**Gottfried Wilhelm
Leibniz Universität Hannover**

Explaining and Applying Graph Neural Networks on Text

Bachelor's Thesis

Computer Science

Author: Nils Cornelius Grünefeld

**First Examiner: Prof. Dr. Avishek Anand
Second Examiner: Dr. Thorben Funke**

March 31, 2022

Selbständigkeitserklärung

Hiermit versichere ich, Nils Cornelius Grünefeld, dass ich diese Arbeit selbstständig verfasst habe und keine weiteren als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, habe ich als solche kenntlich gemacht. Zudem wurde die Arbeit bisher in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt.

Hannover, den 31. März 2022

[illegible]

Nils Cornelius Grünefeld

Contents

Abstract

Text classification is an essential task in natural language processing. While graph neural networks (GNNs) have successfully been applied to this problem both through graph classification and node classification approaches, their typical applications suffer from several issues. In the graph classification case, common graph construction techniques tend to leave out syntactic information. In the node classification case, most widespread datasets and applications tend to suffer from encoding relatively little information in the chosen node features. Finally, there are great benefits to be gained from combining the two GNN approaches. To tackle these concerns, we propose DepNet, a two-stage framework for text classification using GNN models. In the first stage we replace current graph construction methods by utilizing syntactic dependency parsing in order to include as much syntactic information in the GNN input as possible. In the second stage we combine both graph classification and node classification methods by utilizing the former to produce node embeddings for the latter, maximizing the potential of GNNs for text classification. We find that this technique significantly improves the performance of both graph classification and node classification approaches to text classification. Our code is available online[1].

[1]https://git.l3s.uni-hannover.de/grunefeld/gnns-for-text

Acknowledgements

No man is an island, and this thesis would not have been possible if not for the support of a number of people whom I would like to thank.

First and foremost, my supervisor, Thorben Funke, for his input and guidance throughout the last four months and for the simple fact that without him, this thesis would never have been written.

Second, I would like to thank two researchers, Giannis Nikolentzos and Ying-Xin Wu, for being very helpful when exploring their work, as well as the field of graph machine learning at large for it's remarkable openness and accessibility.

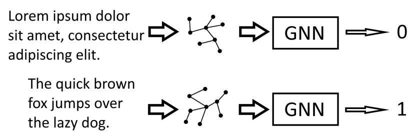
I would also like to thank my friend Sören for his continuous support and advice.

Finally, I would like to thank my family, for their unending support in every way imaginable.

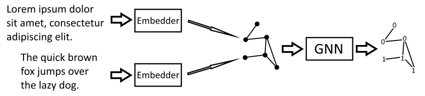
Chapter 1

Introduction

1.1 Motivation


Lorem ipsum dolor sit amet, consectetur adipiscing elit.
GNN
0
The quick brown fox jumps over the lazy dog.
GNN
1

(a) Text classification by graph classification


Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Embedder
The quick brown fox jumps over the lazy dog.
Embedder
GNN

(b) Text classification by node classification

Figure 1.1: The two classical approaches to text classification using graph neural networks

Text classification is a seminal task in natural language processing (NLP). It finds application in several areas of NLP, such as sentiment analysis and question answering [18], news filtering and spam detection [36], and search result organization [25]. As such there has been a tremendous amount of research into this subject, exploring a variety of techniques, from traditional shallow learning methods to modern machine learning and deep learning approaches. Among the latter, especially in recent times, have been graph neural networks (GNNs), which lend themselves very well to this task as natural language documents can in many ways be parsed into and interpreted as graphs.

As shown in Section 2.1 and visualized in Figure 1.1a, the most commonly used type of technique for utilizing GNNs in text classification has been parsing individual documents into graphs to run graph classification methods. This is most often done by deploying word co-occurrence graphs constructed through sliding window algorithms, forming nodes for each unique word in a text and edges between co-occurring words [25, 6, 26, 24, 39]. Another perhaps less common approach is to treat text classification as a node classification task in a network of connected documents [36, 20] as visualized in Figure 1.1b. In this case, documents are represented as nodes which are then provided with some form of embedding on the document level as node features. These cases tend to feature shallow embeddings consisting of handcrafted features such as binary word vectors encoding the presence of words in the associated document [30].

However, both of these approaches tend to suffer from various issues. In the case of text classification by graph classification, the common word co-occurrence technique in graph construction tends to lose syntactic information contained in the text, as sentence structures are not included in the evaluation at all. In order to rectify this issue, alternative methods of graph construction are necessary. Syntactic dependency parsing [11] constitutes such an alternative, encoding syntactical information from given documents in their graph representations. GNNs can presumably utilize this additional information in order to improve their expressivity and performance. In the case of text classification by node classification, commonly problems lie in the document representations that are used as node features. As these features tend to be shallow embeddings, oftentimes they contain relatively little information about the documents they represent. As a result, GNNs operating on such document relation graphs frequently rely heavily on the relations encoded in the graph's edges rather than being able to utilize document-inherent information as well.

Further, predictions produced by artificial intelligence algorithms such as GNNs can be difficult to understand and therefore verify. This constitutes a problem, as the understanding of any decision making process is crucial to its evaluation and to establish inference. Motivated by this challenge, a variety of techniques have been developed to open the black box, analyzing and explaining the predictions made by graph neural network models [38]. This process can further be utilized to compare different GNN models and investigate their differences, e.g. in order to evaluate whether there are genuine differences in terms of model performance instead of mere coincidences.

In this study, we suggest and examine a solution to the described issues in common graph neural network text classification methods. We propose DepNet, a two-stage framework for text classification using GNNs, that both combines the advantages of graph and node classification and provides a solution for the issues that common text classification by graph classification techniques suffer from. For graph classification, we examine the application of syntactic dependency graphs instead of word co-occurrence graphs, which constitutes the first stage of DepNet. In the second stage, we then attempt to utilize document embeddings produced by GNNs operating on the texts themselves as node features for node classification tasks. We hypothesize that this approach can rectify the described issues and yield significant improvements in both graph classification and node classification situations.

1.2 Outline

We begin by providing an overview of the relevant prior work on text classification, both using classical methodologies and, more detailed, in terms of graph neural networks. Then, we examine and explain the set of tools we apply in the course of our experiments. We then present the main body of our work, first in our applied methodology itself, with the results of our experiments following after. Finally, we discuss our observations and report a conclusion.

Chapter 2

Relevant Work

2.1 Prior Research

As a seminal task in natural language processing, text classification has been studied for decades using a multitude of approaches [16]. These approaches have progressed over time as the general toolset in both artificial intelligence and machine learning in general and natural language processing specifically have been progressed and improved. There have further been several studies of the history and development of text classification, such as Yang and Liu [35], Kowsari et al. [16], and Li et al. [18], which have informed the following paragraphs.

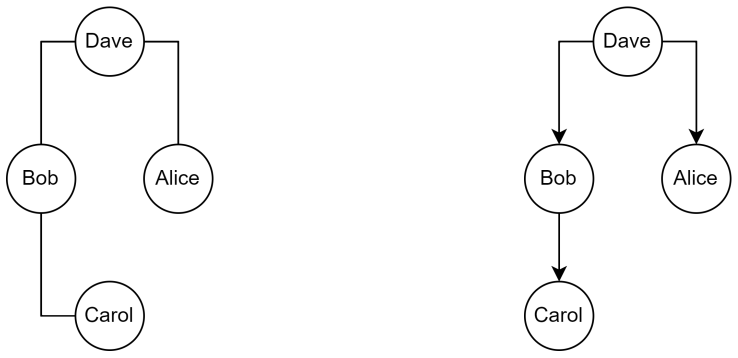
In classical natural language processing, various techniques have been applied to text classification tasks. Among the first such methods applied to text classification were probabilistic graphical models such as Naive Bayes classifiers, going back to the 1960s [21]. In the same decade, further simple classifiers were utilized for text classification, such as the K-Nearest Neighbors algorithm [35]. Even simple linear regression methods such as Yang's Linear Least Squares Fit [34] have been applied, with some success, to the problem at hand. Over time, however, the typically deployed methods have risen in sophistication with more complex models such as decision trees being utilized by Johnson et al. [10] and with the application of support vector machines [9] in the 1990s, the field took big steps in the direction of modern machine learning [18].

With the rise of machine learning, more specifically deep learning, new methods of data classification have been applied to the task of text classification. While the classical NLP toolbox has yielded decent results, they tend to require much effort in the way of feature engineering in order to reach respectable accuracy. In relatively small datasets this is unavoidable and traditional techniques keep producing the highest accuracies but with the rise of "big data", this trade-off can increasingly often be avoided by utilizing data driven machine learning methodologies. This process started in the 2010s with the first comparatively simple methods in the Multilayer Perceptrons [17] and the Recursive Neural Network [8]. Over time, the NLP community turned to more and more sophisticated and complex algorithms, such as Recurrent Neural Networks [19] and Convolutional Neural Networks [13]. Finally, some authors have applied attention mechanisms to deep learning applications. As such, Zhou et al. [40] deploy an attention-based LSTM network to further advance the application of deep learning techniques on text classification tasks, while Vaswani et al. [31] introduce a self-attention based method they call Transformer.

