Understanding Graph Convolutional Networks for Text Classification

Soyeon Caren Han,[1*†] Zihan Yuan,[2*] Kunze Wang,[2] Siqu Long,[2] Josiah Poon[1]

The University of Sydney, NSW, Australia
[1] {caren.han, josiah.poon}@sydney.edu.au
[2] {zyua5587, kwan4418, slon6753}@uni.sydney.edu.au

Abstract

Graph Convolutional Networks (GCN) have been effective at tasks that have rich relational structure and can preserve global structure information of a dataset in graph embeddings. Recently, many researchers focused on examining whether GCNs could handle different Natural Language Processing tasks, especially text classification. While applying GCNs to text classification is well-studied, its graph construction techniques, such as node/edge selection and their feature representation, and the optimal GCN learning mechanism in text classification is rather neglected. In this paper, we conduct a comprehensive analysis of the role of node and edge embeddings in a graph and its GCN learning techniques in text classification. Our analysis is the first of its kind and provides useful insights into the importance of each graph node/edge construction mechanism when applied at the GCN training/testing in different text classification benchmarks, as well as under its semi-supervised environment.

1 Introduction

After the rise of deep learning, text classification models mostly applied sequence-based learning models, CNN or RNN, which mainly captures text features from local consecutive word sequences, but may easily ignore global word co-occurrence in a corpus which carries non-consecutive and long-distance semantics. Graph-based learning models are directly dealing with complex structured data and prioritising global features exploitation. Several recent research efforts on investigating Graph Convolutional Networks (GCN) on NLP tasks include application to text classification (Huang et al. 2019; Yao, Mao, and Luo 2019; Liu et al. 2020). This is largely because they can analyse rich relational structure and preserve the global structure in graph embeddings. The GCN-based text learning should include two main phases: 1) graph construction from free text and 2) graph-based learning with the constructed graph. A straightforward manner of graph construction is to represent relationships between words/entities in the free text. Yao, Mao, and Luo (2019) proposed a text graph-based neural network, named TextGCN, the first corpus-level graph-based transductive text classification model. In TextGCN, a single large textual graph is firstly constructed based on the entire corpus with words and documents as nodes, and co-occurrence relationship between words and documents as edges. Then, a GCN is employed to learn on the constructed text graph. More recent studies applied extra contextual information, such as topic model (Huang et al. 2019), syntactic and semantic information (Liu et al. 2020), pre-trained language model (Zhang et al. 2020b) or utilised different information propagation mechanisms (Wu et al. 2019; Zhu and Koniusz 2021). We noticed that most studies only focus on either hyperparameter testing or performance comparison with other state-of-the-art text classification baselines. It is still unclear what factors in textual graph construction or graph learning are having an impact on the GCN-based text classification. Thus, finding the optimal textual graph construction or learning mechanism itself, the two main phases for GCN-based text learning, remains a *black box* to us. Such observations and limitations lead to several important questions. First, the performance of GCN-based text learning methods is highly affected by the quality of the input graph, which covers the global structure and relations of an entire corpus or a whole dataset. Our first question is *'What is the best textual graph construction approach to understand and represent the whole textual corpus?'*. In a text corpus, we have two main components, documents and words, which can be used as nodes. Then, what feature/embedding is better to represent the node feature for the textual graph? And what edge (relation) information should be used between nodes? Secondly, we use GCN learning in order to capture information from the neighbours of each word or document node. Our second question would be *'How much larger of a neighborhood's information should be integrated in order to produce the better text classification performance?'* In other words, how many GCN layers should be stacked for the best performance on different text classification tasks?

In this work, we focus on answering the above questions. We report the effect of graph construction mechanisms by analysing the variants of defining main components in a graph, including nodes and edges. Then, we present a study to figure out the effect of GCN learning layers by integrating a variant range of neighbourhoods' information. We conduct our evaluation on both a full corpus environment and a semi-

* Equal contribution

† Corresponding author (Caren.Han@sydney.edu.au)

supervised limited environment. The full corpus environment is exactly the same as the original training-testing split of five widely used benchmark corpora including 20NG, R8, R52, Ohsumed and MR, tested by different GCN-based text classification studies (Yao, Mao, and Luo 2019; Liu et al. 2020; Wu et al. 2019). The purpose of traditional GCN models (Kipf and Welling 2016) is to solve semi-supervised classification tasks, so we test on a semi-supervised environment with a very limited amount of labelled data. For this limited setup, we use the above five widely used text classification corpora, as well as four low resource language document classification benchmarks (incl. Chinese, Korean, African). Note that this paper aims to analyse the graph construction and learning mechanism of GCN-based text classification models when there are no extra resources. This is because high-quality resources are not always available, especially for the low resource language or specific domain that requires expertise.

