Graph-based classification of text

Task 1 and Task 3 of the Assignment for Text Analytics (CE807)

Registration Number: 1804588
School of Computer Sciences and Electronics Engineering
University of Essex

ABSTRACT

Given the available computing resources in the last decade or so, there has been a massive surge in research and design of new artificial neural networks (ANN) which have achieved great breakthroughs in a number of fields such as computer vision using convolutional neural networks (CNNs), and in natural language processing using recurrent neural networks (RNN), to name just a few. However, these architectures have traditionally been applied to regular grids, whereas plenty of data come in the form of graphs. Recently, there have been growing interest in research on the topic of learning from graphs using neural networks. These neural networks are known as Graph Neural Networks or GNNs. There have been quite a few works carried out in document classification using GNNs that surpass state of the art. In this project we will review this body of work. We will then implement a text classification using a GNN and then compare our results with both benchmarks and state of the art. We will finally draw conclusions about our work and point to some future work that can be carried out. (this report contains both task 1 and task 3 of the assignment)

KEYWORDS

Graph neural networks; Graph convolutional Networks; Document classification; Semi-Supervised Learning

1 INTRODUCTION

There are plenty of learning tasks in which the data can be represented using graphs. A graph is made up of nodes, which are connected to each other using edges which contain rich relational information about the nodes they connect. Some concrete examples include social networks [1], modelling physics systems [2] and protein-protein interactions [3].

In graph analysis, machine learning can be leveraged in three distinct ways, namely, to predict node value, edge or link values, and for clustering of nodes depending on the task at hand. GNNs are neural networks that operate on graph domains. GNNs have proven to be high performing and of high interpretability [4], which has enabled researchers to apply it to a wide range of problems as mentioned above. One area where graph neural networks have shown great potential is in the area of natural language processing such as text classification, spam filtering, document organization, opinion mining and many more. In this project, we use GNNs for text classification, more specifically, we use GNNs to classify movie reviews into positive and negative reviews.

One important step in a text classification pipeline is text representation, which would have been traditionally done manually, such as converting text into bag-of-words, lemmatization etc. However, with the advent of ANNs, these representations are learned by the neural network. ANNs have been especially useful in findings word embeddings and document embeddings, which can capture relationships between words and documents by the distance that each word or document has in this new embedding coordinate system.

Furthermore, CNNs and RNNs are good at capturing local and sequential relations in text. However, they may ignore global relationships and structure in text. On the other hand, GNNs are good at capturing global structure and relations in text in the form of graph embeddings [5].

In this project, we implement a graph based model for text classification as proposed by Yao et al [5], in which they use a convolutional graph network (CGN) as first proposed by Kipf and Welling [6]. The graph is made up of nodes that represent both documents and words. The problem of text classification is then represented as a node classification problem. The document edges are connected to word edges using a tf-idf measure, and word to word edges are represented using a measure known as PMI. In the following sections we will explain each of these concepts in more depth. We will then analyze our results and we will end this report by drawing conclusions from it and point the reader to future work that can be carried out.

2 LITERATURE REVIEW

2.1 Traditional Text Classification

Traditionally, to use machine learning algorithms for the purpose of text classification, all the documents in a corpus would be converted into a fixed number of features which is then fed into the algorithm. There are several ways to do that. The most commonly used technique is Bag-Of-Words (BOW). Using this representation, a document is converted into a collection of words which occur in it at least once. However, there are many words in a document which occur a lot in a given language but do not carry any useful information with regards to the task at hand and can in fact substantially degrade the performance of the classifier. These words are known as stop words and are removed before creating a BOW. Another way which can be used to decrease the number of words is called stemming. Stemming is removing suffixes from words which are derived from a word which carry the same information as the longer words. For instance, the word "amazing" and "amazement" and "amazed" can be stemmed and be represented using the stem "amaz".

Even after applying the above methods, the remaining set of words can still be huge and therefore can cause curse of dimensionality. Therefore, other methods which rank words based on some criteria and chooses only the top words need to be employed. Some examples of these methods are "Term Frequency–Inverse Document Frequency" (tf-idf), Information Gain (IG), Mutual Information (MI) and Term Strength (TS). Yet, there are other methods that don't rank words but convert words into a lower dimensional representation such as Principal Component Analysis (PCA). There are also other, more complex methods for feature representations such as entities in ontologies [7] and n-grams [8].