In recent years, graph neural networks have experienced rising popularity and in that course, researchers have begun to approach the task of text classification through the GNN framework. In doing so, text classification can be thought of as both a graph classification task or a node classification task. Various literature has emerged studying both strategies, while whether the option of formulating the task at hand as a node classification problem exists depends on whether inter-document relations are available. In such cases, often heterogeneous graphs are chosen, combining nodes representing both documents and text contained within those documents. An early example of this approach is TextGCN [36], with TensorGCN [20] producing further improvements. In text classification via graph classification cases on the other hand, graphs necessarily contain only the content and information of a single document. An example of such an approach is DGCNN [25] which parses a given document into a graph using a word co-occurrence sliding window technique. Huang et al. [6] deploy a similar method, however, they combine these intra-document word co-occurrence graphs with global word co-occurrence information which is incorporated as edge weights. Peng et al. [26], MPAD [24], and TextING [39] further study and improve this technique. In the case of the first two by adding an attention mechanism to the GNN operator and in the case of TextING by focusing on inductive capabilities by strictly limiting training data.

2.2 Graph Neural Networks

The following introduction of graph neural networks closely follows the Graph Representation Learning textbook by William L. Hamilton [4].



(a) Friendship graph (b) Debtors graph

Figure 2.1: Two social network graphs

Graphs are a powerful tool that is used widely in the field of computer science. They are both an abstraction used for modeling and a data structure itself. Formally, a graph $G = (V, E)$ is a tuple of two sets V and E. V is a set which elements represent nodes or vertices, while E contains links or edges between vertices. Any element of E is therefore a pair $(v_1, v_2) : v_1, v_2 \in V$ which represents an edge between v_1 and v_2. For example, a graph may be used to model a social network between friends with nodes representing individuals and edges representing friendships, as visualized in Figure 2.1a. Here, Alice is friends with Dave, Carol is friends with Bob, Bob is friends with Dave and Carol, and Dave is friends with Alice and Bob. Further, edges and therefore graphs may be directed or undirected. In the undirected case, a tuple (v_1, v_2) represents a symmetric connection between the two vertices v_1 and v_2, while in the directed case an edge tuple represents a connection from v_1 to v_2. As an example, consider the social network from the undirected case, however, instead of friendships the edges represent debt, as visualized in Figure 2.1b. In this example, Bob is in debt to Carol, with Dave being in debt to both Alice and Bob. Since one person being in debt to another does not imply the reverse, simple unordered connections can clearly not model such relationships.

Graphs may carry feature information. Beyond the existence of nodes and relationships between them, both elements of graphs may carry information to specify themselves. In the case of nodes, each node may be assigned an n-dimensional vector representing node information. Oftentimes, these vectors come in the form of one-hot encodings. For example in the case of our friendship network, each node might carry a 2-dimensional encoding whether a person enjoys two different activities. In the case of edges, edge information may be represented both as vectors and as simple scalars to weight each edge. In the social network case for example, one-hot edge features might encode the kind of relationship (such as romantic or platonic), while edge weights might represent the length of a friendship.

Due to the prevalence of graphs both as a modeling tool and a data structure and the potency of neural nets as machine learning devices, naturally the question of if and how the two can be applied together arises. Unfortunately, this is not a trivial task. Conventional neural networks operate on fixed size, fixed structure input, such as images or weather data. Graphs on the other hand feature neither. Graphs are of both flexible size and no set structure.

Graph neural networks circumvent these issues by operating directly on each node. Each node embedding is iteratively updated by aggregating neighboring nodes' information in a process called message passing. Similarly to conventional neural networks, iterations are thought of as layers, where the k-th layer producing an embedding h_i^k of the node i can be expressed as

$$h_i^k = \text{update}(h_i^{k-1}, \text{aggregate}(\{h_j^{k-1} \forall j \in N(i)\})$$

where N returns a node's neighborhood:

$$N(x) = \{y : \exists(x, y) \in E\})$$

and aggregate and update are differentiable functions, of which the former maps multiple node embeddings onto an aggregate and the latter produces an updated node embedding from an aggregation of node embeddings and the current node's own one. In combination, these two function are also called a GNN's operator. Notably, update almost always includes some non-linear activity such as a sigmoid or a ReLU function in order to increase the model's expressivity.

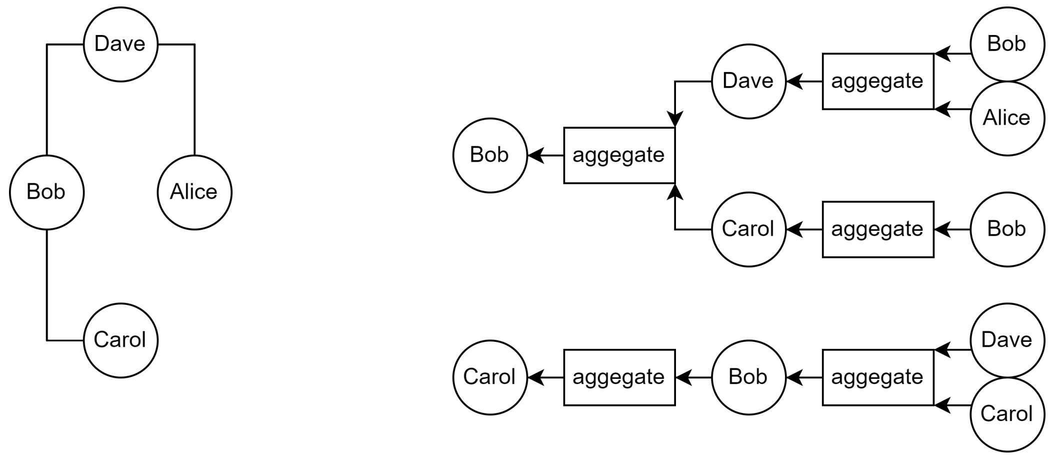


Figure 2.2: Two-layer computation graphs for Bob and Carol in the example friendship graph

In other words, for each node, a layer in a GNN first produces a message, an aggregate of information, from the node's neighborhood and then combines this message with the current node representation to update it's embedding. As with conventional neural nets, an arbitrary number of such layers may be chained. Notably however, there is a distinct characteristic regarding the amount of layers in a given GNN model. When considering a single node $n \in V$, the first layer will draw messages from each neighboring node. A second layer will again draw messages from each neighboring nodes, which in turn have drawn messages from their neighborhood previously, leading to a message passing chain reaching up to two hops from n. In other words, a k-layer GNN draws messages from a k-hop neighborhood around any given node. This neighborhood when thought of as a directed graph with each edge representing the flow of message passing steps is also called a node's computation graph. An example for the computation graph under a two-layer GNN of two nodes from Figure 2.1a is given in Figure 2.2.

After the core procedure of a GNN, it is worth considering the kind of tasks that these models can be applied to solve. Such tasks fall into different categories, two of which we will focus on: node classification and graph classification. In node classification, individual nodes carry labels representing for example certain categories that nodes fall into. However, not all labels are known, a subset of the graph's nodes is of unknown labels, with the task being to predict these labels from both the node's own feature information and it's relations to other nodes. As such, these tasks are usually closely related to the kind of information we store about each node in it's features. In the case of our friendship graph for example, a node classification task might be to predict whether a person is an extrovert or an introvert. Presumably, both the kinds of activities that a person enjoys and the kinds of activities their friends enjoy would carry at least some predictive power in this task and in this case perhaps even the amount of friends that a person has. In graph classification on the other hand, the entire graph is assigned a label. A dataset therefore necessarily contains multiple graphs, with the goal of the task being to label graphs of unknown category. A common example of this is the classification of molecules. For example we might have given the chemical structure of several molecules in the form of graphs, with atoms represented by nodes and chemical bonds represented by edges. In such a situation, one possible task may be to predict the toxicity of any given molecule. A GNN would therefore try and predict toxicity both from each atom's own qualities, and the interactions between atoms within the molecule.

Depending on the task at hand, after iterating through the GNN layers a readout function may be necessary. In node classification tasks, technically it is possible to simply run a set of GNN layers such that the last layer's feature update produces a vector whose dimension is the same as the number of possible labels. However, it is standard practice to add for example a softmax function after the last layer, among other reasons to turn the feature vector into a probability distribution over all possible labels. In the case of graph classification, we necessarily need a pooling function to aggregate the node level information developed by the GNN. An example for such a pooling function would be mean-pooling, which averages all node feature vectors in the graph entry-wise to produce a mean feature vector. Often, this pooled vector is fed again through a softmax function or a perceptron.

