A Picture Is Worth a Thousand Words: Rethinking Graph Neural Networks for Text Classification

Yiping Huang*
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

Abstract

Text classification has been well studied. Recent studies have used graph neural networks for the task by building a heterogeneous graph of the whole corpus. However, early models were mostly limited to context of transductive learning and needed additional complex optimization techniques for mini-batch training. We present a new approach to overcome these issues by treating each document as a separate graph and use graph level pooling for text classification. We show this is a viable alternative for network representation of text by testing on our document graph convolutional networks (DocGCN) model, which have achieved or matched the state-of-the-art results across the well-established benchmark datasets for text classification. By using this approach, our model also works well with small-sized and mislabeled data, which makes it readily applicable in settings where it is difficult to acquire data and correct labels.

1. Introduction

Since its early day, machine learning method has been applied to text classification in many different ways, ranging from spam mail detection (Sahami et al. 1998) to sentiment analysis in tweets (Badjatiya et al. 2017). Recent natural language processing (NLP) discovery, such as recurrent neural networks (RNN) (Badjatiya et al. 2017), Convolutional Neural Networks (CNN) (Kim 2014) has allowed for even more possibilities. These models take text as sequential input and utilize deep learning to capture message in the local neighborhood. However, these models can bias towards local information and ignore the global information present in the document.

On the other hand, graph-based learning has made remarkable progress recently and a variety of models have come up which operate on the graph-structured data. For example, Kipf and Welling 2016 proposed graph convolutional networks (GCN) as a localized first-order approximation of spectral graph convolutions. Veličković et al. 2017) also proposed graph attention networks (GAT) which leverages masked self-attentional layers as an alternative to the spectral-based graph convolution.

Ever since, the graph based learning models have been successfully applied to various domains and for various tasks. For example, Kipf and Welling 2016 used GCN model on the citation network where documents are connected via citations to predict the document labels, which outperformed related methods by a significant margin. Ying et al. 2018 developed an improved GCN model and successfully deployed it to recommend new pins to the user at Pinterest with a product-user bipartite network. In the field of computational pharmacology, Zitnik, Agrawal, and Leskovec 2018 developed Decagon, a general GCN model for multirelational link prediction to predict the side-effect of using multiple drugs with a heterogeneous graph of protein-protein, drug-protein and drug-drug interactions. Stokes et al. 2020 also used message-passing graph models to predict new antibiotics based on molecule structures.

Often, the graph structure these researches worked with already have a natural representation, and one can get this impression from the use of the word 'network' in their dataset names. For instance, in citation network, a node represents a document, and an edge indicates that one end node has cited another. In biology application, the molecular structure forms the underlying graph of study. Recently, some studies have extended the graph-based learning to other non-traditional network domains such as text classification (Yao, Mao, and Luo 2019), which has shown remarkable result over traditional models. However, several challenges still exist in

*This research is carried out in KE@Work program collaborated with CBS, Heerlen.

applying graph-based learning to the new domains. Often, early models were applied in a transductive learning setting, requiring all the data to be present at once. Additionally, they did not utilize optimization techniques such as stochastic gradient descent (SGD) for alleviating computation burden and speeding up iterations.

In this study, we work on a new approach to integrate graph neural networks (GNN) with text classification, which addresses the limitations of the early models. Specifically, we attempt to treat each document as a separate graph and use graph level pooling to classify document. We test our approach on our document graph convolutional networks (DocGCN) model [1] and show it is a viable alternative for network representation of text. We also show robustness of our model by testing it with different parameter and with different learning environment. To summarize, the main contributions of this study are as follows:

- We present a new approach for text classification by treating each document as a separate graph, which makes efficient inductive learning easily possible.
- The model using this approach has achieved or matched results from state-of-the-art text classification method on the widely used benchmark datasets.
- Our approach is very robust with both small-sized training set and with data of high mislabeling rate by utilizing the relational inductive bias of the graph structure.