In summary, the main contributions are as follows:

- We conduct a comprehensive analysis of the role of graph construction and learning in GCN-based text classification over five widely used benchmarks (full corpus environment) and nine benchmarks (limited training environment), including low resource languages
- We perform a comparative analysis of the accuracy performance of different node and edge constructions when GCN is applied for text classifications
- We evaluate the performance of GCN-based Text Classification with different variants of GCN layer stacks
- We make source code publicly available to encourage reproduction of the results[1]

2 Related Work

2.1 Graph Neural Networks in NLP

Graph Neural Networks (GNN) have received increasing attention in the realm of semi-supervised learning (Kipf and Welling 2016; Li, Han, and Wu 2018). Bastings et al. (2017) took word representations produced based on syntactic dependency trees as graph nodes and applied them to GCN learning for machine translation. Tu et al. (2019) proposed a Heterogeneous Document-Entity graph and utilized a GCN to do reasoning over the constructed graph for multihop reading comprehension problems. Cao et al. (2019) designed a Multi-channel Graph Neural Network that learned the alignment-oriented knowledge graph embeddings for entity alignment. Xu et al. (2019) extracted node features from neighbourhoods and applied dual graph-state LSTM networks to summarize graph local features and extracted interaction features between pairwise graphs for entity interaction prediction. Zhang et al. (2020a) proposed automatic sentence graph learning and incorporated it with GCN for headline generation. Dowlagar and Mamidi (2021) combined the GCN graph modelling with multi-headed attention for code-mixed sentiment analysis.

Some recent studies applied graph neural networks for text classification by exploring different approaches of graph structure construction learned from the text data. Henaff, Bruna, and LeCun (2015) and Defferrard, Bresson, and Vandergheynst (2016) simply viewed a document as a graph node. Peng et al. (2018) proposed a sentence-based graph in order to solve a large-scale hierarchical text classification problem. Yao, Mao, and Luo (2019) constructed a large textual graph with word and document nodes and edge features represented as co-occurrence statistics, PMI/TF-IDF values. SGC (Wu et al. 2019) and S^{2}GC (Zhu and Koniusz 2021) constructed a graph as TextGCN, but proposed different information propagation approaches. Vashishth et al. (2019) incorporated syntactic/semantic information for word embedding training via GCNs. Liu et al. (2020) proposed multiple aspect graphs constructed from external resources in terms of semantic, syntactic and sequential contextual information, which are jointly trained. When faced with low resource text classification problems, these approaches either do not fully explore the latent structure within the corpus data itself as they consider only the connections between documents, or are not applicable due to lack of external resource. Most prior studies focused on either hyperparameter testing or performance comparison with other state-of-the-art text classification baselines. Distinct from these works, we examine the important factors in two main phases of GCN-based text learning, textual graph construction and graph learning because they have a critical impact on the GCN-based text classification performance.

3 GCN-based Text Classification

We consider the task of GCN-based text classification with only single-label classification. Figure 1 visualises the architecture of typical graph-based text classification models based on the given corpus with no extra resources. There are various types of GCN-based Text Classification mechanisms introduced in the field. Two commonly used mechanisms are the corpus-level (Yao, Mao, and Luo 2019) and document/sentence-level GCN text classification models(Huang et al. 2019). In this work, we will focus on the former models since it captures the global structure information of a corpus/entire dataset, whereas the latter models consider only local-level information (from a single sentence/document). With the former approaches, we can analyse the rich relational structure and preserve the global structure of a graph. In this section, we give a brief overview of GCN and TextGCN, the first corpus-level GCN-based text classification model.