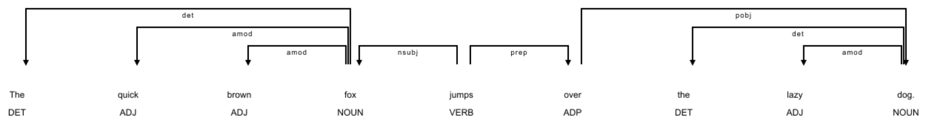
Regardless of what kind of task we are dealing with, in order to train the GNN a loss function is applied to the final output of the model. Frequent examples of such loss functions are a cross-entropy loss function or the mean squared error. The resulting loss is then backpropagated through the model in order to learn it's weights and biases, using standard (stochastic) gradient descent [29, 12] or more sophisticated methods such as Adam [14].

2.3 Dependency Parsing

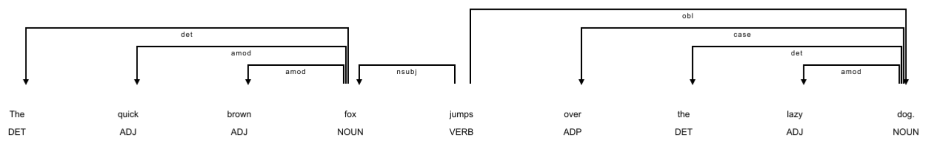
In tackling classification tasks, graph neural networks rely on information encoded in the provided graph. Such information is carried in two distinct ways, first in the node and edge information included through associated features and second through the relational structure of the graph itself. As such, it is crucial in graph construction to maximize the amount of information carried by the produced graph. In the case of text to graph parsing, this information is separated into two complementary parts: syntactic and semantic information.

A syntactic dependency tree is a connected, directed, acyclic graph encoding syntactic relations between individual components of a sentence. In such a graph, each node $n \in V$ represents a token within the sentence while each edge $e \in E$ encodes the syntactic dependency relationship between two tokens. Two examples of such trees are given in Figure 2.3. Each token in the sentence is assigned a part-of-speech tag which represents the type of word the token contains within the given language. Examples of such types and tags are for example noun, verb, or adjective. Dependency relations describe the interactions between each tokens that form their roles within the given sentence. A special case of this is the predicate of the sentence, which is usually linked to a special head node. An example for a regular dependency relation is a noun being turned into a sentence's subject by it's dependence on the predicate. Other dependency relations include prepositions, objects, or multiple words forming a compound.[11]

SpaCy [5] and Stanza [28] both apply greedy transition-based parsing. This algorithm works by first tokenizing a sentence, then successively iterating over each token and performing one of three actions. Each word is either pushed onto a stack, parsed as a dependent child of the current highest ele-



(a) spaCy



(b) Stanza

Figure 2.3: The sentence "The quick brown fox jumps over the lazy dog." parsed into a syntactic dependency tree using both spaCy and Stanza

ment on the stack, or parsed as the head node of the current highest element of the stack. The core of this procedure is the method of determining the action for each token. In both spaCy and Stanza, this is done using a single layer linear perceptron. This perceptron assigns each possible action that may be applied to a given token a score based on the current model state, after which the highest scoring action is taken. This point is also where different transition-based parsers differ, specifically in what decision function is applied and how the model state is fed into this function.

As noticable when comparing Figure 2.3a and Figure 2.3b, different dependency parsers may produce different dependency trees. While sometimes some of these different trees can be incorrect, most often differences in parsing results mirror differing ways of interpreting a sentence which may all be valid.

2.4 Word Embeddings

Given a set W of words which we call a vocabulary, a word embedding is a d-dimensional vector e_w assigned to a given word $w \in W$ that semantically represents the word. [11]

The Word2Vec [22] approach to word embeddings is perhaps best described by a John Rupert Firth quote: ”You shall know a word by the company it keeps.” [2] It derives the semantic meaning of a word by what other words it tends to occur in close proximity to, in a manner similar to word co-occurrence algorithms. More specifically, given a corpus of text, the dictionary W containing all words present in the corpus and a window size m, the simplest Word2Vec model deploys a sliding window algorithm. This process iterates over the given corpus and attempts to use the word c in the center of each window, also called the center word, to predict any word o surrounding it within the window, also called a context word. With the dot product of two vectors representing an approximate measure of vector similarity, the probability of finding a word c in the window surrounding the word c is defined by the equation

$$P_\theta(o|c) = \frac{e^{u_o \cdot v_c}}{\sum_{w \in W} e^{u_w \cdot v_c}}$$

where θ is the Word2Vec model represented by a set containing word vectors. For word $w \in W$ there are two word vectors contained in θ, one each for the case of w being a center word and a context word. We measure the accuracy of a given Word2Vec model instance by it’s capability to use any possible center word to predict all of it’s context words across the entire corpus, formalized in the likelihood function

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m<j<m \\ j\neq 0}} P_\theta(w_{t+j}|w_t)$$

whose optimization is the core of the Word2Vec approach. When training a Word2Vec model, word vectors for each word $w \in W$ is initially initialized as a random vector and then trained by applying stochastic gradient ascent on the likelihood function. After a full training, the center and context vectors of each word are averaged to create a single final word embedding.

In simpler terms, Word2Vec determines word embeddings such that the vector representations of two words that appear close to each other in a large corpus of texts are as similar as possible. It should further be noted that the approach described here is called the skip-gram model. Word2Vec provides a second model, called continuous bag-of-words model, which is very similar to the skip-gram model. The only difference is that while the skip-gram model attempts to predict context words based on a center word, the continuous bag-of-words model does the opposite, it attempts to predict the center word based on context words.

While Word2Vec has seen great success in a multitude of applications across NLP, some alternatives have been proposed that attempt to improve on the basic Word2Vec approach, which we will therefore examine as to whether there is any improvement to be gained. A second popular technique to produce word embeddings is GloVe [27]. The initial significant difference between Word2Vec and GloVe is that while GloVe applies a sliding window technique similar to Word2Vec, it considers the amount of times a word appears in the window of another word. In order to do this, they construct a matrix $X \in \mathbb{N}^{|W| \times |W|}$ where any entry X_{ij} counts the number of times $w_j \in W$ occurs in the context of $w_i \in W$ within a given corpus. From this, we can derive the probability of w_j appearing given w_i:

$$P_\theta(w_j|w_i) = \frac{X_{ij}}{\sum_k X_{ik}}$$

In terms of word vectors, the log probability function is now defined as

$$\log P_\theta(w_i|w_j) = v_i \cdot v_j$$

which leads to

$$\log \frac{P_\theta(x|a)}{P_\theta(x|b)} = v_x \cdot (v_a - v_b)$$

with v_a, v_b, and v_c being word vectors representing the words w_a, w_b, and w_c respectively.

Again similarly to Word2Vec, the GloVe model assigns each word two different vectors v and $\tilde{v}$ to a given word for the case of it being the center word and a context word respectively. Unlike Word2Vec however, the final word embedding is produced by addition of the two. By applying a squared error method, a function f that essentially behaves similar to $min(x, c)$ for some constant c, and bias terms b_i and $\tilde{b_j}$, the objective or loss function for a given model θ is thus given by

$$L(\theta) = \sum_{i,j=1}^{|W|} f(X_{ij})(v_i \cdot \tilde{v_j} + b_i + \tilde{b_j} - \log X_{ij})^2$$

which is again to be minimized similarly to Word2Vec.

In conclusion, GloVe examines the frequency with which different word co-occur within a corpus and then applies a frequentist approach to derive the probability of two words appearing closely together. These probabilities are then used to train word embeddings. The core difference between Word2Vec and GloVe lies in whether or not the frequency of word co-occurrences are considered and following how the probability of a word co-occurrence is defined.

2.5 GNN Explanations

As graph neural networks like most machine learning tools tend to resemble black boxes, oftentimes, they lack transparency. Most predictions produced by GNNs are difficult to explain on their own. Since in practice, most GNN models do not feature explainability-by-design, end-to-end methods are necessary to deal with these black box models. For this reason, a large amount of literature and methodologies dealing with the task of explaining the behavior of GNNs has emerged. Explanations produced by such tools may come in the form of local or global explanations. Local explanations are concerned with individual model predictions, seeking to explain why in a specific given case, a model made the given decision. Global explanations on the other hand attempt to work across all predictions made by a given model, establishing which features determine the output for any arbitrary input. However, the vast majority of existing research focuses on local explanations, while the amount of available global explanation techniques remains significantly smaller. [38, 7]

Depending on the task a given GNN model is performing, such explanations may come in different forms. In the case of graph classification tasks, a common kind of explanation comes in the form of edge masks. Such masks assign each edge in the graph a value representing it's importance to the resulting prediction. Similarly node feature masks may consist of node feature masks, indicating the importance of each entry in the given node feature vectors.

When dealing with node classification, the first contrast to other situations is that not the entire graph that is operated on is relevant. For any node classification task, explanations need only consider the node's computation graph. For these computation graphs, explanations are then similar to those of graph classification tasks. For example, they may consist either of edge masks or node feature masks evaluating the importance of each edge or node feature in the computation graph to the node classification. However, another common type of explanation is a node mask indicating which nodes in the computation graph play a crucial role for the model's result. This is less common in the graph classification case.