2. Related Work

2.1. GCN

Graph Convolutional Network (GCN), as was proposed by Kipf and Welling 2016, conducts convolution on graph structure. The original 2-layer GCN model computes

$$Z = f(X,A) = \hat{A}ReLU(\hat{A}XW^{(0)})W^{(1)} \tag{1}$$

, where X is the node feature matrix, $W^{(0)}$ and $W^{(1)}$ are model parameters for the first and second layers, $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, $\tilde{A} = A + I_N$ and $\tilde{D}$ is the degree matrix of $\tilde{A}$. Thus $\hat{A}$ is a normalized adjacency matrix with added self-loop.

[1] The source code can be found at `https://github.com/hypster/DocGCN`

2.2. TextGCN

TextGCN by Yao, Mao, and Luo 2019, applies GCN model for text classification. The model first constructs a heterogeneous graph of the whole corpus, which consists of two node types: the document node and the word token node. There are two edge types in the graph: the document-word edge and the word-word edge. The document-word edge uses term frequency-inverse document frequency (TF-IDF) for the edge weight, and the word-word edge uses point-wise mutual information (PMI) as edge weight. PMI between $word_i$ and $word_j$ is computed as

$$PMI(i,j) = \log \frac{p(i,j)}{p(i)p(j)} \tag{2}$$

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

$$p(i) = \frac{\#W(i)}{\#W} \tag{4}$$

where $\#W(i)$ is the count of sliding windows that contain $word_i$, $\#W(i,j)$ is the count of sliding windows that contain both $word_i$ and $word_j$, and $\#W$ is the total count of sliding windows. The size of the sliding window is a hyperparameter for which a default value of 20 was chosen. The PMI weight can vary from $-\infty$ to $min(-\log p(i), -\log p(j))$. A negative value between two word tokens indicates they are not likely to appear together. A value of 0 indicates independence, and a positive value indicates semantic correlation in the corpus. The authors only constructed edges with positive weight and exclude those with zero or negative weight. Additionally, there is no direct linkage between document nodes in the TextGCN model. However, messages can be passed between document nodes through stacking multiple GCN layers.

State-of-the-art performance was reported for TextGCN on several public datasets (Yao, Mao, and Luo 2019). Some GCN model variants have followed this approach in the text classification task and have achieved good scores as well (Wu et al. 2019; Zhu and Koniusz 2021). However, there are two issues with this approach. Firstly, TextGCN is only applicable for transductive learning, as documents are represented as nodes, which means during the message propagation, nodes from the validation set and testing set also send messages to the training nodes, although the final loss is only computed against target labels from the training set. The joint message propagation across different groups of nodes might be helpful in node classification. However, this assumes that all nodes contain the correct information.

If the training nodes are often mislabeled, then these training nodes will also send the confused messages to the testing node after each backpropagation of the wrong loss. Thus, in a iterative fashion, the signal from the correct message can get obscured instead of being augmented. More importantly, for transductive learning, when predictions must be made for new node, the model must be retrained before being put to use, which limits its usage in many practical settings where new data are constantly getting produced.

Secondly, the original GCN used full-batch training in node classification as mini-batch training on graph is much harder than on other grid-like structures like text and image. By its design, TextGCN treats every document as a single node and performs node level classification. Therefore, a large graph of the whole corpus needs to be stored in the memory of the GPU in order to be passed as input to the model. As GPU needs to crunch the data in parallel, the choice for the size of the hidden channels can be compromised, which might lead to underfitting for high dimensional output or other more complex training scenarios. Compared with mini-batch training, model parameters take also longer time to converge as they are updated only once per epoch.

3. Method

3.1. Document Level Graph Neural Networks

To address these issues, we propose our novel approach here: instead of viewing the whole corpus as an entire heterogeneous graph, we treat each individual document as an independent subgraph constructed on the document level semantic relations. Thus we perform text classification on graph level instead of on node level as was used in previous approach.

