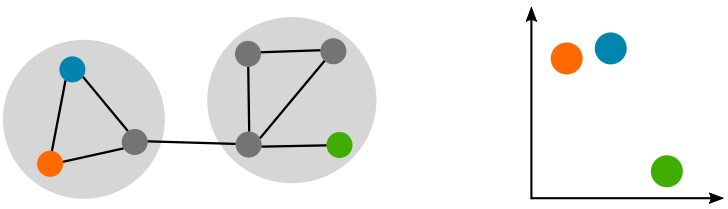


Figure 2.1: Nodes in a graph (left) represented in a continuous embedding space (right). Under the homophily hypothesis, nodes in the same community (enclosed by the shaded ovals) are close in the embedding space, far from embeddings of nodes in a different community.

matrix with entries $A_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$, and 0 otherwise. Each node has an associated feature vector $\mathbf{x}_i \in \mathbb{R}^F$, which for all nodes in the graph we arrange in the rows of a matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$. We are interested in the problem of learning embeddings $\mathbf{z}_i \in \mathbb{R}^D$ for each node in the graph. Ideally, the embeddings should be low-dimensional, that is, $D \ll F$, while still capturing useful information about node features and the graph that can be used in related tasks. We denote with $\mathbf{Z} \in \mathbb{R}^{N \times D}$ the matrix containing all node embeddings.

In the absence of labels or a clearly predefined task for which the representations will be used, unsupervised learning approaches for graphs regularly make use of contrastive methods, which have general applicability when learning representations. These methods work by defining a score S for pairs of samples that follow observations in the data, also known as *positive samples*, and a score $\tilde{S}$ for *negative samples* that deviate from observations. Learning then amounts to minimizing a loss function designed to maximize the score for positive samples, and minimize it for negative samples.

The contrastive method has been applied successfully to the problem of learning representations of words (Mikolov et al., 2013b; Mnih & Kavukcuoglu, 2013; Collobert & Weston, 2008). An example is the Skipgram model, where words are represented as vectors in a continuous space such that co-occurring words are close in the embedding space (Mikolov et al., 2013a). A motivation for embedding words closely based on their co-ocurrence stems from the *distributional hypothesis*, which states that the meaning of a word is characterized by its context (Firth, 1957). Its analog in the context of graphs, the *homophily hypothesis*, states that connected nodes that belong to the same community should be close in the embedding space (Hoff et al., 2002; Fortunato, 2010), as illustrated in figure 2.1.

The homophily hypothesis and the Skipgram model have inspired algorithms for representation learning on graphs, such as DeepWalk and node2vec (Perozzi et al., 2014; Grover & Leskovec, 2016), where the model is encouraged to assign high scores to a node and its close neighbors. More recent approaches still use a contrastive approach, while differing in their specifics.

2.2 Learning algorithms

A complete specification of a learning algorithm entails the definition of a mapping from an initial node representation to an embedding, together with an appropriate loss function and a definition of what constitute positive and negative samples. In this section, we summarize these details for some of the existing methods in the literature.

2.2.1 DeepWalk

Early approaches that learned distributed representations of discrete entities were developed for the task of language modeling, where given a sequence of words, the next has to be predicted (Bengio et al., 2000). More general architectures were proposed by Mikolov et al. (2013a) with the specific purpose of word representation learning, so that words appearing in the same context are close in the embedding space.

DeepWalk extends this idea to representation learning on graphs, by defining the context of a node v_i in a graph as a sequence of nodes that occur in random walk on the graph that passes through v_i:

$$C_i = \{v_{i-w}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+w}\}.$$

Embeddings of nodes are then trained with Stochastic Gradient Descent (SGD) to minimize the negative log-probability of a node in the context:

$$-\log p(v_j|\mathbf{z}_i) \quad \forall v_j \in C_i. \tag{2.1}$$

A shortcoming of modeling the probability in equation 2.1 arises when the number of nodes in the graph grows, so that computing the probability for every node in the graph becomes computationally expensive. This problem has been addressed in language models with *hierarchical softmax*, which gives an approximation to the probability over nodes in the graph that is faster to compute (Mnih & Hinton,

2008); and with *negative sampling*, where the loss function is modified with a similar objective that approximates the log-probability (Gutmann & Hyvärinen, 2012; Mikolov et al., 2013b). With negative sampling, for each node in the graph we maximize the log-probability of its co-occurrence with a positive sample v_p in a random walk, and minimize it for a negative sample v_n randomly sampled from a prior distribution $p(v)$ over all nodes in the graph. The loss function to minimize is the following:

$$\mathcal{L} = -\log p(v_p \in C_i | v_i) - \log\left(1 - p(v_n \in C_i | v_i)\right) \quad \text{s.t. } v_n \sim p(v). \tag{2.2}$$

The probabilities in equation 2.2 can be seen as scores assigned to positive and negative samples, given a node v_i. A common scoring function consists of the inner product of embeddings followed by a sigmoid (Grover & Leskovec, 2016), in which case the resulting loss function per node in the graph is the following:

$$\mathcal{L} = -\log \sigma\left(\mathbf{z}_i^\top \mathbf{z}_p\right) - \log \sigma\left(-\mathbf{z}_i^\top \mathbf{z}_n\right). \tag{2.3}$$

This loss function is therefore an efficient method to learn node embeddings, as it does not require the calculation of a probability distribution over all nodes in the graph. On the other hand, since every node is directly assigned an embedding that is trained via the minimization of equation 2.3, DeepWalk does not take node features into account, although it can be extended via matrix factorization to include them, as shown by Yang et al. (2015b).

If we restrict the context of a node v_i to random walks of length 1, and negative samples are drawn from a conditional distribution $p(v|v_i)$ so that only nodes not in the 1-hop neighborhood of v_i can be selected, the loss function in equation 2.2 turns into the negative log-probability of entries of the adjacency matrix $\mathbf{A}$:

$$\mathcal{L} = -\log p(A_{ip} = 1 | \mathbf{z}_i, \mathbf{z}_p) - \log p(A_{in} = 0 | \mathbf{z}_i, \mathbf{z}_n). \tag{2.4}$$

This special case can be seen as an approximate prediction of the adjacency matrix, and is related to *autoencoder* approaches for graphs that seek to reconstruct the graph structure from low-dimensional node representations.

2.2.2 Graph Autoencoders

Autoencoders have long been used to learn low-dimensional representations of observations, so that they are informative enough to be used to reconstruct the original observation (Hinton & Salakhutdinov, 2006). The process of mapping

an observation to the low-dimensional space is carried out by the *encoder*, while the reconstruction is done by the *decoder*. Usually, the encoder and decoder are neural networks trained with SGD through a loss function that captures the error in the reconstruction. This approach has been utilized in previous works on graph representation learning where embeddings are used to reconstruct the neighborhood of a node (Cao et al., 2016; Wang et al., 2016), or the adjacency matrix (Kipf & Welling, 2016b; Tran, 2018).

An example architecture of a Graph Autoencoder (GAE) consists of an encoder neural network f_θ that maps node features to an embedding: $\mathbf{z}_i = f_\theta(\mathbf{x}_i)$, and a decoder that takes embeddings for a pair of nodes (v_i, v_j) and predicts a link between them. Under the assumption that the probability of an entry in the adjacency matrix is independent of the rest given the embeddings $\mathbf{z}_i$ and $\mathbf{z}_j$, the corresponding reconstruction loss of the adjacency matrix corresponds to the binary cross-entropy loss for each of its entries:

$$\begin{aligned}
\mathcal{L} &= -\log p(\mathbf{A}|\mathbf{Z}) \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{N} \log p(A_{ij}|\mathbf{z}_i, \mathbf{z}_j) \\
&= -\sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij} \log p(A_{ij}=1|\mathbf{z}_i, \mathbf{z}_j) + (1-A_{ij}) \log p(A_{ij}=0|\mathbf{z}_i, \mathbf{z}_j),
\end{aligned} \tag{2.5}$$

where $\mathbf{Z}$ is the matrix containing all node embeddings, and the probabilities can be obtained via an inner product of embeddings, as in DeepWalk.

As suggested by Kipf & Welling (2016b), for graphs with a large number of nodes and high sparsity, this loss can be modified by subsampling entries with $A_{ij} = 0$. Equivalently, for a node v_i we can consider positive samples as any of its neighbors, and negative samples as any node not connected to it, in which case the resulting loss function is the same as the special case of DeepWalk in equation 2.4.

Learning node embeddings can also be cast as a problem of inference of a latent variable, as proposed by Kipf & Welling (2016b) in the Variational Graph Autoencoder (VGAE). In the VGAE, the encoder q parameterizes the posterior distribution of the latent embeddings $\mathbf{Z}$, and a prior distribution $p(\mathbf{Z})$ is defined, such as a standard Gaussian. The model is trained with the reparameterization trick (Kingma & Welling, 2013) to minimize the negative variational lower bound:

$$\mathcal{L} = -\mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] + \mathrm{KL}\left(q(\mathbf{Z}|\mathbf{X},\mathbf{A})\|p(\mathbf{Z})\right) \tag{2.6}$$

The loss function of the VGAE is similar to that of GAE by including a

reconstruction error, with the addition of a regularization term that penalizes a posterior that deviates from the prior. In a related work, Davidson et al. (2018) note that the choice of a Gaussian prior and posterior might not be suitable for graph-structured data, and instead proposes to use a hyperspherical latent space.

A distinguishing feature of GAE and its variational formulations in comparison with DeepWalk, is the use of an encoder that incorporates node features. While for DeepWalk only nodes observed during training are assigned an embedding, encoders are flexible functions that can map features of unobserved nodes to an embedding, which is also known as the *inductive* property.

2.2.3 Deep Graph Infomax

Unsupervised learning methods can also be devised by specifying a loss function that operates on the embedding space, as opposed to the loss in the observation space that DeepWalk and Graph Autoencoders use. This is the approach proposed by Veličković et al. (2018b) in Deep Graph Infomax (DGI). The main purpose of this method is to learn node embeddings that maximize the mutual information with a global representation of the graph. This is achieved through local *patch representations*, defined as a continuous vectors that aggregate features of a node and its neighbors.

Patch representations are obtained in DGI with a Graph Convolutional Network (GCN) (Kipf & Welling, 2016a), which propagates node features across the graph and acts as a node encoder that outputs a representation $\mathbf{z}_i$. A global graph summary is obtained as follows:

$$\mathbf{s} = \sigma\left(\frac{1}{N}\sum_{i=1}^{N}\mathbf{z}_i\right) \tag{2.7}$$

where σ is the sigmoid function.

DGI uses a contrastive learning approach with a discriminator $D(\mathbf{z}_i, \mathbf{s})$, which models the probability that an embedding $\mathbf{z}_i$ belongs to a node in the graph, or to a node in a corrupted version of the graph, given the global summary $\mathbf{s}$. A corrupted graph can be obtained with a permutation of node features, or by adding and removing edges in the graph. Maximization of the mutual information between local and global representations is obtained by minimizing the following loss for each node:

$$\mathcal{L} = -\log D(\mathbf{z}_i, \mathbf{s}) - \log(1 - D(\tilde{\mathbf{z}}_i, \mathbf{s})) \tag{2.8}$$

where $\tilde{\mathbf{z}}_i$ is the embedding of node v_i in a corrupted version of the graph.

2.2.4 Graph2Gauss

All of the methods described so far consider the embedding of a node as a single vector. Graph2Gauss (G2G) (Bojchevski & Günnemann, 2018) instead proposes to represent nodes as Gaussian distributions, so that an embedding of a node v_i is given by a mean vector $\mathbf{z}_{\mu i}$, and a vector $\mathbf{z}_{\sigma i}$ for the diagonal of the covariance matrix, which are obtained with an encoder neural network that takes as input node features.

The method uses an energy-based loss that encourages the energy of positive samples to be low, and high for negative samples. The energy of a pair of nodes (v_i, v_j) measures how distant they are, and is computed with the KL divergence of the Gaussian embedding distributions:

$$E_{ij} = \mathrm{KL}(\mathcal{N}(\mathbf{z}_{\mu i}, \mathrm{diag}(\mathbf{z}_{\sigma i}))\|\mathcal{N}(\mathbf{z}_{\mu j}, \mathrm{diag}(\mathbf{z}_{\sigma j}))). \tag{2.9}$$

For a positive pair of nodes (v_i, v_j) and a negative pair (v_i, v_k), the loss to minimize is the square-exponential loss (LeCun et al., 2006):

$$\mathcal{L} = E_{ij}^2 + e^{-E_{ik}} \tag{2.10}$$

G2G defines positive and negative samples through a ranked strategy, by obtaining a list of nodes in the neighborhood of a node of interest, and sorting it by the distance in the graph in ascending order. By taking consecutive pairs in the list as positive and negative samples, nodes get more distant in the embedding space as the distance increases in the graph.

2.3 Components for unsupervised learning on graphs

The previous summary of methods reveals a pattern in the way they are devised, that can be described as a series of components with a distinct role in the process of learning representations. In this section we present a modular framework under which existing algorithms can be described and extended, which we depict in figure 2.2 and describe next.

2.3.1 Sampling strategies

The first component for unsupervised learning on graphs involves selecting appropriate positive and negative samples for training. Most approaches take into account

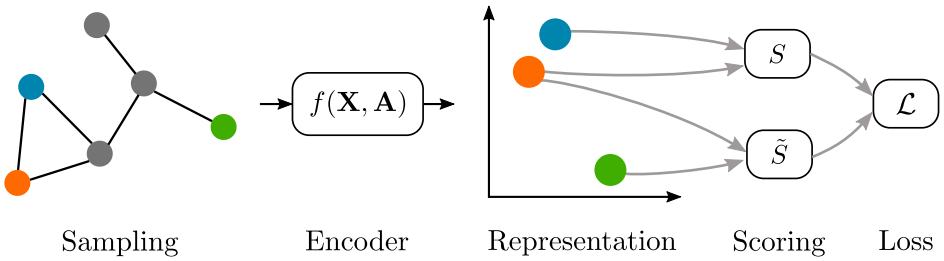


Figure 2.2: Methods for unsupervised learning on graphs can be described by a modular framework consisting of five components. On the left, the process starts with the sampling strategy that selects positive and negative samples. Nodes are embedded into a given representation by encoding node features $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$, and node pairs are assigned scores that are used in the loss function to optimize.

the structure of the graph to select these, so that connected nodes have similar embeddings, whereas distant or disconnected nodes are also distant in the embedding space.