2.1 Classifying Text Using Deep Learning

Text classification using deep learning is the process of applying deep ANNs to text classification. Yao et al [5] divide deep learning approaches into two categories. in the first category, deep learning is used to learn word or document embeddings or both, then using these embeddings, one can use a range of different machine learning approaches, including deep learning, for text classification (Pennington et al [9], Mikolov et al [10]. In this approach, the success of the resulting classification largely is dependent on the quality of the embeddings as shown in [11, 12]. in some works, such as the one carried out by Joulin et al [12] and Le and Mikolov [13], word embeddings learned using unsupervised learning are aggregated to form document embeddings which is then used for document classification. In other works (Tang et al [14]) word embeddings, document embeddings and label embeddings are learned simultaneously which is then used for text classification.

The approach used in this project, which is a reimplementation of the work by Yao et al [5] and Kipf and Welling [6] shares some similarity with this approach in that the GCN also learns word and document embedding, although intrinsically. Yao et al [5] however have managed to extract these embeddings from their GCN.

In the other category of deep learning for text classification, no embedding is employed. Instead deep ANNs such as CNNs and RNNs are used to learn the underlying model. CNNs have been very successful in computer vision tasks. CNNs have also been applied to text classification tasks, with the difference that instead of a two-dimensional convolution, one-dimensional ones are applied to text. Work carried out by Zhang et al [15] and Conneau et al [16] are some examples of successful CNN application in text classification. LSTMs, which is a type of RNN have been very effective in text classification, given their ability to, in a way, remember patterns using attention mechanisms. some examples of successful application of LSTMS in text classification are work carried out by Tai et al [17] and Luo [18]. These methods are very good at capturing local relationships and structure. However, they lack the power to capture global relations and structure, such as global word co-occurrence, which is something GNNs can do well as will be discussed in the next section.

2.1 Text Classification Using GNNs

Over the last few years, research on GNNs has picked up (Cai et al [19]), and there has been a growing number of papers published regarding the application of GNNs in text classification. GNNs were first proposed by Scarselli et al [20], which has been since used in many natural language processing tasks. Defferrard et al [21] were the first to use GNNs in text classification which outperformed traditional CNNs for the same task. Yao et al [5] then built on the work by Defferrard et al [21] by constructing a graph which contained as nodes both words and documents which consequently outperformed the model by Defferrard [21]. However, there are a few shortcomings associated with the above approaches. First, they build a graph based on the corpus, which makes the learned model inflexible in terms of its applicability to new corpora. Second, the above approaches produce a great number of edges which makes training very memory intensive (as is the case with the

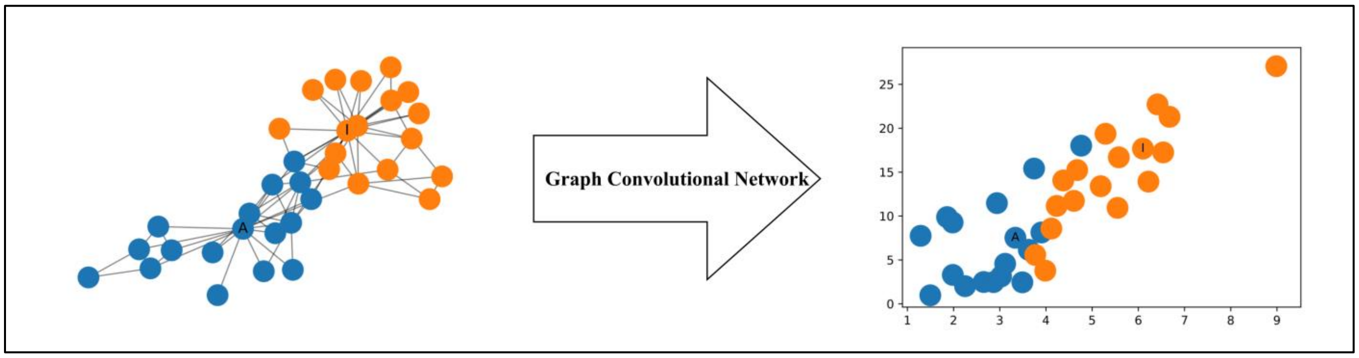

Figure 1. a GCN can produce a good representation of a graph even initially without any training and with randomized weights

approach taken in this project). To this end, Huang et al [22] construct a text level graph instead.
Other examples of the use of GNNs in NLP tasks include relational reasoning (Battaglia et al [23]), neural machine translation (Bastings et al [24]) and sequence labeling (Zhang et al [25]).