3.1 Graph Convolutional Networks

GCN (Kipf and Welling 2016) is a multi-layer neural network generalized from Convolutional Neural Networks, which directly operates on the graph-structured data and learns representation vectors of nodes based on properties of their neighbourhoods. Formally, a GCN graph G is constructed as $G = (V, E, A)$, where V $(|V| = N)$ and E represents the set of graph nodes and edges respectively while $A \in \mathcal{R}^{N\times N}$ is the graph adjacency matrix. Based on the constructed graph G, the GCN learning takes in the input matrix $H_0 \in \mathcal{R}^{N\times d_0}$ containing initial d_0-dimensional features of the N nodes in V and then conducts the propagation

[1] https://github.com/usydnlp/TextGCN_analysis

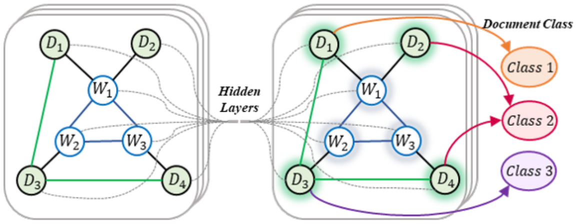

Figure 1: Text Classification with GCN. A single large graph is constructed; words and documents appear as nodes, and co-occurrence between words and documents. Assume that we have only three classes for the document classification

through layers based on the rule in equation (1), which formulates the propagation operation from layer l to the subsequent layer $(l+1)$.

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\hat{A}H^{(l)}W^{(l)}) \tag{1}$$

Here, $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is the normalized symmetric adjacency matrix $\tilde{A} = A + I$ (I refers to an identity matrix for including self-connection of nodes); $\tilde{D}$ is the diagonal node degree matrix, i.e. $\tilde{D}(i,i) = \sum_j \tilde{A}(i,j)$; $W^{(l)} \in \mathcal{R}^{d_l \times d_{l+1}}$ denotes the layer-specific trainable weight matrix for the lth layer (d_l/d_{l+1} is the feature dimension of layer $l/l+1$); σ is a non-linear activation function such as ReLU or softmax, which can be different for a specific layer. The main focus of our analysis lies in exploring the role of node and edge that constructs the graph G as well as the variants of GCN learning techniques when applying to text classification.

3.2 TextGCN

Inspired by a GCN, TextGCN (Yao, Mao, and Luo 2019) constructs an entire corpus based graph, which uses all the words and documents in the corpus as graph nodes and sets the word-word and word-document edges to preserve the global word co-occurrence and word-document relations in the graph structure. Then, it would be modelled by GCN learning. The edge between each word pair is represented by the point-wise mutual information (PMI) value, normally used for measuring semantic similarity in the Term-Sentence-Matrix. The word-document edge is calculated based on the Term Frequency-Inverse Document Frequency(TF-IDF) weight of the word in the document. The constructed graph is fed into a two layer GCN as in equation (1) where the second layer node embeddings for both word and document have the same size as the label set and are passed into a softmax classifier for the output. The cross-entropy loss is then calculated over all labelled documents for training and optimization. Especially, they simply set the initial input word/document node features as one-hot vectors.

4 GCN Analysis on Text Classification

4.1 Graph Node Construction Analysis

Following TextGCN, we set all the words and documents in the corpus as our node set for graph G, i.e. the number of nodes $|V| = N = D + M$ equals to the sum of the number of documents (corpus size D) and the number of unique words in the corpus (vocab size M). With those word and document nodes, we explore the role of initial node representation in GCN-based text classification models with two commonly used input embedding types in NLP: 1) one-hot and 2) BERT embeddings. 1) one-hot embedding is the most widely used categorical input encoding approach in traditional NLP. 2) Bert embedding is one of the most popular contextual word embeddings so we generate word/document node representation. It includes each individual word embedding and a [CLS] token for representing sentence/document-level embedding.