DeepWalk uses random walks of fixed length to find positive samples, and random nodes as negative examples. Properties of the random walk can be modified to obtain embeddings that capture communities or local structural roles, as node2vec (Grover & Leskovec, 2016), which introduces parameters that balance between walks that remain close to a node, and walks that explore other neighborhoods.

In GAE, the reconstruction of the adjacency matrix motivates the use of first order neighbors as positive samples, and non-neighbors as negative samples. G2G proposes a similar sampling strategy, but ranks closer nodes higher than distant nodes, and the size of the sampling neighborhood is a hyperparameter.

To maximize the local mutual information in DGI, the node embeddings of the graph are considered as positive samples. After corrupting the graph, new node embeddings are obtained and used as negative samples. Veličković et al. (2018b) show experimentally that DGI is robust to corruption strategies, such as dropping or adding edges from the graph or shuffling the rows of the feature matrix $\mathbf{X}$, although they observe that the latter yields better performance in node classification.

2.3.2 Node encoders

A key problem in the application of machine learning to graphs is incorporating the structure of the graph within a model. Related methods are based on choosing graph statistics by hand (Bhagat et al., 2011b), or by designing suitable graph kernels to capture similarity between nodes (Vishwanathan et al., 2010; Kriege et al., 2019). An alternative is to learn an appropriate mapping from nodes or subgraphs, to representations that are optimized to preserve the structure of the graph (Hamilton et al., 2017). This is a more flexible approach, as the representations are learned end-to-end from data, avoiding the need to manually select features.

Whether a node is assigned a vector of features, or a one-hot representation, initial representations of nodes in a graph can potentially be very sparse and high dimensional, on the order of the number of nodes or more. This motivates the specification of a function that *encodes* these high dimensional vectors into embeddings on a low dimensional space. The reduction in the dimension of the representation space brings many advantages, such as decreasing sparsity, extracting useful features for downstream tasks, and improving computational and sample efficiency (Belkin & Niyogi, 2002; Bengio et al., 2013).

The simplest graph encoder is a lookup table (LUT) of embeddings $\mathbf{E} \in \mathbb{R}^{N \times D}$, and produces a node embedding by matrix multiplication with a one-hot vector $\mathbf{x}_i$, such that $\mathbf{z}_i = \mathbf{E}\mathbf{x}_i$. This is the approach adopted by graph factorization algorithms (Ahmed et al., 2013a), spectral clustering (Tang & Liu, 2011), and DeepWalk.

By its definition, the LUT encoder disregards any information provided by node features and the structure of the graph. This has motivated the design of encoders that leverage at least one of these aspects. Such encoders have an increased flexibility compared to a LUT encoder, by including learnable parameters and nonlinearities, as in a Multi-Layer Perceptron (MLP). This type of encoder is used by G2G, and is defined with the following propagation rule:

$$\mathbf{H}^{(l+1)} = \mathrm{MLP}(\mathbf{H}^{(l)}) = f\left(\mathbf{H}^{(l)}\mathbf{W}^{(l+1)}\right) \tag{2.11}$$

where we define $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{W}^{(l+1)}$ is the matrix of weights, and f is a nonlinear activation function. A node encoder is then formed by stacking L of these layers, so that the activations in the last layer $\mathbf{H}^{(L)} = \mathbf{Z}$ form the node embeddings.

Other encoders further consider the graph structure, such as Graph Convolutional Networks (GCN), as proposed by Kipf & Welling (2016a), with the following propagation rule for layer $l + 1$:

$$\mathbf{H}^{(l+1)} = \mathrm{GCN}(\mathbf{H}^{(l)}, \mathbf{A}) = f\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l+1)}\right) \tag{2.12}$$

where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the degree matrix, and the term $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is known as the *normalized adjacency matrix* $\tilde{\mathbf{A}}$. In comparison with MLPs, GCNs exploit structural information by introducing the adjacency matrix. In a regular, fully connected neural network, the adjacency matrix is not present in the propagation rule, so each feature vector in the rows of $\mathbf{H}^{(l+1)}$ is only affected by the corresponding row of $\mathbf{H}^{(l)}$, dismissing any relationship between a node and its neighbors. By introducing the adjacency matrix in the propagation rule, GCNs distribute feature information of a node to its neighbors.

The GCN encoder has been considered in learning methods like Graph Autoencoders and Deep Graph Infomax. Other variations of node encoders that can also be considered in unsupervised learning methods, recognize the propagation rule of GCNs as a special case of a message-passing network, as shown by Gilmer et al. (2017), or include additional parameters to model complex interactions between nodes with an attention mechanism (Bahdanau et al., 2015), as in the Graph Attention Network (Veličković et al., 2018a).

The Simplified GCN (SGC), a variant recently proposed by Wu et al. (2019a), argues that k-layer GCNs can be simplified by removing all nonlinearities and using fewer parameters, while preserving the k-hop neighborhood aggregation via powers of the normalized adjacency matrix:

$$\mathbf{H} = \mathrm{SGC}(\mathbf{H}^{(l)}, \mathbf{A}) = \left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)^{k} \mathbf{X}\mathbf{W} = \tilde{\mathbf{A}}\mathbf{X}\mathbf{W} \tag{2.13}$$

2.3.3 Node representations

Node embeddings can be interpreted as continuous representations in a Euclidean space, where a measure of similarity between two nodes can be obtained as the distance between two embeddings in $\mathbb{R}^D$ using the ℓ_2-norm, or based on the inner product between them. This is the case for many algorithms where a node is mapped deterministically to a single vector, such as DeepWalk, GAE, and DGI.

The VGAE model also shows that embeddings can be interpreted as latent variables, sampled from a posterior distribution parameterized by the encoder. As shown in the previous section, this method can assume a Gaussian or a Hyperspherical latent space, and it also admits recent extensions to the VAE framework that add more flexibility to the latent space, such as planar or radial normalizing flows (Rezende & Mohamed, 2015), and Sylvester normalizing flows (van den Berg et al., 2018).

G2G shows that embeddings can alternatively define a Gaussian probability distribution directly, so that a node is represented by a particular set of distribution

parameters. For downstream tasks, however, usually only the mean vector is used as the node embedding.

An additional extension in the direction of representations that diverts from embeddings as single points, considers embedding nodes as discrete probability distributions, or *point clouds* (Frogner et al., 2019). This representation has the advantage of distributing the embedding of a node across different points in the space, allowing for multiple modes that capture different aspects of a node in the representation.

2.3.4 Scoring functions

During training, unsupervised methods make use of a scoring function that is used to evaluate pairs of embeddings. This function can be designed to assign a high score to pairs that belong together, and a low score otherwise.

The choice of a scoring function is closely related to the embedding representation. For vector representations in a Euclidean space, the cosine of the angle between two embeddings is proportional to their inner product:

$$\cos\theta = \frac{\mathbf{z}_i^\top \mathbf{z}_j}{\|\mathbf{z}_i\|\|\mathbf{z}_i\|}. \tag{2.14}$$

Therefore, two nodes that are similar should be close in the embedding space with an angle $\theta \approx 0$. This can be achieved by maximizing the inner product, which is used as the scoring function in DeepWalk, GAE, and VGAE, in addition to the sigmoid function to map the score to the interval $(0, 1)$.

DGI, on the other hand, scores a pair consisting of a node embedding $\mathbf{z}_i$ and a global summary embedding $\mathbf{s}$ via a bilinear product followed by a sigmoid. For G2G, the embedding representation motivates the use of the Kullback-Leibler divergence as the scoring function.

Other embedding representations also allow for different scoring functions. A scoring function for the point cloud representation can be obtained by measuring the *Wasserstein distance* between the distributions defined by the point clouds, which measures the cost of moving all the probability mass from one distribution to another, given a distance function. The point cloud representation will be treated in detail in the next chapter.

2.3.5 Loss functions

Unsupervised methods based on the contrastive method are optimized to maximize a score for positive samples, and minimize it for negative samples. Given a score S for a positive sample, and $\tilde{S}$ for a negative sample, a suitable loss function to minimize for each sample pair is the following:

$$\mathcal{L} = -\log S - \log(1 - \tilde{S}) \tag{2.15}$$

This loss function can be seen as the binary cross-entropy loss for a model of the probability of a certain event y, and is used in DeepWalk, GAE, and DGI. Through negative sampling, DeepWalk uses this loss while modeling the probability that a node appears in a random walk through a node of interest. GAE models the probability of two nodes being linked, and DGI models the probability of whether a certain node embedding is related to a global summary vector, or that it comes from a corrupted version of the graph.

In the energy-based learning approach, the objective consists of minimizing the energy E for positive samples, and maximizing the energy $\tilde{E}$ for negative samples (LeCun et al., 2006). In G2G, the KL divergence is used to measure the energy between pairs of node representations. This energy is then used in a square exponential loss that penalizes energies for negative samples with exponentially decreasing force:

$$\mathcal{L} = E^2 + e^{-\tilde{E}} \tag{2.16}$$

Other losses have been proposed in the literature (LeCun & Huang, 2005; LeCun et al., 2006), and pose alternatives for experimentation in graph representation learning (see figure 2.3). Examples include the hinge loss, defined as

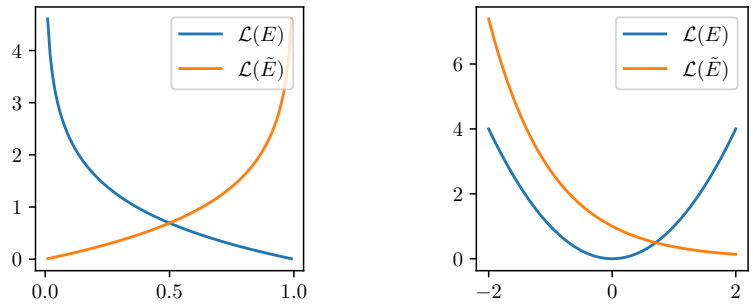
$$\mathcal{L} = \max\left(0, m + E - \tilde{E}\right), \tag{2.17}$$

The hinge loss penalizes differences between the energies of the positive and negative pairs larger than $-m$ (where m is a *margin* hyperparameter), and thus it does not favor any absolute value for each energy term.

A second alternative is the square-square loss, defined as

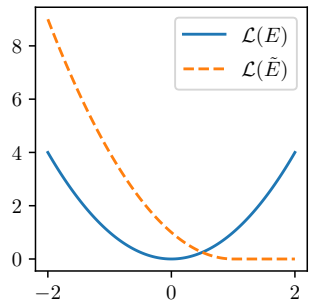
$$\mathcal{L} = E^2 + \left(\max(0, m - \tilde{E})\right)^2. \tag{2.18}$$

When this loss is minimized, the energy of the positive samples is minimized, favoring values of zero, and the energy of the negatives is encouraged to be equal to or larger than the margin m.

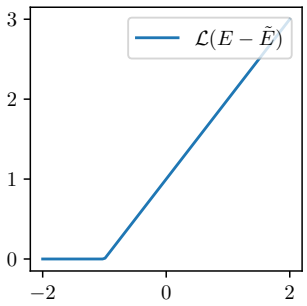


(a) Binary cross-entropy

(b) Square-exponential



(c) Square-square



(d) Hinge

Figure 2.3: Loss functions for unsupervised learning. Some losses treat the scores due to positive and negative samples separately (denoted here as $\mathcal{L}(E)$ and $\mathcal{L}(\tilde{E})$, respectively), while the hinge loss depends on the difference in the scores.

As shown by LeCun et al. (2006), the loss functions described in this section guarantee that minimization will find values where the energy is low for positive samples and high for negative samples. In the context of graph representation learning, this is related to a precise definition of the scoring function, which can vary across methods. In DeepWalk, GAE, and DGI the inner product is followed by a sigmoid that yields a score in the unit interval, which is used in the binary cross-entropy loss. Alternatively, we can use the hinge loss with the *negative* inner product as the energy. For a positive pair of nodes (v_i, v_j) and a negative pair (v_i, v_k), the loss becomes

$$\begin{aligned} \mathcal{L} &= \max(0, m + (-\mathbf{z}_i^\top \mathbf{z}_j) - (-\mathbf{z}_i^\top \mathbf{z}_k)) \\ &= \max(0, m + \mathbf{z}_i^\top \mathbf{z}_k - \mathbf{z}_i^\top \mathbf{z}_j) \end{aligned} \tag{2.19}$$

This loss penalizes the case where $\mathbf{z}_i^\top \mathbf{z}_k > \mathbf{z}_i^\top \mathbf{z}_j - m$, thus encouraging embeddings where the inner product for negative samples is lower than the inner product for positive samples by a margin of m, and it has been used in methods for graphs with

multiple relations between nodes (Yang et al., 2015a).