3 METHOD

The implementation that we have used in this project closely follows the implementations used by Kipf and Welling [6] and Yao et al [5]. It differs from both in certain implementation details which we will point out in this section. Some sections of the code has been directly adopted from an online post by Jepsen [26] which is available at: https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-62acf5b143d0
As an overview, the way our GCN works is first we construct a graph containing both documents and words as nodes. We then treat this problem as a semi-supervised learning task to learn the label of the test documents.

3.1 The Data

The data for this assignment is a dataset about movie reviews on the movie aggregator website "Rotten Tomatoes". From this point on we will refer to this dataset as MR, to keep in line with the abbreviation used in [5]. We have downloaded the data from a database maintained by Cornell university from the following link: https://www.cs.cornell.edu/people/pabo/movie-review-data/.
The reason we have chosen this dataset is firstly because it is a single label classification task and there are only two classes for the label, namely positive and negative, which makes it a relatively simple dataset for the purposes testing our learning model. But also it is chosen as it is one of the datasets used in Yao et al [5] and many other researches which Yao et al [5] use to compare their results with. Each document in this corpus is one sentence and most of the reviews are in English, apart from a few rare cases which are in other languages. There is a total of 10662 reviews. In Yao et al [5] and [27] 2/3 of the data is used for training and the rest for testing, and that is the split that we have used as well. Furthermore, we have used the same exact instances as in Yao et al [5] for training and testing.

3.2 Construction and Rules of the Graph Neural Network

GCNs are multi-layer neural networks that have proven to be a powerful tool for analyzing graphs, even a two layer neural network has the ability to produce useful representation of a graph as shown in Figure 1 (adapted from [26]). a graph of the form $G = (V, E)$, where V represent the set of vertices of the graph and E is represent the set of edges connecting the vertices. A GCN takes as input:

- a feature matrix $X \in \mathbb{R}^{n \times m}$, where n is the number of vertices and m is a feature representation for that vertex.
- An $N \times N$ matrix called the adjacency matrix A

In our implementation, the input feature matrix X is just the one-hot encoded matrix of all the vertices in the graph.
Therefore, a hidden layer of our GCN can be represented as:

$$H^i = f(H^{i-1}, A)$$

Where $H^0 = X$ and f is the propagation rule. Therefore, the first hidden layer is calculated as follows:

$$H^1 = \rho(AXW_0)$$

Where ρ is an activation function such as ReLu, and $W_0 \in \mathbb{R}^{m \times k}$ is the weight matrix.
However, there are a few problems with the above equation. First, vertices in A are not considered a neighbor of themselves, which means information regarding the vertex