Based on this approach, we give here a simple model, which we call document graph convolutional networks (DocGCN). We use PMI computed on the single document to construct the document graph and GCN as our model for training. We give the outline in Figure 1. The detailed steps are as follows:

Tokenization. We use the same preprocessing step as was in Kim 2014, which has been adopted in many other research papers on text classification.

Constructing graph. For document level graph construction, we similarly calculate the PMI value using equation 2, 3 and 4 using a default window size of 5. However, instead of using PMI from the global corpus, we compute PMI for each document separately. Thus we are biased towards the local semantic relations given by the document. For node features, we use the index of the word tokens which the node represents in the vocabulary. When preparing for the mini-batch, we then transform the indices into corresponding one hot features for a good memory usage.

Model architecture. DocGCN incorporates GCN model in its architecture. For training stability, we also add batch normalization after each GCN layer, and after the relu activation function we also add dropout layer and set the default dropout rate to 0.5. In its last layer before the output layer, a global pooling layer is added to extract graph level information across the node dimension. By default we use the mean pooling in our model. We use cross entropy loss for the loss function, as is defined by

$$\mathscr{L} = -\sum_{d \in Y_D, c \in C} Y_{dc} \log SoftMax(Z_{dc}) \tag{5}$$

where D is the batch size, C is the output dimension, and softmax for Z_{dc} is defined as:

$$softmax(Z_{dc}) = \frac{e^{Z_{dc}}}{\sum_{k=1}^{C} e^{Z_{dk}}} \tag{6}$$

4. Experiment

In the experiment we want to mainly determine the effects of those factors on the model performance: the number of layers in the model, the size of the hidden channels, the size of the window, the size of the training data, and the percentage of mislabeled data.

Hardware. We used cloud instance for the model training. The hardware specification is as follows: GPU is GeForce RTX 3090 24 GB, CPU is AMD EPYC 7302, 512G memory, 64 kernel and hard disk is 1.6T SSD.

Dataset. We used the five widely used benchmark datasets in our experiment, which are 20 news group dataset (20*NG*) (Lang 1995), subset of Reuter news consisting of eight categories (*R*8), subset of Reuter news consisting of 52 categories (*R*52) (Debole and Sebastiani 2005), *Ohsumed* medical papers dataset (Joachims 1998), and one-sentence movie review dataset (*MR*) (Pang and Lee 2005).

The 20*NG* dataset contains 18,846 documents, and 20 evenly distributed categories in total. The training set and test set are split by date, which contain 11,314 and 7,532 documents respectively.

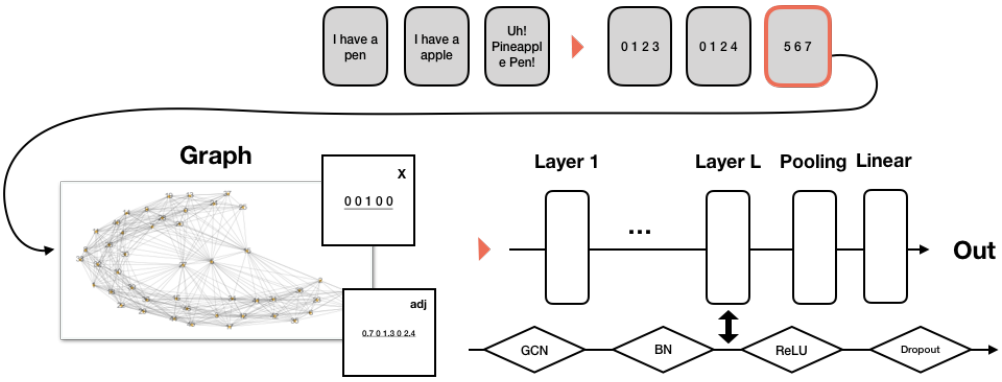


Figure 1: Schematic of DocGCN. The raw text is transformed to indices first before constructing the PMI adjacency matrix, then the model takes together the one hot features from nodes and the graph structure as the input. After GCN layers, messages from different nodes are pooled on graph level before being transformed by the last linear layer to the output with the same dimension as the number of unique class labels.

