Including Attributes in Graph Embeddings

Jérôme Lenßen

Department of Computer Science
RWTH Aachen University

This thesis is submitted for the degree of
Bachelor of Science

October 2018

Abstract

Graph embeddings play an important role in the analysis of large graphs. In a graph embedding the graph is mapped to a latent feature space preserving a pre-defined proximity measure. Those latent features can then be used as an input to machine learning and data mining algorithms which require usually vectors as an input. Till now most of the existing models focused on exploiting the information given in the topological structure of the graph to create an embedding, ignoring the rich source that attributes associated to nodes and edges can be. In this thesis we propose different approaches to the problem of including graph attributes in a graph embedding. We do that by first analysing the state of the art models and then introducing new ways of taking auxiliary information into account. The focus lies on incorporating textual and numerical values. We demonstrate that those approaches can beat state of the art models and introduce a novel way of incorporating literals into KGloVe.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

ANN Artificial Neural Network

BCA Bookmark-Coloring Algorithm

BCV Bookmark-Coloring Vector

CART Classification and Regression Tree

CNN Convolutional Neural Network

GCN Graph Convolutional Network

GMM Gaussian Mixture Model

MTL Multi-task Learning

NER Named Entity Recognition

PPR Personalised Page Rank

SGD Stochastic Gradient Descent

SVM Support Vector Machine

VGAE Variational Graph Auto-Encoders

Chapter 1

Introduction

Graph structures play an important role in our daily life. By using the internet, we access a graph structure containing billions of nodes and edges. Social networks connect people around the world forming a giant graph. Protein-protein interaction (PPI) networks play an important role in drug development, describing how different proteins react to each other and how drugs affect those interactions [63]. Since graph structures seem to form naturally, it is important to get a good understanding of those structures. By analysing the graph and extracting information from it, we can understand the complex rules that for example generate the topology of the graph. Moreover we can optimise processes, such as in a communication systems or in a PPI network and gain knowledge about how different entities are related to each other and interact with each other. The case of Cambridge Analytica[1] showed, that by understanding how a network works, we can influence how it will evolve in the future by controling core points in the network, so called hubs. The information mined from the graph can then be later used in different tasks such as link prediction, node classification and community detection. The knowledge base extracted from the graph can also be used as an additional source for training agents.

The representation of the information that we gained from the graph is crucial for further tasks such as clustering and link prediction. Good representations should highlight the features we are looking for and make therefore tasks easier which depend on those features. It should generalise well so that different tasks can use the representation. A representation is nevertheless usually task-specific and requires a good understanding of the theory underlying the problem which is not dependent on a specific representation. A good example is provided by [28, p.20-22] where the task is to multiply two natural numbers a and b. The operation of

[1] https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election

multiplication is defined as

$$a \cdot b \coloneqq \underbrace{a + \ldots + a}_{\times \text{ b}}. \tag{1.1}$$

Depending if we represent a and b in the decimal system or as roman numerals, we can rather easy multiply the numbers in the decimal system but with roman numerals the task is quite hard. This problem highlights how important a good representation of the objects is, that we are operating on.

Since most machine learning algorithms take vectors as an input, we want our graph to be represented by vectors. A common representation of a graph is its adjacency matrix. The problem of using the adjacency matrix as an input to data mining algorithms is its sparsity, since most graphs are not fully connected networks. To solve the problem of sparsity the notion of graph embeddings was introduced in recent years. The idea is to find a mapping f, that maps the graph into a low-dimensional normed vector space V. By mapping the graph to V, the problem of sparsity is resolved, since features have to be represented densely. We make hereby use of the inherent properties of vector spaces that are useful in this regard, that they are structured objects with well-defined operations such as vector addition and scalar multiplication.

There are different ways of embedding a graph, depending on the task that we want to solve. Earlier work focused on factorisation based methods such as Laplacian Eigenmaps [2] that performs an Eigenvalue decomposition of the Laplacian matrix of the given graph as well as Locally Linear Embedding (LLE) [42] that can also be reduced to an Eigenvalue decomposition problem. The problem with those methods is, that they can only preserve local proximity in a graph and that they are computationally costly. Modern factorisation methods such as HOPE [33] can preserve higher order proximity and can be computed more efficiently through Singular Value Decomposition (SVD). Due to the fact that matrix factorisation is costly, more recent approaches make use of random walks over the graph and optimise the log-likelihood of a node given its surrounding nodes in a random walk [37], [17]. Interestingly it can be shown, that methods based on random walks carry out an implicitl matrix factorisation [39]. With the increased interest in deep learning techniques, many models based on deep artificial neural networks (ANNs) were proposed in recent time. An example is the Variational Graph Auto-Encoder (VGAE) [22] uses a graph convolutional neural network (GCN) to reconstruct the adjacency matrix. Including attributes in a graph embedding gathers more and more interest in the graph embedding community in recent years. The underlying idea is, that attributes of a graph offer additionally information about how entities in the graph are related to each other. Therefore the goal is to embed this information together with the structural information into one graph embedding. DANE

[61] and ANRL [15] are two recent models, that use multitask-learning via hard- and soft-parameter sharing to embed the topological structure of the graph together with the attribute information. Another approach is made by Conv use additional knowledge bases to exploit further information about entities in the graph [53]. Most models have no closed solution due to their highly non-linear objective function and are therefore optimised with gradient descent based methods. To remedy this fact, NetHash proposes in [60] learns an embedding with a set of hashing functions to perform dimension reduction via Locality-sensitive Hashing (LSH) [10]. A survey of the development of different embedding techniques is presented by Goyal in [35].

1.1 Outline

This thesis is structured as follows. First a use case is presented to motivate the use of attributes in a graph embedding. Then we define the goal of the thesis in section 1.3 and what contributions are going to be made. Chapter 2 introduces the necessary background needed to understand different embedding techniques. Different approaches to create a graph embedding, starting off with embeddings methods which do not make use of attributes and then presenting state of the art models which include attributes, are presented in chapter 3. After having analysed existing models, the requirements and problems for creating graph embeddings are summarised and the most important techniques and methods are compared in section 3.3. In chapter 4, we then present different approaches to incorporate auxiliary information into graph embeddings. Having defined those models, the realisation, implementation and evaluation of those models is described in chapter 5. Chapter 6 reviews the different approaches and summarises improvements and problems regarding the different approaches.

1.2 Motivation

Consider a social network where users can connect by searching their name or by using filters to search for users with common interests. Each user can upload a description of himself, personal information such as age, sex, current relationship status and multimedia such as videos, pictures and music.
For the maintainer and developer of the social network it is of interest, that as much user as possible use the social network on a regular basis. Therefore he wants to be able to analyse users and their relations efficiently. Moreover he wants to introduce a recommender system based on machine learning, which recommends to each user possible new friends to keep the

network interesting. The maintainer therefore decides to generate a graph embedding of the network, since most machine learning algorithms take vectors as an input. The recommender system scans the neighbourhood of each node in the friendship graph and suggests on this basis new friends to each user. A possible new friend is defined as a user that is a friend of a friend.
By only scanning the neighbourhood of a user, possible new friends with the same interests can easily get missed. For example two people studying at the same university, taking the same lectures and with the same interests would be a perfect match, but will be missed if they have no common friends. To improve the recommendation system the maintainer decides to take into account additional information, without simply relying on the graph structure. He decides to use all of the users descriptions as additional features. The problem is, that he needs to find a good representation of those features to be able to compare efficiently the attributes between two users. To solve the problem he uses different embedding tools for each attribute, to generate an attribute vector for each user. As an example he uses a natural language processing tool to compute a vector summarising the user description represented as text. Having generated the feature vector for each user, the recommendation system can now efficiently compare two attribute vectors using a predefined distance metric. The system can then suggest new friends based on similar descriptions of the users and their neighbourhood relations.
By effectively combining all information uploaded by the users, the friendship recommendation system can benefit from this extra information and improve its prediction capabilities on a specific task. The problem on how to incorporate this additional information into a graph embedding is subject of this thesis and will be further investigated in the next chapters.

1.3 Thesis Goal

The goal of this thesis is to experiment with different approaches to incorporate auxiliary information into a graph embedding. This can be graph attributes but also external sources such as a knowledge base. This will be achieved by first reviewing the state of the art models and classical graph embedding methods. Next, we present own approaches, based on the previously introduced models. We then experiment with different ways to augment existing graph embedding methods with auxiliary information. In a last step we investigate new ways of embedding the graph structure and the literals into the same vector space. The focus of this work will be on textual literals, such as short abstracts describing an entity. Textual literals are important, since many knowledge graphs such as DBPedia [24], YAGO [52] or Wikidata [57] contain abstracts describing the entities in the graph.

Additionally we will use attributes represented through a binary feature vector, stating if an entity has a certain property or not. There are several datasets, for example citation networks such as PubMed [32], Cora or CiteSeer [26] providing those attributes for each entity in the graph This task will then be further investigated when a boolean vector is not available due to the fact that properties can take infinite number of values (for example the income of a person in a social network). Having found models which take literals into account, they will be evaluated on different tasks against state of the art models on common datasets such as FB15k [7] and PubMed. The assumption is, that models using auxiliary information from the graph, should outperform state of the art models ignoring the attributes on common tasks such as node classification or link prediction.
Literals can have different types and values, therefore it is important to find out how to combine the information included in the attributes with the graph structure. The question of the representation of attributes is going to be discussed for different types of attributes. Hereby methods from natural language processing (NLP) are going to be explored such as word2vec [30] and GloVe [36] to represent words in a vector space. Named Entity Recognition (NER) can extract relation between entities in text based attributes, that either can be embedded using for example GloVe or to augment the graph with new edges between those entities describing the extracted relation. Therefore an own NER system will be implemented to do named entity recognition.

Chapter 2

Background

When working with graph attributes we first need to think about how to represent them in a vector space. Since attributes can have different types, most attributes will need to undergo a pre-processing step, to obtain a feature vector, before we can use them in a graph embedding model. In the first part we introduce preliminary definitions. The rest of the chapter is dedicated to present common techniques for feature extraction of different attributes, focusing on text and graph structures. Additionally we introduce methods and algorithms which are necessary to understand models, that will be introduced in later chapters.

2.1 Notations and Definitions

In this section, we give a formal definition of important structures such as graph, knowledge graph, graph embedding and related terms.

Definition 1. *A graph G is a tuple (V,E), where V is an arbitrary set and $E \subseteq V \times V$ is a relation. An undirected graph G is a graph, where E is symmetric. Otherwise, we call the graph directed. A weighted graph is a graph $G_{weight} = (V,E,f)$, with $f : V \to \mathbb{R}$. A graph is called connected, if there is a path between each node in the graph.*

Definition 2. *For a given undirected graph $G = (V,E)$ and $N = |V|$, the adjacency matrix $A \in \mathbb{R}^{N \times N}$ is defined as*

$$A_{ij} = \begin{cases} 1, \text{ if } (v_i, v_j) \in E \\ 0, \text{ otherwise.} \end{cases}$$

Definition 3. *For a given graph $G = (V,E)$ and a node $v \in V$ the adjacency list $\Gamma(v)$ of the node v is defined as*

$$\Gamma(v) = \{u \in V \,|\, (u,v) \in E \text{ or } (v,u) \in E\}$$

Definition 4. *For a node $v \in V$ in a directed graph $G = (V,E)$, we define the in- and out-degree of the node as*

$$deg^-(v) = |\{u \in V \,|\, (u,v) \in E\}|$$
$$deg^+(v) = |\{u \in V \,|\, (v,u) \in E\}|$$

The number of neighbours is given as $deg(v) = deg^-(v) + deg^+(v)$.

Definition 5. *For a given undirected graph $G = (V,E)$ and $N = |V|$ the degree matrix $D \in \mathbb{R}^{N \times N}$ is defined as*

$$D_{ij} = \begin{cases} deg^-(v_i) + deg^+(v_i), \text{ if } i = j \\ 0, \text{ otherwise.} \end{cases}$$

Definition 6. *For a given graph $G = (V,E)$ and $N = |V|$, the Laplacian matrix $L \in \mathbb{N} \times \mathbb{N}$ is defined as $L = D - A$.*

Definition 7. *A knowledge graph $G_{know} = (V,E,L_V,L_E)$ is a directed graph, where V is a set of entities, and E a set of relations. L_V and L_E are labelling functions, which assign a unique label to each entity respectively relation.*

A knowledge graph is often represented as a set of triples (h,r,t), where $h,t \in V$ are entities, connected by a relation $r \in E$. We call h the head entity or subject, r relation or predicate and t the tail entity or object.

Definition 8. *The class of all graphs is defined as*

$$\mathscr{G} = \{G \mid G \text{ is a graph according to definition } 1 \text{ or } 5\}$$

Two important terms when analysing graphs is the first- and second-order proximity. Those measures are an indicator of the similarity between two nodes, regarding their neighbourhood.

Definition 9. *For a graph $G = (V,E)$ the first-order proximity between two adjacent nodes $u,v \in V$ is the weight of the edge (u,v), If two nodes are not adjacent, the first-order proximity is 0. For unweighted graphs, we assume that the edge weight is one.*

Definition 10. *For a graph $G = (V,E)$ the second-order proximity between two nodes $u,v \in V$ is the similarity of their first-order proximity to each node in their neighbourhood.*

Note, that there are different ways of defining a measure for the second-order proximity. The Jaccard Coefficient or the Personalised Page Ranks (PPR) for each node are to examples of a second-order proximity measure. Both will be introduced later. Given the definition of the similarity of nodes, we can now define a graph embedding.

Definition 11. *Let $U \subseteq \mathcal{G}$. A graph embedding is a function $f : U \to V$, where V is D-dimensional normed vector space.*

The function f has the property, that it preserves some pre-defined proximity measure of the graph. Note, that f maps a graph $G \in U$, to one vector or a set of vectors depending on the representation needed.

2.2 Artificial Neural Networks

With the recent improvements in automatic speech recognition, computer vision and other fields, achieved through the use of deep neural networks, the question arises how to apply neural networks to graph embedding techniques. In recent years a variety of models integrating artificial neural networks have been proposed to generate graph embeddings. In this section we will introduce the feedforward neural network and the auto-encoder model. There are many more types of neural networks such as convolutional neural network (CNN) or recurrent neural networks (RNN). An introduction to this topic is given by Goodfellow et al., in [16].

2.2.1 Feedforward Neural Network

A commonly used neural network is the feedforward neural network. It is represented by the function

$$f(x;\theta) = f^{(n)}\left(f^{(n-1)}\left(\dots f^{(2)}\left(f^{(1)}(x)\right)\dots\right)\right). \tag{2.1}$$

The number n denotes the number of layers in the neural network. $f^{(1)}$ is called the input layer and $f^{(n)}$ the output layer. The other layers are called the hidden layers. In general a layer is represented by a non-linear transformation given by

$$f^{(k)}(x;\theta) = \sigma(W^{(k)}x + b^{(k)}), \tag{2.2}$$

where $k \in \{1,\dots,n\}$, σ is a differentiable function called the activation function.
$\theta = \{W^{(k)}, b^{(k)} \mid k \in \{1,\dots,n\}\}$ are the weights and $b^{(k)}$ is the bias of each layer.
We define three commonly used activation functions, namely, Softmax, Sigmoid and the Rectified Linear Unit (ReLU) for ANNs.