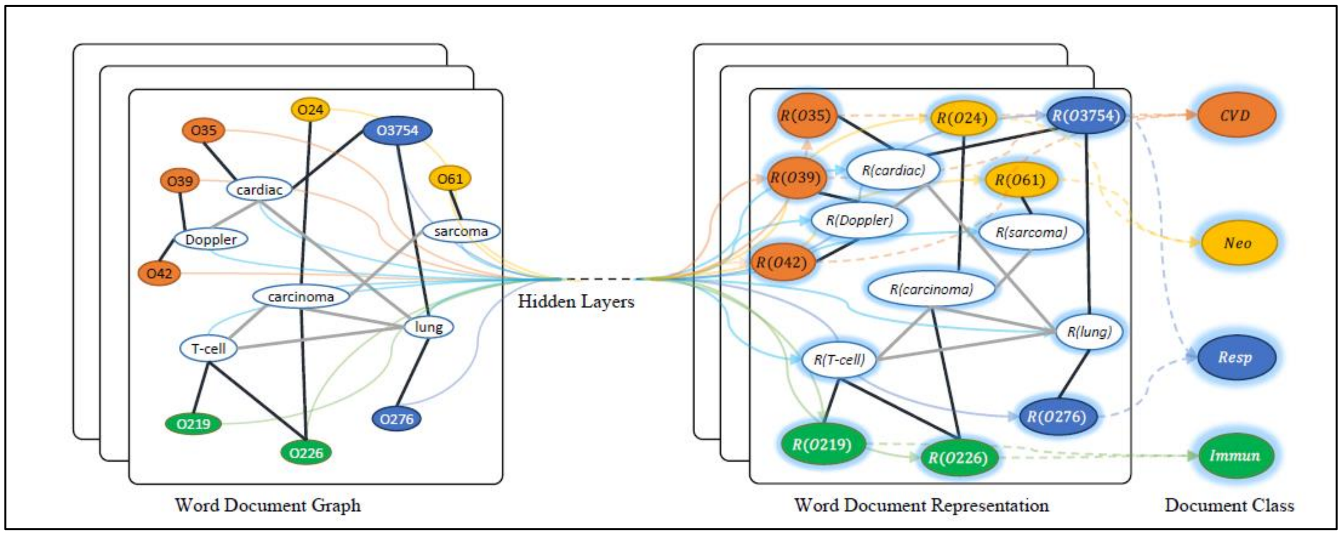

Figure 2. This figure shows the overall structure of a convolutional graph network. The graph on the left-hand side shows the heterogeneous graph representation of a corpus called Ohsumed. Nodes beginning with O are document nodes and the white nodes are word nodes. The graph on the right-hand side is the embedding of the nodes on the left after the graph has passed through a GCN.

is not taken into account in the neural network. To get around this problem we can add self-loops to A by adding to it the identity matrix I which will give us $\hat{A}$.
Second problem is that vertices with plenty of neighbors or too few neighbors can cause problems with vanishing or exploding gradients. To get around this problem, we can use a degree matrix $D \in \mathbb{R}^{n\times n}$, where $D_{ii} = \sum_j A_{ij}$ as follows:

$$\tilde{A} = D^{-0.5}\hat{A}D^{-0.5}$$

Therefore, our updated equation for a hidden layer calculation becomes:

$$H^{i+1} = \rho(\tilde{A}H^{i}W_{i})$$

Now that we have defined how each layer is constructed, we stack the layers next to each other to get our full neural network. In this project, we have used two hidden layers. In the first hidden layer, there are 4 neurons and the second hidden layer has 2 neurons. We have chosen these numbers as this is what has been used in [26]. The reasoning given in that article is that having a two-node hidden layer makes it easy to visualize. Finally, we use a logistic regression layer as the output layer which is used for node classification.
In the implementation of the neural network in code, it has been separated into two components. The first component, called `features` contains the graph convolutional layers and the other component called `classifier` contains the classification part of the network, which is the last layer. The reason the implementation is divided as such is to make the visualization of the activation of the hidden layers easier. There is also a `LogisticResgressor` classification layer that performs logistic regression by summing over the features of each node from the previous layer.

3.3 Construction of the Graph

As mentioned earlier, we follow the graph making procedure as outlined in [5], in which a heterogeneous graph is constructed using the documents and words in the corpus. like Yao et al [5], our graph has word-word edges and word-document edges but no document-document edges. Figure 2 is an illustration of a heterogenous graph which is fed to a GCN. The figure is taken from Yao et al [5]. In their implementation, Yao et al do not seem to use any data preprocessing, whereas we preprocess the data as follows:

- Punctuations, as defined in the "string.punctuation" module in python are removed.
- In the next step of preprocessing, all the stop words are removed. Stop words are words that happen very frequently in a language but do not convey any information regarding the document they appear in. the stop words we have removed are those defined in "nltk.corpus.stopwords" module in python.
- Also, we use a module in python to calculate the tf_idf of every word in a document, and one of the side effects of this module is that it removes all the single letter words and numbers such as "a" and "1".

Other than what is mentioned in the bullet points above. Every other word is remained in the dataset.