The *Ohsumed* dataset contains medical abstracts from the MeSH (Medical Subject Headings) categories of the year 1991. The dataset contains 13,929 abstracts, and 23 cardiovascular diseases categories in total. We further exclude documents that belong to multiple categories to make the result comparable across the datasets. In the end, there are 7,400 documents, of which 3,357 are in the training set and 4,043 are in the test set.

*R*8 dataset contains 7,674 documents, of which 5,485 are in the training set and 2,189 are in the test set. *R*52 dataset contains 9,100 documents, of which 6,532 are in the training set and 2,568 are in the test set. Unlike other datasets in the group, the class distribution of the *R*8 dataset is quite unbalanced. The largest class contains 3,923 documents and the smallest class contains only 51 documents, so model performance on this dataset is usually quite high.

Lastly, *MR* dataset contains 10,662 documents, half of which belong to the positive category and half of which belong to the negative category. We followed Tang, Qu, and Mei 2015 for the training/test split.

Overall, these datasets have different characteristics in their content, length and number of classes to predict, and provide a solid base for the evaluation of the model performance. We also provide the statistics of these datasets in Appendix A.

Overall performance. We chosen 2 as the default number of layer, and 64 as the default hidden channel size, and 5 as the default window size. We present the test accuracy of our model on the five benchmark datasets together with these from other baseline models in Table 1. As is shown, our model has achieved or matched the state-of-the-art results across all datasets. Specifically, our model scored the best accuracy on 20*NG* and *R*8, with 86.50% and 97.12% respectively.

Effect of number of layers. Table 2 shows the test accuracies of DocGCN on 20*NG* and *Ohsumed* when we varied the number of layers (2, 3 and 5). For both datasets, models with two layers gave the best accuracy. This observation is consistent with findings from other works (Kipf and Welling 2016; Yao, Mao, and Luo 2019) that GCN tends to work better with a shallow network structure. The underlying issue for this is called oversmoothing, where nodes tend to get similar representations from messages of their exponentially increasing neighborhood members when we increase the depth of the model. A full theoretical account of the limitations of different GNN variants including GCN can be found in Xu et al. 2018.

Effect of hidden channel size. Table 3 shows test accuracies of our models on 20*NG*, *Ohsumed* and *R*8 with different hidden channel size (32, 64, 128 and 256). We found the choice of hidden channel size did not affect the model performance much across the datasets with the exception in *Ohsumed* where we saw a 2.65% increase from size 32 to size 256. Meanwhile, we found that a good choice still depends on the specific dataset: for 20*NG*, setting it to 64 was more than enough, for *Ohsumed*, a larger value like 256 worked better, and for *R*8, a middle value like 128 produced the best result.

Effect of window size. Table 4 shows the test accuracies of our models on 20*NG* and *Ohsumed* with different window size (2, 5, 10 and 20). For 20*NG*, the best test accuracy was attained with size 5, and for *Ohsumed*, the best one was found with size 2. We observed that

Model	20NG	R8	R52	Ohsumed	MR
TF-IDF + LR	0.8319 ± 0.0000	0.9374 ± 0.0000	0.8695 ± 0.0000	0.5466 ± 0.0000	0.7459 ± 0.0000
CNN-non-static	0.8215 ± 0.0052	0.9571 ± 0.0052	0.8759 ± 0.0048	0.5844 ± 0.0106	**0.7775 ± 0.0072**
Bi-LSTM	0.7318 ± 0.0185	0.9631 ± 0.0033	0.9054 ± 0.0091	0.4927 ± 0.0107	0.7768 ± 0.0086
TextGCN	0.8634 ± 0.0009	0.9707 ± 0.0010	**0.9356 ± 0.0018**	**0.6836 ± 0.0056**	0.7674 ± 0.0020
DocGCN	**0.8650**	**0.9712**	0.9280	0.6404	0.7456

Table 1: Test accuracy of different models on 5 datasets. All numbers except from last row on DocGCN are cited from Yao, Mao, and Luo 2019.