Definition 12. *Let V be a vector space, $x,y \in V$ and $\phi : V^D \to \mathbb{R}$, $D \in \{1,2\}$. Then the Softmax, Sigmoid and ReLU activation functions are defined as*

$$\begin{aligned} \text{Softmax}(x_i,y) &= \frac{\exp(\phi(x_i,y))}{\sum_j \exp\left(\phi(x_j,y)\right)}, \\ \text{Sigmoid}(x,y) &= \frac{1}{1+\exp(-\phi(x,y))}, \\ \text{ReLU}(x) &= \max(x,0). \end{aligned} \tag{2.3}$$

Note, that Softmax defines a probability distribution, whereas ReLU and Sigmoid do not. The function ϕ is defined as, if not otherwise stated, the dot-product for vectors. If $y = 1$, we use for ϕ the identity function. In this case we omit y in σ.
The goal of a neural network is to learn a function f^*. In general, we are given training pairs $(x_i, f^*(x))_{i\in\mathbb{N}}$ and we want to minimise the error between $f(x)$ and $f^*(x)$. This is commonly, done by defining a (differentiable) objective function $J(\theta)$. The neural network is then trained using gradient-based methods like Newton's method [43]. By using the chain-rule, each entry in the weight vectors and matrices is updated separately and the next step in the gradient-descent algorithm is computed.

2.2.2 Auto-encoder

An auto-encoder is a neural network that consists of two parts. An encoder function f_{enc} and a decoder function f_{dec}. For input x and an arbitrary loss function L, the auto-encoder aims to minimise the error between the input x and the output $(f_{\text{dec}} \circ f_{\text{enc}})(x)$ resulting in an objective function

$$J(\theta) = L(x, (f_{\text{dec}} \circ f_{\text{enc}})(x)). \tag{2.4}$$

In order to make sure that the auto-encoder does not learn the identity function, which is the trivial solution to the reconstruction problem, the output dimension of the encoder, is set to a smaller value than the input dimension. The output of the encoder is called the code or code layer. The encoder is then forced to emphasise the most salient features in the input vector to minimise the reconstruction error. Such an auto-encoder is called an under-complete auto-encoder [16]. Figure 2.1 gives an overview of the architecture of an auto-encoder.
The encoding and decoding function is commonly implemented as feedforward neural networks and trained with back-propagation. On the other hand we could choose any trainable function as encoder or decoder, as long as the output dimension of the encoder and decoder fixed.
There are several methods to improve the performance of the auto-encoder during training.

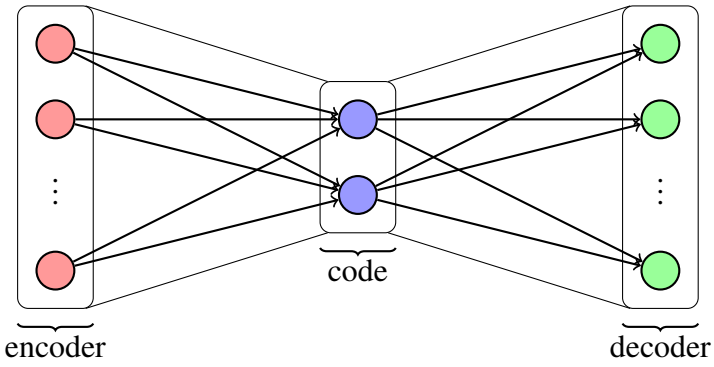

Fig. 2.1 Architecture of an auto-encoder

One method is to apply regularisation to the code layer. This introduces sparseness in the representation and is especially suited for further classification tasks on the generated features. The second method is called denoising auto-encoders. The idea here is, that the auto-encoder should not simply copy the input to the output, but should also act as an inverse filter to random noise applied to the input. The objective function is the same as the generic auto-encoder given by

$$J(\theta) = L(x, (f_{\text{dec}} \circ f_{\text{enc}})(\tilde{x})), \tag{2.5}$$

where $\tilde{x}$ is the corrupted input. This process results in general in more robust features. A more in depth overview of the different approaches using auto-encoders gives [16, Ch. 14]

2.2.3 Multi-Task Learning

Most models in machine learning and especially neural networks learn by minimising an objective function. This function encodes the information we want to learn and the way the information will be represented. When humans try to learn a new task, they usually do not rely on one type of information, but make use of information of different domains that are not directly connected to the task we try to learn. For example the ability to speak French, helps us learning Spanish. We adopt this method by introducing multiple objective functions and weigh them using hyper-parameters. The general structure has the form

$$J(\theta) = \sum_{i=1}^{N} \alpha_i \cdot J_i(\theta).$$

The main task is split into multiple sub tasks $L_i(\theta)$ and weighted with α_i, so that $\sum_i^N \alpha_i = 1$. This idea is called Multi-task Learning (MTL) and is a field in machine learning that studies

objective functions with multiple tasks and how they can be efficiently learned. An overview of the field is presented in [44].

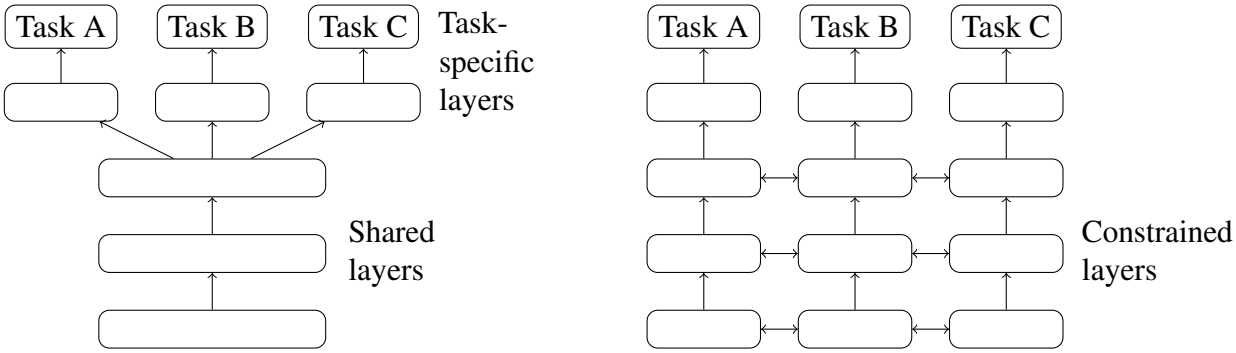

Fig. 2.2 Overview of soft and hard parameter sharing in neural networks. (Source: [44])

Two common techniques are soft- and hard-parameter sharing. With hard-parameter sharing, most of the hidden layers in the neural network are shared by all objective functions. Soft-parameter sharing assigns each objective function its own layers. To share information, the weights of the layers are regularised across the objective functions. This forces the weights to be similar and therefore share information. One type of regularisation is given by

$$J(\theta) = \sum_{i}^{N} ||W_i^{(l)} - \frac{1}{N}\sum_{j}^{N} W_j^{(l)}||_2^2. \tag{2.6}$$

Here the weights $W_i^{(l)}$ of layer l for each task $i \in \{1,\dots,N\}$ are pushed towards the mean of all weights for layer l. Figure 2.2 gives an overview of the architecture for both methods.

2.3 Natural Language Modelling

When working on a graph structure like Wikipedia, we are naturally confronted with unstructured text describing entities in the graph. As we pointed out in section 1.3, only taking the graph structure into consideration for generating a graph embedding is not sufficient, since relations between entities are described in the text. Therefore we are interested in methods for automatic feature extraction from unstructured text. This section will give an overview of three methods for extraction features from unstructured text.

2.3.1 Bag-of-Words Model

A document D is a collection of sentences in a natural or formal language. Each line D_i consists of a sentence $w_1^{N_i} = (w_1,\dots,w_{N_i})$, where $N_i \in \mathbb{N}$. Given a sentence $w_1^{N_i}$ we call

$w_i \in V$ a word, which represents the atomic parts of a document. V is the vocabulary of the document.
A Bag-of-Words model (BoW) for the document D is a vector $b \in \mathbb{R}^{|V|}$, where b_i represents the number of times w_i appeared in the document. For a set of documents, the Bag-of-Words model counts the number of appearance of word w_i across all documents.
Note, that the BoW vector does not capture how words are related to each other. We will demonstrate in section 5.4, that despite its simplicity, the Bag-of-Words model contains enough information to enhance graph embeddings on the node classification task.

2.3.2 Word Embeddings

To remedy the fact, that BoW vectors do not capture how words are related to each other, we introduce in this section word embeddings Word embedding models compute, similar to graph embeddings, for each word in a corpus a vector. This vector represents the word in a vector space. The models aims to preserve a proximity measure defined on the graph. In general, we want that a word embedding catches some of the linguistic regularities of a language. Given the vector representation of king, man, queen and woman, generated by word2vec [30] a word embedding method, fulfils following equation:

$$\text{vec}(\text{'King'}) - \text{vec}(\text{'Man'}) = \text{vec}(\text{'Queen'}) - \text{vec}(\text{'Woman'}). \tag{2.7}$$

The embedding represents concepts through distances between pairs of words. In this example it learned that king is related to man as queen is related to woman and expresses this fact by positioning the entities of both pairs equidistant to each other in the vector space.

2.3.3 word2vec

We will now give an overview of the afore mentioned word2vec model. word2vec is a neural network based model. There are two version of the model. The first model is the Skip-gram model and the second one is the Continuous Bag-of-Words (CBOW) model. Figure 2.3 presents the model architectures. word2vec is trained on a text corpus C with vocabulary V. The idea of the Skip-gram model is that for a given word we estimate the surrounding context. More formally, given a sentence $w_1^N = (w_1, \ldots, w_N) \in C$ with $w_i \in V$ we want to maximise the probability

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{-c \leq j \leq c, j \neq 0} \log \Pr\left(w_{i+j} | w_i\right). \tag{2.8}$$

$c \in \mathbb{N}$ represents the surrounding context of our focus word and the probability $\Pr\left(w_{i+j}|w_i\right)$ is computed using the Softmax function. The probability is then defined as

$$\Pr\left(w_{i+j}|w_i\right) = \text{Softmax}(\tilde{v}_{i+j}, \tilde{v}_i) = \frac{\exp(\tilde{v}_{i+j}^T \tilde{v}_i)}{\sum_{t=1}^{|V|} \exp(\tilde{v}_t^T \tilde{v}_i)}. \tag{2.9}$$

Where $\tilde{v} \in \mathbb{R}^D$ is a corresponding embedding vector of the projection layer and $D \in \mathbb{N}$ the embedding dimension. Note, that since we need to sum over the whole vocabulary, which can consists of hundred of thousands of words, the computation of the Softmax function is replaced by a hierarchical Softmax model to speed up the training [31].

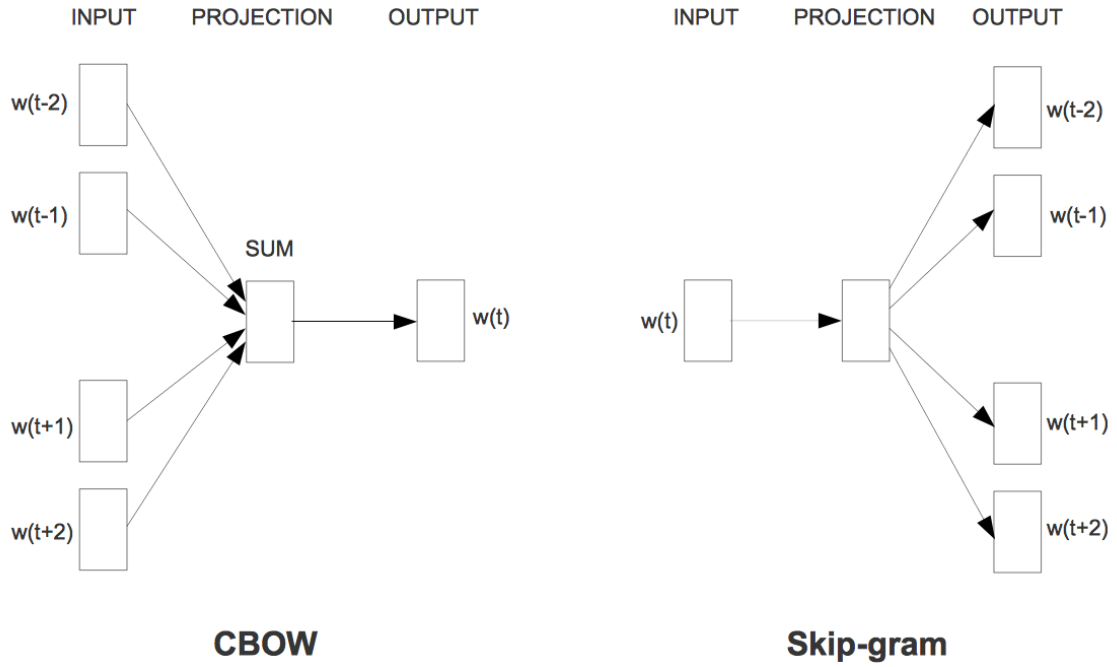

Fig. 2.3 CBOW and Skip-gram model architecture (Source: [29])

As can be seen in figure 2.3, word2vec is implemented as a three layer artificial neural network. It consists of an input layer, a projection layer and an output layer. The projection layer represents the learned embedding matrix after training, since the model is trained to reconstruct the context given the embedding. This means, that the actual information is stored in the projection layer. The model is trained using the back-propagation algorithm with a stochastic gradient descent method. The CBOW model aims to predict from the surrounding context the focus word. It follows the exact opposite approach of Skip-gram. Changing the conditional probability in the loss function to $\Pr\left(w_i|w_{i+j}\right)$, maximises the probability of the focus word given the context.

2.3.4 GloVe

More recently GloVe (Global Vectors) [36] was proposed. GloVe is a word embedding method which generates embeddings using a word-word co-occurrence matrix. This matrix, captures global features, by computing statistics for the whole corpus. This is in direct opposition of word2vec, which learns local features by training on short sentences with a fixed context size.
The word-word co-occurrence matrix is denoted by $X \in \mathbb{R}^{|V|\times|\mathscr{V}|}$ where X_{ij} is the count of how often w_j appears in the context of w_i given a (anti-)symmetric window of length $c \in \mathbb{N}$, for $w_i, w_j \in \mathscr{V}$. The probability, that w_j appears in the context of w_i, is defined as

$$\Pr(w_j|w_i) = \text{Softmax}(X_{ij}) = \frac{X_{ij}}{\sum_{k=1}^{|V|} X_{ik}}. \tag{2.10}$$

To learn the embedding, GloVe uses two embedding matrices $V, \hat{V} \in \mathbb{R}^{|\mathscr{V}|\times D}$, where D is the embedding dimension. The context window can be antisymmetric, therefore we have to differentiate between context word and focus word. $\hat{V}$ stores the context vectors. If the context window is symmetric, the co-occurrence matrix will be symmetric and V will be equivalent to $\hat{V}$.