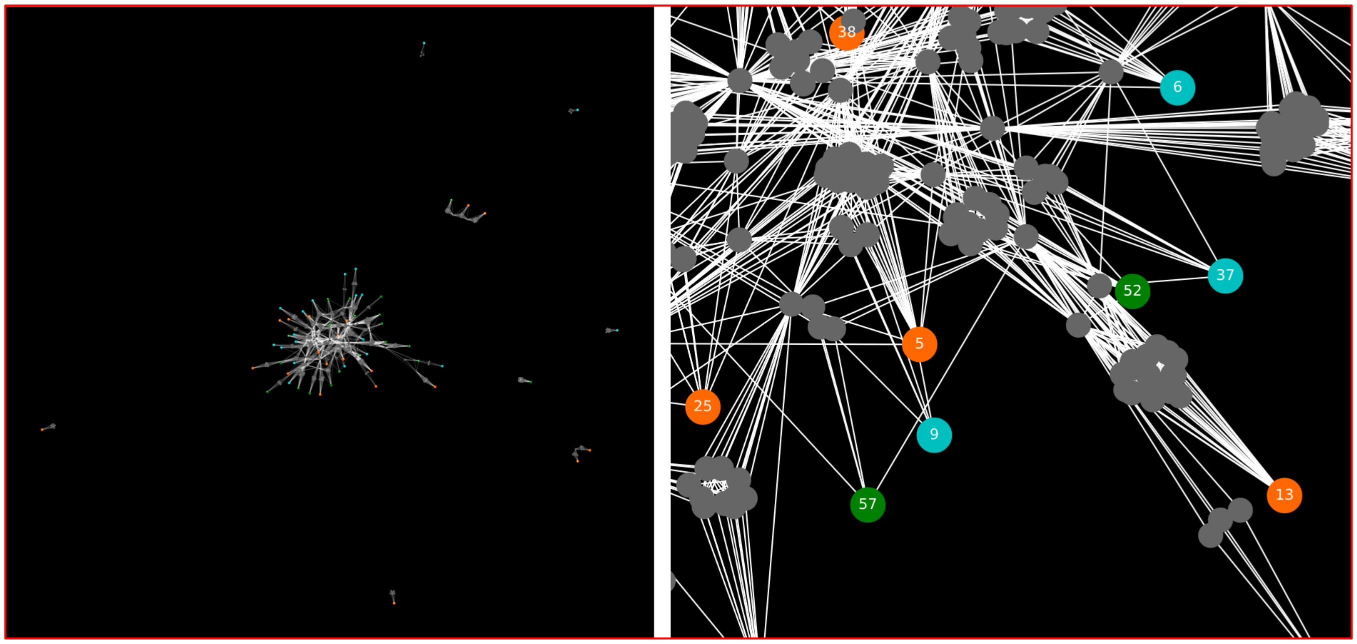

Figure 3. The image on the left shows a graph made using 60 movie reviews form the MR dataset The image on the right is a zoomed-in version of the same graph where grey nodes are word nodes and the colored nodes are document nodes, with the orange nodes representing positive reviews, cyan representing negative reviews and green representing nodes to be labeled i.e. documents in the text sets.

After the above preprocessing steps are applied, we then use tf-idf to calculate the edge weights between words and documents for the adjacency matrix A. The calculation used for the word-word edges in A is the same as the one used in [5]. This measure is called point-wise mutual information or PMI for short which is a measure of association which is used in information theory and it is calculated using the following formula:

$$PMI(i,j) = \log \frac{p(i,j)}{p(i)p(j)} \tag{1}$$

$$p(i,j) = \frac{\#W(i,j)}{\#W} \tag{2}$$

$$p(i) = \frac{\#W(i)}{\#W} \tag{3}$$

Where $\#W(i)$ is the number of times the word i appears in the corpus, $\#W(i,j)$ is the number of times the words i and j appear together within a window W and $\#W$ is the total number of sliding windows in the corpus. the length of the sliding window we have used is 20, as the average sentence in the English language is between 15 to 20 words [28]. If the PMI value is negative, it means that the words are weakly correlated and therefore we will set the value equal to zero, which means there would be no edge between the two words. Otherwise the value of the PMI will become the value of the edge between the two words.

We now have everything we need to make the graph. Figure 3 Shows a graph made using only 60 randomly chosen reviews from the dataset. As can be seen from the image on the left in Figure 3, there are some disconnected document nodes on the periphery of the main graph, which perhaps is a potential problem if any of these peripheral nodes are unlabeled, as they are not connected to any labeled nodes. However, as the number of nodes increases, the chances of having disconnected node decreases as there are more words to connect nodes together. This problem is especially prevalent with corpora with very short documents as is the case with our corpus. as the number of words in each document increases, the chances of having a disconnected node decreases, as there are more words, which consequently decreases the chances disconnected graphs.