An attempt to use the negative inner product in a loss such as the square-exponential would not be as successful, as we obtain

$$\mathcal{L} = (-\mathbf{z}_i^\top \mathbf{z}_j)^2 + e^{\mathbf{z}_i^\top \mathbf{z}_k} \tag{2.20}$$

Overall, this loss penalizes the magnitude of any inner product, which is not beneficial for positive samples. On the other hand, it does so exponentially for the negative samples, and quadratically for positive samples, a difference that could allow learning useful embeddings. We can gain more insight by examining the gradient of equation 2.20 with respect to the model parameters $\boldsymbol{\theta}$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = -2\mathbf{z}_i^\top \mathbf{z}_j \frac{\partial(\mathbf{z}_i^\top \mathbf{z}_j)}{\partial \boldsymbol{\theta}} + e^{\mathbf{z}_i^\top \mathbf{z}_k} \frac{\partial(\mathbf{z}_i^\top \mathbf{z}_k)}{\partial \boldsymbol{\theta}} \tag{2.21}$$

We note that in the first term the sign of the gradient could be reversed during training, depending on the sign of the inner product, therefore the use of this loss could lead to problems in convergence when using SGD. We can conclude that that it is possible to exchange the loss function in methods for representation learning to learn embeddings with certain favorable properties, as long as they are coupled with an appropriate scoring function.

In many cases, minimizing the loss function can cause the norm of the embeddings to grow without bound. This is the case when the inner product is used in the scoring function. As the norm increases, the loss will decrease accordingly until overfitting occurs. For this reason, it is often necessary to include regularization in methods for graph representation learning. Techniques used in the literature include weight decay (Yang et al., 2015a), the KL divergence as used in variational approaches like VGAE, and early stopping, used in DGI and G2G. Even though early stopping does not add an explicit term to the loss, it effectively reduces the parameter space to a neighborhood around the initial value, which has a regularizing effect (Bishop, 1995; Sjöberg & Ljung, 1995).

2.4 Conclusion

We have listed some of the existing methods in the literature for unsupervised graph representation learning, and we have described them under a modular framework where each component is flexible in terms of possible variations. In this view, existing methods can be seen as a particular choice for each of the components, as we show in table 2.1. We find that this choice is evaluated as a whole, while an assessment

Table 2.1: Components of unsupervised learning methods on graphs in existing algorithms: DeepWalk (Perozzi et al., 2014), GAE (Kipf & Welling, 2016b), $\mathcal{S}$-VGAE (Davidson et al., 2018), DGI (Veličković et al., 2018b), and G2G (Bojchevski & Günnemann, 2018). vMF($\mathbf{z}$) corresponds to the von Mises Fisher distribution.

Method	**Encoder**	**Representation**	**Score**	**Loss**	**Sampling**
DeepWalk	LUT	$\mathbf{z}_i \in \mathbb{R}^D$	$\sigma(\mathbf{z}_i^\top \mathbf{z}_j)$	$-\log S - \log(1-\tilde{S})$	(+) random walk neighbors (-) non-neighbors
GAE	GCN	$\mathbf{z}_i \in \mathbb{R}^D$	$\sigma(\mathbf{z}_i^\top \mathbf{z}_j)$	$-\log S - \log(1-\tilde{S})$	(+) 1st order neighbors (-) non-neighbors
$\mathcal{S}$-VGAE	GCN	$\mathbf{z}_i \sim \text{vMF}(\mathbf{z})$	$\sigma(\mathbf{z}_i^\top \mathbf{z}_j)$	$-\log S - \log(1-\tilde{S})$	(+) 1st order neighbors (-) non-neighbors
DGI	GCN	$\mathbf{z}_i \in \mathbb{R}^D$	$\sigma\left(\mathbf{z}_i^T \mathbf{W} \mathbf{s}\right)$	$-\log S - \log(1-\tilde{S})$	(+) original graph (-) corrupted graph
G2G	MLP	$\mathbf{z}_{\mu i} \in \mathbb{R}^D$ $\mathbf{z}_{\sigma i} \in \mathbb{R}^D$	$\text{KL}(\mathcal{N}_i \| \mathcal{N}_j)$	$S^2 + \exp(-\tilde{S})$	(+) n order neighbors (-) $n+1$ order neighbors

of the effect of individual elements is often missing. These results thus motivate a series of questions:

- Given an existing method, what is the effect of changes in its components?
- Can we devise novel methods by leveraging the framework and including related results in the literature, such as the use of Wasserstein distances in scoring functions?
- What are the experimental advantages and limitations of this variations in downstream tasks like node classification and link prediction?

These questions will be addressed in the following chapters.

Chapter 3

Graph Wasserstein Embeddings

The most common approach in methods for graph representation learning is to embed each node in the graph as a single point in Euclidean space. This representation is convenient, as it allows the use of simple scoring functions such as the inner product, or the KL divergence, which has a closed form for distributions like the Gaussian. However, this representation causes all the information captured by an embedding to be concentrated in a specific region of the space, which prompts us to question the effectiveness of such representation to capture aspects that occur naturally in data, such as uncertainty and multimodal distributions.

A similar remark has been made in the related area of word representation learning, where words can have multiple meanings, depending on the context, that point embeddings can fail to learn. Li & Jurafsky (2015) propose to use a Chinese Restaurant Process (Blei et al., 2003) to find word embeddings with multiple senses, although they find that their approach can be easily matched by a single point embedding with a comparable number of parameters. Other methods that capture word meaning uncertainty have been proposed, by embedding words as Gaussian distributions (Vilnis & McCallum, 2015), or using a VAE to find the posterior distribution of a word embedding, given its context (Brazinskas et al., 2018). These methods often result in increased performance, which motivates the application of similar approaches for graph representation learning.

Among the methods that we have reviewed, G2G and VGAE provide a mechanism towards embeddings that capture uncertainty, by representing nodes with the mean and covariance of a Gaussian distribution. Ideally, learning the covariance allows to capture uncertainty when representing a node, according to what is observed in the data. Bojchevski & Günnemann (2018) show that in G2G, the learned covariance is correlated with the class diversity of the neighborhood of a node: nodes with

neighborhoods of different classes result in higher covariance, and vice versa. This is a surprising result, since class information is not used during training. In spite of these results, G2G and VGAE require a higher number of parameters in comparison to methods like DeepWalk and GAE, and more importantly, they still makes use of a single point (the mean of the distribution) for the tasks of link prediction and node classification.

A new promising direction, introduced by Frogner et al. (2019), consists of learning embeddings as discrete probability distributions, or *point clouds*, and measuring the *Wasserstein distance* beween two node embeddings, which quantifies the cost of moving all the probability mass from one distribution to the other, according to a specified cost function.

Under this approach, a node is represented as a set of points scattered throughout the space on locations learned from data, providing an increase in flexibility in comparison with point embeddings. Furthermore, Frogner et al. highlight theoretical results that enable the Wasserstein distance to preserve properties of the space being embedded, such as the shortest path distance on a graph.

In the previous chapter we introduced a modular framework for representation learning on graphs. In this chapter we propose a method that uses point clouds as the representation component, and the Wasserstein distance in the scoring function component of the framework. We introduce the theoretical and computational background of the problem, and we present preliminary results on real-world networks.

3.1 Embedding nodes as distributions

We begin by considering discrete probability distributions, which can be defined as a set of probability masses located in certain locations of a space. More formally, let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a subset of a domain space $\mathbb{R}^S$. We call $\mathcal{X}$ the *support* set, which contains n locations to which a discrete probability distribution p_X assigns nonzero mass. Let $\mathbf{a} \in \mathbb{R}^n$ be the *probability vector* containing the mass for each $x_i \in \mathcal{X}$, such that

$$\sum_{i=1}^{n} a_i = 1. \tag{3.1}$$

The distribution can then be defined as a sum of Dirac deltas located at the support points:

$$p_X = \sum_{i=1}^{N} a_i \delta_{x_i} \tag{3.2}$$

We are interested in embedding nodes as distributions of the form 3.2. In order to use this formulation in our modular framework for unsupervised representation learning, we must provide a definition of similarity between node embeddings that can be employed in the scoring component. Therefore, we need a measure of similarity between a pair of discrete probability distributions.

Let p_X be the distribution assigned to a node v_x. Similarly, let $\mathcal{Y} = \{y_1, \ldots, y_m\}$ be the support of the distribution p_Y assigned to a node v_y, and let $\mathbf{b} \in \mathbb{R}^m$ be the corresponding probability vector. In the special case where $\mathcal{X} = \mathcal{Y}$, a straightforward way to measure the similarity of the pair (v_x, v_y) is the Kullback-Leibler divergence of their associated distributions:

$$\mathrm{KL}(p_X \| p_Y) = \sum_{i=1}^{n} a_i \log \frac{a_i}{b_i} \tag{3.3}$$

As highlighted by Bojchevski & Günnemann (2018), this asymmetric divergence can be suitable for directed graphs, otherwise we can use the symmetric Jensen-Shannon divergence. However, the assumption of equal supports for both distributions is limiting, as it requires the support points for all nodes to be in the exact same locations.

3.1.1 The Wasserstein distance

We can define an alternative measure of similarity between distributions as follows: let $\mathbf{C} \in \mathbb{R}^{n \times m}$ be a *cost matrix*, where the C_{ij} entry contains the cost of moving a unit of mass from location $x_i \in \mathcal{X}$ to the location $y_i \in \mathcal{Y}$. The *optimal transport cost* is the minimum cost of transporting the masses in the support $\mathcal{X}$ towards the masses in $\mathcal{Y}$.

This problem is central in the theory of optimal transport, and its early formulation, known as the *Monge problem* (Monge, 1781), requires every point in $\mathcal{X}$ to be assigned to exactly one point of $\mathcal{Y}$, that is, probability masses cannot be split. This might not be achievable when $n \geq m$, and it is not possible when $n < m$.

The *Kantorovich relaxation* (Kantorovich, 2006) addresses this by formulating the assignment problem through a permutation matrix $\mathbf{P} \in \mathbb{R}^{n \times m}$, also known as the *coupling matrix*, where the P_{ij} entry contains the amount of mass at location x_i that is moved to the location y_i. Note that this requires that the rows of $\mathbf{P}$ add to $\mathbf{a}$, and its columns to add to $\mathbf{b}$. The optimal transport problem then consists of finding

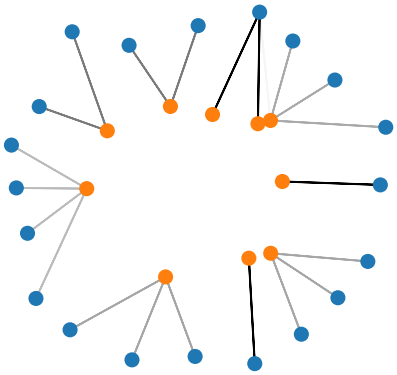


Figure 3.1: A solution of the optimal transport problem under the Kantorovich relaxation, which consists of finding the assignment that minimizes the cost of transporting the probability mass from one distribution (orange) to another (blue); shown here for a pair of distributions with support points in $\mathbb{R}^2$. The strength of the lines represents the fraction of the mass that is transported, with the darkest being a fraction of 1.

the minimum transportation cost:

$$L_{\mathbf{C}} = \min_{\mathbf{P}} \langle \mathbf{C}, \mathbf{P} \rangle = \min_{\mathbf{P}} \sum_{ij} C_{ij} P_{ij} \tag{3.4}$$

subject to $\mathbf{P}\mathbf{1}_m = \mathbf{a}$ and $\mathbf{P}^\top \mathbf{1}_n = \mathbf{b}$, where $\mathbf{1}_n$ and $\mathbf{1}_m$ are column vectors of n and m ones, respectively. An illustration of this problem is shown in figure 3.1.

The definition of the cost matrix is problem dependent, although in particular we can associate it with a metric, such as the Euclidean distance. Let c be a function defined on the domain space $\mathbb{R}^S$. c is a *distance function* or *metric* if it satisfies the following conditions for all $x, y, z \in \mathbb{R}^S$:

1. **Non-negativity:** $c(x, y) \geq 0$
2. **Simmetry:** $c(x, y) = c(y, x)$
3. **Identity of indiscernibles:** $c(x, y) = 0 \Leftrightarrow x = y$
4. **Triangle inequality:** $c(x, z) \leq c(x, y) + c(y, z)$

If we use a metric c to calculate the pairwise costs, such that $C_{ij} = c(x_i, y_j)^p$, we call c the *ground metric* and the minimum transportation cost defines the **p-Wasserstein distance** between the distributions p_X and p_Y:

$$\mathcal{W}_p(p_X, p_Y) = L_{\mathbf{C}}^{1/p} \tag{3.5}$$

Given this definition, the p-Wasserstein distance is a metric on the set of discrete probability distributions, as it satisfies the four conditions listed above (Villani, 2008), hence giving rise to the *Wasserstein space*. In contrast with the KL divergence, the Wasserstein distance does not constrain the distributions to have the same support. This is therefore a suitable metric to measure the similarity of two nodes embedded as discrete distributions.

The last issue to be addressed is the solution of the optimal transport problem: how to find the coupling matrix $\mathbf{P}$ that minimizes the transportation cost? A well known method in the optimal transport literature is the *Hungarian algorithm* (Kuhn, 1955). Its applicability for representation learning on graphs is limited, as it assumes that the distributions are uniform and have the same number n of support points, and it has complexity $O(n^3)$ (Jonker & Volgenant, 1987). Furthermore, the algorithm is not differentiable with respect to the cost matrix, so it is not suitable for optimization with SGD and backpropagation.

In the next section we describe a modification of the problem that allows to find an iterative and differentiable algorithm that makes the computation of Wasserstein distances more amenable to their use in deep learning.