	2	3	5
20NG	**86.50%**	83.83%	80.23%
Ohsumed	**64.04%**	62.21%	56.07%

Table 2: Effect of model depth on performance. We measured the test accuracies on 20*NG* and *ohsumed* by varing the number of model layers (2, 3 and 5).

	32	64	128	256
20NG	85.26%	**86.50%**	86.36%	86.07%
Ohsumed	61.39%	62.97%	64.01%	**64.04%**
R8	95.61%	96.25%	**96.98%**	95.98%

Table 3: Effect of hidden channel size on performance. We measured the test accuracies on 20*NG* and *Ohsumed* by varying the hidden channel size (32, 64, 128, 256).

extending the window too long would cause a drop in performance. We think a larger window size might introduce more noises in the model by connecting unrelated words together. We also noticed the similar behavior when we changed the number of layers in the model. We think this is not by accident, since a larger window size also widens the receptive field of the node as increasing the number of convolution layers does, although later does a more complex transformation of the messages with the nonlinear activation functions.

	2	5	10	20
20NG	85.42%	**86.50%**	85.36%	83.92%
Ohsumed	**64.53%**	64.04%	63.62%	62.80%

Table 4: Effect of window size on performance. We measured the test accuracies on 20*NG* and *Ohsumed* by varing the window size (2, 5, 10 and 20).

Effect of training size. We tested the model with different training sizes. Table 5 shows the test accuracies of our models under different training data ratios (0.05, 0.1, 0.5 and 0.9) on 20*NG*, *R*8 and *Ohsumed*. We found using a ratio of 0.5, our model performed almost as well as using 90% of the training data on 20*NG*, and with 5% usage of training data, our model reached nearly 90% accuracy on *R*8. We found DocGCN performance did not suffer too much when we halved the dataset size, which makes it a appropriate candidate in settings where it is difficult to gather more data.

	0.05	0.1	0.5	0.9
20NG	66.01%	74.00%	84.72%	**86.50%**
R8	89.58%	94.43%	96.76%	**97.12%**
Ohsumed	34.45%	41.63%	56.76%	**64.04%**

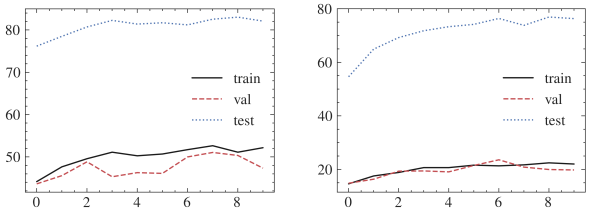
Table 5: Effect of training size on performance. We measured the test accuracies on 20*NG*, *R*8 and *Ohsumed* by varying the percentage usage of the training data (0.05, 0.1, 0.5 and 0.9).

Effect of noise on label. In many real-life settings, data often do not come in good arrangement. For example, the respondent could fill out the wrong label information in a survey, or text quality is suffered due to preprocessing issues. To model the setting with mislabeled data, we assigned the training set (including both the training and validation data) to a randomly chosen label under different perturbation rate (0.1, 0.5 0.7, 0.8, 0.9 and 1.0). Table 6 shows the test accuracies of DocGCN on 20*NG*, *R*52 and *Ohsumed* under different perturbation rates. We observed our model is exceptionally good at counteracting the negative effect of mislabeled data. With a percentage of 70% mislabeled data on the training set, our model still attained more than 80% test accuracy within a span of 10 epochs on 20*NG*. Even with a rate of 0.9, our model still obtained nearly 60% test accuracy on 20*NG*. For all cases, it was only until all the training data were mislabeled that our model started to randomly predict the class.