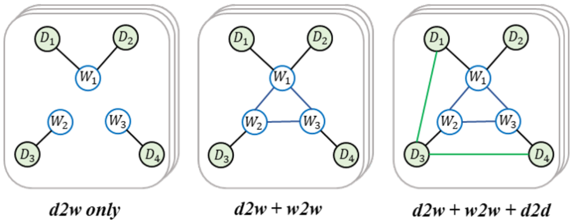

Figure 2: Different Variants of Edge Construction in an entire corpus-based graph. Assume that we have four documents and three words.

$$B_{D_d} = \text{BERT}(D_d) \tag{2}$$

$$H_{node_{D_d}} = B_{D_d}^{[CLS]} \tag{3}$$

$$H_{node_{W_m}} = \min_{d \in D_{W_m}} (B_{D_d}^{W_m}) \tag{4}$$

For one-hot embedding, we simply set the feature matrix H_0 as an identity matrix $I \in \mathcal{R}^{N \times N}$ for the one-hot vector input. For BERT embedding, the calculation for word and document nodes are illustrated in Equations (2)-(4). Concretely, we feed each document D_d into the BERT model as in equation (2), resulting in the sequence representation B_{D_d}. For example, a document D_d such as “John feels happy” will result in the B_{D_d} as “$B_{D_d}^{[CLS]}$ $B_{D_d}^{John}$ $B_{D_d}^{feels}$ $B_{D_d}^{happy}$ $B_{D_d}^{[SEP]}$”. We directly take the [CLS] representation $B_{D_d}^{[CLS]}$ as the node embedding for D_d, as in equation (3). Then for a word W_m, we collect all the documents containing this word, denoted as D_{W_m}, and apply min pooling over all the BERT representation $B_{D_d}^{W_m}$ for this word from documents in D_{W_m}, as is illustrated in equation (4). The essential difference between these two types of embedding is that one-hot embedding incorporates no external knowledge or semantic information but purely indicates which word or document in the corpus the node is representing. Comparatively, BERT embedding is the output representation from the BERT model pretrained on large text corpus, which can impart the common sense semantic information to the represented nodes and differentiate each document node based on the document-specific word context. The detailed analysis of these two embedding types is provided in Section 6.1.

4.2 Graph Edge Construction Analysis

For edge construction, we intend to fully analyse all possible co-occurring relations between every two types of nodes, which no studies have yet explored. We utilise document-document edges in addition to word-word and word-document edges as can be seen in Figure 2. The construction details are provided in equation (5). Refer to the TextGCN(Yao, Mao, and Luo 2019), we use the PMI value between word pairs as a word-word edge feature, and utilise Term Frequency-Inverse Document Frequency(TF-IDF) weight to weight word-document edges.

$$\text{PMI}(i,j) = \max(\log\frac{p(i,j)}{p(i)p(j)}, 0) \tag{5}$$

$$p(i,j) = \frac{\#W(i,j)}{\#W} \tag{6}$$

$$p(i) = \frac{\#W(i)}{\#W} \tag{7}$$

4.3 Graph Learning

As mentioned before, GCN only captures information about the immediate neighbours with one layer of graph convolution. When multiple GCN hidden layers are stacked, information about larger neighbourhoods is integrated.

We adopt this GCN propagation rule in equation (1) for modelling the constructed graph. Specifically, we explore a various number of hidden layers $L \in \{1, 2, 3, 4, 5\}$ to find the optimal range of neighbours to be integrated. For the first l ($l \in \{1, .., L-1\}$) layers, we apply ReLU as activation functions as in equation (10). Here $H^{(1)}$ is the initial feature matrix of nodes using one-hot or BERT as described in Section 4.1. We set the last layer output for both word and document node embeddings to have the same size as the label set and apply a softmax classifier over the output as in equation (11). The cross-entropy loss is then calculated over all the labelled documents as in equation (12), in which $\mathcal{Y}_\mathcal{D}$ is the set of document indices with labels available and F is the output feature dimension (equals to the number of classes). Y denotes the label indicator matrix. We provide the analysis for different numbers of hidden layers in Section 6.3.

$$H^{(l+1)} = \text{ReLU}(\hat{A}H^{(l)}W^{(l)}) \tag{8}$$

$$Z = \text{softmax}(\hat{A}H^{(L)}W^{((L))}) \tag{9}$$

$$L = -\sum_{d\in\mathcal{Y}_\mathcal{D}}\sum_{f=1}^{F} Y_{df}\ln Z_{df} \tag{10}$$