For training V and $\hat{V}$ are initialised randomly. Then GloVe aims to approximate the log-scaled word-word co-occurrence count for each word pair by the dot product of the corresponding word embeddings. The embedding matrices are updated after each step of a gradient descent method. The objective criterion is given by

$$J(\theta) = \min \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} f(X_{ij})(v_i^T \hat{v}_j + b_i + \tilde{b}_j - \log(X_{ij}))^2, \tag{2.11}$$

where $v \in V$, $\hat{v} \in \hat{V}$, b_i, $\hat{b}_j \in \mathbb{R}$ are biases for the word and the context word and $\theta = \{V, \hat{V}, b, \hat{b}\}$ are the parameters to be learned. The function $f : \mathbb{R} \to [0,1]$, $x \mapsto f(x)$ is a weighting function that reduces the importance of more frequent words that use to encode less information.

2.4 PageRank Based Models

In a graph embedding a proximity measure we want to preserve is how different nodes are related to each other. An algorithm that aims to capture this information is the PageRank

algorithm. For a directed graph $G = (V,E)$, the PageRank algorithm computes a PageRank vector $p \in \mathbb{R}^{|V|}$. The algorithm simulates a random walk on a transition matrix.

Definition 13. *For a directed graph $G = (V,E)$, with $N = |V|$, the transition matrix $P \in \mathbb{R}^{N \times N}$ is defined as*

$$P_{ij} = \begin{cases} \frac{1}{deg^+(i)}, \text{ if } (i,j) \in E, \\ 0, \text{ otherwise.} \end{cases}$$

The initial idea is to model the behaviour of users on the web graph by random walks on the graph. The resulting PageRank vector represents the importance of each page on the web. A user will not always follow the links on a page, but will eventually jump (apparently) randomly to a new webpage. To model this behaviour, we change our matrix to

$$P = \alpha \cdot P + (1 - \alpha) \cdot I, \tag{2.12}$$

for $0 < \alpha < 1$. This gives us two important properties of P. First P becomes a connected matrix, and second P is the transition matrix from an ergodic Markov chain. The ergodic theorem for Markov chain guarantees us then a stationary distribution π. A stationary distribution π, is a vector, satisfying

$$\pi = \pi P. \tag{2.13}$$

The PageRank of a graph is $p = \pi$ and gives us the probability that a user reaches the website i with probability p_i. We can also compute the Personalised PageRank (PPR) using the PageRank algorithm. The PPR models a user, who instead of jumping to a random node, always jumps back to our focus node. The PPR vector indicates how relevant each node is in context of our focus node.

2.4.1 The Bookmark-Colouring Algorithm

Since the PageRank algorithm assigns to each node a value, the vector is dense. In the case of PPR vectors most nodes will not play a role for our focus node, but get a value assigned. The Bookmark-Colouring Algorithm (BCA) provides a fast way to approximate a sparse PPR vector called Bookmark-Colouring Vector (BCV) [4]. The idea of the BCA, is to inject a unit amount of paint in to our focus node and then let the paint flow through the outgoing edges, while retaining a fraction of the paint in each node.

The amount of paint, that is retained in the node, is given by α, where $0 < \alpha < 1$. If the amount of paint flowing through a node is lower than a threshold $\varepsilon > 0$, the paint does not flow further. Since in every step of the algorithm the paint is reduced by α and $\varepsilon > 0$,

Algorithm 1 BC$(b, w, \alpha, \varepsilon)$ Bookmark-Colouring Algorithm

Input: Focus node b, colour amount w, retention coefficient α, and threshold ε.
Output: BCV p
begin
    $p_i = \begin{cases} \alpha, \text{ if } i = b, \\ 0, \text{ otherwise} \end{cases}$
    if $w < \varepsilon$ or $\deg(b) = 0$ **then**
        return p
    end if
    for $(b, j) \in E$ **do**
        $p = p + BC(j, (1-\alpha) \cdot \frac{w}{\deg^+(b)}, \varepsilon)$
    end for
    return p
end

the algorithm stops after a finite amount of steps. It can be shown, that if $\varepsilon = 0$ the BCV is equal to the PPR vector. Figure 2.4 demonstrates how the BCA works on a simple linear graph. We first inject $w = 1$ amount of red colour into the first node, and let it flow through the graph. We reduce the amount of colour that flows further in each step by a factor of $1 - \alpha = 0.5$. After three steps, we have $1 \cdot (1-\alpha)^3 = \frac{1}{8} < 0.2 = \varepsilon$ that flows to the last node. The stopping criterion is met and the algorithm terminates.

Fig. 2.4 Bookmarking-Colouring Algorithm on simple linear graph with $(\alpha, \varepsilon) = (0.5, 0.2)$ and unit amount of red colour injected at the beginning.

2.5 Named Entity Recognition

The task of Named Entity Recognition (NER) is to extract structured information from unstructured text. Given a sequence $w_1^N = (w_1, \ldots, w_N)$ the idea is to find a mapping from each word to one or more classes $C = \{C_1, \ldots, C_K\}$. A NER task could be to recognise entities of an RDF graph in a set of documents and linking the recognised words to the RDF identifier. Naive NER systems use a simple dictionary approach to perform entity matching. For each string in the text, the NER system tries to match the string with an entity. More sophisticated heuristics and handcrafted rules can be introduced to cope with the complexity of natural language relying on linguistic expert knowledge. Another approach

is to use statistical methods such as hidden Markov models (HMM) or ANNs [62]. The HMM is trained with the Viterbi algorithm, to maximise the likelihood that a given string is correctly labeled. These models do not rely on expert knowledge but are dependent on available data for training. NER has several applications for graph embeddings. For example it provides a method to extract additional information from an external data source. This is done by disambiguating entities and relationships. Those relationships can then be added to the knowledge graph .

Chapter 3

Related Work

In section 3.1 we introduce classical graph embedding models, such as node2vec and TransE. Those models do not incorporate graph attributes and operate solely on the graph structure. In section 3.2 we present several state of the art models, that combine attributes with graph structure information. The last part of the chapter lists requirements for graph embedding.

3.1 Classical Graph Embedding Models

In this section we present classical graph embedding models and techniques. We begin with models based on matrix factorisation. We introduce recent models using techniques such as random walk on graphs and methods from natural language processing.

3.1.1 Factorisation-based methods

Early work on graph embedding focuses on the factorisation of different matrix representations for a given (un-)directed graph $G = (V, E)$ such as the Laplacian or the adjacency matrix of the graph. Using the rows or columns of the adjacency matrix as latent features for the nodes in the graph is not sufficient. One reason is that the embedding vectors become large for real-world networks and the second problem is that they encode poorly the information contained in the graph.

Locally Linear Embedding and Laplacian Eigenmaps

Locally Linear Embedding (LLE) [48] and the Laplacian Eigenmaps (LE) [3] are two prominent factorisation methods. The idea is that surrounding nodes provide a description of the focus node. Both perform a dimensionally reduction to create embeddings, compensating

the sparsity of the adjacency matrix.
Having a data matrix $X \in \mathbb{R}^{N \times N}$, LLE follows a two step approach. In the first step,

$$J_1(W) = \sum_{i=1}^{N} ||X_i - \sum_{j=1}^{N} W_{ij} \cdot X_j||_2^2 \tag{3.1}$$

with $\sum_{j=1}^{N} W_{ij} = 1$ is minimised. Each vector is described as the weighted linear combination of its surrounding vectors. In the second step

$$J_2(Y) = \sum_{i=1}^{N} ||Y_i - \sum_{j=1}^{N} W_{ij} \cdot Y_j||_2^2 \tag{3.2}$$

is minimised where $Y \in \mathbb{R}^{M \times M}$, $M \ll N$, by solving an Eigenvector problem. Laplacian Eigenmaps constructs a graph of the K-Nearest Neighbours of each vector. The weights are computed using an arbitrary kernel function, so that $W_{ij} = \phi(X_i, X_j)$, where ϕ is the kernel. The objective function minimises

$$J(Y) = \sum_{i,j=1}^{N} (Y_i - Y_j) \cdot W_{ij} \tag{3.3}$$

by solving an Eigenvector problem.

HOPE

High-Order Proximity preserved Embedding (HOPE) [34] is a model operating on a directed, weighted graph $G_{\text{weigh}} = (V, E, f)$. For $S \in \mathbb{R}^{N \times N}$ with $N = |V|$, S_{uv} describes the second-order proximity of $(u, v) \in E$. S can for example be the PPRs of all nodes, the Katz Index [20] or the Adamic/Adar [1] score. We observe, that the similarity matrix of those measures can be factorised into two matrices, $S = M_1^{-1} \cdot M_2$. For PPR we get

$$S = (I - \alpha \cdot P)^{-1} \cdot (1 - \alpha) \cdot I. \tag{3.4}$$

The objective function of HOPE is defined as

$$J(\theta) = \|S - U^s {U^t}^T\|_2^2, \tag{3.5}$$

where U^s, U^t are the embedding vectors. The approximation of the embedding vectors can be efficiently computed using Singular Value Decomposition (SVD) [56].

3.1.2 Random Walk Based Methods

Based on the results on semantic relation tasks in NLP, several models for graph embedding aim to adopt word embeddings techniques to compute graph embeddings. Especially GloVe and word2vec which were introduced in section 2.3 perform well on those tasks. We will first introduce DeepWalk and node2vec which use the Skip-gram model for graph embedding. Secondly, we present KGloVe, which is based on GloVe.

DeepWalk

The first model is called DeepWalk [37]. The idea of DeepWalk is to generate 'sentences' from a graph $G = (V,E)$. This allows us to use word2vec. The sentences are generated by performing short random walks of length t on the graph. A random walk of length $t \in \mathbb{N}$ is a path $v_1^N = (v_1, \ldots, v_N)$ where $v_i \in V$. For two consecutive nodes v_j, v_{j+1} in v_1^N we have an edge $(v_j, v_{j+1}) \in E \subseteq V \times V$ in G. First DeepWalk performs for each node in the graph $\gamma \in \mathbb{N}$ random walks. Then it trains a Skip-gram model on the corpus of sentences. The word embeddings from the Skip-gram model are used as graph embeddings. Using random walks, DeepWalk explores higher proximity orders of each node, since for every walk parts of the neighbourhood of each node is summarised in a sentence. As Skip-gram aims to estimate the context for a node, the embedding of the node will represent the local neighbourhood of a node.

node2vec

node2vec [17] is a generalisation of DeepWalk that explores biased random walks. node2vec introduces two parameters to control the random walk. The return parameter p and the in-out parameter q. The return parameter controls the probability of returning to a node just visited before and the in-out parameter balances the walk between Breadth-first search (BFS) and Depth-first search (DFS). Lets assume we are in the middle of a random walk and we jumped from node t to u. The probability of jumping from node u to v is defined as

$$\Pr(v|u) = \alpha_{pq}(t,v) \cdot P_{uv}, \tag{3.6}$$

where P is the transition matrix of the graph. α is called the search bias and is defined as

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p}, \text{ if } d_{tx} = 0 \\ 1, \text{ if } d_{tx} = 1 \\ \frac{1}{q}, \text{ if } d_{tx} = 2 \end{cases} \tag{3.7}$$

We define d_{tx} as the shortest path between the previously visited node and the next node. If p is large, the probability of returning to the same node again is low. If q is large, nodes further apart from our previously selected node, will be assigned a lower probability. This results in a random walk resembling BFS. For $p = q = 1$ we get the DeepWalk model. The difference between node2vec and DeepWalk is that node2vec makes an informed choice on how to explore the neighbourhood of a node.

RDF2Vec

RDF2Vec [41] is a model that creates embeddings for RDF graphs. RDF graphs are knowledge graphs which have labeled nodes and edges. Instead of using random walks to explore the sub-structures of the graph, it uses Breadth-first search (BFS) with search-depth t. An improvement upon BFS is proposed, using Weisfeiler-Lehmann Subtree Graph Kernels [50]. Note, that with RDF2Vec in comparison to node2vec, we get an embedding for nodes and edges. After generating the training corpus, the model trains node and edge embedding vectors using the CBOW word-embedding model.

KGloVe

Instead of exploiting local structures as in RDF2Vec and DeepWalk, KGloVe [12] aims to capture the global structure of the graph. KGloVe uses GloVe to obtain graph embeddings. GloVe relies on the definition of a context between nodes to be able to compute a co-occurrence matrix. In KGloVe the context of a node is described using the PPR of the node. To compute the PPR for each node takes a considerable amount of time. Therefore KGloVe approximates the PPR for each node in the graph using BCA. BCA is presented in section 2.4.1. The resulting BCV is given as

$$p^{(v)} \in [0,1]^{|V|},\ v \in V \text{ with } \sum_{j=1}^{|V|} p_j^{(v)} = 1,$$

where an entry $p_j^{(v)} = 0$ means, that the node v_j is not connected to v and therefore not important. $p_j^{(v)} = 1$ means, that the node v_j is important for node v. Furthermore biased walks are introduced instead of uniform distribution of paint in each step. For example the inverse frequency of an entity in a triple is taken into account. The idea of the bias is that less frequent nodes contain more information then frequent nodes that are adjacent to almost all nodes in the graphs. The context of GloVe is computed using a symmetric window. The BCA algorithm operates on directed edges. This means, that a node that has a directed edge

to our focus node but no edge back, will be not in the context of our focus node. To remedy this problem KGloVe computes two rounds of the BCA, where the last round is computed on the graph with reversed edges. The two BCVs per nodes are merged to a single BCV, which is used for the co-occurrence matrix. The co-occurrence matrix is used as an input to GloVe to generate the node embeddings.

3.1.3 Models based on Affine Projections

TransX represents a family of graph embedding models (Trans{E, H, R, D, Sparse}, TorusE), that maps every node and edge in the graph into a latent feature space. The relations between entities are encoded by geometric operations such as translation and rotation. TransX models operate on triples consisting of a head h, a tail t and a relation r. Triples (h,r,t) are common representation of data in knowledge graphs. The simplest model is TransE [7], which maps every node and edge in the graph to a vector in a low dimensional vector space. For each triple (h,r,t) in the graph, TransE learns an embedding, so that for the corresponding vectors the equation $h+r \approx t$ holds. We do not explicitly differentiate between an entitiy and its vector. It should be clear given the context. The relation between two entities is encoded as a translation in the semantic space.
Assume a graph with an entity h, connected to t_1 and t_2 by r. TransE fails to encode this relation, since $t_1 \approx h+r \approx t_2$ but $t_1 \neq t_2$. This means that the performance of TransE decreases if there are many 1-to-n, n-to-1 or n-to-n relations in the graph.
To improve upon TransE, different models have been proposed, namely TransH [59], TransR [25], TransD [18] and TorusE [14] which encode relatedness by using different geometric operations such as rotation, translation and projection. All models use Stochastic Gradient Descent (SGD). The models aim to minimise the error of the following objective function with slight variations on how h,t and r are embedded

$$J(\theta) = ||h+r-t||_2^2, \tag{3.8}$$

where $\theta = \{E,R\}$ and $E \in \mathbb{R}^{N\times D}$, $R \in \mathbb{R}^{T\times D}$ are the embedding matrices of the entities and relation vectors. N is the number of entities in the graph and T the number of relations. In this thesis TransE is going to be used as a baseline model, due to it's simplicity and consistent use as a baseline model in other papers.