To get an better understanding of this robustness behavior, we also show the training curves on 20*NG* when we set the mislabeled rate to 0.5 and 0.8 respectively in Figure 2. We observe that instead of fully saturating the training accuracy to close to 100% as our model often did for correctly labeled data (so as the other deep learning models like CNN, LSTM), the training accu-

racy with mislabeled data only raised slightly over the correctly-labeled rate. For a 50% mislabeled data rate, the best training accuracy was 52.65%, and for a 80% rate, the best training accuracy was 23.59%. The similar behavior was also exhibited for other datasets in the experiment. Therefore, we can see DocGCN, or GCN in general, has a high relational inductive bias for the graph structure, so much so that it is hard to overfit our model with the wrong target label. Meanwhile, when the correct label was used in the loss function, the model can easily fit the data again. In this way, the graph structure acts as a flexible internal regularizer, and we think this sets our model apart from most other deep learning models that has a tendency to overfit the data without external regularizers in the objective functions (which can still overfit in many cases).

For further inspection, we also plot the softmax distributions of our models under different mislabeled-data rates (0, 0.1, 0.5, 0.8 and 1.0) in Figure 3 for a randomly selected document in the test set of 20*NG*, which can be found in Appendix B. We found the probability distribution got more randomly distributed when we increased the mislabeled data ratio. However, the right output channel, in this case 'comp.os.ms-windows.misc', still stands out for ratios of 0.5 and 0.8. For a ratio of 0.1, the model gives the highest probability to the class 'comp.sys.ibm.pc.hardware', while the probability for the right class is 9% lower, and both classes belong to the same computer category. The complete random model with 1.0 ratio also predicted the document in the computer category but misclassified it as 'comp.sys.mac.hardware'. This fully shows how powerful GCN models are in general in separating out the data without relying much on the correct label information.



(a) 50% mislabeled data (b) 80% mislabeled data

Figure 2: Training curves under different mislabeled data rates on 20*NG*. We present the relation between epoch (x-axis) and the accuracy during training (y-axis) for both the 50% and 80% mislabeled data rate.

Model interpretability. GCN models are good at detecting hidden graph structures. To inspect what model has learned in the process and to understand the contribution of each edge to the output prediction, we applied the feature attribution method on our correctly trained model on the same document as was chosen for the previous section. Specifically, we compute the integrated gradients (Sundararajan, Taly, and Yan 2017) of the target function with regard to the graph input using an under limit equivalent with no graph input and an upper limit of a binary adjacency matrix. The computed result reflects the importance of each edge to the target label prediction.

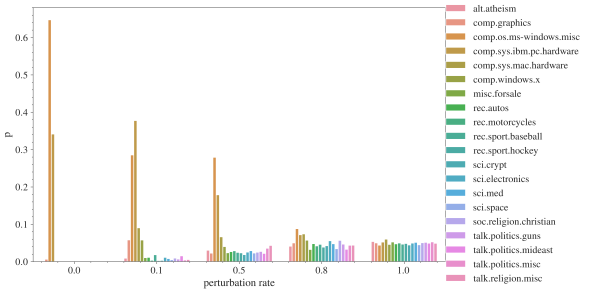


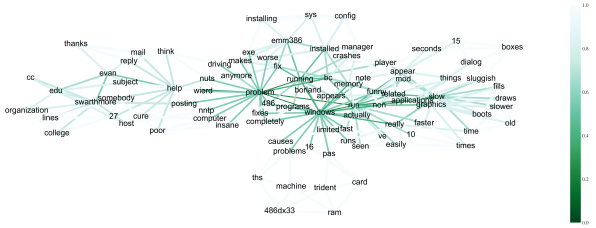
Figure 3: Softmax distribution of the model prediction on a single document under different ratios of mislabeled-data (0.0, 0.1, 0.5, 0.8 and 1.0).