5 Experiment setup

5.1 Datasets

We conduct the comprehensive analysis on both a full corpus environment and a semi-supervised limited environment. The full environment is exactly the same as the original training-testing split of five widely used benchmark corpora including 20NG, R8, R52, Ohsumed and MR, followed by different GCN-based text classification studies (Yao, Mao, and Luo 2019; Liu et al. 2020; Wu et al. 2019). The limited environment aims to cover semi-supervised text classification with a very limited amount of labelled data. We randomly sample 1% as a training set and use the remaining 99% for testing on nine benchmarks, including the above five benchmarks and additional four low-resource language (incl. Chinese, Korean, African) document classification datasets[2]. 1)The **20NG** contains 18,846 news documents in total, which are evenly categorized into 20 classes. 2)**R8** and 3)**R52** (all-terms version) are subsets of the Reuters 21578 dataset. R8 has 8 topic categories with 7,674 documents, and R52 is based on 52 categories with 9,100 documents. 4)**Ohsumed** is collected from the MEDLINE, which is a bibliographic database of biomedical information. Only single-label classification task (7,400 documents) is selected. 5)**MR** is a binary sentiment (positive and negative) classification dataset, which includes 10,622 short movie review comments.

The following list shows four additional document classification benchmarks in low-resource languages, including Chinese, Korean, and African. 6)**Waimai** is a binary sentiment analysis dataset collected from a Chinese online food ordering platform, which provides 11,987 Chinese comments about the food delivery service. 7)**ChSenti** contains 7,766 Chinese documents of hotel service comment with binary class. 8)**KrHate** provides 2,000 binary hate speech comments collected from the Korean radical Anti-male online community, named Womad. 9)**Xhosa** is a Xhosa dataset from the NCHLT Text Corpora collected by South African Department of Arts and Culture & Centre for Text Technology, which contains 4,000 documents of 11 categories.

5.2 Implementation Details

Graph Node Setup We use one-hot and BERT embedding for the analysis. The dimension of one-hot embedding corresponds to the number of nodes. For BERT embeddings, “bert-base-uncased”(for English-based) and “bert-base-multilingual-uncased”(for non-English-based) developed by Hugging Face (Wolf et al. 2019) is used with the input dimension of 768. **Graph Edge Setup** In order to construct the edge, the window size is set to 20 for PMI calculation (word-word edges). The threshold 0.2 is applied when calculating Jaccard similarity measure for doc-doc edges. **Graph Learning Setup** Each hidden layer's dimension is defined as 200, and the dimension of output layer is the number of classes. The training hyperparameters include: 0.02 as the learning rate; 0.5 as the dropout rate; 0 as the L_2 loss weight; 200 as the maximum number of epochs with early stopping of 10 epochs. Adam (Kingma and Ba 2015) is used to train the model.

[2] Dataset Links: **1)**http://qwone.com/~jason/20Newsgroups/ **2)3)**https://www.cs.umb.edu/~smimarog/textmining/datasets/ **4)**http://disi.unitn.it/moschitti/corpora.htm **5)**http://www.cs.cornell.edu/people/pabo/movie-review-data/ **6)**https://github.com/SophonPlus/ChineseNlpCorpus/ **7)**https://github.com/SophonPlus/ChineseNlpCorpus/blob/master/datasets/ChnSentiCorp_htl_all/intro.ipynb **8)**https://www.kaggle.com/captainnemo9292/korean-extremist-website-womad-hate-speech-data **9)**https://github.com/praekelt/feersum-lid-shared-task

	20NG	**R8**	**R52**	**Ohsumed**	**MR**
Node feature					
onehot	**0.8607**	**0.9692**	**0.9345**	**0.6824**	0.7641
BERT	0.7206	0.9510	0.8440	0.4618	**0.7821**
Edge feature					
d2w only	0.8475	0.9493	0.9169	0.6667	0.7462
+w2w	**0.8617**	**0.9693**	0.9333	**0.6841**	0.7600
+w2w+d2d	0.8607	0.9692	**0.9345**	0.6824	**0.7641**

Table 1: Test accuracy by different node and edge construction variants on the full environment