3.2 Including Auxiliary Information into Graph Embeddings

In recent years the attention in research has shifted towards how to incorporate attribute information into graph embeddings. Graph attributes offer another perspective into relations between graph entities. Including auxiliary information describing relations in the graph, will likely improve the performance on tasks such as node classification and link prediction. Attributes can have different types, they can be images, text or numeric values. Given the different types it is hard to realise a holistic approach including every possible type. Therefore most approaches existing today focus on a specific type of literal. In this section we give an introduction into different attempts to incorporate auxiliary information into graph embeddings. We try to cover a variety of different techniques and models.

3.2.1 Extending Classical Models

LiteralE

LiteralE [23] is a model, that functions as a plug-in for existing graph embedding models. LiteralE operates on a knowledge graph G with attribute matrix $X \in \mathbb{R}^{K\times|V|}$. If entity i in the graph has no attribute j, we set X_{ij} to zero. $X_i \in \mathbb{R}^K$ is the attribute vector of entity i.
The idea is to introduce a learnable function $g : \mathbb{R}^D \times \mathbb{R}^K \to \mathbb{R}^D$, that maps attribute and embedding vectors into the same semantic space. The image of a vector under g is called literal-enriched vector. To train the model, we use the objective function $J(\theta)$ of the existing model. Instead of using the triple (h,r,t), we substitute $h = g(h,X_h)$ and $t = g(t,X_t)$. Note, we assume that attributes are represented as vectors.
As an example, the objective function of TransE is

$$J(\theta) = ||h+r-t||_2^2. \tag{3.9}$$

When using LiteralE we get

$$J(\theta) = ||g(h,X_h)+r-g(t,X_t)||_2^2. \tag{3.10}$$

Conv

Conv [53] is a model that operates on knowledge graphs and uses an additional knowledge base to enhance existing models. It achieves that through extracting textual relations from the knowledge base. The relations are extracted using pre-defined patterns. Figure 3.1

demonstrates how the textual relations are extracted.

Barack Obama is the 44th and current President of United States.

$$\text{SUBJECT} \xleftarrow{\text{subj}} \text{president} \xrightarrow{\text{prep}} \text{of} \xrightarrow{\text{obj}} \text{OBJECT}$$

Fig. 3.1 Extraction of textual relations for a given text (Adapted from [53]).

Given the relation, Conv does a lookup for each word in the relation in a word embedding matrix V. It convolutes the vectors using a convolution kernel C, so that we get

$$p_i = \tanh\left(C \cdot V_{i:i+3} + b\right). \tag{3.11}$$

In the last step, max-pooling is performed to obtain $r = \max\left(p_i\right)$. r represents the textual relation in a single vector. The CNN is jointly trained with an existing model such as TransE. Let $T_{\text{KG}}, T_{\text{KB}}$ be the set of triples in the knowledge graph respectively the knowledge base and f the scoring function of the existing model. Then the objective function is given as

$$J(\theta) = L_{KG}(T_{\text{KG}}) + \alpha \cdot L_{KB}(T_{\text{KB}}) + \beta ||\theta||_2^2, \tag{3.12}$$

where

$$L(T) = -\sum_{(h,r,t)\in T} \log \Pr(h|r,t) + \log \Pr(t|r,h) \tag{3.13}$$

and Softmax is used to compute the conditional probability. The network parameters are regularised with l_2 regularisation. Additionally, the extracted triples are added to the knowledge graph.

3.2.2 Models based on Deep Learning

CANE

The Context-Aware Network Embedding (CANE) [54] is a sophisticated model, that incorporates textual node information. For each node v in the graph, we are given a sequence of words $w(v)_1^N = (w_1, \ldots, w_N)$ of arbitrary length N. CANE generates for every node v, a structural embedding v_{struct} capturing first order proximity and a text embedding vector $v_{text}^{(u)}$ depending on a context node u. This introduces a mutual attention mechanism between two nodes.

The objective function is composed of a structural and a text based objective function, which

is split into three functions

$$J(\theta) = \sum_{e \in E} L_{struct}(e) + L_{text}(e) = \sum_{e \in E} L_{struct}(e) + \alpha \cdot L_{tt}(e) + \beta \cdot L_{st}(e) + \gamma \cdot L_{ts}(e). \quad (3.14)$$

α, β and $\gamma \in \mathbb{R}$ are hyper-parameters and $L(e)$ is defined as

$$L_{xy}(e) = A_{ij} \cdot \log \text{Softmax}(v^x, u^y), \quad (3.15)$$

where A is the adjacency matrix and $x, y \in \{s,t\}$. We notice, that the model captures the first-order proximity by maximising the probability of two adjacent nodes. At the same time, it pulls structural and textual embeddings closer together for adjacent nodes.
To generate embeddings from text sequences regarding node context, CANE uses a Convolutional Neural Network (CNN) with an attention mechanism. Let u, v be two nodes in the graph, with $A_{uv} = 1$. In the first step, we do a look up for each word in the sequences $w(v)_1^N, w(u)_1^M$ to get the embedding vector sequences $\tilde{w}(v)_1^{N_1}, \tilde{w}(u)_1^{N_2}$. This is done by generating a one-hot encoding vector for each word in the sequence and multiplying each vector with the word embedding matrix.
We then use a convolution kernel C to generate $P_i = C \cdot \tilde{w}(v)_{i:i+l-1} + b$, same for $\tilde{w}(u)$ to get Q_i. To get the attention vectors, we compute a weighted correlation score between P and Q given as

$$F = \tanh\left(P^T W Q\right), \quad (3.16)$$

where W is the attention matrix. We calculate the row and column wise mean of F to get the importance of each word given the other sequence and normalise the mean vectors using the Softmax function

$$a^x = \text{Softmax}\left(\frac{1}{N_i}\sum_i F_i\right), \quad x \in \{p,q\}, i \in \{1,2\}. \quad (3.17)$$

Finally, we generate the context dependent text vector using the attention weights

$$u_t^{(v)} = Pa^p, \quad v_t^{(u)} = Qa^q. \quad (3.18)$$

To get the final context aware node embedding, we concatenate the structural and textual node embedding vectors. CANE generates for a given node and for each adjacent node a different embedding vector.
To train the neural network, CANE uses SGD. We present the architecture of the context aware embeddings in figure 3.2.

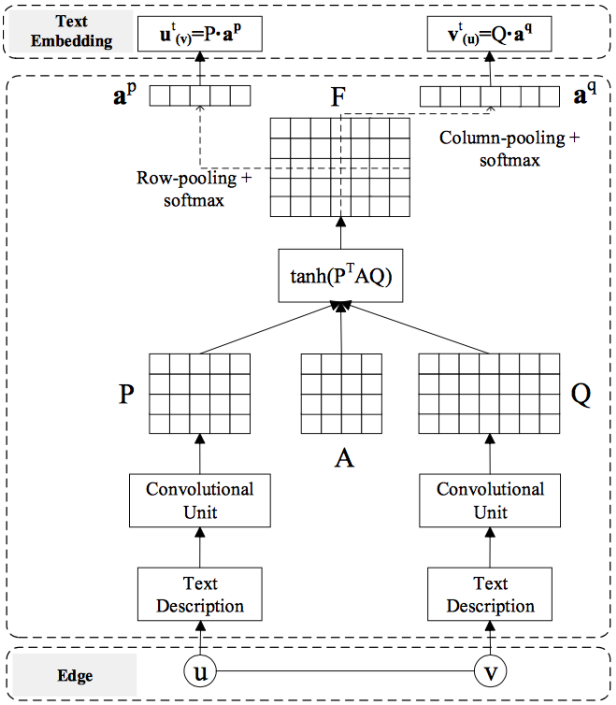

Fig. 3.2 CANE architecture for generation of context aware embedding vectors (Source: [54])

ANRL

The Attributed Network Representation Learning (ANRL) [61] model consists of an auto-encoder f that uses hard parameter sharing to jointly train an embedding on the graph structure with auxiliary information. The decoder minimises the reconstruction error

$$J_{rec}(\theta) = \sum_{i}^{N} ||T(v_i) - f(X_i)||_2^2, \tag{3.19}$$

where v_i is a node in the graph and $T(v_i) = \frac{1}{deg(v_i)} \sum_j^N w_{ij} \cdot X_j$. T encodes the first-order proximity by pulling adjacent nodes closer together. A weight matrix W helps to balance the importance of surrounding nodes. To explore the local neighbourhood, ANRL generates random walks and trains on the Skip-gram objective function. Together we get

$$\begin{aligned} J(\theta) = &- \sum_{i}^{N} \sum_{c \in C} \sum_{-b \leq j \leq b} \log \text{Softmax}(u, y) + \alpha \cdot \sum_{i}^{N} ||T(v_i) - f(X_i)||_2^2 \\ &+ \frac{\beta}{2} \sum_{k}^{K} \left(||W^{(k)}||_F^2 + ||\hat{W}^{(k)}||_F^2 \right). \end{aligned} \tag{3.20}$$

The last term regularises the weights of the auto-encoder, where $||\cdot||_F$ is the Frobenius norm.

DANE

The Deep Attributed Network Embedding (DANE) [15] uses two auto-encoders f^1, f^2 to learn a latent representation of the neighbourhood H^M and the attributes H^X for each node in the graph. The auto-encoders share information via soft-parameter sharing. DANE gets as input to the first network the higher proximity matrix

$$M(t) = P + P^2 + \ldots + P^t, \tag{3.21}$$

where P is the transition matrix of the graph. The second network takes as input the attribute matrix X. The final embedding vectors for each node are given as $H_i^M \oplus H_i^X$. The loss function consists of two parts. First, the reconstruction error needs to be minimised

$$J_{rec}(\theta) = \sum_i^N ||X_i - f^1(X_i)||_2^2 + \sum_i^N ||M(t)_i - f^2(M(t)_i)||_2^2. \tag{3.22}$$

where $N = |V|$ and $f^i = f^i_{dec} \circ f^i_{enc}$. Then, we want to preserve first-order proximity by defining the probability, that two nodes are connected using the sigmoid function

$$\Pr(H_i^{S_1}, H_j^{S_2}) = \text{Sigmoid}(H_i^{S_1}, H_j^{S_2}), \tag{3.23}$$

where $S_i \in \{M, X\}$. This implicates, that H^M and H^X have the same embedding dimension. The objective function is defined as

$$J_{prox}(\theta) = -\sum_{ij}^N A_{ij} \log \Pr\left(H_i^M, H_j^M\right) - \sum_{i,j}^N A_{ij} \log \Pr\left(H_i^X, H_j^X\right). \tag{3.24}$$

To combine the graph data with auxiliary information, soft-parameter sharing is used resulting in

$$J_{soft}(\theta) = -\sum_i^N \left(\log \Pr\left(H_i^M, H_i^X\right) - \sum_{i,j}^N A_{ij} \log\left(1 - \Pr\left(H_i^M, H_j^X\right)\right) \right). \tag{3.25}$$

$J_{soft}(\theta)$ pushes nodes apart which are not in the local neighbourhood, while attribute and node embedding belonging to the same node are pulled towards each other.
Finally, combining all objective functions, we get

$$J(\theta) = J_{rec}(\theta) + J_{prox}(\theta) + J_{soft}(\theta) \tag{3.26}$$

To minimise the objective function, SGD is used to optimise the parameters in θ.

3.2.3 Other Models

There are many more models that incorporate auxiliary information into the graph embedding. For example the Variational Graph Auto-encoder (VGAE) introduced by Kipf and Welling [22] which is based on Graph Convolutional Networks (GCN) [21]. VGAE takes the adjacency matrix such as an attribute matrix of a graph as an input and produces a latent representation of the graphs nodes. It aims to minimise the reconstruction error for the adjacency matrix. Including more models would be out of the scope of this thesis. More graph embeddings techniques, that do or do not incorporate auxiliary information are presented in [35], [8], [58] and [45].

3.3 Requirements

After introducing different embedding models and techniques, we define in this section requirements that a graph embedding model should satisfy. Those requirements are extracted from the models that we presented so far.

- **Dimensionality**: Features should be represented in a dense manner. This reduces the memory requirements as well as the computational effort for algorithms operating on the embeddings.
- **Adaptability**: Different machine learning algorithm require different features of the graph. An embedding model should be generic enough to fit most of the algorithms and should not be tailored to a single algorithm.
- **Input**: A model should be able to process different types of graphs, such as directed and undirected graphs as well as homogeneous and heterogeneous graphs.
- **Embedding output**: Different tasks require different representations of the graph. Therefore a model should be able to generate embeddings with different levels of granularity. For example only node, node and edge, subgraphs and even whole graph embeddings.
- **Auxiliary Information**: For a given attributed graph, a model should generate embeddings that preserve graph data and attribute information in the same semantic space to represent features more densely.

- **Graph Exploration**: A model should combine local and global exploration to embed all properties of the graph topology in one semantic space.
- **Multistep Pipeline**: A model should be trained end-to-end, without additional pre-processing steps.
- **Memory Requirement**: Since graphs, such as social-networks are in general large, a model should use memory resources economically. Most data will not fit in the RAM today, so that this has to be considered, while designing a model.

Chapter 4

Conceptual Approach

In this chapter we present and discuss approaches to the problem of incorporating literals into a graph embedding. We differentiate between two approaches. The first one is to enrich the graph structure with additional information such as additional nodes or edges, using existing graph embedding models. The second approach enriches the embedding itself, either by changing the objective criterion of the model and therefore the way a model learns or by introducing auxiliary information to the model not depending on the graph.

4.1 Enriching the Graph Structure

By enriching the graph structure, we can make use of existing models, without changing the internal working of those methods. Since this approach only changes the input data, it is easier to understand for people already familiar with existing models. On the other side it has its limitations, since we can not directly influence the way a model is learning and representing the features by changing the input data. Since we are working on an arbitrary graph structure $G = (V, E)$, there are only two main parameters that we can control. We can either change the set of nodes V or the set of edges E. By introducing more features to the graph such as a weighting of the edges, we could use more parameters. In this section we focus on a directed graph $G = (V, E)$ with labeled edges and nodes.