Figure 4 shows both the document graphs with the original edge weight and with the edge importance assigned by the method. Figure 4a shows that the initial PMI weight values were close to each other, and edges incident to the frequent words like 'window' and 'problem' were assigned initially a lower weight (Notice we have reverted the colormap for readability). We found this is because the PMI calculation would favour word pairs that appeared once or so in the context over those which were joined by the frequent words, and for a context of a small-sized document, this bias can be further enlarged.

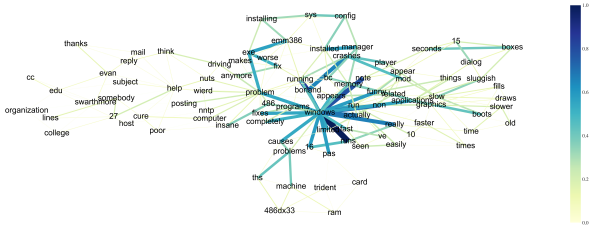
However, Figure 4b shows a different picture. Our model attributes a much higher value for edges that form the important connections in the document, such as 'window-application' and 'problem-exe', than those that are not representative of the document content, such as 'thank-mail' and 'cc-edu'. We can also identify clusters in the graph such as those centered around words like 'window' and 'exe', and also longer paths and bridges such as 'windows-problem-exe-em386' and 'installing-sys-config'. It is the power to detect these hidden structures from the graph that makes it possible for our models to predict accurately in the presence of the mislabeled data.

	0.0	0.1	0.5	0.7	0.8	0.9	1.0
20NG	**86.50%**	84.29%	82.54%	80.39%	76.39%	58.10%	5.16%
Ohsumed	**64.04%**	61.12%	58.15%	54.07%	50.53%	40.00%	3.22%
R52	**92.80%**	90.34%	89.88%	86.64%	84.85%	80.14%	8.10%

Table 6: Effect of mislabeled data on performance. We measured the test accuracies on 20*NG*, *R*52 and *Ohsumed* by varying the ratio of mislabeled data in the training set (0.1, 0.5, 0.7, 0.8, 0.9 and 1.0).



(a) Initial edge weight



(b) Attributed value

Figure 4: Visualization of learning outcome in form of the hidden structure in the graph. We visualize the document graph both with the initial edge weight (divided by the maximum weight) and with the attributed values decided by the integrated gradient method (Sundararajan, Taly, and Yan 2017). Edge importance is indicated by both the edge color and the edge thickness.

5. Discussion

We have proposed document level graph neural networks as a competitive alternative approach for doing text classification. We summarize the advantages of treating document as graph here again. Firstly, we can do inductive learning easily which is not possible with previous models. Secondly, we can use mini-batch training for efficient parameter update. Last but not least, the relational inductive bias of our model can work as a flexible internal regularizer to overcome the issues with mislabeled data.

We have tested this approach using our DocGCN model. We showed that DocGCN achieved or matched state-of-the-art results and it is a robust text classifier that does not dependent too much on the hyperparameters, and can learn well with small training size and mislabeled data. In many practical scenarios, getting the correct labeled data is quite costly, and our model provides a viable approach to learning from sparsely labeled data.

Future study could incorporate more NLP research results. The current graph is simple, built with PMI value and consists of only one node type. We could also extend our graph by building a knowledge graph with multiple relations using pretrained NLP parsing models. For the training models, we could also replace current GCN model with other GNN models like GAT (Veličković et al. 2017) to allow auto weight adjustment for different types of edges.

References

[Bad+17] Pinkesh Badjatiya et al. “Deep learning for hate speech detection in tweets”. In: *Proceedings of the 26th international conference on World Wide Web companion*. 2017, pp. 759–760.

[DS05] Franca Debole and Fabrizio Sebastiani. “An analysis of the relative hardness of Reuters-21578 subsets”. In: *Journal of the American Society for Information Science and technology* 56.6 (2005), pp. 584–596.