6 Discussion and Analysis

6.1 Effect of Node Embedding

The upper block in Table 1 shows the test accuracy by using either one-hot or BERT embeddings as initial node features on the five benchmark datasets under full environment. First, it can be seen that only MR achieves better result with BERT embedding than one-hot embedding, which might be attributed to the fact that MR as a sentiment analysis task benefits better from the general semantics learned from a large external text. In addition, for the other four datasets with higher accuracy via one-hot embedding, we found that datasets with larger size of classification categories, i.e. 20NG, R52, Ohsumed, tend to generate a bigger performance gap between the embedding types compared to R8. We suppose that a smaller size of classification category may exert itself better on pretrained embeddings since it does not have sufficient information to train the global information. We also provide the comparative results under limited environment on all the 9 datasets in Table 2 (Appendix). It is reasonable to see an overall performance drop compared to the full setting. For the first five benchmark datasets, MR still prefers BERT embedding while R8 changes the preference to BERT from one-hot embedding. Both these have a relatively small number of classification categories. This further supports the claim that small-category datasets benefit more from BERT than large-category counterparts. When it comes to the low resource datasets (Waimai, ChSenti, KrHate, Xhosa), it can be seen that all of them produce better accuracy with one-hot embedding than BERT embedding, which might be due to the quality of pre-training on low resource language corpora. Note that we used the edge set (d2w+w2w+d2d) for the node construction testing since it produces the overall highest performance in both full and limited environment.

6.2 Effect of Edge Construction

We also analyse the usage of different edge features for both full and limited environment in the bottom half of Table 1 and 2 respectively. More specifically, three types of edge features are evaluated: **(1) d2w only**, utilises only the word-doc edges in the constructed graph; **(2) +w2w**, uses both word-doc and word-word edges; **(3) +w2w+d2d**, apply all the three types of edges including doc-doc edges. Similar patterns can be found in both settings. Overall, a d2w-only graph always results in the lowest performance, implying insufficient global structural information conveyed by only the word-document co-occurrence. With the w2w edge, the performance increased in all datasets and the amount of increase mostly varies around 0.01 to 0.04 in the two settings. In addition, d2w+w2w+d2d using all the three types of edges, further rises the accuracy for R52 and MR under full environment and for most of the datasets under the limited environment. This highlights the benefit of using full co-occurring relationships (with all types of edges) in a entire graph. The similar trend can be further observed with the persistent spatial gap between the lowest blue line (d2w only) and the other two lines in Figure 3 (Appendix). It illustrates the corresponding accuracy for the three edge features on the five benchmark datasets when increasing the training proportion from extremely few labelled setting (1%) to 99%.

6.3 Effect of GCN Learning

The main aim of GCN learning is to capture information about immediate neighbours with a layer of convolution. When multiple GCN layers are stacked, information from much larger neighbourhoods is extracted and integrated. In Figure 4 (Appendix), we conducted the text classification evaluation to find the optimal range of neighbours' information about each node. We stacked 1 to 5 GCN layers on different text classification in the full environment. It can be seen that the highest performance is achieved by using 2 GCN layers for all five datasets and the performance drops down as the layer decreases or increases. This indicates capturing 2 levels of neighbourhood nodes is the best and increasing the level of neighbourhoods will gradually lead to indifferentiable node representation. 20NG and MR have a similar overall trend and perform more consistently in the three evaluation metrics. Comparatively, the other three datasets are observed to have much lower overall Macro F1 than Accuracy/Weighted F1. Those trends can also be found when switching from the full to the limited environment in Figure 5 (Appendix). Even though low resource language datasets have extremely few labelled data, it is still 2 layers that performs the best overall, shown in Figure 5 (Appendix). When layer number increases, the performance does not always decrease sharply as in previous cases. Specifically, performance of the two Chinese datasets ChSenti and Waimai goes up again at 4 layers after the decrease at 3 layers. Xhosa only achieved a rather stable performance degradation when increasing from 2 to 5 layers. It can be seen that different languages may preserve different patterns on the metrics with the change of GCN layers.