4.1.1 Adding Complementary Edges and Nodes

Given a graph $G = (V, E)$ and a partition $C = \{C_1, \ldots, C_n\} \subseteq \mathscr{P}(V)$ of V, where $v \in V$ is in class C_i, if $v \in C_i$ and $\mathscr{P}(V)$ is the power set of V. A classical graph embedding model, ignoring auxiliary information, is not able to incorporate this class information into the embedding. Therefore, we add for each class C_i, a class node u_{C_i}. Next, we add for each node

$v \in C_i$ an (undirected) edge (v, u_{C_i}) to the graph. In chapter 2 we learned, that most models encode first-order proximity. Furthermore, we can assume, that nodes from the same class do not have to be necessarily adjacent. By connecting them using an intermediate class node, we introduce information to the model, that is not given in the topological structure of the graph and would be missed by just taking the first-order proximity into account. It is important to notice, that this method is only applicable if we are in a supervised learning scenario, where the nodes for each class are given. KGloVe profits from this representation, since it uses the Bookmark Colouring Algorithm. By connecting all nodes of the same type to one node, we create a sink, where the paint is collected and can not flow further. An assumption that has to be tested is

$$\hat{u}_{C_i} = \frac{1}{N_i} \sum_{v \in C} \hat{v} \tag{4.1}$$

for $\hat{u}, \hat{v} \in \mathbb{R}^D$ the corresponding embedding vectors and $N_i = |C_i|$.
Instead of connecting nodes through an intermediate class node, we can directly connect nodes of the same class. Each class induces then a fully connected subgraph, a clique.
A problem concerning this approach is scalability. A fully connected graph with N nodes has $\binom{N}{2}$ edges, connecting nodes of the same class with edges will significantly increase the computation time of the BCA when using KGloVe. Therefore we have to be careful, which model to choose. Its computational complexity should not be mainly influenced by the number of edges in the graph. Instead of generating a fully connected subgraph for each class, we can also connect each entity of a class in a cycle. To approach a fully connected subgraph, we can introduce additional forward edges as in depicted in figure 4.1.

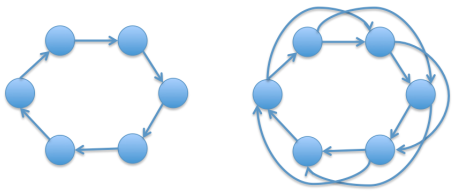

Fig. 4.1 (left) nodes in one class connected in a cycle, (right) nodes connected through forward edges

Instead of connecting a node in the cycle only with its successor, we can connect it with the next $N \in \mathbb{N}$ nodes in the cycle. As N approaches the cycle length, we approach a fully connected subgraph. We denote this approach with ring-N.

4.1.2 Statistical Approach

Usually, we are not given classes for each node in the graph but continuous values, describing features of the node or edge. Using the same approach as in the last section would lead to a large number of classes with few nodes in each class. To be able to use the same approach as in the section above, we need to automatically discretise the continuous features. The idea is that entities with features that are close to each other, tend to share a relation. As an example think about the birth year of a person. The person will be more likely to share features with people in the same age class. To automate the process of introducing new edges or nodes, we use statistical methods such as K-means or a Gaussian Mixture Model (GMM). The number of clusters K is a hyper-parameter, that needs to be evaluated experimentally. We can either cluster the whole feature vector or use clustering for each feature alone or a subset of features. The latter resulting in different clusters per feature. If we cluster each feature, we introduce more edges to the graph, but reveal more connections between entities. For knowledge graphs, we can even add new relations based on the clustering to differentiate between new edges.

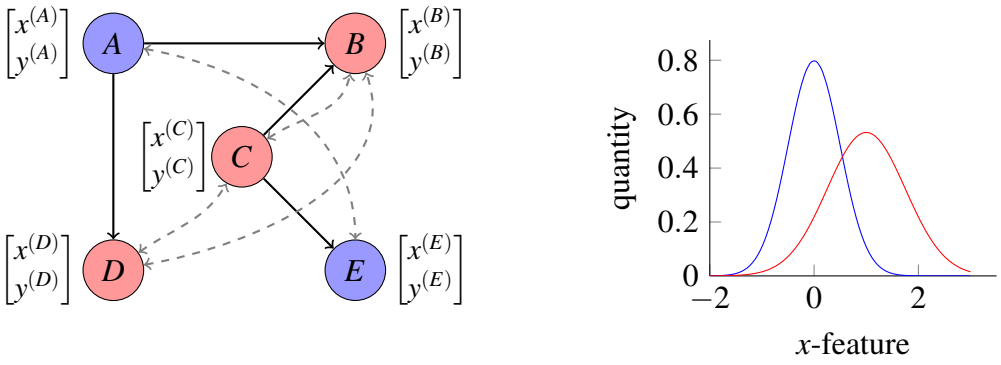

Fig. 4.2 (left) For an attributed graph, we cluster the first feature of each vector using a Gaussian Mixture Model (right). We then connect the nodes which have their first feature in the same cluster by undirected edges represented by grey edges.

4.2 Enriching the Graph Embedding

In section 3.2.1 we propose, motivated by the approach of LiteralE [23], different approaches to directly enrich existing embedding vectors instead of adjusting the graph structure.

4.2.1 Concatenating Attribute Vectors

In an attributed graph, we assign different attributes to each node. For example in a social network we haven given text describing the entity, numerical values such as birth year and age or binary values, such as the relationship status. The goal is to represent all of those attributes for one node including the graph embedding as one vector. We presented different embedding strategies for different types of attributes, word2vec for text or simply the attribute itself when given numerical values. Lets assume we are given a node embedding $x \in \mathbb{R}^{D_1}$ and attribute embeddings $y_j \in \mathbb{R}^{D_j}$, where $D_j \in \mathbb{N}$, for $j \in \{1,\dots,N\}$. We propose a way to merge those vectors by concatenating them. This results in

$$x_{\text{conc}} = \bigoplus_{j=1}^{N} y_j \oplus x \in \mathbb{R}^{\sum_j D_j}.$$

Although, this is a simple approach, we embed all necessary information into one vector. The disadvantage of this approach is that the dimension $D = \sum_j D_j$ of the final vector becomes large when concatenating multiple features. Furthermore, the information that is encoded in the different vectors is not shared between the entries in the vectors.

4.2.2 Merging Attribute and Graph Embedding Vectors

The approach in the last section comes with two flaws. The first one is the growing dimension size. With growing dimension, the computational effort increases for algorithms operating on the embeddings. The second one is that we do not merge the information into a single feature space, since the attributes still reside in their own vector space. When comparing features between vectors of different nodes using the approach above, it would be the same as simply comparing the vectors feature wise.

The goal of this approach is to merge the information, so that each entry in the vector incorporates some information of the other entries. To remedy those flaws we propose a model based on an auto-encoder architecture to efficiently merge all information into a low-dimensional vector space.

Let f_{enc} be the encoding part and f_{dec} be the decoding part. The proposed auto-encoder is trained given the embedding vector $x \in \mathbb{R}^D$ to minimise the objective function defined as

$$J(\theta) = \sum_{i=1}^{|V|} \left\| f_{\text{dec}} \circ f_{\text{enc}}(x_{\text{conc}}^{(i)}) - x_{\text{conc}}^{(i)} \right\|_2^2. \tag{4.2}$$

The reconstruction task allows us to perform a dimension reduction of the feature vector. As a result the feature vectors become dense.

4.2.3 Incorporating Textual Literals Into KGloVe

This model is an extension of the KGloVe model presented in 3.1. It is motivated by the observation, that there are several knowledge graphs that contain short textual abstracts, describing the entities in the graph. Nodes containing attribute information about an entity are called literals. The idea is to incorporate textual literals into the embedding of KGloVe. The proposed model takes as input a knowledge graph and additional unstructured text if available. In a first step, a named entity recognition system extracts all abstracts from the knowledge graph and concatenates them into one file. On the other side, we use KGloVe to compute a co-occurrence matrix for the graph. As a result we get an entity and predicate word list, containing all the identifiers for the entities and the predicates in the graph. With the entity and predicate word list a named entity recognition step is performed on the abstracts. Every matched word is replaced by the corresponding identifier of the matched entity. The NER system adds all words in the abstract that were not matched in the entity recognition step to the existing entity and predicate list. The word list is then used together with the tagged abstracts as an input to GloVe to generate a co-occurrence matrix for the text.

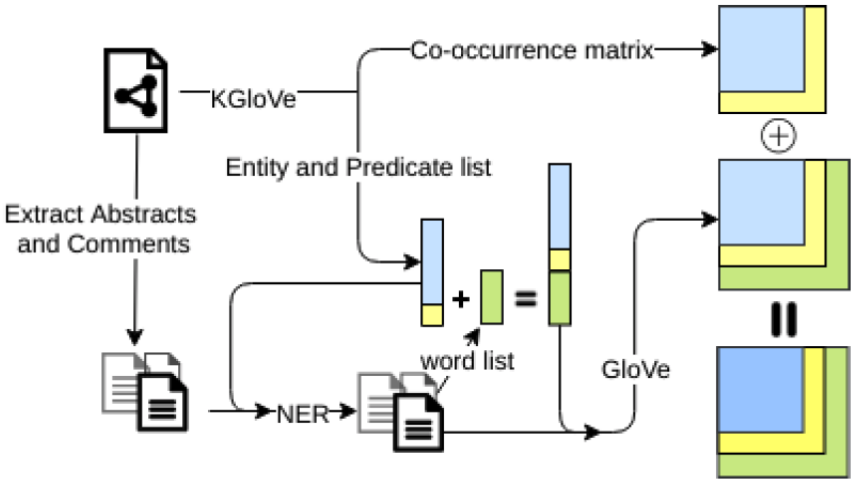

Fig. 4.3 Co-occurrence matrix creation pipeline (Source: [11])

The underlying idea behind this process is, that we compute additional co-occurrence statistics from the text. Many knowledge graphs are incomplete and therefore distort the actual co-occurrence count. Adding additional statistics to the matrix can be seen as countermeasure against the distortion. The problem now is how to merge those statistics. The approximated PPRs of KGloVe are not directly comparable with simple co-occurrence counts. In a first

step we normalise the GloVe co-occurrence matrix. This is done by dividing all entries in the matrix by the 100th largest entry. This implies that there will be some entries in the matrix, that have values greater than one. The problem is that the first entries have such a high count that if used for normalisation they would reduce nearly all entries close to zero. This is due to Zipf's law [38]. By normalising the matrix we approach the characteristics of the PPRs. PPRs are stochastic vectors and therefore their entries sum up to 1. After normalising, both matrices are merged by simply adding each entry with the corresponding one in the other matrix. To generate the embedding vectors GloVe is applied to the merged co-occurrence matrix. The process of generating the merged co-occurrence matrix is presented in figure 4.3.

4.2.4 Re-inject KGloVe

In this section we propose a theoretical framework for incorporating attributes in KGloVe based on attribute similarity. The purpose of re-inject KGloVe is to overcome a flaw of the Bookmark-Colouring Algorithm applied on disconnected graphs. The problem is that for real-world graphs, we can not assume, that the graph is connected. Since the idea of the BCA is based on paint, that flows from node to node through edges, the paint will not be able to flow, if the nodes are not connected. Therefore those nodes will have zero entries for the corresponding nodes in their BCVs. We discussed in section 1.2 that only relying on the topological structure of the graph can lead to dismissing nodes that would match perfectly according to their attribute similarity. To remedy this problem, we propose a re-injection model. The idea of the algorithm is to save a fraction of the paint to distribute to nodes with similar attributes. We introduce a hyper-parameter $\gamma \in (0,1)$, that indicates the amount of paint retained for the attributes. The algorithm consists of three steps. In the first step for each node in the graph the BCV is computed, using the Bookmark-Colouring Algorithm. Secondly, we compute the distance between the attributes for each node, based on a pre-defined metric. A disadvantage is, that this metric has to be defined for each type of attribute separately. In the last step, we choose the K nearest nodes according to our distance metric, inject the saved amount of paint into each node and re-compute the BCV vector accordingly. This allows the BCA to let paint flow from one node to another even if they are are not connected. It solves the problem of disconnected nodes being set to 0 in the PPR. On the other hand, attribute information is taken into account, since the paint is distributed to the K-nearest neighbours with similar attributes. Additionally the BCA captures the neighbourhood of the node that we jumped to. This allows us to discover relations between previously dismissed entities and our focus node based on their attribute similarity.

The new BCV $\tilde{p}$ is defined as

$$\tilde{p}^{(v)} = (1-\gamma) \cdot p^{(v)} + \frac{\gamma}{K} \cdot \sum_{k=1}^{K} p^{(k)}. \tag{4.3}$$

This formula is compliant to the original BCA formulation. We remove $1-\gamma$ of paint of $p^{(v)}$ and inject into each node $\frac{\gamma}{K}$ amount of paint. The amount of colour has to be split between the nodes, therefore we divide by K. To compute for each node the BCV after injection the paint is equal to multiplying $p^{(k)}$ with $\frac{\gamma}{K}$ according to the recursive BCA formula. Since we already computed the BCV, the computation cost of the re-inject model depends on the calculation of the distances between attributes.

Chapter 5

Experiments

In this chapter we experiment with the methods presented in chapter 4. We start by presenting the relevant datasets in section 5.1.1. To be able to measure the performance of our models, we introduce in section 5.1.2 the link prediction problem [51] and the node classification task [5], which are commonly used methods to evaluate graph embeddings [35]. Additionally, we define metrics on those tasks. The models based on adjusting the graph are evaluated in section 5.3, those aiming to enrich existing embeddings in section 5.4.

5.1 Evaluation

To evaluate the proposed models we need to be able to compare them with the state of the art models presented in chapter 3. For reasons of comparability, we introduce in this section commonly used datasets and evaluation methods. We start by presenting the datasets.

5.1.1 Datasets

In this thesis we work with two different types of graphs. The first type are knowledge graphs, with labeled nodes and edges. The second type are citation networks. In a citation network each node represents a paper. A directed edge between two nodes represents a citation.

Knowledge Graphs

The first dataset is FB15k which was introduced in [7], it is a subset of the Freebase knowledge base [6] which belongs to Google but is no longer maintained and was merged into Wikidata. We use FB15k, since it is used by multiple models such as TransE and LiteralE to evaluate the embedding performance. The second dataset is YAGO3-10 which is presented

in [13]. The dataset is a subset of YAGO3 [40] and contains triples describing attributes of people and places such as gender, birthplace and location. Each entity in YAGO3-10 has at least 10 relations. Both datasets consists of a training, test and validation set. Additionally, we use YAGO3-10+, which is a subset of YAGO, divided into a training and test set. The dataset provides facts with numerical values such as birth year and date of creation for places. It is used as an extension to YAGO3-10 to experiment with additional literal information. In a pre-processing step, we remove all entities from the test and validation set, which do not appear in the training set. This guarantees, that for each entity exists an embedding vector after training. For example, for YAGO3-10 we removed 18 triples in the test and 22 triples in the validation set. The afore mentioned datasets are summarised in table 5.1. For DBPedia, we report the sum of the entities and relations.