2.5.1 GNNExplainer

GNNExplainer [37] is one of the earliest methods to generate explanations of arbitrary graph neural network models. It can be used to explain both node classification and graph classification models, returning an edge mask and a node feature mask in either case.

The basic approach of GNNExplainer is to create an edge mask M over the adjacency matrix and a node feature mask F over the node feature vectors in a given graph. These masks are multiplied element-wise with the adjacency matrix and each node feature with the resulting products being fed into the given GNN model to produce a prediction. This process is treated as a model itself, with the edge and node feature mask representing a form of trainable weights, that are to be minimized while keeping the GNN's produced prediction consistent. Formalized as an objective function this looks as follows

$$\min_{M,F} \log F_{\Theta,c}(G = A_G \odot \sigma(M), X = X \odot F)$$

which is to be minimized using stochastic gradient descent or a related method such as Adam, with $F_c(G, X)$ representing the result of the given GNN Θ on the Graph G and the node features X with respect to the class c, and A_G representing the graph G's adjacency matrix. The practice of this is described in Algorithm 1.

Algorithm 1 $GNNExplainer_{\Theta,G,epochs}(y)$

```
1: Initialize edge mask $M$ and feature mask $F$ as random
2: for $i = 0, ..., epochs$ do
3:     $loss = \log F_{\Theta,y}(G = A_G \odot \sigma(M), X = X \odot F)$
4:     Propagate loss backwards using Adam
5: return $M, F$
```

2.5.2 PGM-Explainer

PGM-Explainer [32] deploys a statistical approach to GNN explanation. It starts by introducing the idea of perturbation in which the feature vector of a chose node or edge is perturbed, for example by setting every entry in it's vector to the average of the vector's entries.

Algorithm 2 $PGM - Explainer_{\Theta,G}(y)$

```
1: Choose 50% of $G$'s nodes at random as set $N_1$
2: for each $n \in N_1$ do
3:     Perturb $n$'s features
4: Initialize $P_1 = \{\}$
5: for each $n \in G$ do
6:     Insert $p = \tilde{\chi}^2(n, y)$ into $P_1$
7: Choose top 20 nodes by p-value as set $N_2$
8: for each $n \in N_2$ do
9:     Perturb $n$'s features
10: Initialize $P_2 = \{\}$
11: for each $n \in G$ do
12:     Insert $p = \tilde{\chi}^2(n, y)$ into $P_2$
13: return $P_2$
```

Algorithm 2 displays PGM-Explainer's procedure in pseudocode form, explaining the prediction y by the GNN Θ over the graph G. First, it randomly perturbs half of all nodes in the graph. It then performs the chi-squared test on all nodes, determining the significance of each node to the final prediction. It then repeats this process with the twenty most important nodes. This procedure effectively creates a probabilistic graphical model, also called a Bayesian network, over a given graph's nodes, quantifying their importance to the given model's prediction. From these node values, an edge mask is then derived by multiplying each edge's adjacent node's importance values.

2.5.3 Zorro

In contrast to GNNExplainer and PGM-Explainer, Zorro [3] attempts to produce hard node masks, i.e. a discrete subset of nodes, as an explanation of a given prediction. In order to do this, the authors apply a greedy algorithm that iterates over the given model's input graph. In this process, it successively selects the nodes to be included in the produced explanation of a given prediction. It should be noted that Zorro also provides a subset of features that is important to the explanation, however we focus on nodes in this case.

Zorro utilizes a concept called fidelity. Given a perturbed input, fidelity essentially measures the difference between the resulting prediction under the perturbed input versus the original input. This can be applied to measure the fidelity of an explanation, for example a node subset, by randomly perturbing nodes that are not included and then measuring the fidelity. Given an explanation S, it's fidelity is mathematically defined as

$$F(S) = \mathbb{E}_{Y_S|Z\sim\mathcal{N}}[\mathbb{1}_{\Theta(X)=\Theta(Y_S)}]$$

and computed empirically as displayed in Algorithm 3.

Zorro's process utilizes this concept of fidelity by first initializing the explanation, i.e. the set of relevant nodes, is as an empty set. Following, the algorithm iteratively chooses one node at a time to include in the explanation, selecting the node that maximizes the explanations fidelity in each step. Finally, the selection process terminates once the explanation's fidelity can no longer be improved. The resulting set of nodes represents the subset of the input graph that is relevant to the given model's result. The pseudocode

Algorithm 3 $F_{\Theta,G,samples}(S)$

```
1: Initialize $c = 0$
2: for $i = 0, ..., samples$ do
3:     Compute $G_S$ by randomly perturbing nodes in $G$ other than those in $S$
4:     if $\Theta(G_S) = \Theta(G)$ then
5:         Increment $c$ by 1
6: return $\frac{c}{samples}$
```

for this procedure is displayed in Algorithm 4.

Algorithm 4 $Zorro_{\Theta,G,\tau,K}(y)$

```
1: Initialize $S$ as empty set
2: while $F(S) \geq \tau$ do
3:     $\tilde{n} = \arg\max_{n \in top_K(G)} F_{\Theta,G}(S \cup n)$
4:     Insert $\tilde{n}$ into $S$
5: return $S$
```

A second variant of this method is Soft Zorro[1]. The idea of Soft Zorro is to keep the basic procedure of Zorro but combining it with the approach of learning by gradient descent. The reasons for this are twofold. One, that while Zorro tends to yield good results, these come at the cost of relatively high computational complexity through it's combinatorial exploration of the input graph. And two, that GNNExplainer tends to not produce very useful results in practice. Soft Zorro therefore attempts to tackle the core issues of both techniques by combining their advantages. This is done by adapting the fidelity scoring mechanism of Zorro where instead of exploring all possible fidelity improvements and taking the absolute best in an arg max manner, a softmax approach is applied, with the masks then being optimized using Adam. The process is displayed in Algorithm 5.

[1]As yet unpublished, material provided by Thorben Funke

Algorithm 5 $SoftZorro_{\Theta,G,\tau,K}(y)$

```
1: Initialize edge mask $M$ and feature mask $F$ as random
2: for $i = 0, ..., epochs$ do
3:     Calculate loss using the modified softmax fidelity
4:     Propagate loss backwards using Adam
5: return $M, F$
```

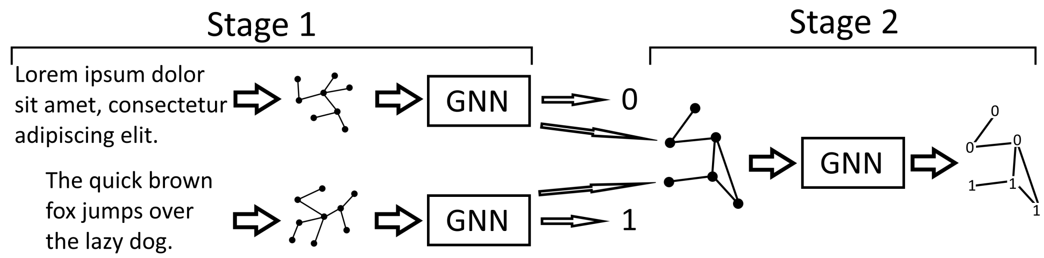
To conclude, Zorro and Soft Zorro are two methods of explaining a graph neural network's prediction by providing discrete subsets of the given graph's nodes and features that are crucial to the computed result. While Zorro achieves this through combinatorial means of comparatively iterating through possible combinations of node and feature subsets, Soft Zorro utilizes gradient descent learning similar to GNNExplainer in order to iteratively learn node and feature masks.

Chapter 3

Experiments

3.1 Methodology

3.1.1 Classification



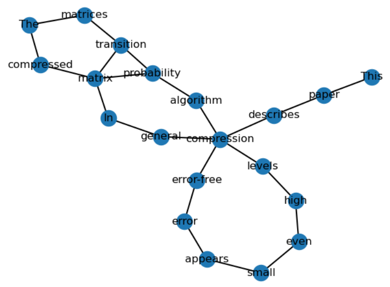
Stage 1
Stage 2
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
The quick brown fox jumps over the lazy dog.
GNN
GNN
0
1
GNN
0
0
0
1
1
1

Figure 3.1: The DepNet pipeline

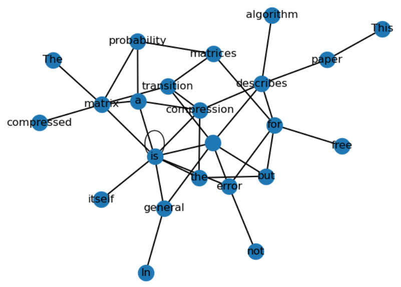
We provide a novel two-stage methodology we call DepNet for text classification utilizing both text-inherent information and inter-document relations. This methodology consists of two stages. In the first stage, we perform text classification by graph classification over a given dataset of documents by parsing each document into a syntactic dependency graph. In the second stage we perform text classification by node classification by constructing an inter-document relation graph from all documents in the dataset. To augment this process we use text embeddings produced by the first stage to increase the amount of information encoded in the document graph, combining the power of text-inherent information and inter-text relations for text

This paper describes a compression algorithm for probability transition matrices. The compressed matrix is itself a probability transition matrix. In general the compression is not error-free, but the error appears to be small even for high levels of compression.