3.3 Training the GCN

finally, we train the GCN as follows. First, we initialize a binary cross entropy loss function, `cross_entropy` and a stochastic gradient descend optimizer called `trainer` which learns the parameters of the network. Then the network is trained for a certain number of epochs, where in each epoch, the loss is calculated and the error is backpropagated using the function `loss.backward()`. The model parameters are then updated using `trainer.step()`. To summarize, adjacency matrix *A* and the one hot encoded feature matrix *X* is fed into the network, as well as the indices for the training instances and text instances and then the network tries to learn the labelling for the instances in the test set.

RESULTS

Given that there are over 10,000 instances, the resulting adjacency matrix end up having a size of over 20,000 including the word nodes, which makes it very memory intensive as well as CPU intensive. We therefore use the cluster at University of Essex to train our model. We trained our network with a variety of parameters such as different number of hidden layer nodes, using lemmatization and training with and without stop words. We also tried training with different proportions of the datasets, starting from a minimum dataset size of 120 and doubling the size of the dataset up to the full dataset. However, only those programs with a dataset size of 480 were completed in time for the project and none of them were able to achieve performance better than random guessing. There was one program with a dataset size of 960 which completed but also did not achieve a performance better than random guessing. It could be the case that we need to train the network on the full dataset to achieve better results.

References

[1] Hamilton, W., Ying, Z. and Leskovec, J. *Inductive representation learning on large graphs*. City, 2017.
[2] Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R. and Battaglia, P. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242* (2018).
[3] Fout, A., Byrd, J., Shariat, B. and Ben-Hur, A. *Protein interface prediction using graph convolutional networks*. City, 2017.
[4] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. and Sun, M. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).
[5] Yao, L., Mao, C. and Luo, Y. *Graph convolutional networks for text classification*. City, 2019.
[6] Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[7] Agrawal, S. O., Chenthamarakshan, V. E. and Dlas, E. W. B. *Method for an efficient electronic messaging system*. Google Patents, City, 2011.
[8] Wang, S. and Manning, C. D. *Baselines and bigrams: Simple, good sentiment and topic classification*. Association for Computational Linguistics, City, 2012.
[9] Pennington, J., Socher, R. and Manning, C. D. *Glove: Global vectors for word representation*. City, 2014.
[10] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. *Distributed representations of words and phrases and their compositionality*. City, 2013.
[11] Shen, D., Wang, G., Wang, W., Min, M. R., Su, Q., Zhang, Y., Li, C., Henao, R. and Carin, L. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. *arXiv preprint arXiv:1805.09843* (2018).
[12] Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).
[13] Le, Q. and Mikolov, T. *Distributed representations of sentences and documents*. City, 2014.
[14] Tang, J., Qu, M. and Mei, Q. *Pte: Predictive text embedding through large-scale heterogeneous text networks*. City, 2015.
[15] Zhang, X., Zhao, J. and LeCun, Y. *Character-level convolutional networks for text classification*. City, 2015.
[16] Conneau, A., Schwenk, H., Barrault, L. and Lecun, Y. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781* (2016).
[17] Tai, K. S., Socher, R. and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075* (2015).
[18] Luo, Y. Recurrent neural networks for classifying relations in clinical notes. *Journal of biomedical informatics*, 72 (2017), 85-95.
[19] Cai, H., Zheng, V. W. and Chang, K. C.-C. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30, 9 (2018), 1616-1637.
[20] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. and Monfardini, G. The graph

neural network model. *IEEE Transactions on Neural Networks*, 20, 1 (2008), 61-80.
[21] Defferrard, M., Bresson, X. and Vandergheynst, P. *Convolutional neural networks on graphs with fast localized spectral filtering*. City, 2016.
[22] Huang, L., Ma, D., Li, S., Zhang, X. and WANG, H. Text Level Graph Neural Network for Text Classification. *arXiv preprint arXiv:1910.02356* (2019).
[23] Battaglia, P., Pascanu, R., Lai, M. and Rezende, D. J. *Interaction networks for learning about objects, relations and physics*. City, 2016.
[24] Bastings, J., Titov, I., Aziz, W., Marcheggiani, D. and Sima'an, K. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675* (2017).
[25] Zhang, Z., Cui, P. and Zhu, W. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202* (2018).
[26] Jepsen, T. S. *How to do Deep Learning on Graphs with Graph Convolutional Networks*. City, 2018.
[27] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. and Mei, Q. *Line: Large-scale information network embedding*. City, 2015.
[28] Acropolitan, T. *Sentence Length Has Declined 75% in the Past 500 Years*. Medium, City, 2019.