	#Entities	#Relations	#Triples
FB15k	14,951	1,345	592,213
DBPedia	8,876,675	8,876,675	83,206,114
YAGO3-10	123,143	37	1,089,040
YAGO3-10+	85,941	5	111,406

Table 5.1 Overview of the RDF graphs used for evaluation

Most entity recognition systems link against DBPedia. Additionally, KGloVe is only evaluated on DBPedia. Therefore, we use the DBPedia[1] knowledge graph to perform the experiment on incorporating literals into KGloVe embeddings. The DBPedia graph contains among others for each entity in the graph a short abstract describing the entity. Furthermore DBPedia provides an interlinking[2] between the DBPedia graph and the datasets presented in table 5.1. The interlinking is used to evaluate the model on other datasets.

Citation Networks

We use three more datasets[3] to evaluate the performance of the proposed methods. Namely CiteSeer, Cora [26] and PubMed [32]. All of these datasets represent citation networks. Each node in the graph represents a paper. A directed edge between to nodes indicates, that the tail is citing the head. Each paper is assigned exactly one label, representing the category in which the paper was published. The PubMed dataset contains papers about diabetes type 1, 2 and experimental treatments. Cora and CiteSeer represent citation networks in the domain of artificial intelligence and machine learning. Each node is assigned a Bag-of-Words vector.

[1] https://wiki.dbpedia.org/downloads-2016-10
[2] https://wiki.dbpedia.org/services-resources/interlinking
[3] https://linqs.soe.ucsc.edu/data

The BoW vector indicates if a paper contains a word from a dictionary or not. The dictionary is constructed by first removing stop-words from all papers, then stemming the words and removing those that occur less then ten times. Table 5.2 presents the statistics of the datasets.

	#Entities	#Edges	#Attributes	#Categories
CiteSeer	3,312	4,732	3,703	6
Cora	2,708	5,278	1,433	7
PubMed	19,717	44,338	500	3

Table 5.2 Overview of the paper citation networks used for evaluation

We removed all entities that do not have a Bag-of-Words vector or cite themselves, since it makes no sense that a paper cites itself. For example, for CiteSeer we removed 16 entities. The distribution of nodes between the categories is presented in figure 5.1

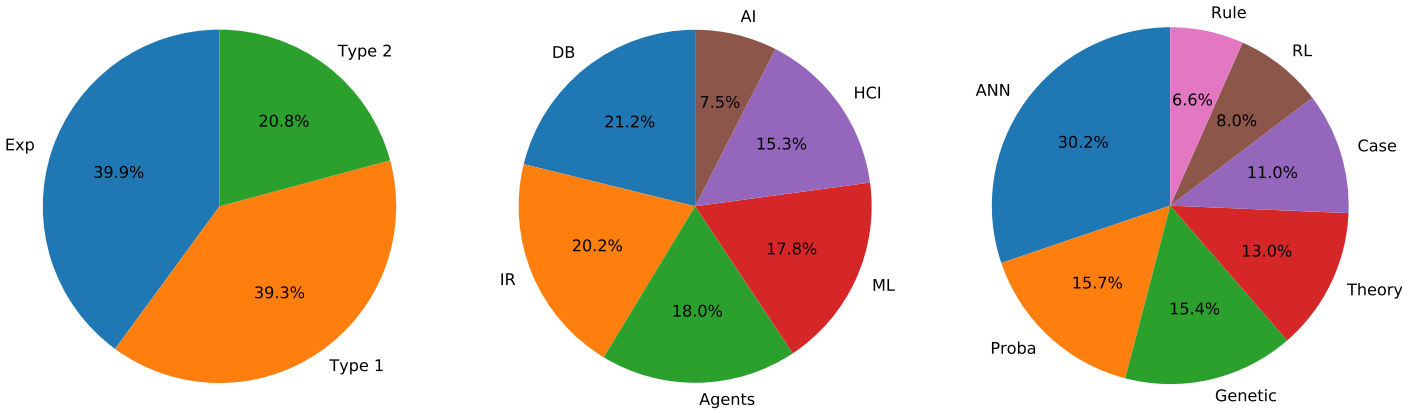

Fig. 5.1 Distribution of categories in citation networks. PubMed (left), CiteSeer (middle) and Cora (right).

5.1.2 Evaluation Tasks and Metrics

There are numerous ways to evaluate the performance of a graph embedding model. For example graph compression, community detection, graph visualisation and node recommendation [8], [35]. In this thesis we use the node classification task and the link prediction problem, which are two common tasks to evaluate the performance of our graph embedding models. We present both tasks and corresponding metrics in this section.

Link Prediction

For a given graph $G = (V, E)$ the link prediction problem is to infer missing or new links in the graph, given the intrinsic information of the graph. This problem is motivated by different

applications in real-world scenarios. For example recommending new friends to an user in a social network [51] as introduced in section 1.2 or inferring new knowledge in a knowledge graph [27].

Predicting missing links in a graph is hard [46]. We can assume that most real-world graphs are sparse [9] and therefore $|E| \ll |V|^2$. The set of possible missing links is then defined as $M_E = V \times V - E$. Choosing uniformly a random edge, the probability of choosing the right edge from $e \in M_E$ is

$$\Pr(e) = \frac{N}{|V|^2 - |E|}, \tag{5.1}$$

where $N \ll |V|^2$ is the number of correct edges. For a large graph the probability becomes small due to the fact that the difference between $|E|$ and $|V|^2$ tends to increase. Luckily, we can make use of the inherent information in the graph, so that we do not have to draw uniformly from the set of missing links. Note, that predicting links in a knowledge graph is more difficult since we need to choose the right relation and not only predict if there is an edge or not.

Algorithm 2 Template for Link Prediction Evaluation

1: **Input:** Graph $G = (V, E)$
2: **Output:** Evaluation score for link prediction
3: **begin**
4: Generate $G = (V, E')$, where $E' \subseteq E$
5: **for** $(h, r, t) \in G'$ **do**
6: Score (h', r, t) and $(h, r, t') \; \forall \, h', t' \in V$
7: **end for**
8: Compute rank R_h, R_t according to scoring function for each $(h, r, t) \in G'$.
9: Compute evaluation score given R_h, R_t, $\forall (h, r, t) \in G'$
10: **end**

To evaluate the performance on the proposed task, we use algorithm 2. To score each triple, we use a heuristic function $f : V \times E \times V \to \mathbb{R}$ and compute the score for each triple (h, r, t). We create corrupted triples (h', r, t) and (h, r, t') and score them as well. Let R_h and R_t be the rankings, where R_h describes the rank of the correct triple (h, r, t) and h is variable in respect to f. Analogous for R_t. The goal is that the correct triple is ranked first. If a corrupted triple is an actual triple in the training set, we have a problem since a perfect model would rank both first. Therefore we differentiate between filtered link prediction, where we only take corrupted triples in account that do not appear in the training or test set and raw link prediction [7]. Given the rankings of our correct triple, we can compute different metrics to evaluate our embedding.

Definition 14. *Let R_h and R_t be the ranks for the triple (h,r,t) and $N \in \mathbb{N}$. hits@N is the defined as*

$$HITS(N) = \frac{1}{2 \cdot |T|} \sum_{(h,r,t) \in T} \mathbb{1}_{[1,N]}(R_h) + \mathbb{1}_{[1,N]}(R_t). \tag{5.2}$$

Definition 15. *Let R_h and R_t be the ranks for the triple (h,r,t). Then the mean rank is defined as*

$$\text{MR} = \frac{1}{2 \cdot |T|} \sum_{(h,r,t) \in T} R_h + R_t. \tag{5.3}$$

T is the set of test triples. We define the Mean Reciprocal Rank (MRR) as the mean of the sum of the reciprocals of R_h, R_t.
As a baseline for comparison we introduce two link prediction heuristics.

Definition 16. *Let $G = (V,E)$ be a graph, $v,u \in G$. We define the Jaccard Coefficient as*

$$J(u,v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} \tag{5.4}$$

The Jaccard Coefficient measures the similarity of the neighbourhood between two nodes. If $u = v$, then $J(u,v) = 1$. Two nodes with similar neighbourhood will therefore have a high probability of being linked. Another score is Adamic/Adar or Frequency-Weighted Common Neighbours [1].

Definition 17. *Let $G = (V,E)$ be a graph, $v,u \in G$. We define the Adamic/Adar score as*

$$AA(u,v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|}. \tag{5.5}$$

Let u and v be adjacent nodes, that are connected to the same node with small in- and out-degree. This node will be more informative about the relation or properties u and v share, then a hub node that is connected to many nodes. Adamic/Adar follows this idea by giving those nodes more weight. As a third heuristic we use the Personalised PageRank (PPR) vectors described in section 2.4. For evaluation we use the Python networkx[4] library [49] which provides implementation of the three heuristics.

Node Classification

Due to the way information is encoded in the different objective criterions of graph embedding models, the link prediction problem demands a different scoring function for each model.

[4] https://networkx.github.io

A simpler way to evaluate a model is to predict categories of each node given the node embedding. Therefore, we introduce the node classification task in this section.
Let $G = (V,E)$ be a graph. Each node in V is assigned one or more categories in $\mathscr{C} = \{C_1,\ldots,C_n\}$ with $C_i \subseteq V$. Let $\tilde{V}$ be the corresponding embedding matrix of the graph. The node classification task is to minimise for a classifier $f : \tilde{V} \to C$ (e.g. Logistic Regressor) the error of the classifier. The performance of the classifier depends on the quality of the embedding. Therefore the node classification task is a good way of comparing graph embeddings. To measure the error we use the F1 score [47]. To define the F1 score, that requires the definition of precision and recall.

Definition 18. *Let* $\mathrm{t}\{\mathrm{p},\mathrm{n}\}$ *be the number of true positives/negatives and* $\mathrm{f}\{\mathrm{p},\mathrm{n}\}$ *the number of false positives/negatives, then precision is defined as*

$$\text{precision} = \frac{\mathrm{tp}}{\mathrm{tp}+\mathrm{fp}} \tag{5.6}$$

and recall is defined as

$$\text{recall} = \frac{\mathrm{tp}}{\mathrm{tp}+\mathrm{fn}}. \tag{5.7}$$

Given precision and recall, we define the F1 score as the harmonic mean of both scores. F1 is then defined as

$$\mathrm{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{5.8}$$

A perfect F1 score is given when recall and precision are equal to 1. This means the classifier generated no false positive as well as no false negatives. Therefore, the higher the F1 score, the better the classifier. Note, that the F1 score is a combination of two scores. This means that a model with strong recall but poor precision can be as good as a model with strong precision but poor recall. The metric gives the classifier the possibility to balance between to scores and is not as strict as accuracy for example. In a multi-label classification scenario, we compute the F1 score for each class and then average the scores. For multiple queries (e.g. K-fold cross validation), we define the macro-precision and macro-recall as the mean of the precision respectively recall over all queries. The macro-F1 score is defined as the harmonic mean of the macro-precision and macro-recall score.
For the evaluation of the node classification task, we use five classification algorithms. Namely Logistic Regression (Logistic), Classification and Regression Trees (CARTs), Support Vector Machines (SVMs), K-Nearest Neighbours (K-NN) and a fully connected neural network with three hidden layers of size (300 - 200 - 100) and ReLU as activation function.

We use the Python library scikit-learn[5] that implements all of the classifier above. For training and testing we use 10-fold cross validation and report the macro-F1 score.

5.2 Analysis of KGloVe and node2vec

For the experiments we will use KGloVe and node2vec to generate embeddings. We use KGloVe, since it is the only model in chapter 3, that uses global statistics of the graph, obtained from the BCA, to generate an embedding. Furthermore, the way that context in the graph is computed offers different approaches to embed auxiliary information as we have seen in section 4.2.3 and 4.2.4. node2vec is used, since it is an established model with multiple implementations available. Moreover node2vec provides stable embeddings that we can use for evaluation on the link prediction and node classification task.
In this section we evaluate the sensitivity of KGloVe and node2vec when changing the dimension size and experiment with different hyper-parameter settings.
For a graph embedding it is important to choose the right embedding dimension. If the dimension size is chosen to small, the model is not able to encode all the information in the graph. This will restrict the model to generalise well. On the other hand, if the embedding dimension is chosen to big, the computational effort increases and training takes in general longer. The goal is to find a balance between computational effort and expressiveness of the model.
For node2vec we use the Python implementation provided by the Stanford SNAP library[6]. For KGloVe we use the C++ implementation that is used in [12].
We evaluate the performance of the models for using the PubMed dataset on the node classification task. Both node2vec and KGloVe have several hyper-parameters which need to be tuned. In node2vec we have the return parameter p and the in-out parameter q. For KGloVe we can adjust the amount of paint α that is injected into the graph as well as the cut-off parameter ε. To find a good set of hyper-parameters we perform grid search. For node2vec we found that for $p = 0.25$ and $q = 0.25$, it performs well on the classification task. This is also the parameter set which is reported in the original paper [17]. For KGlove the search is performed for different $\varepsilon \in \mathbb{R}$ and $\alpha \in \mathbb{R}$. Since the results are inconclusive, we choose $\alpha = 0.5$ and $\varepsilon = 0.0001$. These parameter reduce the computational cost due to a high cut-off value. We fix those hyper-parameters throughout this section, if not noted otherwise.
With the parameters obtained from the grid search, each model is trained to generate embeddings for $D \in \{1, 10, 20, 50, 100, 150, 200, 500, 1000\}$, where D is the embedding dimension.

[5] http://scikit-learn.org
[6] https://github.com/snap-stanford

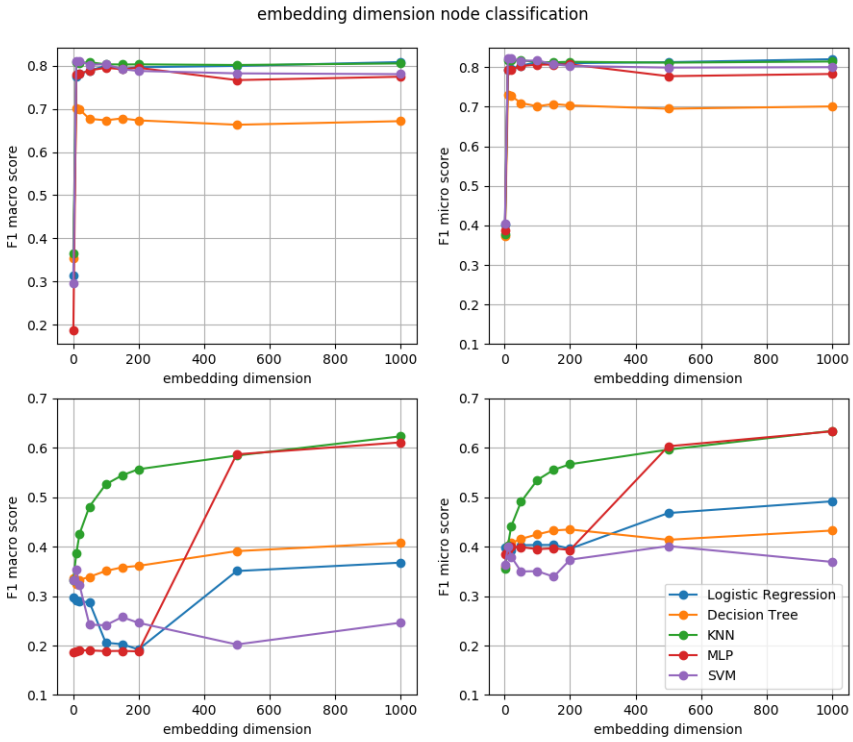

Fig. 5.2 Effect of different embedding dimension sizes for node2vec, with $p = 0.25$, $q = 0.25$ and 50% of the dataset as training set on the node classification task.