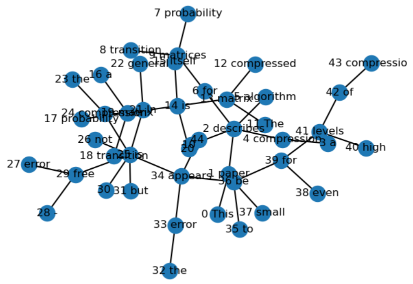
(a) Raw text



(b) CO Model



(c) Mixed Model



(d) DepNet

Figure 3.2: A sample text parsed by the three evaluated graph construction models

classification. Further, the first stage can be applied by itself when inter-document relations are not available.

The first stage of our methodology is in principle a simple text classification by graph classification task similar to much previous work. However, we provide a novel approach to graph construction. As shown in Section 2.1, most previous text classification by graph classification applies a word co-occurrence technique to graph construction, where nodes only represent unique words and edges represent co-occurrence in proximity. We instead deploy syntactic dependency parsing for graph construction. The resulting graphs contain all syntactic tokens within the text as nodes, with edges representing dependency relations between these tokens.

In our basic DepNet model, we disregard the specific labels associated with each dependency graph edge that specify the kind of relation the edge represents. However, we also examine whether their inclusion can improve the model's accuracy. We parse these dependency relations into edge feature vectors that can be fed into a GNN model by encoding them as one-hot vectors, with each entry in the vector representing a specific type of dependency relation.

In order to evaluate this approach, we also deploy a word co-occurrence methodology with a window size of three to build a baseline dataset. Notably, in this case during text preprocessing we also perform stopword removal [1] in order to maximize consistency with previous literature. Word embeddings for out-of-vocabulary words are drawn from a continuous uniform random distribution $\mathcal{U}_{[-0.01,0.01]}$.

In addition to the conventional word co-occurance model and our DepNet framework, we also consider a mixed approach. In this mixed model, we tokenize the given text and create a node in the graph for each unique word. However, instead of deriving edges between these nodes from the word co-occurance approach, we parse the text again using a syntactic dependency parser, applying dependency relations produced by this parser as edges. Effectively, this version of our graph construction is similar to the DepNet model except that we merge tokens that occur multiple times into a single node. The resulting model can be thought of as either a word co-occurance graph with dependency relation edges or a syntactic dependency graph with only unique nodes. Examples for each graph construction model we examine are given in Figure 3.2.

To incorporate as much semantic information in our dataset as possible while combining it with the syntactic information encoded in the graphs by dependency parsing, word vectors are chosen as node features in every model variant. To keep consistency with the existing literature [39], we consider two of the most widely used word vector models, Word2Vec and GloVe.

If inter-document relations are available, the second stage of our methodology can be applied. In this stage, we construct a graph from the given set of documents with each node representing a text and each edge representing an inter-document relation. Further, we apply a text embedding produced by our first stage as the feature vector for each node in this text relation graph. This represents the main novelty of our second stage, as in most text classification by node classification cases conventionally node features contain at most shallow embeddings of each document. In the case of CORA-ML [30] for example, node feature vectors provided in the dataset merely contain binary word vectors encoding the presence of words from the entire dictionary of the dataset in the corresponding text. Further, in order to preserve consistency by keeping the same GNN operators across all experiments, we process these edge features into learnable edge weights instead of applying them directly.

While our focus is on graph construction as a framework that can be used with arbitrary GNN operators, in order to evaluate our approach we run concrete graph and node classification experiments in both stages. To try and keep these experiments as focused on the graph construction approach as possible, we choose two GNN operators in both stages in the GCN [15] and the GraphConv [23] operator activated by the ReLU function, which are represented by the following equations for each GNN layer, with the former representing the GCN operator and the latter the GraphConv operator:

$$h_i^k = \text{ReLU}(w^T \sum_{j \in N(i) \cup i} \frac{e_{i,j}}{\sqrt{\hat{d}_i \hat{d}_j}} h_i^{k-1})$$

$$h_i^k = \text{ReLU}(w_0 h_i^{k-1} + w_1 \sum_{j \in N(i)} e_{i,i} * h_j^{k-1})$$

where w, w_0, and w_1 are trainable weights, h_i^k is the feature vector of node i at layer k, $e_{i,j}$ is the edge weight associated with the edge (i, j), and $\hat{d}_i$ is defined as follows:

$$\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$$

Notably, in the case of the GCN operator, we do not add self-connections. This is mostly to preserve compatibility with explainers such as PGM-Explainer.

We keep the rest of the model as simple as possible as well. For the graph classification tasks in the first stage, we apply mean pooling as the graph aggregation function and a one-layer linear perceptron f as the final readout, resulting in the following readout function:

$$o = f(\frac{1}{|V|}\sum_{v \in V} h_v)$$

For the node classification tasks in the second stage, we apply a softmax readout function after the last GNN layer:

$$o_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$$

with o_i and h_i representing the i-th element of the corresponding vector.

3.1.2 Explanation

When considering whether possible accuracy improvements from our graph construction framework are robust or merely coincidental, it is helpful to examine explanations of model predictions. Accordingly, we conduct a qualitative explainability analysis comparing our approach to previous techniques in order to both explain our models' results and to verify the accuracy gains over the baseline approach.

As explained in Section 2.5, when running graph classification tasks such as the first stage of our methodology, explanations of a model prediction commonly consist of an edge mask that assigns each edge in the graph a value representing it's importance for the resulting classification. We apply two methods of generating such edge masks, GNNExplainer and PGM-Explainer[1], on sample documents from our studied dataset. We compare the resulting edge evaluations in order to determine whether the model's decision making processes differ significantly. Additionally, we run a small quantitative comparison of the applied explanation methods in order to evaluate and

[1]For both explainers we use the implementations provided by Wang et al. [33]

support their results. This comparison is done by utilizing mutual information between two explanations, which is a measure of how much one variable predicts another or how much information one variable contains about another. For example, a mutual information value of zero indicates represents complete independence between two variables.

As the second stage of our method consists of node classification, we turn to a different explanation technique. In node classification, the task is to explain the prediction a model produced for a given node instead of the entire graph. Therefore, explanations explore the computation graph associated with the given node. Typically, such explanations consist either of an edge mask evaluating the importance of each edge in the computation graph to the node classification, similar to the graph classification case, or a node mask indicating which nodes in the computation graph play a crucial role for the model's result. We again apply two explanation methods to compare our DepNet model to the baseline, however, in this stage we use GNNExplainer to draw edge masks and both Zorro and Soft Zorro to draw node masks. We again compare the results between DepNet and baseline across all three explainers to investigate differences between the models and perform a supplementary comparison of the applied explainers themselves. For the latter, we use the Jaccard index, which is a measure of the similarity between two sets and defined as the ratio of the size of two sets' intersection to the size of their union.

3.2 Results

3.2.1 Experimental Setup

In order to examine the first stage, we run several experiments by building various datasets from the CORA-ML[2] set of documents. As word embeddings we consider Word2Vec[3] and GloVe[4] embeddings, and as dependency parsers we consider spaCy and Stanza. In the case of Word2Vec embeddings, we choose 300-dimensional vectors trained on Google News data with a dic-

[2]https://github.com/abojchevski/graph2gauss/tree/master/data
[3]https://code.google.com/archive/p/word2vec/
[4]https://nlp.stanford.edu/projects/glove/

tionary size of three million, while in the case of GloVe embeddings we choose 300-dimensional vectors trained on Wikipedia and Gigaword data with a dictionary size of 400,000. For each of these experiments, we randomly split the examined dataset into a training set and a testing set at an 8:2 ratio. The applied GNN model is then trained for 100 epochs at a learning rate of 0.01 using the Adam optimizer. All reported results are the average accuracy and the corrected sample standard deviation from ten independently run experiments. All experiments are done twice using the GCN and the GraphConv operator using two layers, except for the very first one, in which case only the GCN operator is used.

As the second stage consists of node classification tasks, we change the experimental setup slightly. For each experiment in this phase, we randomly select 5% of the graph's nodes as the training set, while the other 95% are used for accuracy testing. In each experiment we again conduct ten independent runs of training for 100 epochs using the Adam optimizer with a learning rate of 0.01. We again report the average accuracy and corrected sample standard deviation. We perform every experiment twice, using two layers of both the GCN and the GraphConv operator.