3.1.2 Entropic regularization

We begin by defining the *entropy* of a matrix:

$$H(\mathbf{P}) = -\sum_{ij} \mathbf{P}_{ij} \log \mathbf{P}_{ij}. \tag{3.6}$$

A matrix with a low entropy will be sparser, with most of its non-zero values concentrated in a few points. Conversely, a matrix with high entropy will be smoother, with the maximum entropy achieved with a uniform distribution of values across its elements. The optimal transport problem can be modified by including the entropy of the coupling matrix in the objective:

$$\begin{aligned} L_{\mathbf{C}}^{(\varepsilon)} &= \min_{\mathbf{P}} \langle \mathbf{C}, \mathbf{P} \rangle - \varepsilon H(\mathbf{P}) \\ \text{subject to} \quad \mathbf{P}\mathbf{1}_n &= \mathbf{a} \\ \mathbf{P}^\top \mathbf{1}_m &= \mathbf{b} \end{aligned} \tag{3.7}$$

where ε is a regularization coefficient. The entropic version of the problem has a number of important properties. First, it is convex, so it has a unique minimum. Second, its unique solution converges to the solution of the original formulation

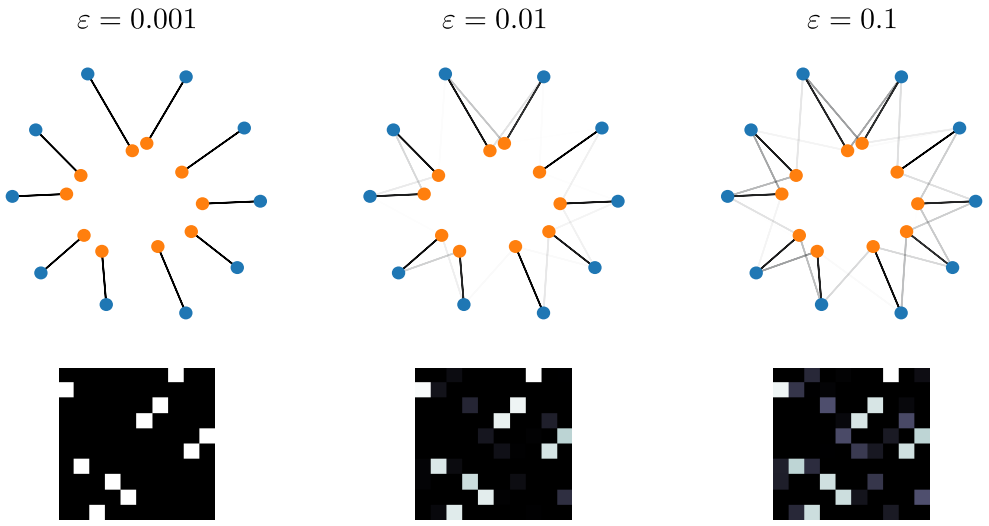


Figure 3.2: Solutions of the regularized optimal transport problem for different values of ε. The associated permutation matrix is shown at the bottom, with brightness depicting the magnitude of the entries. As ε goes to zero, the solution approaches the solution of the unregularized problem. As it increases, the assignments become more diffuse and the entropy of the permutation matrix increases.

as $\varepsilon \to 0$ (Peyré & Cuturi, 2019), as we illustrate in figure 3.2. Lastly, it can be solved using a sequence of differentiable operations that converge to the optimal permutation matrix.

The unique solution to problem 3.7 is of the form $\mathbf{P} = \text{diag}(\mathbf{u})\mathbf{K}\text{diag}(\mathbf{v})$, where $\mathbf{K}$, also known as the *Gibbs kernel*, has entries $K_{ij} = \exp(-C_{ij}/\varepsilon)$. From the constraints on the rows and columns of the permutation matrix, we find that the following two conditions must hold:

$$\mathbf{u} \odot (\mathbf{K}\mathbf{v}) = \mathbf{a} \quad \text{and} \quad \mathbf{v} \odot (\mathbf{K}^\top \mathbf{u}) = \mathbf{b} \tag{3.8}$$

where $\odot$ is the element-wise product.

The Sinkhorn algorithm for solving the regularized optimal transport problem (Yule, 1912; Sinkhorn, 1964; Cuturi, 2013) consists on initializing $\mathbf{u}$ and $\mathbf{v}$ with ones, and iteratively updating them using the conditions 3.8:

$$\mathbf{u}^{(k+1)} = \frac{\mathbf{a}}{\mathbf{K}\mathbf{v}^{(k)}} \tag{3.9}$$

$$\mathbf{v}^{(k+1)} = \frac{\mathbf{b}}{\mathbf{K}^\top \mathbf{u}^{(k+1)}} \tag{3.10}$$

where the divisions are element-wise. These sequential updates are also known as the *Sinkhorn iterations*, which have been shown to converge in a finite number of steps (Franklin & Lorenz, 1989).

Algorithm 3.1 Wasserstein distance between distributions p_X with support $\mathcal{X} = \{x_1, \dots, x_n\}$ and probability vector $\mathbf{a}$, and p_Y with support $\mathcal{Y} = \{y_1, \dots, y_n\}$ and probability vector $\mathbf{b}$.

Inputs:

p_X, p_Y

Regularization coefficient $\varepsilon > 0$

metric $c : \mathcal{X} \times \mathcal{Y} \to [0, \infty)$

Compute cost $\mathbf{C}$: $C_{ij} = c(x_i, y_j)^p \ \forall x_i \in \mathcal{X}, y_j \in \mathcal{Y}$

Compute kernel $\mathbf{K}$: $K_{ij} = \exp(-C_{ij})$

$\mathbf{u} \leftarrow \mathbf{1}_n$

$\mathbf{v} \leftarrow \mathbf{1}_m$

while not converged **do**

  $\mathbf{u} \leftarrow \mathbf{a}/\mathbf{K}\mathbf{v}$

  $\mathbf{v} \leftarrow \mathbf{b}/\mathbf{K}^\top\mathbf{u}$

end while

$\mathbf{P} \leftarrow \text{diag}(\mathbf{u})\mathbf{K}\text{diag}(\mathbf{v})$

return $\langle \mathbf{P}, \mathbf{C} \rangle^{1/p}$

An important aspect of the Sinkhorn iterations is the relation between the coefficient ε and the number of iterations required for convergence. Assuming the same number n of support points, for a fixed coefficient $\varepsilon = \frac{4\log(n)}{\tau}$ and $\tau > 0$, the iterations converge to a permutation matrix $\mathbf{P}$ such that

$$\langle \mathbf{P}, \mathbf{C} \rangle \leq L_{\mathbf{C}} + \tau$$

in $O(n^2 \log(n) \tau^{-3})$ iterations (Altschuler et al., 2017). This means that i) the transportation cost that the Sinkhorn iterations yields approximates the unregularized optimal transport cost, with an error of up to τ, and ii) the number of iterations required to guarantee this is inversely proportional to the error τ in the approximation.

The Sinkhorn algorithm thus serves as a method to calculate an approximation of the Wasserstein distance, which is also known as the *Sinkhorn* distance, and we denote as $\mathcal{W}_p^{(\varepsilon)}(p_X, p_Y)$. We outline it in algorithm 3.1.

We can see that the Sinkhorn algorithm consists of a sequence of linear, differentiable operations, so we can use it as a module in a deep learning architecture that takes as input a pair of supports and probability vectors, and outputs the Wasserstein distance. This, together with the increased flexibility of Wasserstein spaces, has motivated its applications to problems in generative modeling (Genevay et al., 2018; Ambrogioni et al., 2018) and metric embedding (Frogner et al., 2019).

In their work, Frogner et al. (2019) explore learning embeddings on Wasserstein spaces of the kind we have discussed in this section. In particular, they propose to use them to embed nodes in graphs, so that the geometry in the Wasserstein space preserves the properties of the graph. Following this motivation, we explore the use of such spaces in the scoring and representation components of the framework for graph representation learning outlined in the previous chapter.

3.2 The point cloud representation

We will now consider two specific choices in the components of our framework for graph representation learning:

- **Representation:** Given the output $\mathbf{h}^L \in \mathbb{R}^D$ of the node encoder, we will represent nodes as discrete, uniform probability distributions with n support points in $\mathbb{R}^S$, such that $n \times S = D$. Even though the probability vector $\mathbf{a}$ could be learned as well, we choose to use a uniform distribution with $a_i = \frac{1}{n}$, as learning the weights has shown not to yield improved results (Frogner et al., 2019; Kloeckner, 2012; Claici et al., 2018). This means that the output of the node encoder is interpreted as the locations of the support points. We call this the *point cloud* representation.

- **Scoring:** Similarly as in Graph2Gauss, where the scoring function uses the KL divergence between the distributions, we can use the 1-Wasserstein distance. Given a pair of nodes v_x and v_y embedded as point clouds p_X and p_Y, respectively, the scoring function is given in terms of $\mathcal{W}_1^{(\varepsilon)}(p_X, p_Y)$, which is calculated with the Sinkhorn algorithm to enable end-to-end learning with SGD. As a distance-based score, this function is compatible with *generalized margin losses* (LeCun et al., 2006), which include the hinge, square-square, and square-exponential losses. Minimizing these losses when using the Wasserstein distance will therefore yield distributions that are close, in the optimal transport sense, for positive samples, and distant for negative samples.

The point cloud representation offers an advantage with respect to point embeddings due to its ability to represent multiple modes, without committing to parametric distributions restricted by the tractability of computations (as is the case of the Gaussian in the VAE, for example). These intuitive benefits have been supported by formal results on related works that explore the *representational capacity* of Wasserstein spaces.

We can make this notion precise by defining an embedding as a map $\phi : \mathcal{A} \to \mathcal{B}$ between two metric spaces $(\mathcal{A}, d_{\mathcal{A}})$ and $(\mathcal{B}, d_{\mathcal{B}})$, where $d_{\mathcal{A}}$ and $d_{\mathcal{B}}$ are metrics on $\mathcal{A}$ and $\mathcal{B}$, respectively. We also call $\mathcal{B}$ the *target* or *embedding* space. A space with large representational capacity preserves distances of objects that are embedded into it, which is measured as the distortion in the distance between two objects in the embedding space, compared to the distance in the original space (Frogner et al., 2019). More formally, for a pair (u, v) in the original space, and for $L > 0$ and $C \geq 1$, the distortion of the embedding is the smallest C such that the following holds:

$$Ld_{\mathcal{A}}(u, v) \leq d_{\mathcal{B}}(\phi(u), \phi(v)) \leq CLd_{\mathcal{A}}(u, v) \tag{3.11}$$

Previous work has shown that Wasserstein spaces can embed a wide variety of spaces with low distortion. This includes ℓ^1 (the space of absolutely convergent series) (Bourgain, 1986), and finite metric spaces on $\mathbb{R}^3$ (Andoni et al., 2018). Based on these results, Frogner et al. (2019) propose to use Wasserstein spaces to embed nodes in a graph with the shortest-path distance as the original metric, and they show that low distortion can be achieved for synthetic and small networks. In the next section we present preliminary results on the use of Wasserstein spaces to embed nodes in real-world networks.

3.3 Preliminary results

In order to verify experimentally the distortion properties of Wasserstein embeddings, we train embeddings that minimize the distortion of the shortest-path metric on a graph. As node encoder we use a two-layer GCN with 128 output units. We use these to parameterize the locations of the support points. Given a fixed number of output units, we change the number of support points, which effectively changes the dimension of the support space. In particular, we run experiments with 1 point in $\mathbb{R}^{128}$, 2 in $\mathbb{R}^{64}$, and 8 in $\mathbb{R}^{16}$. For the ground metric we use the L^2 distance. Note that in the case of 1 point in $\mathbb{R}^{128}$, the Wasserstein distance in this case is equal to the L^2 distance between point embeddings.

For each node v_x in the graph we randomly sample a node v_y within its k-neighborhood, where k is a hyperparameter that we set to 10. We then minimize the distortion loss:

$$\frac{|\mathcal{W}_1^{(\varepsilon)}(p_X, p_Y) - d_{\mathcal{G}}(v_x, v_y)|}{d_{\mathcal{G}}(v_x, v_y)} \tag{3.12}$$

where $d_{\mathcal{G}}(v_x, v_y)$ is the shortest-path distance between v_x and v_y. We calculate the mean loss for all nodes and minimize it with Adam (Kingma & Ba, 2015), with a

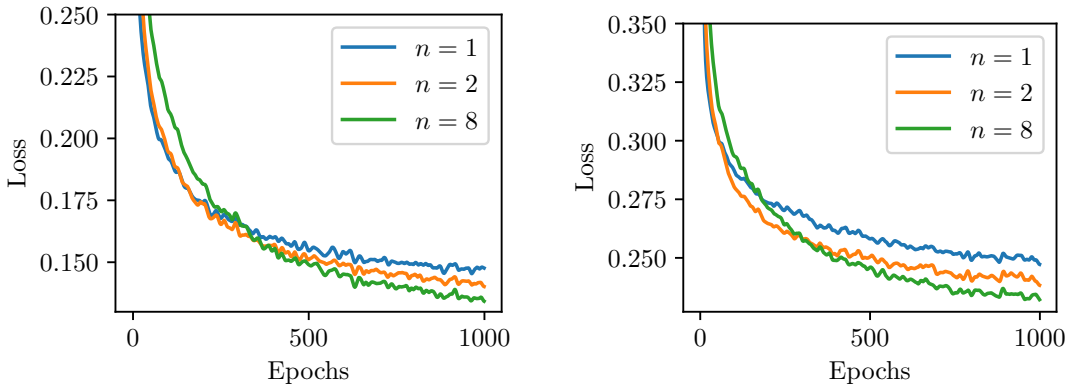


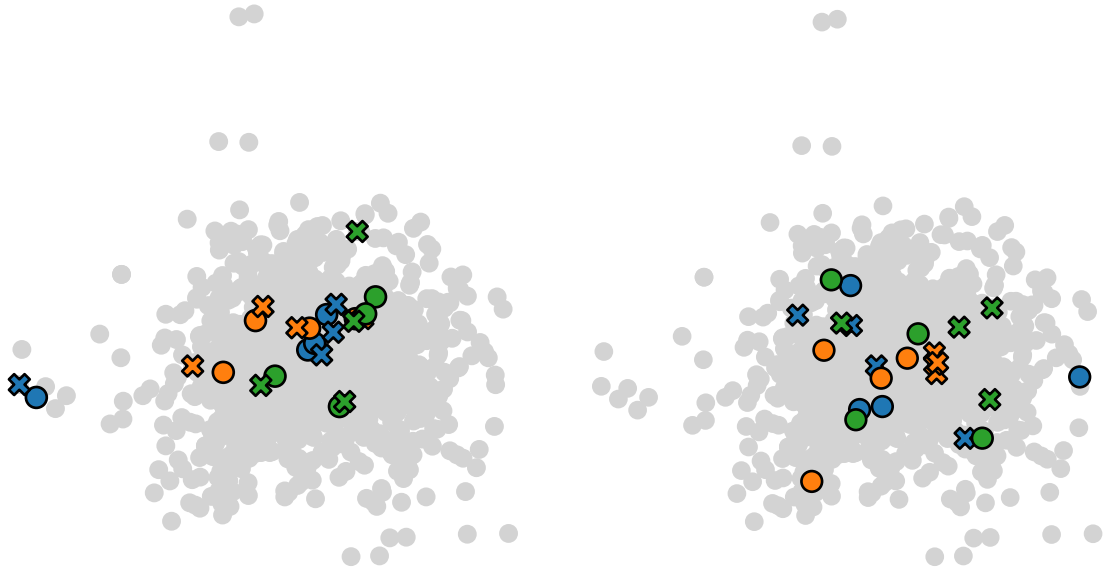
Figure 3.3: Mean distortion when embedding the shortest-path distance on the Wasserstein space, with varying number of support points n. Results shown for Cora (left) and Citeseer (right).

learning rate of 0.01 for 1000 epochs. We train the embeddings using the Cora and Citeseer datasets (Sen et al., 2008), which correspond to citation networks where nodes represent publications, and edges are present if a document cites another. Cora and Citeseer have 2,708 and 3,327 nodes, respectively.