Although, the graph is directed, we train both models on an undirected graph. This is due to the fact, that in [15] the authors use undirected edges for training. To guarantee comparability, we follow the same setting. For KGloVe we add for each edge the reversed edges to the graph.

The classification performance of the different embedding dimensions is presented in figure 5.2. For dimension $D < 10$ both models fail to learn a good representation of the graph. node2vec generates stable embeddings for dimensions greater then $D = 10$. In comparison to KGloVe, node2vec performs for a low embedding dimension already reasonably well. The performance does not change when increasing the embedding dimension. This implicates that node2vec does not benefit from additional dimensions once it is provided with the minimum amount of dimensions to learn a representation. KGloVe compared to node2vec behaves unstable when changing the embedding dimensions. Nevertheless, we observe a clear improvement when increasing the dimension size. Interestingly, KGloVe performs different for different classifiers. The best classifier for KGloVe is K-NN. This implicates, that nodes with the same label are placed close to each other in the latent feature space.

5.3 Adding Auxiliary Information to the Graph

In this section we present and analyse the experiments based on enriching the graph structure with auxiliary information presented in section 4.1.1.

5.3.1 Adding Edges and Nodes to the Graph

We present the results of the approaches described in section 4.1.1. For evaluation we use the Cora citation network and the CiteSeer document classification network. Cora has seven classes, CiteSeer six and each node is assigned to exactly one class. For both models we set the embedding dimension to $D = 128$. In the experiment we evaluate node2vec with class nodes on Cora. The results are listed in table 5.3.

	with	without
Logistic	0.9601	0.8055
CART	0.8241	0.6659
KNN	0.8864	0.7988
MLP	0.9599	0.8076
SVM	0.9589	0.8076

Table 5.3 Macro-F1 scores using node2vec ($p = 0.25$, $q = 0.25$ and $D = 128$) with and without class nodes on node classification task for Cora.

We observe, that node2vec clearly outperforms the baseline node2vec, which does use class nodes. To be able to generalise, we perform a second experiment on CiteSeer. Additionally, we use the proposed ring method where nodes in a class are connected through a ring. Table 5.4 presents the classification results, where ring-N is the ring approach with $N \in \mathbb{N}$ forward jumps.

	node2vec	class nodes	ring-1	ring-10	ring-50
Logistic	0.5370	0.9651	0.8705	0.9964	1.0000
CART	0.5078	0.8032	0.6905	0.8755	0.9906
KNN	0.5813	0.8551	0.8865	1.0000	1.0000
MLP	0.6036	0.9723	0.9124	0.9980	1.0000
SVM	0.4845	0.8904	0.7769	0.9869	1.0000

Table 5.4 Macro-F1 scores using node2vec ($p = 0.25$, $q = 0.25$) with class nodes and $D = 128$ on node classification task for CiteSeer with 10% as training set.

Again node2vec with class nodes outperforms the baseline model. Furthermore, we get nearly perfect scores as we increase the number of jumps to $N = 50$ in the ring model. In figure 5.3 we plotted the two dimensional t-SNE [55] vectors for node2vec with class nodes

to visualise how the nodes are embedded in the latent feature space. We can see that the assumption that class nodes represent centroids for each cluster is confirmed. Therefore, we do not only get a graph embedding but also a representation of the whole class, when using the class node approach.

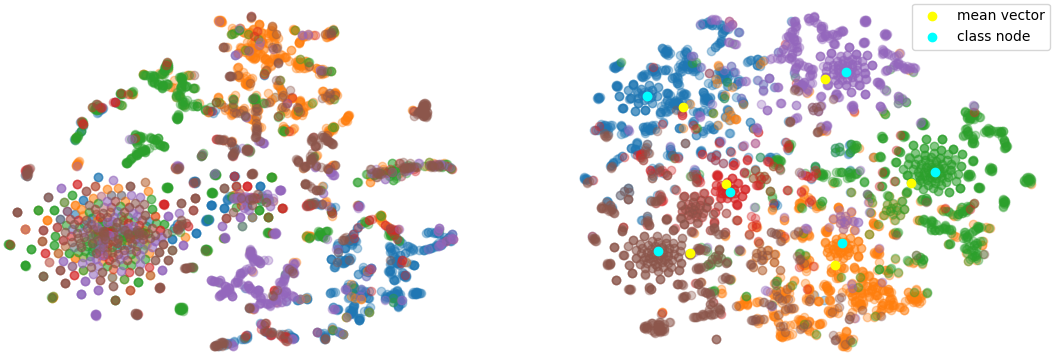

Fig. 5.3 Two dimensional t-SNE visualisations of node2vec ($p = 0.25$, $q = 0,25$) vectors on CiteSeer. (left) node2vec embeddings. (right) node2vec with class nodes, mean vectors per class (yellow) and class vectors (cyan).

Since the ring-N approach provides nearly perfect scores on the node classification task, we want to know, if we are still preserving the graph structure. Therefore we evaluate both approaches on the link prediction task. To additionally show that we are actually learning meaningful representation, we report on the average results of uniformly independently drawn vectors from $[-1,1]^{|V|}$, which we use as graph embeddings over 100 runs.

	MAP
node2vec	0.9822
class nodes	0.9826
ring	0.9253
ring-10	0.7812
ring-50	0.6492
PPR	0.9216
Adamic/Adar	0.6066
Jaccard Coefficient	0.6058
Random	0.5012

Table 5.5 Mean Average Precision scores for node2vec ($p = 0.25, q = 0.25$) with class nodes and $D = 128$ for the link prediction task on CiteSeer.

We observe from figure 5.5 that the node2vec embeddings perform much better than random vectors. This implicates that they actually encode useful features for link prediction. Moreover the performance in link prediction using class nodes is approximately the same as without class nodes. We can conclude, given the results of figure 5.3, that class nodes perform clustering while preserving the graph structure. On the other hand, the performance in link prediction quickly deteriorates as the number of additionally edges between nodes increases using the ring-N approach. Compared to class nodes, we get a good performance in node classification at the expense of link prediction performance, while class nodes improve the performance on both tasks compared to the original model.

5.3.2 Statistical Approach

The approach in section 5.3.1 is based on the assumption of a supervised learning scenario. In a real world scenario we will be usually confronted with-partially labeled or unlabelled data. The task to determine the political orientation of each user in a social network is an example of a semi-supervised learning scenario, since some user will reveal their political orientation while others will not. We learned in the section above that we can encode class information by introducing class nodes. In this experiment we use fuzzy K-means for different $K \in \mathbb{N}$ to determine a cluster for each attribute vector in the graph. For the embedding, each cluster is assigned a class node and then trained using node2vec. We evaluate the performance of node2vec on CiteSeer.

	node2vec	2-Means	4-Means	7-Means	10-Means
Logistic	0.5370	0.5077	0.5391	0.5635	0.5714
CART	0.5078	0.3627	0.4125	0.4330	0.4579
KNN	0.5813	0.5747	0.5693	0.6144	0.6122
MLP	0.6036	0.6077	0.5994	0.6159	0.6348
SVM	0.4845	0.4607	0.4862	0.4976	0.5258

Table 5.6 Macro-F1 scores using node2vec ($p = 0.25$, $q = 0.25$) with class nodes and $D = 128$ on node classification task for CiteSeer with 10% as training set.

We observe from table 5.6 that we obtain an improvement in the node classification task using class nodes generated by clustering. Additionally we see that by augmenting the number of clusters we obtain better results in comparison to the original model.
Since CiteSeer is a small dataset with already pre-processed Bag-of-Words vectors, we evaluate the approach in a second experiment using YAGO3-10. YAGO3-10 does not contain labels for each entity. Therefore, we evaluate the embedding on the link prediction task. For each triple in the test set we generate two times 500 corrupted triples, where the tail

respectively the head is variable. Before evaluating KGloVe on the link prediction task, we need to define how a triple is ranked by KGloVe. Let $s, o, p \in \mathbb{R}^D$ be the embedding vectors of a triple to be ranked. An intuitive measure motivated by TransE is given by

$$f(s,p,o) = \|s+p-o\|_2^2. \tag{5.9}$$

The problem is, that KGloVe is not trained to encode information via translation. Recall the objective function of GloVe, that is defined as

$$J(\theta) = \min \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} f(X_{ij})(v_i^T \hat{v}_j + b_i + \tilde{b}_j - \log(X_{ij}))^2. \tag{5.10}$$

The objective function aims to minimise the difference between the dot product and the log-scaled co-occurrence count. Therefore, the information is encoded using the dot product. We propose upon this observation to rank triples using KGloVe according to

$$f(s,p,o) = \frac{1}{3} \cdot [s^T p + o^T p + s^T o]. \tag{5.11}$$

If the co-occurrence count is high for each pair in the triple, we expect f to be large. We evaluated the link prediction performance of KGloVe for different equations such as the equations 5.10 and 5.11 and established, that equation 5.11 performs best. Therefore we will use this score to rank triples in the following experiments. Table 5.7 reports Mean Rank and hits@10 values for KGloVe trained on YAGO3-10.

	MR	MR tail	MR head	hits@10
KGloVE	282.1845	390.3935	176.8883	0.0406
KGloVe Class Nodes	137.8706	70.1294	206.1891	0.3178

Table 5.7 Mean Rank and hits@10 scores for KGloVe ($\alpha = 0.5$, $\varepsilon = 0.0001$) on YAGO3-10 with class nodes.

To incorporate auxiliary information, we use the YAGO3-10+ dataset. The attributes consist of five additional relations, such as *wasBornOnDate* and *happenedOnDate*. To generate class nodes, first for each relation we collect all values in the dataset. Secondly, we sort those values according to their natural ordering. To generate a clustering we divide the ordered collection into K equally sized bins. The idea is that for a birth year for example, we might have more data from this century than 2000 years ago. This means, that bins with ancient birth years will span over a bigger interval and thereby connecting persons from different epochs. We introduce historical information into the embedding and implicitly cluster them

according to their underlying order. From table 5.7 we observe, that the clustering approach outperforms the baseline KGloVe model. The Mean Rank differs when choosing the tail entity or the head entity freely for corrupted triples. Note, that the results still represent a poor link prediction capability, since the average of the correct triples is not ranked in the upper 10% of the randomly selected triples. This might be due to the fact, that KGloVe is not directly optimised for link prediction. Nevertheless, the approach of automatic clustering with class nodes outperformed all baseline models.

5.4 Enriching Graph Embeddings

This section explores different methods to enrich the graph embedding vector without altering the graph structure itself. The corresponding experiments are described in section 4.2.

5.4.1 Concatenating Embeddings

In this experiment we test different approaches on how to incorporate Bag-of-Words vectors into graph embeddings by concatenating them. To generate the graph embeddings we use KGloVe and node2vec. We use PubMed for evaluation, since the dataset provides a 500 dimensional Bag-of-Words vector for each node in the graph. Furthermore, this vector is not 0/1-valued but contains the term frequency-inverse document frequency (tf-idf) values of each word in the vector. The tf-idf value is the product of the term-frequency (tf) and the inverse document-frequency (idf).
For each node a one hot encoded vector $x_{\text{lit}}^{v} \in \mathbb{R}^{500}$, where the i-th entry is equal to one, if a word is in the Bag-of-Words vector and otherwise zero, is computed. Additionally, we train node2vec and KGloVe embeddings on the graph structure. We concatenate the resulting embedding vectors with the attribute vectors. The concatenated vector is then defined as

$$x_{\text{conc}}^{(v)} = x_{\text{lit}}^{(v)} \oplus x_{\text{graph}}^{(v)} \in \mathbb{R}^{D+500}, \tag{5.12}$$

where $D = 150$ denotes the embedding dimension.
To compute a baseline we use the attribute vectors as node embeddings and evaluate them on the classification task together with the graph embeddings from node2vec and KGloVe. The macro-F1 scores are presented in table 5.8. Additionally we report the averaged scores over ten runs for random vectors drawn uniformly from $[0,1]^D$. We evaluate the concatenated vectors in table 5.9.
Using the Bag-of-Words vectors as a substitution for the node embeddings gives us a better score than the random vectors. In addition to that the concatenated vectors outperform the

	random	BoW	tf-idf	node2vec	KGloVe
Logistic	0.39354	0.8735	0.8153	0.7967	0.4376
CART	0.35541	0.8072	0.8170	0.6784	0.4063
KNN	0.3562	0.7154	0.7478	0.7986	0.5410
MLP	0.35925	0.8708	0.8671	0.7873	0.5429
SVM	0.38461	0.8637	0.8045	0.7923	0.3561

Table 5.8 Macro-F1 scores for node classification on PubMed using different baseline methods (random vectors, weighted binary vectors)

	baseline			BoW	
	BoW	node2vec	KGloVe	node2vec	KGloVe
Logistic	0.8735	0.7967	0.4376	0.8927	0.8753
CART	0.8072	0.6784	0.4063	0.7859	0.7849
KNN	0.7154	0.7986	0.5410	0.8337	0.7290
MLP	0.8708	0.7873	0.5429	0.8904	0.8671
SVM	0.8637	0.7923	0.3561	0.8814	0.8609

Table 5.9 Macro-F1 scores for node classification on PubMed for KGloVe and node2vec ($D = 150$) with concatenated Bag-of-Words (BoW) vectors.

original embeddings for both models. In comparison to the other vectors the random vectors perform poorly, indicating that the embeddings generated from our methods actually encode useful information about the graph. Although, KGloVe performs poorly on the classification task, we see that adding the Bag-of-Words embedding improves the F1 score above the KGloVe baseline and the Bag-of-Words score. We can conclude from that, that KGloVe although it does not encode well specific node information. We also see a small improvement upon the baseline when using KGloVe, which shows clearly that the graph embedding has a positive effect on the baseline, but also the graph embedding on the Bag-of-Words embedding.

A problem with the Bag-of-Words representation is, that the entries in the vector do not contain any information about how the words are related to each other. It therefore does not capture the semantics of the words. To remedy this fact, we use in a second approach pre-trained word embeddings generated by fasttext [19]. Fasttext is a program from Facebook, that offers word embeddings that were trained on large corpora of text. The idea is to combine the word embeddings to one vector, representing all words in the Bag-of-Words vector. To get a single vector representation we first replace each word in the Bag-of-Words vector by its corresponding word vector and then compute the mean vector over all vectors for each node. Table 5.10 presents the results using the concatenated fasttext vectors. We see, that the fasttext word embeddings perform worse then the Bag-of-Words vectors, but they still

improve the performance of node2vec and KGloVe. Especially, the performance of KGloVe is increased.