3.2.2 Classification

	spaCy	Stanza
Word2Vec	0.7787 ± 0.011	0.7686 ± 0.0322
GloVe	0.7504 ± 0.016	0.7284 ± 0.0191

Table 3.1: Mixed model performance across two different word embeddings and dependency parsers

	spaCy	Stanza
Word2Vec	0.7941 ± 0.0119	0.7861 ± 0.0087
GloVe	0.7610 ± 0.015	0.7341 ± 0.0193

Table 3.2: DepNet model performance across two different word embeddings and dependency parsers

Initially we compare the accuracy of the first stage of our method across different word embeddings and dependency parsers for both our mixed and our DepNet model. The results are displayed in Tables 3.1 and 3.2. Word2Vec significantly outperforms GloVe as the choice of word embeddings in all cases, by at least three percentage points. Similarly, the performance of spaCy as the dependency parser is strictly higher that that of Stanza, to a slightly lesser extent, leading to the conclusion that the combination of the spaCy dependency parser and Word2Vec word embeddings is the superior choice in all situations. The standard deviation remains very small in all cases.

Dataset	GCN	GraphConv
CO	0.7610 ± 0.0154	0.7666 ± 0.0232
CO-0	0.7687 ± 0.0260	0.7550 ± 0.0297
Mixed	0.7787 ± 0.0110	0.7802 ± 0.0153
Mixed-0	0.7709 ± 0.0108	0.7760 ± 0.0150
DepNet	0.7941 ± 0.0119	0.7987 ± 0.0147
DepNet-0	0.7955 ± 0.0179	0.7973 ± 0.0113
DepNet-EF	0.7921 ± 0.0170	0.7899 ± 0.0199

Table 3.3: First stage model performance

Next, we compare the first stage of our method using Word2Vec vectors and the spaCy dependency parser as well as our mixed model to the conventional word co-occurrence technique as a baseline. This is done by building different datasets from the set of documents, which we denote by "CO" for the co-occurrence baseline, "Mixed" for the mixed approach, and "DepNet" for our main model. Additionally, in this experiment we compare zero vectors to vectors drawn from a uniform random distribution as word embeddings for out-of-vocabulary words, denoted by the suffix "-0" in the corresponding datasets. Further, we considered syntactic dependency relation types as edge features and whether their inclusion might improve the observed accuracy. The dataset in this case is denoted by the suffix "-EF". In order to preserve comparability and consistency with the rest of our results, the inclusion of edge features happens in the form of running each edge feature vector through a one-layer perceptron to produce edge-weights that are further adapted in training. The results are shown in Table 3.3.

In this stage, replacing word co-occurrence graphs with syntactic dependency graphs yields an accuracy improvement of about three percentage points, with a small variance between the usage of zero vectors and randomly sampled vectors for out-of-vocabulary words. This increase is consistent across both applied GNN operators. Further, the mixed model's performance falls almost exactly in between the co-occurrence approach and the first stage of DepNet, further strengthening the case for syntactic dependency graphs providing significant amounts of information for the graph classification model.

Notably, we find that the inclusion of edge feature information does not yield an improvement in model accuracy. While our results even show a slight decrease, we believe that the difference in performance here is too small to be of any strong significance. The lack of improvement might be explained by the semantic information encoded in the word embeddings already providing most additional information that dependency relation types incorporate. For example, in the case of a negation, the word "not" might already contribute any benefits that the dependency relation type would specify through it's word vector.

Dataset	Baseline	CO	Mixed	DepNet
Baseline	0.7665 ± 0.0184			
Input		0.5884 ± 0.0855	0.5435 ± 0.0795	0.5956 ± 0.0689
Hidden		0.7739 ± 0.0397	0.8143 ± 0.0253	0.8410 ± 0.0243
Output		0.5250 ± 0.1357	0.7746 ± 0.0817	0.8415 ± 0.0255
Output-0		0.8107 ± 0.0476	0.8259 ± 0.0799	0.7548 ± 0.0952

Table 3.4: Second stage GCN performance

Dataset	Baseline	CO	Mixed	DepNet
Best embedding		Output-0	Output-0	Output
Accuracy	0.7665 ± 0.0184	0.8107 ± 0.0476	0.8259 ± 0.0799	0.8415 ± 0.0255
Improvement		0.0442	0.0594	0.075

Table 3.5: Second stage GCN improvements over baseline

Dataset	Baseline	DepNet
Baseline	0.7729 ± 0.0095	
Input		0.6476 ± 0.0687
Hidden		0.8373 ± 0.0247
Output		0.8495 ± 0.0266
Output-0		0.7872 ± 0.0345

Table 3.6: Second stage GraphConv performance

Dataset	Baseline	DepNet
Best embedding		Output
Accuracy	0.7729 ± 0.0095	0.8495 ± 0.0266
Improvement		0.0765

Table 3.7: Second stage GraphConv improvements over baseline

Finally, we deploy the second stage of our methodology. We draw text embeddings from different layers of the graph classification network and apply them as node features to a node classification GNN running on the inter-document relation graph from the CORA-ML dataset. Specifically, in the first case we draw these graphs embeddings from initial input word embeddings across the graph, in the second case from the embeddings produced by the first layer of the first stage GNN, and in the third case from the final embeddings produced by the second layer. We denote these cases by "Input", "Hidden", and "Output" respectively. Additionally, we draw the output layer's embeddings using zero vectors for out-of-vocabulary words, denoted by the suffix "-0". We compare the accuracy of this technique to the usage of the conventional node features included in the dataset, denoted as "Baseline". These results are displayed in Tables 3.4 and 3.6 for the GCN and GraphConv operator respectively. Tables 3.5 and 3.7 further display which layer's embedding from the first stage lead to the best accuracy and the improvement over the baseline results.

Here, we observe significant increase in accuracy when replacing the baseline binary word presence vectors with text encodings produced by a graph classification model, up to a robust 7.5 percentage point increase in accuracy when utilizing the first stage of our methodology. This increase is observed in both the GCN and the GraphConv scenarios. We further find that the gains in accuracy in the second stage is directly linked to the gain in accuracy in the first stage in the choice of model to produce node embeddings, which represents further evidence for the robustness of our findings. These improvements are most likely explained by DepNet combining the advantages of both graph classification and node classification, utilizing both intra-text information in the first stage and inter-text information in the second stage. Notably, we perform the second stage using a very small percentage of our dataset for the training phase which suggests significant inductive capacities.

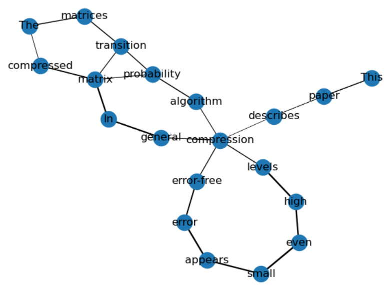
Some inconsistencies do arise however, most significantly in the differing performance between node embeddings produced using zero vectors versus randomly sampled vectors for out-of-vocabulary words. While the zero vector case seems to perform better for node embeddings produced by the word co-occurrence model and the mixed model, this reverses in the case of DepNet. Despite the DepNet model performance being consistent across both examined GNN operators, the reason for this difference is not immediately obvious and may require further research.

We also consider the direct joint application of both stages of DepNet. In this setup, the first model is called directly within the forward function of the second model, in a loop over all nodes and their documents in the graph, to produce model-internal text and thus node embeddings. In this case, both models can be jointly trained within one process, using a model that directly takes both the inter-document relation graph and the raw documents as input. However, this experiment yields no practically viable results. We do not report any preliminary findings under this setup, as a single run of training takes about eight times as long as a stage one training and orders of magnitude longer than a stage two iteration, while producing merely about fifty percent accuracy. The question of why this approach failed is somewhat puzzling, however. One would imagine that the joint training would give the model the opportunity to hone the node embeddings produced by the first stage, optimizing them for the final node classification. While a possible explanation is the training set only being five percent of all nodes and thus documents in the graph, which might not give the first stage enough material to generalize from, we repeated the experiment with a training set of eighty percent without finding significant improvements. There is still a reasonable chance that this joint model might after some adjustment outperform the two-stage version, so we believe that further research on this issue would be useful.

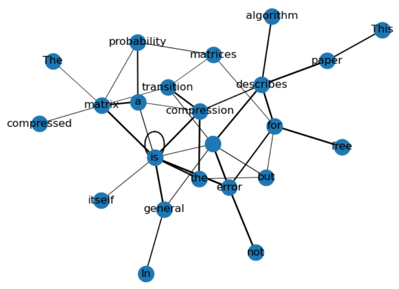
In contrast to the accuracy gains we observe, a possible disadvantage of DepNet is the additional effort needed compared to previous methods. While the graph construction by dependency parsing did not take significantly more time in our experiments than word co-occurrence parsing, the usage of the second stage of DepNet requires an entire additional course of graph classification over an entire given dataset compared to the baseline technique. Nonetheless, this process is only needed once in preprocessing, which is why we argue it is worthwhile. In any case, in any pure graph classification scenarios there is little downside to choosing syntactic dependency graphs over word co-occurrence graphs.