We show the loss curves for both graphs in figure 3.3. These curves confirm the results on small networks shown by Frogner et al. (2019), and demonstrate the feasibility of using Wasserstein spaces to embed nodes while preserving the graph structure, such as shortest-path distances. We also observe the advantage of using point clouds ($n > 1$), as opposed to point embeddings ($n = 1$), as the former achieves lower embedding distortion, highlighting the potential of Wasserstein spaces to embed other metric spaces on graphs.

It is interesting to note that even when the encoder outputs 128 units, Wasserstein embeddings can be visualized in the plane without any dimensionality reduction techniques, when the support space is $\mathbb{R}^2$. This allows us to evaluate the embeddings qualitatively to verify their properties for unsupervised learning on graphs. With this aim, we train embeddings for the Cora dataset using the same settings as before, except for the sampling strategy, for which we use the first-neighbors sampling of GAE. This strategy uses 1-hop neighborhoods as positive samples, and any other non-neighbor nodes as negative samples. This should produce point clouds that are close for linked nodes in the graph, and separated otherwise.

For visualization, we sample 100 nodes at random and we plot their support on the plane. We then select 3 pairs, where for each pair the first support is drawn with a cross, and the second with a circle. We visualize these pairs for positive and



(a) Positive pairs (b) Negative pairs

Figure 3.4: Wasserstein embeddings for positive and negative samples. Pairs (p_X, p_Y) are shown with the same color, with the support of p_X shown with crosses, and the support of p_Y with circles. Gray circles show supports for other nodes in the graph. In this example, nodes are embedded as discrete distributions in $\mathbb{R}^2$ with 4 support points.

negative samples in figure 3.4. The visualization shows that point clouds for positive pairs are effectively pushed closer, whereas point clouds are separated for negative pairs. Hence we can see that Wasserstein embeddings are able to capture information on the graph, which in this case corresponds to first neighborhood structure.

3.4 Conclusion

The prospect of higher flexibility of Wasserstein spaces, and theoretical and experimental results on their capacity, have motivated us to introduce a representation and a scoring function component for experimentation in our modular framework. Preliminary results show that these spaces allow for low-distortion embeddings that preserve information present in the structure of the graph, such as shortest paths and 1-hop neighborhoods. We have yet to evaluate how this generalizes to downstream tasks, such as node classification and link prediction. We will address this aspect in the next chapter.

Chapter 4

Experiments

The proposed modular framework motivated a series of questions regarding the effect of changes in the components of existing methods, when the embeddings are used in downstream tasks. This type of analysis is not present in related work, and as noted by Shchur et al. (2018), the evaluation of machine learning models on graphs can be negatively affected due to different training procedures, and the repeated use of fixed splits and small datasets that give a biased estimate of generalization.

We address these issues by running experiments that test such changes, which allows us to obtain insights on the properties of existing methods, and novel variants, when used for link prediction and node classification on real-world networks. We then evaluate the use of Wasserstein spaces for these tasks. Following the results of Shchur et al. (2018), we propose a consistent and reproducible evaluation framework with randomized splits, and uniform hyperparameter search and computational budgets across different models.

The implementation of our modular framework is released as an open source library for representation learning on graphs, together with the code to reproduce our experiments[1].

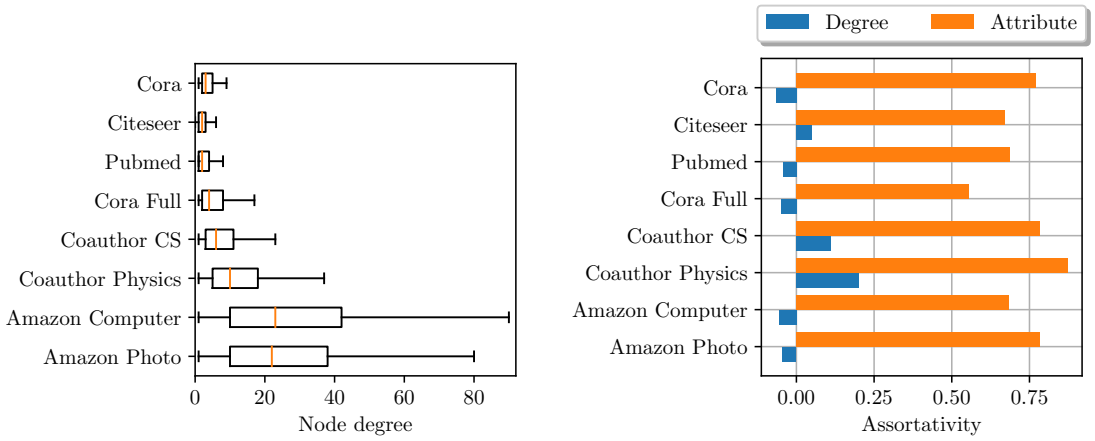
4.1 Datasets

We make use of standard datasets of different sizes, that have been used in the literature to evaluate the performance of machine learning algorithms on graphs. The first group of datasets is comprised by Cora, Citeseer (Sen et al., 2008), and Pubmed (Namata et al., 2012). These are citation networks that represent documents as nodes

[1] https://github.com/dfdazac/graph-learn

Table 4.1: Statistics for different datasets used for the experiments.

Dataset	Classes	Nodes	Features	Edges
Cora	7	2,708	1,433	5,278
Citeseer	6	3,327	3,703	4,552
Pubmed	3	19,717	500	44,324
Cora Full	67	18,703	8,710	62,421
Coauthor CS	15	18,333	6,805	81,894
Coauthor Physics	5	34,493	8,415	247,962
Amazon Computers	10	13,381	767	245,778
Amazon Photo	8	7,487	745	119,043



(a) (b)

Figure 4.1: Properties of the datasets used in the experiments, showing (a) average node degree, and (b) degree and attribute assortativity.

and citations between them as edges. Each node has an associated feature vector that corresponds to a bag-of-words representation of the contents of a document, and a label denoting a topic. We also consider Cora-Full, an extension of the Cora dataset.

Furthermore, we consider recently proposed datasets for the evaluation of models that operate on graphs (Shchur et al., 2018), which contain bigger networks than the ones described before. In the Amazon Computers and Amazon Photo graphs, nodes represent goods and edges link products bought together. Each node is assigned a feature vector with bag-of-words reviews and a label showing a category. The Coauthor CS and Coauthor Physics datasets are co-authorship networks, where each node represents an author, and authors are linked if they have co-authored a paper.

In this case features correspond to publication keywords, and labels represent a field of study.

Dataset statistics can be found in table 4.1. We additionally show plots that illustrate properties of the different graphs. In figure 4.1a we show the average node degree, which is computed by counting the total number of edges connected to a node, and dividing by the total number of nodes. This diagram shows that the datasets cover a wide range of node degrees, which is of interest to evaluate the performance of our methods under different conditions.

We can obtain more information about the properties of the graphs by using the node *assortativity*, which measures the average correlation between connected nodes acording to a certain feature (Newman, 2003). We show its values for the datasets in figure 4.1b. The *degree assortativity* measures the correlation according to node degree, so that if the correlation is high, nodes with a large number of edges are interconnected. We note that even though Cora, Citeseer, and Pubmed are all citation networks, Citeseer exhibits positive degree assortativity, suggesting that it contains particular characteristics that differentiate it from the rest. Newman (2003) shows empirically that technological networks (e.g. documents in the World Wide Web, or software dependencies) tend to have negative degree assortativity, whereas social networks have positive degree assortativity. This is also the case for the Coauthor networks, which represent authors and academic interactions among them.

The *attribute assortativity* measures the correlation in terms of classes. When it is positive, nodes of the same class are highly interconnected, otherwise nodes tend to connect to nodes of different classes. This corresponds to the concept of homophily described in chapter 2. From figure 4.1b, we observe that all datasets have positive attribute assortativity, which shows that all the networks conform to the homophily hypothesis.

4.2 Evaluation

In order to asses the quality of the learned embeddings, we evaluate their use in two tasks that are common in the literature of graph representation learning: link prediction and node classification. In the link prediction task we are interested in predicting whether there is an edge between a pair of nodes, given their embeddings. The node classification task uses the embedding of a node to predict a label. This is achieved by training a simple classifier, such as a logistic regression model, that

takes as input the embedding.

To train embeddings for each of these tasks, the data preparation is different as we outline next.

Link prediction Since some encoders, such as GCN and SGC, make use of the graph structure to encode a node, it is important to guarantee that the embeddings are trained with a disjoint list of edges for the training and test splits. To achieve this, we randomly sample a number of negative edges equal to the number of edges in the graph. We then select 85%, 5%, and 10% from these lists to create training, validation, and test splits, respectively.

For a given unsupervised learning method, we use its scoring component to obtain a score for pairs of embeddings. Given a list of predicted scores and the ground truth (i.e. an edge exists or not), the precision-recall curve consists of points (P_i, R_i) with the precision P_i and recall R_i obtained at different values of a threshold, which is applied to the score to predict an edge. To calculate a summary of this curve, we report the average precision:

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n, \tag{4.1}$$

where n is the number of threshold values used to construct the curve.

Node classification For this task we train the models without removing any edges of the graph, except for GAE, which is trained in the same way as for the link prediction task. However, unlike in the link prediction case, we randomly sample negative samples at each epoch to train the GAE.

We employ the learned node embeddings to train a logistic regression classifier with 3-fold stratified cross-validation using 10% of the labeled nodes, for a maximum of 300 iterations. We report the accuracy on the remaining 90% of the data.

4.3 Experiments

In all our experiments, the embedding dimension is 128. When using MLP and GCN encoders, we use two layers with 256 and 128 units, respectively. Given its formulation, models with the SGC encoder only have a single layer with 128 output units. For the nonlinearities we use the ReLU activation function (He et al., 2015). We train the models for 200 epochs, using the Adam optimizer (Kingma & Ba, 2015). We select the best learning rate based on the performance on the validation set.

As observed by Shchur et al. (2018), results of representation learning methods on graphs can vary significantly depending on specific splits for training and evaluation. To improve our estimates on performance we run every experiment 20 times, and report the mean and standard deviation in the results. We further guarantee that all models and baselines are trained on the same random splits. Our implementation uses PyTorch (Paszke et al., 2017) and the PyTorch Geometric library (Fey & Lenssen, 2019).

4.3.1 Comparative study

We start by comparing the existing methods that we have reviewed: DeepWalk, DGI, GAE, and G2G. We also consider the use of raw node features, which omits the encoder component. When using raw features for link prediction, we use the inner product as the score.

To evaluate the effect of changes in the components of a method, we experiment with the following variations:

- **Node encoder:** DGI and GAE are originally formulated with a GCN encoder, and we explore the use of an MLP and an SGC model in its place. For G2G we consider the original MLP encoder, and a GCN variant.

- **Representation:** G2G represents nodes as Gaussian distributions. Compared to DGI and GAE, this model requires twice the number of parameters in the last layer of the encoder, to account for the mean and variance. We experiment with a variant that discards the variance, and treats the mean as a point embedding. Instead of the KL divergence in the scoring component, we use the L^2 distance. We call this variant Graph2Vec (G2V).

Since GAE and G2G contrast direct neighbors against non-neighbors in the loss function and sampling strategies, they are better equipped for the task of link prediction than DGI. To explore how the performance of DGI can be improved in the task of link prediction, we train an edge predictor that uses the learned embeddings (which are not fine-tuned), so that the score for a pair is given by the bilinear form $\mathbf{z}_i^\top \mathbf{W} \mathbf{z}_j$. We denote this model as DGI-B.

The results of the experiments are shown in table 4.2 for link prediction, and table 4.3 for node classification. We discuss these results next.