7 Conclusions

We focused on understanding the underlying factors that may influence GCN-based text classification, and proposed to examine graph construction and graph learning mechanisms. We systematically examined the role of node and edge in a corpus-level textual graph, and found the optimal range of neighbours' information by testing the different number of GCN layer stacks. The empirical results of experiments on various real datasets in both full environment and limited training environment supported our analysis.

A Appendix

Table 2, Figure 3,4, and 5 can be found in the next page. Once the paper is accepted, we will add those figures and tables in the main content with 6 pages (one additional page).

References

Bastings, J.; Titov, I.; Aziz, W.; Marcheggiani, D.; and Sima'an, K. 2017. Graph Convolutional Encoders for Syntax-aware Neural Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1957–1967.

Cao, Y.; Liu, Z.; Li, C.; Li, J.; and Chua, T.-S. 2019. Multi-Channel Graph Neural Network for Entity Alignment. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 1452–1461.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*.

Dowlagar, S.; and Mamidi, R. 2021. Graph convolutional networks with multi-headed attention for code-mixed sentiment analysis. In *Proceedings of the First Workshop on Speech and Language Technologies for Dravidian Languages*, 65–72.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

Huang, L.; Ma, D.; Li, S.; Zhang, X.; and Houfeng, W. 2019. Text Level Graph Neural Network for Text Classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3435–3441.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Liu, X.; You, X.; Zhang, X.; Wu, J.; and Lv, P. 2020. Tensor graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 8409–8416.

Peng, H.; Li, J.; He, Y.; Liu, Y.; Bao, M.; Wang, L.; Song, Y.; and Yang, Q. 2018. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 world wide web conference*, 1063–1072.

Tu, M.; Wang, G.; Huang, J.; Tang, Y.; He, X.; and Zhou, B. 2019. Multi-hop Reading Comprehension across Multiple Documents by Reasoning over Heterogeneous Graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2704–2713.

Vashishth, S.; Bhandari, M.; Yadav, P.; Rai, P.; Bhattacharyya, C.; and Talukdar, P. 2019. Incorporating Syntactic and Semantic Information in Word Embeddings using Graph Convolutional Networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3308–3318.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv*, abs/1910.03771.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*, 6861–6871. PMLR.

Xu, N.; Wang, P.; Chen, L.; Tao, J.; and Zhao, J. 2019. MR-GNN: Multi-Resolution and Dual Graph Neural Network for Predicting Structured Entity Interactions. In *IJCAI*.

Yao, L.; Mao, C.; and Luo, Y. 2019. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7370–7377.

Zhang, R.; Guo, J.; Fan, Y.; Lan, Y.; and Cheng, X. 2020a. Structure Learning for Headline Generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9555–9562.

Zhang, Y.; Yu, X.; Cui, Z.; Wu, S.; Wen, Z.; and Wang, L. 2020b. 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*, 334–339.

Zhu, H.; and Koniusz, P. 2021. Simple spectral graph convolution. In *International Conference on Learning Representations*.

	20NG	**R8**	**R52**	**Ohsumed**	**MR**	**Waimai**	**ChSenti**	**KrHate**	**Xhosa**
					Node feature				
onehot	**0.6937**	0.8973	**0.7990**	**0.4092**	0.6178	**0.8301**	**0.7548**	**0.9048**	**0.9952**
BERT	0.6585	**0.9147**	0.7962	0.3893	**0.7255**	0.8059	0.6970	0.8256	0.7878
					Edge feature				
d2w only	0.6239	0.8698	0.7776	0.3906	0.5995	0.8149	0.5297	0.7445	0.9874
+w2w	0.6680	0.8949	0.7978	**0.4099**	0.6158	0.8283	0.7410	0.7609	**0.9953**
+w2w+d2d	**0.6937**	**0.8973**	**0.7990**	0.4092	**0.6178**	**0.8301**	**0.7548**	**0.9048**	0.9952

Table 2: Test accuracy by different node and edge construction variants on the limited environment