3.2.3 Explanation

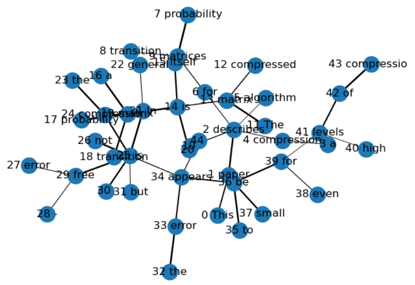
In order to explain the graph classification in the first stage of our method, we draw edge masks representing the relative importance of each edge in the graph to the explanation produced by the model. These explanations are produced for the conventional word co-occurance graphs, our mixed model, and our DepNet model. We deploy two different explanation techniques to produce these edge masks, GNNExplainer and PGM-Explainer. Figure 3.3 displays the GNNExplainer results for a sample text from the CORA-ML dataset, while Figure 3.4 displays the PGM-Explainer results for the same sample.



(a) CO Model



(b) Mixed Model



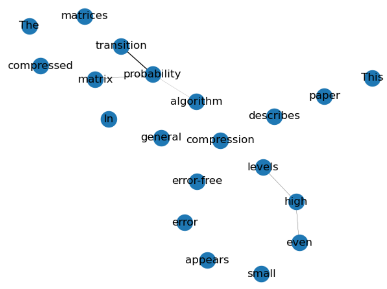
(c) DepNet

Figure 3.3: First stage GNNExplainer
Edge thickness indicates corresponding importance value

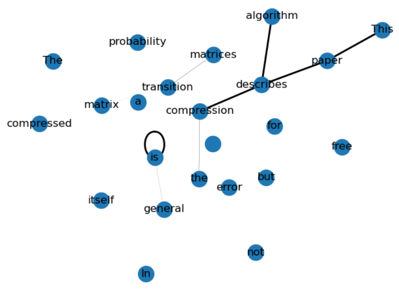
Dataset	MI
COM	0.0386
Mixed	0.0622
DepNet	0.0575

Table 3.8: Average mutual information between GNNExplainer and PGM-Explainer explanations across 30 documents classified by each model

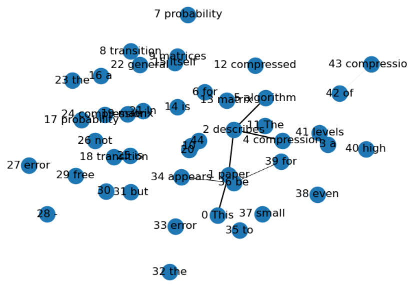
Our GNN explanation evidence supports the observed accuracy gains by the first stage of DepNet. The GNNExplainer results seem to be inconclusive as it appears that GNNExplainer simply does not produce explanations of the necessary detail to evaluate differences between the evaluated models. This observation is further supported by two quantitative examinations of the produced GNNExplainer results. First, we compare the histograms of edge masks returned by GNNExplainer and PGM-Explainer on a sample



(a) CO Model



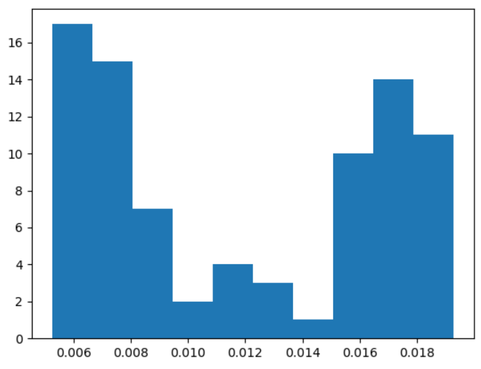
(b) Mixed Model



(c) DepNet

Figure 3.4: First stage PGM-Explainer
Edge thickness indicates corresponding importance value

document, displayed in Figure 3.5. The explanation produced by GNNExplainer clearly carries much less information than those produced by PGM-Explainer, with the entire range of values only reaching a fifth as high and the edge mask clearly containing significantly more noise. Second, we pick a sample of twenty documents from the CORA-ML dataset. We produce classifications of these texts using all three model variants, then produce explanations for these classifications using both GNNExplainer and PGM-Explainer. Finally, we calculate the mutual information between the two explanations for each document and model. We calculate the averages across the 30 sample documents per model variant which are displayed in Table 3.8. The figures we found are clearly very low, in the single digit percentage range, indicating that explanations produced by GNNExplainer tend to be significantly less expressive than and therefore inferior to those produced by PGM-Explainer.


(a) GNNExplainer

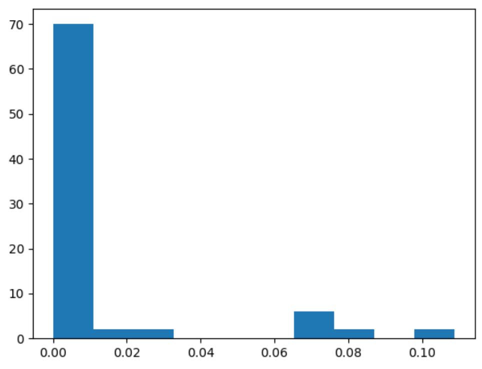
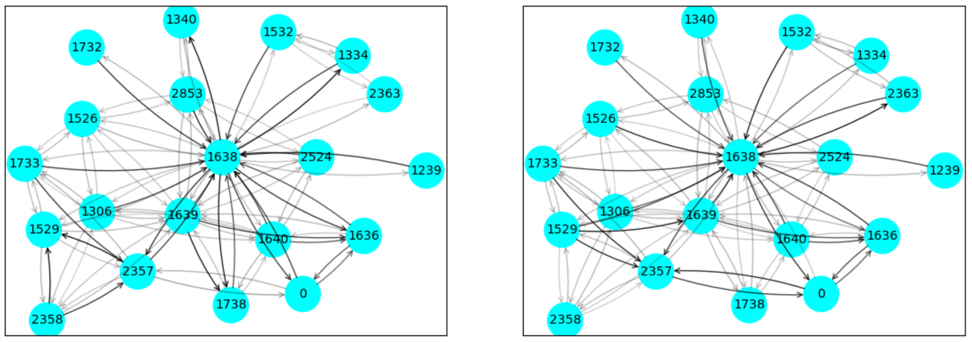

(b) PGM-Explainer

Figure 3.5: Histograms of edge mask explanations over a sample text

PGM-Explainer on the other hand provides significant supporting indications. The evidence here clearly shows that the GNN model's focus shifts when presented with both our mixed model and DepNet compared to the word co-occurrence model baseline. This change being consistent between the mixed model and DepNet indicates that it does indeed stem from the syntactic information encoded in dependency relation graphs. The accuracy difference between these two models perhaps then stems from the additional semantic information included in the DepNet graph, since the presented word nodes and therefore word vectors more closely represent their actual distribution within the parsed text.

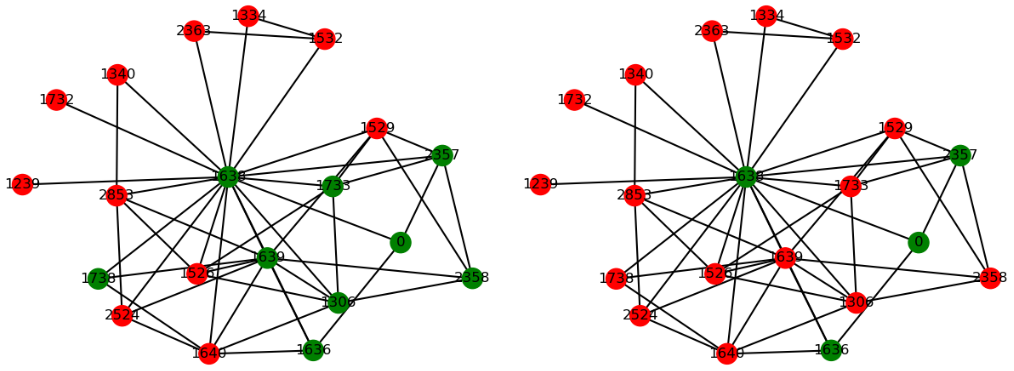
As the second stage consists of node classification, we turn to two different forms of explanations to investigate the difference between DepNet and the baseline model. First, we again run GNNExplainer, this time returning an edge mask indicating each edge's importance to the final prediction over the computation graph of a selected sample node, in our case being node 0. The results of this are displayed in Figure 3.6. Additionally, we opt to run two further explanation methods in Zorro and Soft Zorro. In contrast to GNNExplainer and PGM-Explainer, these do not return edge masks but rather a discrete subset of the computation graph, indicating which nodes are important to the resulting prediction of node 0. The results for Zorro are



(a) Baseline (b) DepNet

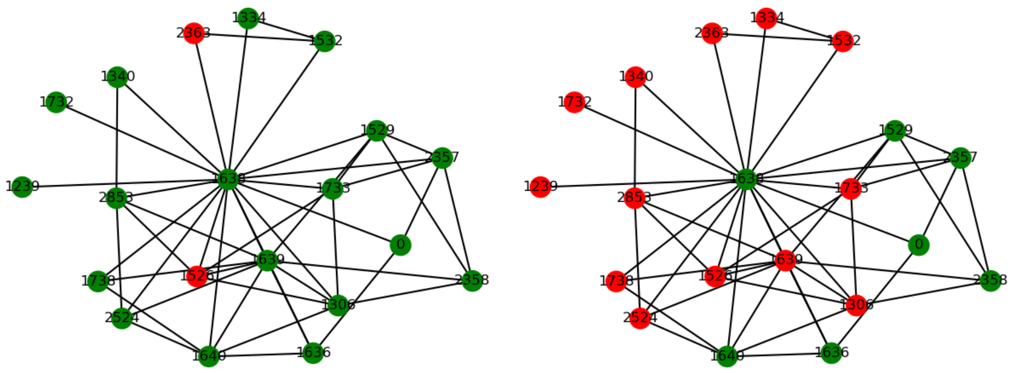
Figure 3.6: Second stage GNNExplainer on node 0
Edge thickness indicates corresponding importance value

displayed in Figure 3.7, the results for Soft Zorro in Figure 3.8.