Raw features This method does not involve any learning, so in general we expect its performance to be much worse compared to representation learning methods. We observe this for most of the cases, except for the Coauthor datasets. This can be explained by noting that node features in these networks are *keywords*, which are usually concise and explanatory, unlike the *bag-of-words* features used in the rest of the networks, which are formed by fixing a vocabulary depending on word frequency. In this sense, it is more likely for two connected nodes in the Coauthor datasets to have very similar keywords, so that for link prediction the value of the inner product for connected nodes will be higher on average than for other datasets. This effect is increased by the high homophily of the Coauthor datasets, which are among the top 3 networks with the highest degree and attribute assortativity, as shown in figure 4.1b. This behavior is also noticeable in node classification.

DeepWalk For this method we observe consistently good performance across datasets compared to using raw features. On link prediction, we can see a notorious decrease in performance on the Amazon datasets, which contain networks with the highest average node degree. This suggests that for the link prediction task, the random walk used by DeepWalk might not be sufficient to traverse sufficiently diverse neighborhoods for nodes with high degree. The performance on node classification is more robust against the average node degree.

DGI Using an MLP encoder results in worse performance, which is expected because the mutual information maximization on which DGI is based leverages patch representations. These representations are lost when using an MLP encoder since it only uses node features, and discards the structure of the graph. As we hypothesized, the original formulation of DGI, which uses a GCN encoder, has low link prediction performance.

We found that two different variants result in improved performance: training a link prediction model (GCN-DGI-B), and using an SGC encoder. In spite of the improvements that an additional link prediction model brings, it requires more parameters and resources for additional training. Using an SGC encoder, on the other hand, uses less parameters and can result in competitive performance for some datasets. We observed that SGC allows to obtain higher scores in the training set compared to the GCN, while also generalizing well to the test set, which we attribute to the lower number of parameters of the SGC. However, for networks with high average node degree, as in the Amazon datasets, the use of multiple layers and nonlinearities in the GCN yields better performance than the SGC.

Table 4.2: Link prediction results (average precision on the test set, in percent). For DGI, GAE, and G2G we highlight in bold the highest score for variants of the method, and the highest across all methods is underlined. We do not consider GCN-DGI-B in this comparison as it requires training an additional link prediction model.

Method	Cora	Citeseer	Pubmed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo
Raw features	76.8 ± 1.2	88.2 ± 0.9	89.6 ± 0.2	89.9 ± 0.2	92.3 ± 0.2	93.3 ± 0.1	67.4 ± 0.3	66.8 ± 0.5
DeepWalk	92.0 ± 0.6	91.3 ± 0.5	91.3 ± 0.3	96.4 ± 0.1	95.7 ± 0.2	96.1 ± 0.1	85.9 ± 0.1	88.9 ± 0.2
MLP-DGI	62.3 ± 2.2	69.1 ± 2.7	75.4 ± 0.9	66.8 ± 0.8	79.9 ± 0.8	80.9 ± 0.8	66.8 ± 0.5	66.0 ± 0.6
GCN-DGI	88.5 ± 1.2	91.7 ± 1.2	93.2 ± 0.4	89.5 ± 0.4	87.4 ± 1.3	86.7 ± 0.9	**83.2 ± 1.1**	**79.0 ± 2.0**
SGC-DGI	**92.4 ± 0.7**	**92.4 ± 1.1**	**96.2 ± 0.3**	**96.5 ± 0.2**	**87.7 ± 2.3**	**92.9 ± 0.5**	81.8 ± 0.4	76.6 ± 0.7
GCN-DGI-B	93.2 ± 0.8	93.9 ± 0.7	92.8 ± 1.0	95.8 ± 0.4	94.1 ± 1.1	96.1 ± 0.5	93.4 ± 2.0	95.0 ± 1.0
MLP-GAE	79.9 ± 1.5	83.1 ± 1.5	87.5 ± 0.6	83.7 ± 0.7	94.1 ± 0.2	94.3 ± 0.1	92.2 ± 0.2	94.2 ± 0.2
GCN-GAE	91.9 ± 0.8	92.9 ± 0.9	94.9 ± 0.2	95.4 ± 0.2	96.2 ± 0.2	97.0 ± 0.1	95.8 ± 0.2	**96.6 ± 0.1**
SGC-GAE	**93.1 ± 0.8**	**94.6 ± 0.8**	**96.0 ± 0.4**	**96.5 ± 0.2**	**96.7 ± 0.1**	**97.2 ± 0.1**	**96.1 ± 0.2**	94.7 ± 0.2
MLP-G2G	**92.7 ± 0.8**	**94.3 ± 0.7**	92.8 ± 0.3	97.7 ± 0.2	63.2 ± 0.6	71.7 ± 0.7	63.6 ± 0.3	63.6 ± 0.5
MLP-G2V	92.3 ± 0.8	93.9 ± 0.8	92.9 ± 0.3	**97.8 ± 0.1**	65.7 ± 0.4	71.5 ± 0.4	63.7 ± 0.4	63.7 ± 0.5
GCN-G2G	92.5 ± 0.9	92.6 ± 0.8	93.7 ± 0.4	97.5 ± 0.2	**96.6 ± 0.1**	**97.2 ± 0.1**	**81.7 ± 4.9**	**90.2 ± 1.1**
GCN-G2V	**92.7 ± 0.8**	92.5 ± 0.9	**94.0 ± 0.4**	97.7 ± 0.2	**96.6 ± 0.1**	**97.2 ± 0.1**	75.3 ± 2.9	86.3 ± 1.9

On node classification, the SGC instead resulted in overfitting, that is, while it preserved the high performance in the training set that was observed for link prediction, the performance on the test set is lower. With the exception of the Amazon datasets, the GCN presents better generalization properties for this task.

GAE For link prediction, this method resulted in superior performance compared to the rest of the methods. This is expected, since GAE is based on the reconstruction of the adjacency matrix. The performance of the original model with the GCN encoder is significantly improved when using an SGC in its place, except in the Amazon Photo dataset. GAE with an SGC encoder also results in improved performance on node classification in most of the cases. A notorious exception occurs with the Coauthor datasets where the MLP encoder, often yielding poor results in other datasets, gives the best performance. Similarly as in the analysis with raw features, the properties of node features for these networks seem to be sufficient to discard the graph structure *in the encoder*. It is important to note that even when using an MLP encoder, GAE still induces the structure of the graph through the sampling strategy and loss function.

Table 4.3: Node classification results (accuracy in percent). For DGI, GAE and G2G we highlight in bold the highest score for variants of the method, and the highest across all methods is underlined.

Method	Cora	Citeseer	Pubmed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo
Raw features	64.0 ± 1.1	65.9 ± 0.6	84.4 ± 0.4	53.5 ± 0.4	92.3 ± 0.3	94.9 ± 0.1	80.4 ± 0.7	87.8 ± 0.5
DeepWalk	76.7 ± 1.4	53.9 ± 1.4	80.1 ± 0.3	59.6 ± 0.3	86.9 ± 0.2	92.7 ± 0.1	87.4 ± 0.3	91.0 ± 0.4
MLP-DGI	41.1 ± 2.4	38.5 ± 2.0	70.4 ± 0.9	17.1 ± 0.6	85.1 ± 1.0	91.1 ± 0.6	59.5 ± 1.4	61.6 ± 1.4
GCN-DGI	**82.8 ± 0.9**	**70.5 ± 0.9**	**84.6 ± 0.3**	**62.8 ± 0.4**	**91.5 ± 0.2**	**95.1 ± 0.1**	88.4 ± 0.5	92.7 ± 0.3
SGC-DGI	76.3 ± 1.3	63.7 ± 1.0	82.7 ± 0.6	56.8 ± 0.4	91.1 ± 0.2	94.5 ± 0.2	**88.5 ± 0.3**	**93.1 ± 0.3**
MLP-GAE	73.4 ± 1.2	68.1 ± 0.8	79.3 ± 0.5	41.4 ± 0.9	**93.2 ± 0.2**	**95.3 ± 0.1**	83.0 ± 0.6	92.0 ± 0.4
GCN-GAE	81.7 ± 1.2	70.5 ± 0.8	**83.1 ± 0.4**	59.4 ± 0.6	91.6 ± 0.2	94.9 ± 0.1	87.9 ± 0.5	92.7 ± 0.6
SGC-GAE	**82.3 ± 0.9**	**71.4 ± 0.7**	82.0 ± 0.5	**62.9 ± 0.4**	91.7 ± 0.2	95.0 ± 0.1	**88.0 ± 0.5**	**93.1 ± 0.4**
MLP-G2G	79.7 ± 1.1	68.7 ± 0.9	84.3 ± 0.3	57.8 ± 0.3	69.4 ± 0.5	87.8 ± 0.1	49.1 ± 0.5	52.9 ± 0.7
MLP-G2V	78.7 ± 1.4	68.1 ± 0.7	**84.4 ± 0.3**	57.9 ± 0.4	64.0 ± 0.5	85.0 ± 0.2	48.1 ± 0.6	51.7 ± 0.9
GCN-G2G	**79.8 ± 1.1**	**69.0 ± 0.7**	83.8 ± 0.3	**62.0 ± 0.4**	**91.0 ± 0.2**	**94.4 ± 0.1**	**87.8 ± 0.2**	**92.0 ± 0.3**
GCN-G2V	79.2 ± 1.0	**69.0 ± 0.8**	83.9 ± 0.3	**61.9 ± 0.6**	**90.9 ± 0.2**	**94.3 ± 0.1**	**87.7 ± 0.5**	**92.0 ± 0.4**

G2G In both tasks, simplifying the node representation from Gaussian distributions measured with the KL divergence (G2G), to Euclidean embeddings measured with the L^2 norm (G2V), resulted in almost identical performance at a lower complexity, suggesting that the additional complexity of G2G is not required, and its strength instead lies in its loss function and sampling strategy. The initial formulation with an MLP encoder is competitive for small to medium sized graphs. The GCN encoder shows a similar behavior, but for larger graphs it vastly outperforms the MLP encoder, making it a reasonable default to learn embeddings with G2G, specially in large graphs where the node classification task seems to benefit from neighborhood aggregation.

Our experiments show that variants of existing methods often exhibit significantly improved performance, and graph properties can have an important effect depending on the method of choice. As observed in G2G, we find that certain modeling choices made in previous works are less important than previously believed, and that other optimization-related modeling choices are sometimes responsible for improved performance. We have thus shown that the proposed modular framework can be of practical use for devising new methods for unsupervised learning on graphs, and highlighting strengths and weaknesses of existing approaches and possible variations.

In the previous experiments we explored the effect of changes in the encoder and representation components. Instead of conditioning additional experiments on

Table 4.4: Values used for each component in the hyperparameter study.

Component	**Values**
Encoder	MLP, GCN, SGC
Representation	Euclidean, Gaussian
Scoring	Inner product, KL divergence, Euclidean distance
Loss	Binary cross-entropy, square-square, square-exponential, hinge
Sampling	First-neighbors, ranked, graph corruption

existing methods, we can evaluate new variants by exploring the space of possible combinations of components.

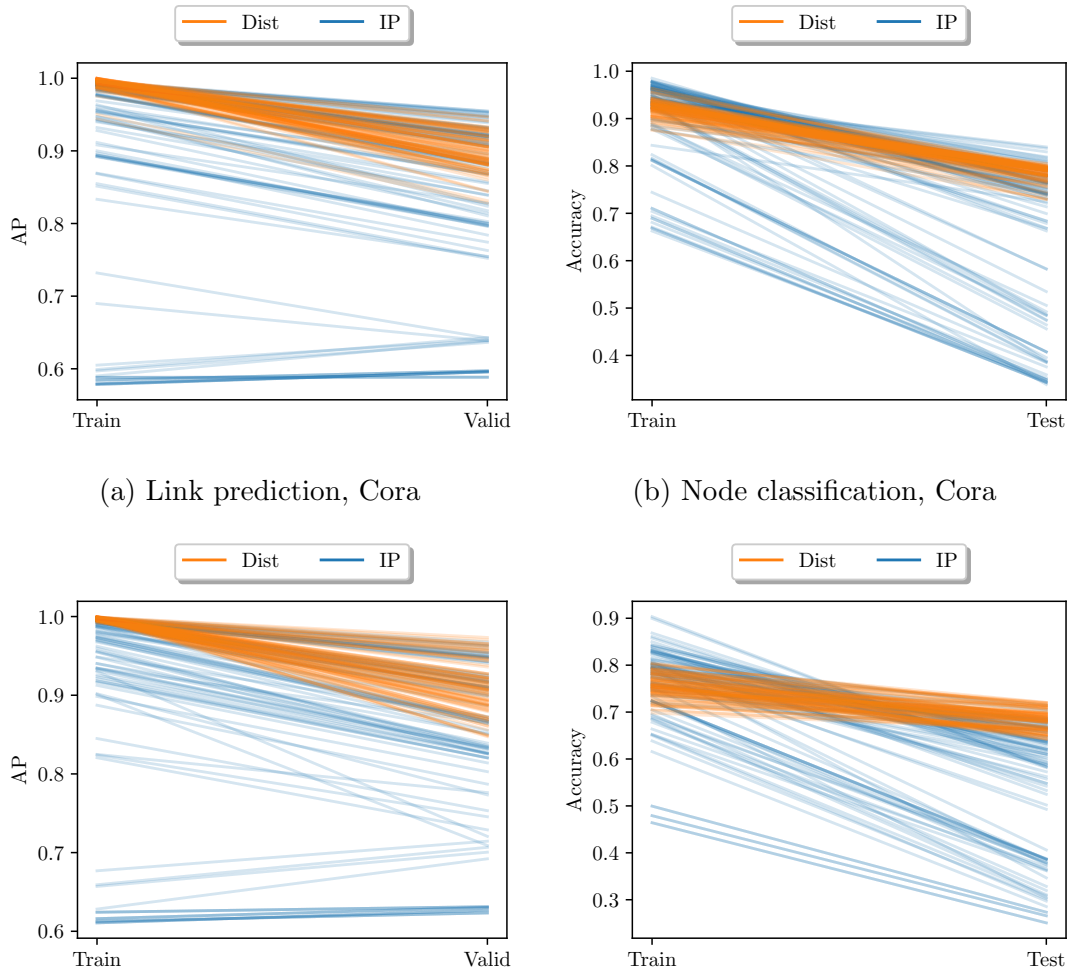
4.3.2 Hyperparameter study

Our modular framework for graph representation learning poses a natural question: given a graph, what is the best combination of components in terms of performance on link prediction and node classification? So far in our work we have identified a series of guidelines to couple components in a principled way, that result in embeddings with good predictive performance. These include an appropriate correspondence between scoring and loss functions, and the use of the SGC encoder due to its simplicity and performance in comparison with the MLP and GCN encoders.