[Joa98] Thorsten Joachims. “Text categorization with support vector machines: Learning with many relevant features”. In: *European conference on machine learning*. Springer. 1998, pp. 137–142.

[Kim14] Yoon Kim. *Convolutional Neural Networks for Sentence Classification*. 2014. arXiv: 1408.5882 [cs.CL].

[KW16] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[Lan95] Ken Lang. “Newsweeder: Learning to filter netnews”. In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 331–339.

[PL05] Bo Pang and Lillian Lee. “Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales”. In: *arXiv preprint cs/0506075* (2005).

[Sah+98] Mehran Sahami et al. “A Bayesian approach to filtering junk e-mail”. In: *Learning for Text Categorization: Papers from the 1998 workshop*. Vol. 62. Madison, Wisconsin. 1998, pp. 98–105.

[Sto+20] Jonathan M Stokes et al. “A deep learning approach to antibiotic discovery”. In: *Cell* 180.4 (2020), pp. 688–702.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG].

[TQM15] Jian Tang, Meng Qu, and Qiaozhu Mei. “Pte: Predictive text embedding through large-scale heterogeneous text networks”. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1165–1174.

[Vel+17] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[Wu+19] Felix Wu et al. “Simplifying graph convolutional networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 6861–6871.

[Xu+18] Keyulu Xu et al. “How powerful are graph neural networks?” In: *arXiv preprint arXiv:1810.00826* (2018).

[Yin+18] Rex Ying et al. “Graph convolutional neural networks for web-scale recommender systems”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 974–983.

[YML19] Liang Yao, Chengsheng Mao, and Yuan Luo. “Graph convolutional networks for text classification”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7370–7377.

[ZAL18] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. “Modeling polypharmacy side effects with graph convolutional networks”. In: *Bioinformatics* 34.13 (2018), pp. i457–i466.

[ZK21] Hao Zhu and Piotr Koniusz. “Simple spectral graph convolution”. In: *International Conference on Learning Representations*. 2021.

A. statistics of datasets

Dataset	# Docs	# Training	# Test	# Words	# Classes	Avg Len
20NG	18,846	11,314	7,532	42,757	20	221.26
R8	7,674	5,485	2,189	7,688	8	65.72
R52	9,100	6,532	2,568	8,892	52	69.82
Ohsumed	7,400	3,357	4,043	14,157	23	135.82
MR	10,662	7,108	3,554	18,764	2	20.39

Table 7: Corpus statistics. The table is cited from Yao, Mao, and Luo 2019

B. Document sample used for visualization

From: edorn1@cc.swarthmore.edu (Evan Dorn) Subject: Please HELP!! (Wierd Problem) Nntp-Posting-Host: mac6.trotter1.swarthmore.edu Organization: Swarthmore College Lines: 27 Somebody help me cure my poor computer before I go insane! I have a problem with my 486 when running windows that appears to be memory-related. It's actually not limited to windows, but that's where it causes most of my problems. Ths machine's 486DX33, 8Meg RAM, 256Kcache, TRIDENT TVGA card, PAS-16 soundcard. 1) Windows runs REALLY, REALLY slow most of the time. Slower than on my old 386SX16. Graphics draws/fills are slow, boots are slow, applications are sluggish, dialog boxes take up to 15 seconds to appear. (Note: some of my other non-windows applications do funny things that appear to be related. Several run slow, my .mod player crashes the system etc..) 2) Running Borland C++ 3.0 before running windows (or any of the other programs) COMPLETELY fixes the problem. Windows will run as fast as I've ever seen it run, easily 10 times faster for graphics than when I don't run BC beforehand. 3) I don't have a memory manager installed in config.sys. Installing emm386.exe does not fix the problem- it makes it worse. After emm386 is installed, running BC will not fix the problem anymore. This problem is driving me nuts. If you think you can help, please reply through the mail. Thanks, Evan