(a) Baseline (b) DepNet

Figure 3.7: Second stage Zorro on node 0
Green nodes are determined to be relevant to the prediction



(a) Baseline (b) DepNet

Figure 3.8: Second stage Soft Zorro on node 0
Green nodes are determined to be relevant to the prediction

	Baseline	DepNet
Zorro	0.4078	0.2599
Soft Zorro	0.3697	0.2708

Table 3.9: Average share of the computation graph included in a node's explanation

As in the first stage, evidence from generated explanations of the GNN model support the case for the robustness of the observed accuracy improvements. While GNNExplainer again produces explanations of relatively little informational value, in this case they at least appear to suggest a clearer hierarchy and information flow within the explained node's computation graph. Both Zorro and Soft Zorro, on the other hand, yield strong supportive evidence. In both cases, using the node embeddings produced by the first stage of DepNet significantly reduces the size of the produced explanation, indicating that significantly fewer neighboring nodes are necessary for a successful classification. In order to confirm that this finding is not limited to the sample node we visualize but a consistent observation, we select a larger sample of thirty nodes, generate explanations for each node using both Zorro and Soft Zorro for both the baseline case and DepNet, and compute the average share of each node's computation graph that is included in the resulting explanation. The results are displayed in Table 3.9, indicating that the application of DepNet reduces the relevant share of a node's computation graph for it's classification by at least a quarter. This might even suggest stronger inductive capabilities when deploying DepNet over baseline methods.

Baseline	0.2911
DepNet	0.4033

Table 3.10: Jaccard indices between Zorro and Soft Zorro explanations across 30 sampled baseline and DepNet classifications

Lastly, we provide a brief comparative analysis of Zorro and Soft Zorro in order to validate their explanations. If either of the two provided methods were to produce questionable explanations, we would expect to observe a strong divergence, such as for example between GNNExplainer and PGM-Explainer. If the two tools on the other hand produce explanations that prove similar, it would appear to be relatively unlikely that both methods provide largely invalid results. As the explanations produced by Zorro and Soft Zorro consist of discrete subsets of a node's computation graph, we turn to the Jaccard index in order to perform this comparison and evaluate the disparities between both explanation techniques. We choose a sample of thirty nodes, classify them using both the baseline model and DepNet, explain each classification using both Zorro and Soft Zorro, calculate the Jaccard index between the two explanations for each model and node, and then calculate the mean of the Jaccard indices per model. The results are displayed in Table 3.10. As is clearly visible, Zorro and Soft Zorro diverge to a far less significant extent than GNNExplainer and PGM-Explainer in their explanations. In fact they tend to agree to a considerable extent, indicating both methods provide valid and credible explanations.

Chapter 4

Conclusion

By focusing on graph construction we provide a model-agnostic approach we call DepNet to text classification in the presence of inter-document relations via graph neural networks. This approach is of a two-part nature. First, we perform text classification via graph classification by constructing individual semantic dependency graphs from each text. Second, we apply graph embeddings produced by the first model as node features to a inter-document relation graph, on which we then perform node classification. Additionally, we apply various GNN explanation techniques to DepNet in order to explain and verify the observed differences in performance between our model and previous techniques.

We find that the application of syntactic dependency parsing instead of word co-occurrence techniques leads to considerable increases in accuracy when performing text classification by graph classification. Further, the application of node embeddings produced by a graph classification model run on the respective text significantly improves the results of text classification by node classification. Both of these observations are supported by qualitative GNN explanation evidence. We can therefore confirm our hypothesis and conclude that DepNet is able to both improve on the status quo of text classification by graph classification methods and provide a useful framework for text classification by node classification tasks.

Our findings are not without limitations. Further research is needed, especially to study the behavior and results of DepNet when utilizing more complex GNN operators, such as Graph Attention Networks. Further, while we believe that our observations represent a genuine advantage of syntactic dependency graphs over word co-occurrence graphs in GNN applications, more experiments examining DepNet on other datasets are needed to confirm and generalize our results.

Bibliography

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[2] JR Firth. A synopsis of linguistic theory, 1930—55. *InF. R. Palmer (ed.)*, 1968.

[3] Thorben Funke, Megha Khosla, and Avishek Anand. Zorro: Valid, sparse, and stable explanations in graph neural networks. *CoRR*, abs/2105.08621, 2021.

[4] W.L. Hamilton. *Graph Representation Learning.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.

[5] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy: Industrial-strength natural language processing in python. 2020.

[6] Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Text level graph neural network for text classification. pages 3435–3441, 01 2019.

[7] Mark Ibrahim, Melissa Louie, Ceena Modarres, and John Paisley. Global explanations of neural networks: Mapping the landscape of predictions. pages 279–287, 01 2019.

[8] Ozan İrsoy and Claire Cardie. Deep recursive neural networks for compositionality in language. In Z. Ghahramani, M. Welling, C. Cortes,

N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2096–2104. Curran Associates, Inc., 2014.

[9] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, pages 137–142, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[10] David Johnson, Frank Oles, Tong Zhang, and Thilo Goetz. A decision-tree-based symbolic rule induction system for text categorization. *IBM Systems Journal*, 41:428 – 437, 02 2002.

[11] Daniel Jurafsky and James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 2. 02 2008.

[12] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462 – 466, 1952.

[13] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[15] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[16] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4), 2019.

[17] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32

of *Proceedings of Machine Learning Research*, pages 1188–1196, Bejing, China, 22–24 Jun 2014. PMLR.

[18] Qian Li, Hao Peng, Jianxin Li, Congyin Xia, Renyu Yang, Lichao Sun, Philip S. Yu, and Lifang He. A survey on text classification: From shallow to deep learning. *ArXiv*, abs/2008.00364, 2020.

[19] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. In *IJCAI*, 2016.

[20] Xien Liu, Xinxin You, Xiao Zhang, Ji Wu, and Ping Lv. Tensor graph convolutional networks for text classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:8409–8416, 04 2020.

[21] M. E. Maron. Automatic indexing: An experimental inquiry. *J. ACM*, 8(3):404–417, jul 1961.

[22] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[23] Christopher Morris, Martin Ritzert, Matthias Fey, William Hamilton, Jan Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609, 07 2019.

[24] Giannis Nikolentzos, Antoine Jean-Pierre Tixier, and Michalis Vazirgiannis. Message passing attention networks for document understanding. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 8544–8551, 2020.

[25] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Bao Mengjiao, Lihong Wang, Yangqiu Song, and Qiang Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. pages 1063–1072, 04 2018.

[26] Hao Peng, Jianxin Li, Senzhang Wang, Lihong Wang, Gong Qiran, Renyu Yang, Bo Li, Philip Yu, and Lifang He. Hierarchical taxonomy-aware and attentional graph capsule rcnns for large-scale multi-label text classification. *IEEE Transactions on Knowledge and Data Engineering*, PP:1–1, 12 2019.

[27] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. volume 14, pages 1532–1543, 01 2014.

[28] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.

[29] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 06 2017.

[32] Minh N. Vu and My T. Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *CoRR*, abs/2010.05788, 2020.

[33] Xiang Wang, Ying-Xin Wu, An Zhang, Xiangnan He, and Tat-Seng Chua. Towards multi-grained explainability for graph neural networks. In *Proceedings of the 35th Conference on Neural Information Processing Systems*, 2021.

[34] Yiming Yang and Christopher G. Chute. An example-based mapping method for text categorization and retrieval. *ACM Trans. Inf. Syst.*, 12(3):252–277, jul 1994.

[35] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, page 42–49, New York, NY, USA, 1999. Association for Computing Machinery.

[36] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. *CoRR*, abs/1809.05679, 2018.

[37] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32:9240–9251, 12 2019.

[38] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *CoRR*, abs/2012.15445, 2020.

[39] Yufeng Zhang, Xueli Yu, Zeyu Cui, Shu Wu, Zhongzhen Wen, and Liang Wang. Every document owns its structure: Inductive text classification via graph neural networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 334–339, Online, July 2020. Association for Computational Linguistics.

[40] Xinjie Zhou, Xiaojun Wan, and Jianguo Xiao. Attention-based LSTM network for cross-lingual sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 247–256, Austin, Texas, November 2016. Association for Computational Linguistics.