Since the effect of components like scoring and loss functions can be more subtle and data-dependent, and it can also vary depending on the task on which the embeddings are evaluated, we propose to leverage our framework to carry out hyperparameter search across components, which allows us to evaluate the combined effect of multiple configurations.

Using the same training settings as in the previous experiments, we perform hyperparameter search separately for link prediction and node classification. The set of values for each component is shown in table 4.4, which we use to generate a grid of hyperparameters that amounts to a total of 192 configurations. Some choices in the components are constrained, e.g. the inner product score can only be used if the representation is Euclidean.

We start by running the experiments using the Cora, Citeseer, and Pubmed datasets. In figure 4.2 we show the performance on link prediction and node classification in Cora and Citeseer. Following our previous remarks, our experiments demonstrate how a particular choice of components can result in drastic differences in performance.



(a) Link prediction, Cora

(b) Node classification, Cora

(c) Link prediction, Citeseer

(d) Node classification, Citeseer

Figure 4.2: Results of the hyperparameter study for Cora and Citeseer, in link prediction and node classification. Each line represents a particular configuration of components. We use different colors for configurations based on an inner product (IP) and a distance (Dist) score. For node classification we do model selection via cross-validation, and we use the test set only to report final performance.

We note a distinction between configurations that use either the inner product or a measure of distance in their scoring functions. As highlighted in figure 4.2, the inner product score is more unstable with respect to choices of other components, namely the encoder and the loss function. The worse models use an MLP encoder and employ the square-square or square-exponential loss. This follows our analysis in chapter 2 on the compatibility between scoring and loss functions, and the lower performance of the MLP on graphs.

Table 4.5: Results on the test set for the best variants, and the best method obtained via hyperparameter search.

(a) Link prediction (average precision).

Method	Cora	Citeseer	Pubmed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo
Raw features	76.8 ± 1.2	88.2 ± 0.9	89.6 ± 0.2	89.9 ± 0.2	92.3 ± 0.2	93.3 ± 0.1	67.4 ± 0.3	66.8 ± 0.5
DeepWalk	92.0 ± 0.6	91.3 ± 0.5	91.3 ± 0.3	96.4 ± 0.1	95.7 ± 0.2	96.1 ± 0.1	85.9 ± 0.1	88.9 ± 0.2
DGI	92.4 ± 0.7	92.4 ± 1.1	96.2 ± 0.3	96.5 ± 0.2	87.7 ± 2.3	92.9 ± 0.5	83.2 ± 1.1	79.0 ± 2.0
GAE	93.1 ± 0.8	94.6 ± 0.8	96.0 ± 0.4	96.5 ± 0.2	96.7 ± 0.1	97.2 ± 0.1	96.1 ± 0.2	96.6 ± 0.1
G2G	92.7 ± 0.8	94.3 ± 0.7	94.0 ± 0.4	97.8 ± 0.1	96.6 ± 0.1	97.2 ± 0.1	81.7 ± 4.9	90.2 ± 1.1
Best method	**95.4 ± 0.7**	**96.7 ± 0.6**	**97.3 ± 0.2**	**98.5 ± 0.2**	**98.0 ± 0.1**	**98.0 ± 0.1**	**96.9 ± 0.1**	**97.2 ± 0.1**

(b) Node classification (accuracy).

Method	Cora	Citeseer	Pubmed	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo
Raw features	64.0 ± 1.1	65.9 ± 0.6	84.4 ± 0.4	53.5 ± 0.4	92.3 ± 0.3	94.9 ± 0.1	80.4 ± 0.7	87.8 ± 0.5
DeepWalk	76.7 ± 1.4	53.9 ± 1.4	80.1 ± 0.3	59.6 ± 0.3	86.9 ± 0.2	92.7 ± 0.1	87.4 ± 0.3	91.0 ± 0.4
DGI	82.8 ± 0.9	70.5 ± 0.9	84.6 ± 0.3	62.8 ± 0.4	91.5 ± 0.2	95.1 ± 0.1	88.5 ± 0.3	93.1 ± 0.3
GAE	82.3 ± 0.9	71.4 ± 0.7	83.1 ± 0.4	62.9 ± 0.4	93.2 ± 0.2	**95.3 ± 0.1**	88.0 ± 0.5	93.1 ± 0.4
G2G	79.8 ± 1.1	69.0 ± 0.7	84.4 ± 0.3	62.0 ± 0.4	91.0 ± 0.2	94.4 ± 0.1	87.8 ± 0.2	92.0 ± 0.3
Best method	**84.0 ± 0.8**	**72.1 ± 0.5**	**85.3 ± 0.2**	**65.2 ± 0.3**	**92.6 ± 0.3**	94.2 ± 0.3	**89.0 ± 0.4**	**93.4 ± 0.4**

For link prediction, distance-based methods often result in very high training performance that does not generalize well to the test set, showing a sign of overfitting. Methods based on the inner-product score (aside from pathological cases where the score does not match with the loss function appropriately) yield slightly lower training performance but better generalization.

In the case of node classification, methods based on the inner product score yield better training and test performance in Pubmed (not shown in figure 4.2), and Cora. In Citeseer, distance-based methods exhibit better generalization performance. We note that Citeseer differs from Cora and Pubmed in that it has positive degree assortativity, although there is no formal justification for this not to be more than a correlation, and other graph statistics might explain this result.

Having observed the results of multiple methods, we select the best combination and proceed to apply it to the rest of the datasets. We find that it continues to outperform other methods in link prediction on all datasets, and in node classification on all but one dataset. We show the results in tables 4.5a and 4.5b, compared against the best variants of existing methods that we found in the previous section.

Our findings demonstrate the potential of the modular framework to devise new methods for graph representation learning. For the networks we have studied and the tasks of link prediction and node classification, the exhaustive search across

components allows us to identify a very strong method for link prediction, which outperformed all other methods we have studied thus far. The method uses an SGC encoder, the inner product as a score, the hinge loss, and the first-neighbor sampling strategy of GAE. Its superior performance also generalizes to node classification, with the exception of Citeseer, where the best method instead uses an L^2 distance score (while the rest of the components is unchanged from the best method in link prediction).

We can shed light on the best performing method by examining its architecture. For a node embedding $\mathbf{z}_i$ and positive and negative samples $\mathbf{z}_j, \mathbf{z}_k$, the method minimizes the hinge loss with the inner product score:

$$\mathcal{L} = \max(0, 1 - \mathbf{z}_i^\top \mathbf{z}_j + \mathbf{z}_i^\top \mathbf{z}_k) \tag{4.2}$$

This loss attains a global minimum when $\mathbf{z}_i^\top \mathbf{z}_j = A_{ij} = 1$ and $\mathbf{z}_i^\top \mathbf{z}_k = A_{ik} = 0$, or equivalently, when $\mathbf{Z}\mathbf{Z}^\top = \mathbf{A}$. Given an SGC encoder $\mathbf{Z} = \text{SGC}(\mathbf{X}, \mathbf{A})$ (defined in equation 2.13), the minimum is therefore achieved for a weight matrix $\mathbf{W}$ such that

$$\begin{aligned}
\mathbf{A} &= \mathbf{Z}\mathbf{Z}^\top \\
&= \text{SGC}(\mathbf{X}, \mathbf{A})\text{SGC}(\mathbf{X}, \mathbf{A})^\top \\
&= (\tilde{\mathbf{A}}^k \mathbf{X}\mathbf{W})(\tilde{\mathbf{A}}^k \mathbf{X}\mathbf{W})^\top \\
&= \tilde{\mathbf{A}}^k \mathbf{X}\mathbf{W}\mathbf{W}^\top \mathbf{X}^\top \tilde{\mathbf{A}}^k
\end{aligned} \tag{4.3}$$

This result shows that method found via hyperparameter search is a linear model, which describes a factorization of the adjacency matrix through feature propagation across k-hop neighborhoods. Matrix factorization methods have been used in related works to obtain node embeddings (Ahmed et al., 2013b), and in particular some approaches consider higher order neighborhoods that are captured by powers of the adjacency matrix (Cao et al., 2015; Ou et al., 2016).

The feature propagation performed by the best variant can be seen as low-pass filtering of node features. Previous works have shown that this operation is sufficient to obtain sufficient predictive performance in the datasets we have used, by removing high frequency noise in node features (Wu et al., 2019a; Maehara, 2019), which explains the superior performance of this method.

A second interpretation stems from equation 4.3, where we can see that the model recovers the adjacency matrix by starting from its normalized version $\tilde{\mathbf{A}}$ to perform feature propagation, from which it obtains the embeddings by reducing the dimensionality with the weight matrix. The process is then reversed with the tranpose of the weight matrix. This can be seen as a Linear, Symmetric Graph

Table 4.6: Performance on downstream tasks of embeddings obtained with LSGAE and PCA.

(a) Link prediction (average precision).

Method	Cora	Citeseer	Pubmed
LSGAE	95.4 ± 0.7	96.7 ± 0.6	97.3 ± 0.2
PCA	92.0 ± 0.9	94.8 ± 0.7	97.1 ± 0.1

(b) Node classification (accuracy).

Method	Cora	Citeseer	Pubmed
LSGAE	84.0 ± 0.8	72.1 ± 0.5	85.3 ± 0.2
PCA	83.6 ± 0.8	73.0 ± 0.6	85.8 ± 0.3

Autoencoder (LSGA). It has been shown that linear autoencoders are closely related to Principal Components Analysis (PCA), as they learn the principal subspace of the data (Bourlard & Kamp, 1988; Baldi & Hornik, 1989). From this we can conclude that LSGAE works by approximating the principal components of node features smoothed across k-hop neighborhoods, which results in strong predictors for link prediction and node classification.

We examine the experimental relationship between LSGAE and PCA in both tasks by running experiments on Cora, Citeseer, and Pubmed. In the case of PCA, we obtain embeddings by applying the algorithm to the matrix of filtered features $\tilde{\mathbf{A}}\mathbf{X}$ to reduce the dimension down to 128. The results are shown in table 4.6, where we observe similar results for these methods, especially in node classification.

4.3.3 Wasserstein embeddings

We now turn our attention to the use of Wasserstein spaces to embed nodes in a graph, as motivated in the previous chapter. We defined the representation and scoring components, and we experimented with multiple combinations of encoders, loss functions and sampling strategies, using the same values listed in table 4.4, as for the hyperparameter study. We used 128 units for the embeddings, as in the previous experiments. These were used to represent the locations of the support points, in spaces of varying dimension. We run experiments with 1, 4, 8, 16, and 32 support points.

The results show that the MLP encoder yields the best performance for link prediction. The GCN and SGC encoders produce similar results on the training set, but their generalization is lower, as we show in figure 4.3 for Cora. We observed similar results on the rest of the datasets.

We observed that the loss functions that result in better performance are those that contain an explicit margin hyperparameter, such as the hinge loss and the square-square loss, for which we used a fixed margin of 1. These losses resulted in

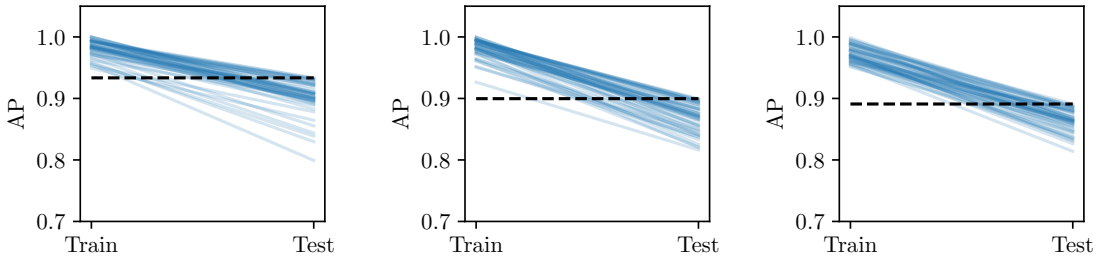


Figure 4.3: Link prediction performance of Wasserstein embeddings on the training and test sets, when using different encoders: MLP (left), GCN (center), and SGC (right). The dashed line shows the highest value obtained in the test set. The MLP encoder shows better generalization than encoders that use the structure of the graph.

Table 4.7: Average precision in percent on the link prediction task using Wasserstein embeddings. The results are obtained by using 128 real numbers to represent the locations of a varying number of support points.

(a) Link prediction (average precision).

# points	Cora	Citeseer	Pubmed
1	93.3	95.8	96.0
4	93.0	95.4	95.8
8	93.1	95.2	95.6
LSGAE	95.4	96.7	97.3

(b) Node classification (accuracy).

# points	Cora	Citeseer	Pubmed
1	79.0	67.1	84.6
4	79.7	70.2	82.5
8	77.6	69.8	82.6
LSGAE	84.0	72.1	85.3

a better separation of scores between positive and negative samples, compared to the square-exponential, which is also a generalized margin loss that does not have an explicit margin hyperparameter. The use of binary cross-entropy loss yields the lowest performance among all loss functions, which indicates that energy-based losses are better suited when using a score based on a distance.

We show the effect of the number of support points for the link prediction task in table 4.7a. We observe that values higher than 1 do not bring significant improvements in performance, while increasing the computational cost of the procedure. When using one point in the support, the distance essentially reduces to the Euclidean distance, and the representation is not different from that used by other methods, such as GAE and DGI. We also note that this method is also outperformed by LSGAE.