	BoW	fasttext	BoW node2vec	BoW KGloVe	fasttext KGloVe	fasttext node2vec
Logistic	0.8746	0.8322	0.8912	0.8753	0.8040	0.8040
CART	0.8086	0.5559	0.7873	0.7849	0.6873	0.6873
KNN	0.7177	0.6501	0.8316	0.7290	0.8033	0.8033
MLP	0.8713	0.8514	0.8885	0.8671	0.8148	0.8148
SVM	0.8647	0.7335	0.8787	0.8609	0.7857	0.7857

Table 5.10 Comparison of macro-F1 scores for node2vec and KGloVe concatenated with pre-trained word embeddings generated by fasttext.

Still, the fasttext vectors have their legitimacy since they represent a compressed version of the Bag-of-Words. The problem with the Bag-of-Words representation is, that the dimension grows quickly as we add more words to the vector, while the dimension of the fasttext vectors will remain the same. Another point that needs to be considered is that every word is stemmed to generate the Bag-of-Words vectors. A nearest word search had to be performed to find the closest corresponding fasttext vector. This leads in some cases to vectors, that are less common and therefore not representative.

5.4.2 Merging Embeddings with Auto-encoders

As we saw in the last section, concatenating feature vectors with node embeddings can improve the performance when using those embeddings on machine learning tasks such as classification or regression. A problem with concatenation is, that the dimension of the embedding vectors increases quickly as we add more vectors. This leads to an increased computational cost for classification and regression. Thus it is of interest to reduce the dimension size while preserving the information encoded in the vectors. To achieve this, we use two different techniques. The first one is the Principal Component Analysis (PCA) which is a common method for dimensionality reduction, the second one is an auto-encoder model. We presented the architecture of the auto-encoder in section 4.2.2

As a baseline, we use the results of node2vec and the Bag-of-Words vector used as node embedding on the node classification task. We additionally report the results of the concatenated vectors denoted as node2vec+. In the experiment, we use for $f_{\text{enc}}, f_{\text{dec}}$ simple feedforward neural networks. We experiment with different activation functions and different layer sizes. To implement the neural network, we perform a grid search over the depth, layer size and activation function of the neural network. Since we use 0/1-valued vectors concatenated

	BoW	node2vec	node2vec+	PCA-50	PCA-1000	Net-50	Net-1000
Logistic	0.7278	0.7864	0.8390	0.8157	0.8308	0.8380	0.8110
CART	0.5926	0.6499	0.6705	0.7040	0.6294	0.3867	0.5648
KNN	0.3962	0.7949	0.7549	0.8170	0.7457	0.7539	0.8064
MLP	0.7310	0.8147	0.8284	0.8225	0.8289	0.8144	0.8062
SVM	0.7001	0.7490	0.7529	0.8111	0.7505	0.7530	0.7150

Table 5.11 Macro-F1 scores on Cora for different dimension reduction techniques.

with node2vec embeddings we get sparse vectors. Therefore when using ReLU as activation function the network tends to learn zero vectors. Instead we use tanh as activation function, with encoding layers of dimension (1,000 - 600 - 200). The results are presented in table 5.11. For comparison we also list the performance of PCA reduced embedding vectors. Interestingly the network performs poorly at the node classification task using CARTs, while achieving good macro-F1 scores for low dimension. PCA and the auto-encoder architecture perform almost the same. Since the original embedding vectors have dimension 1561 and we reduce them to dimension 50, we are able to merge the most salient features effectively in a low-dimensional vector space.

5.4.3 Adding Literals to KGloVe

In this section, we present the experimental results of incorporating textual literals into KGloVe as described in section 4.2.3. The experiment is performed on the DBPedia 2016-10[7] dump. In the first step, we extract manually the abstracts from the graph. This is done by using Jena[8] a Java library for operating on serialised resource description formats (RDFs) such as Turtle or the N-triple format. To perform entity recognition on the abstracts, different NER systems were tested. Since all of them have a restriction on the number of queries we can perform per day, an own NER system was implemented. In the first step, the system reads all triples of the graph into memory. Then, the identifier of each entity are cleaned by removing special characters such as the URL prefix. Next, we construct a character trie from the identifier of each subject, object and predicate in the graph. This allows us to perform efficiently substring matching. To do the actual entity recognition, we concatenate all abstracts into one file and use the file as an input to our NER system. The system uses a backtracking algorithm to match the longest sentence possible. This allows us to match entities where identifier consists of multiple words. The matched sentence is then replaced by the identifier of the entity. For the DBPedia 2016-10 dataset, we match 37.24% of the words

[7] https://wiki.dbpedia.org/downloads-2016-10
[8] https://jena.apache.org

with an entity out of 348,077,826 words in the corpus. As described, we train in section 4.2.3 a KGloVe embedding of the graph in parallel. From KGloVe we obtain the entity and predicate list, that we concatenate with the word list which we obtain from our NER system. With the word list and the annotated abstracts, we generate a co-occurrence matrix. After normalising, we add both matrices entrywise and train GloVe on the merged matrix.

	Classification					Regression		
	NB	KNN	SVM-100	SVM-1000	C45	LR	KNN	M5
KGloVe with literals	1.6	1.74	1.2	1.38	1.76	1.6	1.46	1.6
Normal KGloVe	1.4	1.26	1.8	1.62	1.24	1.4	1.54	1.4

Table 5.12 The average ranking of the classifier (resp. regression model) evaluated on 10-fold cross validation for 5 different classification (resp. regression) tasks for different algorithms.

To evaluate the model, we use five different datasets to predict classes and properties of entities in the Cities, Metacritic Movies, Metacritic Albums, AAUP and Forbes dataset[9]. We evaluate the embedding on regression and classifcation tasks and report the average ranking of the classifier. For classification Naive Bayes (NB), K-NN, CARTs and SVMs are used. For regression we use Linear Regression, M5Rules and K-NN. The results are listed in 5.12. The model in the second row of the table is the best performing KGloVe model from [12] which we use as a baseline. By comparing the values from the table it is not clear which model performs better. KGloVe with literals outperforms the normal KGloVe model when using SVMs for classification otherwise they perform similar. The comparison is not easy to make, since the best performing KGloVe model is compared with a first experiment. Furthermore, the model seems to work, since although we changed the co-occurrence matrix, the model is still able to learn a representation of the graph.

[9] https://bit.ly/2J8WBdN

Chapter 6

Conclusion

In this thesis, we presented different approaches on embedding auxiliary information together with the graph structure. In chapter 2, we introduced techniques for extracting features from graphs and texts. Throughout the thesis those techniques were used by different embedding models such as node2vec which is based on word2vec or KGloVe which is based on the Bookmark-Colouring Algorithm. Building upon chapter 2, we presented a variety of graph embedding models in chapter 3. The models were categorised based on the underlying methods and algorithms they used (e.g. factorisation and random walks). Additionally, we sorted them as far as possible in a chronological order to present how graph embedding models evolved over time. We observed that early work is based on factorisation methods while recent work is based on deep learning techniques, such as auto-encoders and word embedding models. Based on the models in chapter 3 we established two categories for embedding auxiliary information. The first category adds additional information to the graph structure. The second category enriches the graph embedding by taking attribute information into account. Having defined the two categories we presented in chapter 4 six approaches for including attributes in an embedding and evaluate them in chapter 5.

With the node class experiment, we were able to demonstrate that we can efficiently encode label information in the graph and at the same time preserve graph attributes. Additionally, we demonstrate that class nodes represent centroids for each class in the graph. The ring-N model captured the label information but masked the graph structure, so that the performance of node2vec decreased when adding more nodes. Furthermore, we showed that by automatically generating class nodes using clustering we still perform better than our baseline models. Through concatenation of the embedding and attribute vectors, we were able to establish that we can enrich the embedding vector in a simple way. Additionally, we experimented with different representations of text using fasttext. We were able to show, that by projecting the concatenated vectors in a common latent feature space, we obtain a dense representation

of our features. This dense representation performs still as good as the original embedding. The experiment of incorporating literals into KGloVe was created in collaboration with Dr. Michael Cochez, Martina Garofalo and Maria Angelo Pellegrino. The model is summarised in [11] and was accepted at the fourth workshop on Semantic Deep Learning (SemDeep-4). Further research has to be done on how to merge the co-occurrence matrices and the use of sophisticated named entity recognition systems should be investigated. Concluding, we demonstrated that it is possible to include textual and numerical information into a graph embedding in different experiments.

References

[1] Adamic, L. A. and Adar, E. (2003). Friends and neighbors on the web. *Social Networks*, 25:211–230.

[2] Belkin, M. and Niyogi, P. (2002). Laplacian eigenmaps and spectral techniques for embedding and clustering. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 585–591. MIT Press.

[3] Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396.

[4] Berkhin, P. (2006). Bookmark-coloring approach to personalized pagerank computing. *Internet Mathematics*, 3(1):41–62.

[5] Bhagat, S., Cormode, G., and Muthukrishnan, S. (2011). Node classification in social networks. *CoRR*, abs/1101.3291.

[6] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA. ACM.

[7] Bordes, A., Usunier, N., García-Durán, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q., editors, *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.*, pages 2787–2795.

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

[9] Chung, F. (2009). Graph theory in the information age.

[10] Cochez, M. (2014). Locality-sensitive hashing for massive string-based ontology matching. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 01*, WI-IAT '14, pages 134–140, Washington, DC, USA. IEEE Computer Society.

[11] Cochez, M., Garofalo, M., Lenssen, J., and Pellegrino, M. A. (2018). A first experiment on including text literals in kglove. *International Semantic Web Conference (ISWC), SemDeep-4*.

[12] Cochez, M., Ristoski, P., Ponzetto, S. P., and Paulheim, H. (2017). Global RDF vector space embeddings. In d'Amato, C., Fernández, M., Tamma, V. A. M., Lécué, F., Cudré-Mauroux, P., Sequeda, J. F., Lange, C., and Heflin, J., editors, *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, volume 10587 of *Lecture Notes in Computer Science*, pages 190–207. Springer.

[13] Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. (2017). Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476.

[14] Ebisu, T. and Ichise, R. (2017). Toruse: Knowledge graph embedding on a lie group. *CoRR*, abs/1711.05435.

[15] Gao, H. and Huang, H. (2018). Deep attributed network embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3364–3370. International Joint Conferences on Artificial Intelligence Organization.

[16] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[17] Grover, A. and Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864, New York, NY, USA. ACM.

[18] Ji, G., He, S., Xu, L., Liu, K., and Zhao, J. (2015). Knowledge graph embedding via dynamic mapping matrix. In *ACL (1)*, pages 687–696. The Association for Computer Linguistics.

[19] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

[20] Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43.

[21] Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

[22] Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders. *CoRR*, abs/1611.07308.

[23] Kristiadi, A., Khan, M. A., Lukovnikov, D., Lehmann, J., and Fischer, A. (2018). Incorporating literals into knowledge graph embeddings. *CoRR*, abs/1802.00934.

[24] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2015). DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6(2):167–195.

[25] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. In Bonet, B. and Koenig, S., editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2181–2187. AAAI Press.

[26] Lu, Q. and Getoor, L. (2003). Link-based classification. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 496–503. AAAI Press.

[27] Mahdisoltani, F., Biega, J., and Suchanek, F. M. (2015). Yago3: A knowledge base from multilingual wikipedias. In *CIDR*.

[28] Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., New York, NY, USA.

[29] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[30] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

[31] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In Cowell, R. G. and Ghahramani, Z., editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252. Society for Artificial Intelligence and Statistics.

[32] Namata, G. M., London, B., Getoor, L., and Huang, B. (2012). Query-driven active surveying for collective classification. In *Workshop on Mining and Learning with Graphs*.

[33] Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016a). Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1105–1114, New York, NY, USA. ACM.

[34] Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016b). Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1105–1114, New York, NY, USA. ACM.

[35] Palash Goyal, E. F. (2017). Graph embedding techniques, applications, and performance: A survey. *CoRR*, abs/1705.02801.

[36] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *In EMNLP*.

[37] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652.

[38] Powers, D. M. W. (1998). Applications and explanations of zipf's law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, NeMLaP3/CoNLL '98, pages 151–160, Stroudsburg, PA, USA. Association for Computational Linguistics.

[39] Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. (2017). Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. *CoRR*, abs/1710.02971.

[40] Rebele, T., Suchanek, F. M., Hoffart, J., Biega, J., Kuzey, E., and Weikum, G. (2016). YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II*, pages 177–185.

[41] Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., and Paulheim, H. (2018). Rdf2vec: Rdf graph embeddings and their applications. *Semantic Web Journal*.

[42] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326.

[43] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

[44] Ruder, S. (2017). An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098.

[45] S., N. S. (2013). Graph embedding and dimensionality reduction - a survey. *International Journal of Computer Science and Engineering Technology (IJSCET)*, 4(1).

[46] Sadraei, A. (2014). Link prediction algorithms, what will facebook look like tomorrow?

[47] Sasaki, Y. (2007). The truth of the f-measure.

[48] Saul, L. K. and Roweis, S. T. (2000). An introduction to locally linear embedding. Technical report.

[49] Schult, D. A. (2008). Exploring network structure, dynamics, and function using networkx. In *In Proceedings of the 7th Python in Science Conference (SciPy*, pages 11–15.

[50] Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561.

[51] Srinivas, V. and Mitra, P. (2016). *Link Prediction in Social Networks: Role of Power Law Distribution*. Springer Publishing Company, Incorporated, 1st edition.

[52] Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: A Core of Semantic Knowledge. In *16th International Conference on the World Wide Web*, pages 697–706.

[53] Toutanova, K., Chen, D., Pantel, P., Poon, H., Choudhury, P., and Gamon, M. (2015). Representing text for joint embedding of text and knowledge bases. ACL Association for Computational Linguistics.

[54] Tu, C., Liu, H., Liu, Z., and Sun, M. (2017). Cane: Context-aware network embedding for relation modeling. In *ACL*.

[55] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

[56] Voss, H. (2007). A jacobi-davidson method for nonlinear and nonsymmetric eigenproblems. *Comput. Struct.*, 85(17-18):1284–1292.

[57] Vrandečić, D. and Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

[58] Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

[59] Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In Brodley, C. E. and Stone, P., editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1112–1119. AAAI Press.

[60] Wu, W., Li, B., Chen, L., and Zhang, C. (2018). Efficient attributed network embedding via recursive randomized hashing. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2861–2867. International Joint Conferences on Artificial Intelligence Organization.

[61] Zhang, Z., Yang, H., Bu, J., Zhou, S., Yu, P., Zhang, J., Ester, M., and Wang, C. (2018). Anrl: Attributed network representation learning via deep neural networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3155–3161. International Joint Conferences on Artificial Intelligence Organization.

[62] Zhou, G. and Su, J. (2002). Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 473–480, Stroudsburg, PA, USA. Association for Computational Linguistics.

[63] Zitnik, M. and Leskovec, J. (2017). Predicting multicellular function through multi-layer tissue networks. *CoRR*, abs/1707.04638.