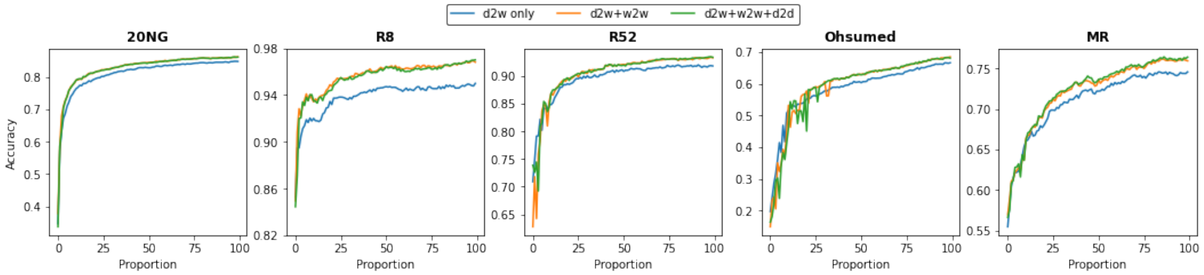

Figure 3: Test accuracy by varying training data proportions (from 1% to 99%)

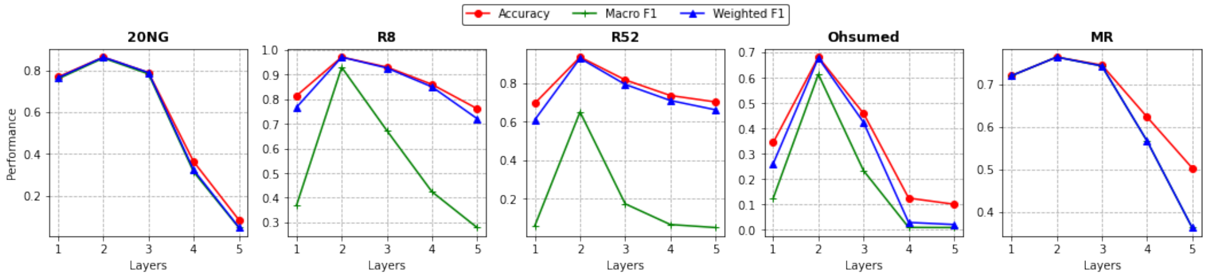

Figure 4: Test performance by varying GCN hidden layer stacks on the full environment

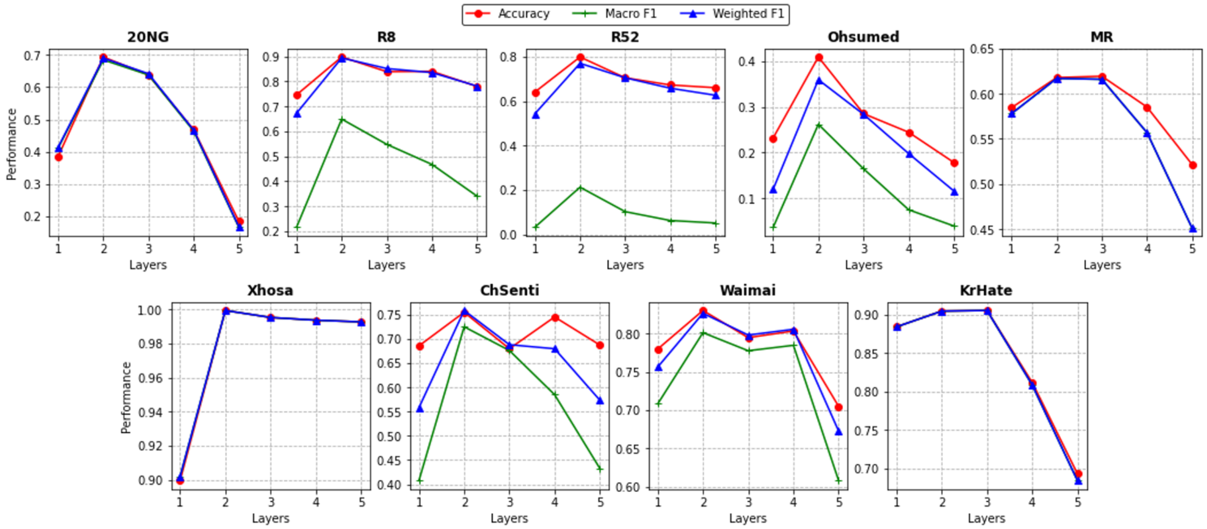

Figure 5: Test performance by varying GCN hidden layer stacks on the limited environment