When using Wasserstein embeddings for node classification, we note that using a

logistic regression classifier (as in the previous experiments for other methods) would induce a dependency on the order in which the locations of the support points are passed to the classifier. A more suitable architecture is invariant to permutations of the support points, which has been proposed in the literature under the name of PointNet (Qi et al., 2017) or Deep Set networks (Zaheer et al., 2017).

We follow this approach and train a classifier that uses the locations of the support points to predict a class. For a node v_x with support $\mathcal{X}$ learned by the unsupervised method, we pass every location $\mathbf{x}_i \in \mathcal{X}$ through a two-layer MLP with ReLU activations, and then take the element-wise maximum operator across all locations. The result is passed through a linear transformation to obtain the logits $\tilde{\mathbf{c}}_x$. The function represented by the classifier can thus be written as follows:

$$\tilde{\mathbf{c}}_x = \mathbf{W} \max_i \left(\{ \mathrm{MLP}(\mathbf{x}_i) | \mathbf{x}_i \in \mathcal{X} \} \right) + \mathbf{b} \tag{4.4}$$

The conditional distribution of classes for a given node is then obtained by passing the logits through a softmax layer.

The results are shown in table 4.7b. As noted before, we note that the benefit of using more than one support point is often not clear, and sometimes it can hurt performance. In the case of Citeseer, using 4 points results in improvements in accuracy. As in the case of link prediction, LSGAE still outperforms the use of Wasserstein embeddings, and its accuracy is significantly larger than when using 1 point, which corresponds to a point embedding in Euclidean space. We also experimented with classifiers that included regularization in the form of weight decay and dropout, but under different values we could not find a setting with improved generalization.

4.4 Summary

The experimental evaluation of our modular framework highlight its potential for studying and improving methods for graph representation learning. In particular, our experiments show that information about the graph needs to be present in at least one module (e.g. encoder, or sampling strategy), to increase the predictive performance of the representations. For instance, DGI fails with an MLP encoder, because the sampling strategy considers the graph as the positive sample, and a corrupted version as the negative sample. Without a graph neural network to encode the features, the method does not have any way to leverage information about the graph. Some choices in the representation might not be necessary as they add more

complexity but they don't provide significantly improved performance. This is the case of G2G, where the Gaussian representation can be simplified as vectors in a Euclidean space with pairs evaluated with the L^2 distance, to obtain similar results while halving the number of parameters.

We observe differences in the kind of methods that work well for link prediction and node classification, with multiple assumptions that can have a positive or negative impact depending on a specific network. It is in principle difficult to obtain a model for link prediction that fits the training data very well while also generalizing, since negative samples in the training set can be due to edges that have been removed from the graph and are present in the test set, or actual cases of two unconnected nodes. Furthermore, when using the inner product to score a pair, we maximize the correlation of embeddings of nodes that are linked, whereas for node classification, two nodes might be likely to be connected but do not necessarily belong to the same community. The performance of a specific method is then tied to the homophily characteristics of the network and other graph statistics.

The hyperparameter study led us to a linear method to learn embeddings that consistently outperforms other methods for all datasets, that is close to the formulation of GAE, but instead uses an SGC encoder and a hinge loss. The fact that linear models are sufficient to obtain performance similar to more complex alternatives for the networks that we have used, has been observed in previous works. This motivates the use of different datasets, especially for link prediction, where in many cases the performance is already high; and methods with improved performance for node classification.

The modular framework admits including a variant to embed nodes in Wasserstein spaces. In the previous chapter, preliminary experiments showed that such method can indeed learn distributions that are close for positive samples, and far for negative samples. However, when testing the method in link prediction and node classification we observe that in spite of obtaining results that compete with existing methods, the best number of points is often 1, which discards the distributional representation of the embeddings. We note that Wasserstein embeddings fit very well to the training data, but as used in our methods, they do not generalize for link prediction and node classification. Using graph encoders can make this problem worse by increasing overfitting.

Frogner et al. (2019) indicate that while Wasserstein spaces have been shown to have high representational capacity, their generalization capabilities remain as an

open question. The authors also report[2] that the flexibility of these spaces produces overfitting when applied to graph representation learning, and proper regularization techniques towards ameliorating this problem are an open problem.

[2]Personal communication.

Chapter 5

Conclusion

In this thesis, we have explored current methods for unsupervised representation learning on graphs, and we have identified a modular framework that can be used to determine the extent to which existing algorithms can be improved. Our framework is validated by findings that show that in some cases, changes in certain components can lead to significant improvements or a reduction in the complexity of a model. Furthermore, its modular nature allows it to continue being used to devise new methods as research continues in the field, for instance in the topics of graph neural network architectures, and variational inference for more complex representations.

Aiming to explore richer representations for nodes, we explored embeddings in Wasserstein spaces. We found that conventional regularization techniques are not enough to prevent their high representational capacity from causing overfitting, and improving generalization is a promising direction for future work. Instead, in our benchmarks, a linear model with regularization resulted in the best performance. This gives an incentive for the use of new benchmark datasets that benefit from the use nonlinear functions for representation learning.

The networks that we have used in our evaluation experiments have a wide range of sizes. However, many real-world examples are much bigger and additional measures must be taken when they do not fit in memory. An interesting direction of future work consists of extending the evaluation to such graphs, by adapting methods to a larger scale. Our experiments also showed that statistics of graphs can be diverse, and can have an impact on a particular downstream task of interest. We hope to gain improvements by further studying the relationship between these statistics, the downstream task of interest, and the type of components that a method of representation learning uses, while also considering additional downstream tasks, such as graph classification.

Bibliography

Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., and Smola, A. J. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pp. 37–48. ACM, 2013a.

Ahmed, A., Shervashidze, N., Narayanamurthy, S. M., Josifovski, V., and Smola, A. J. Distributed large-scale natural graph factorization. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pp. 37–48, 2013b.

Altschuler, J., Weed, J., and Rigollet, P. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 1961–1971, 2017.

Ambrogioni, L., Güçlü, U., Güçlütürk, Y., Hinne, M., van Gerven, M. A. J., and Maris, E. Wasserstein variational inference. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 2478–2487, 2018.

Andoni, A., Naor, A., and Neiman, O. Snowflake universality of wasserstein spaces. In *Annales Scientifiques de l'Ecole Normale Superieure*, volume 51, pp. 657–700. Societe Mathematique de France, 2018.

Backstrom, L. and Leskovec, J. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pp. 635–644, 2011.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Baldi, P. and Hornik, K. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Belkin, M. and Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pp. 585–591, 2002.

Belkin, M., Niyogi, P., and Sindhwani, V. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov):2399–2434, 2006.

Bengio, Y., Ducharme, R., and Vincent, P. A neural probabilistic language model. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pp. 932–938, 2000.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Bhagat, S., Cormode, G., and Muthukrishnan, S. Node classification in social networks. In *Social network data analytics*, pp. 115–148. Springer, 2011a.

Bhagat, S., Cormode, G., and Muthukrishnan, S. Node classification in social networks. In *Social Network Data Analytics*, pp. 115–148. 2011b.

Biederman, I. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987.

Bishop, C. Regularization and complexity control in feed-forward networks. In *Proceedings International Conference on Artificial Neural Networks ICANN'95*, volume 1, pp. 141–148. EC2 et Cie, January 1995.

Blei, D. M., Griffiths, T. L., Jordan, M. I., and Tenenbaum, J. B. Hierarchical topic models and the nested chinese restaurant process. In *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, pp. 17–24, 2003.

Bojchevski, A. and Günnemann, S. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

Bourgain, J. The metrical interpretation of superreflexivity in banach spaces. *Israel Journal of Mathematics*, 56(2):222–230, 1986.

Bourlard, H. and Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

Brazinskas, A., Havrylov, S., and Titov, I. Embedding words as distributions with a Bayesian Skipgram model. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pp. 1775–1789, 2018.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Cao, S., Lu, W., and Xu, Q. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pp. 891–900, 2015.

Cao, S., Lu, W., and Xu, Q. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 1145–1152, 2016.

Claici, S., Chien, E., and Solomon, J. Stochastic wasserstein barycenters. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 998–1007, 2018.

Clark, E. V., Gelman, S. A., and Lane, N. M. Compound nouns and category structure in young children. *Child development*, pp. 84–94, 1985.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167. ACM, 2008.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pp. 2292–2300, 2013.

Davidson, T. R., Falorsi, L., De Cao, N., Kipf, T., and Tomczak, J. M. Hyperspherical variational auto-encoders. *34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*, 2018.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

Dorogovstev, S. and Mendes, J. Evolution of networks, 2003.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. *CoRR*, abs/1903.02428, 2019.

Firth, J. A synopsis of linguistic theory 1930-1955. In *Studies in Linguistic Analysis*. Philological Society, Oxford, 1957. reprinted in Palmer, F. (ed. 1968) Selected Papers of J. R. Firth, Longman, Harlow.

Fortunato, S. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

Fouss, F., Pirotte, A., Renders, J., and Saerens, M. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. Knowl. Data Eng.*, 19(3):355–369, 2007.

Franklin, J. and Lorenz, J. On the scaling of multidimensional matrices. *Linear Algebra and its applications*, 114:717–735, 1989.

Frogner, C., Mirzazadeh, F., and Solomon, J. Learning entropic Wasserstein embeddings. In *International Conference on Learning Representations*, 2019.

Genevay, A., Peyré, G., and Cuturi, M. Learning generative models with sinkhorn divergences. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, pp. 1608–1617, 2018.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 1263–1272, 2017.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016.

Gutmann, M. and Hyvärinen, A. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13:307–361, 2012.

Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Herzig, R., Raboh, M., Chechik, G., Berant, J., and Globerson, A. Mapping images to scene graphs with permutation-invariant structured prediction. In *Advances in Neural Information Processing Systems*, pp. 7211–7221, 2018.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Hinton, G. E., McClelland, J. L., Rumelhart, D. E., et al. *Distributed representations*. Carnegie-Mellon University Pittsburgh, PA, 1984.

Hoff, P. D., Raftery, A. E., and Handcock, M. S. Latent space approaches to social network analysis. *Journal of the american Statistical association*, 97(460): 1090–1098, 2002.

Jonker, R. and Volgenant, A. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

Kantorovich, L. V. On the translocation of masses. *Journal of Mathematical Sciences*, 133(4):1381–1382, 2006.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2013.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Kloeckner, B. Approximation by finitely supported measures. *ESAIM: Control, Optimisation and Calculus of Variations*, 18(2):343–359, 2012.

Kriege, N. M., Johansson, F. D., and Morris, C. A survey on graph kernels. *arXiv preprint arXiv:1903.11835*, 2019.

Kuhn, H. W. The H1ungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

LeCun, Y. and Huang, F. J. Loss functions for discriminative training of energy-based models. In *AIStats*, volume 6, pp. 34, 2005.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Li, J. and Jurafsky, D. Do multi-sense embeddings improve natural language understanding? In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pp. 1722–1732, 2015.

Liang, X., Hu, Z., Zhang, H., Lin, L., and Xing, E. P. Symbolic graph reasoning meets convolutions. In *Advances in Neural Information Processing Systems*, pp. 1858–1868, 2018.

Liben-Nowell, D. and Kleinberg, J. M. The link prediction problem for social networks. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pp. 556–559, 2003.

Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Marcus, G. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

Marr, D. and Nishihara, H. K. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 200(1140):269–294, 1978.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013b.

Mnih, A. and Hinton, G. E. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pp. 1081–1088, 2008.

Mnih, A. and Kavukcuoglu, K. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in neural information processing systems*, pp. 2265–2273, 2013.

Monge, G. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie royale des sciences de Paris*, 1781.

Namata, G., London, B., Getoor, L., and Huang, B. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, 2012.

Newman, M. E. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

Nickel, M., Tresp, V., and Kriegel, H. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pp. 809–816, 2011.

Osherson, D. N. and Smith, E. E. On the adequacy of prototype theory as a theory of concepts. *Cognition*, 9(1):35–58, 1981.

Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1105–1114, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Peyré, G. and Cuturi, M. Computational optimal transport: With applications to data science. 2019.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 77–85, 2017.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 1530–1538, 2015.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NIPS 2018*, 2018.

Sinkhorn, R. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

Sjöberg, J. and Ljung, L. Overtraining, regularization and searching for a minimum, with application to neural networks. *International Journal of Control*, 62(6): 1391–1407, 1995.

Tang, L. and Liu, H. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

Tran, P. V. Learning to make predictions on graphs with autoencoders. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 237–245. IEEE, 2018.

van den Berg, R., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pp. 393–402, 2018.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018a.

Veličković, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. In *International Conference on Learning Representations*, 2018b.

Villani, C. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Vilnis, L. and McCallum, A. Word representations via Gaussian embedding. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Wang, D., Cui, P., and Zhu, W. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1225–1234, 2016.

Wasserman, S., Faust, K., et al. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

West, D. B. et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996.

Weston, J., Ratle, F., Mobahi, H., and Collobert, R. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pp. 639–655. Springer, 2012.

Wu, F., Zhang, T., Souza Jr., A. H., Fifty, C., Yu, T., and Weinberger, K. Q. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019a.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019b.

Xu, D., Zhu, Y., Choy, C. B., and Fei-Fei, L. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5410–5419, 2017.

Yang, B., Yih, W., He, X., Gao, J., and Deng, L. Embedding entities and relations for learning and inference in knowledge bases. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015a.

Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015b.

Yule, G. U. On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society*, 75(6):579–652, 1912.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 5171–5181, 2018.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Zhou, T., Lü, L., and Zhang, Y.-C. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.

Zhu, X., Ghahramani, Z., and Lafferty, J. D. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919, 2003.