TGCN: A Novel Deep Learning Model for Text Classification

Yuan Li
New York University
Brooklyn, NY
yl6606@nyu.edu

Dongzi Qu
New York University
Brooklyn, NY
dq394@nyu.edu

Abstract

Text classification is a traditional problem in natural language processing. Using word embeddings as the main features during classification has been studied for a long time. However, only a few number of studies have explored novel graph convolutional neural networks (GCN) for text classification. In this work, we reimplement a text GCN (TGCN) model from Yao [16]. The graph built can perform relations between word-word and word-document together. Unlike traditional methods, which need word embeddings, we use one-hot representation for word and document as inputs, then TGCN learns the embeddings for word and document automatically, with the input labels for documents. The experimental results show that TGCN performs better than other baseline models, even without pre-prepared embeddings, where the embeddings will be fine-tuned after the training process. Besides accuracy, TGCN also receives high performance on small datasets and other language dataset, which shows the robustness of TGCN on multiple tasks.

1. Introduction

Text classification is an essential and classical problem in natural language processing. There are various applications related to text classification, such as: news filtering, spam detection and document integration/organization. In order to solve this task, most traditional methods take use of features engineering by constructing the embeddings (representations) for documents. Recently, deep models are widely appied to learn the representations for documents. Under the help of neural networks, the semantic and syntactic information inside the document can be captured. However, both traditional methods and deep models may ignore the word co-occurrence in some long-term or non-consecutive sequence analysis. Yao [16] then propose to a text graph convolutional network (Text GCN) for text classification. The word co-occurrence and document word relations are represented in a text graph generated as the training data. The experimental results from Yao shows a vanilla Text GCN without any external word embeddings or knowledge outperforms state-of-the-art methods for text classification in English datasets.

In this work, we reimplemented a text GCN for Chinese text classification using tensorflow [1] 2.0 version. The GCN model is built based on Yao's Text GCN model [16] and measured the performance of this text GCN on Chinese datasets. Our model achieves highest accuracy on all datasets. Then we analyze these results, and discuss the influence of different hyper-parameters on TGCN model. For future research, we think this model can be used on more different language datasets. To summarize, our main contributions are as follows:

- We reimplemented TGCN model for Chineses text classification based on [16].
- We experiment this model on six different datasets. TGCN achieves highest accuracy on all datasets.
- We analyze the reasons for TGCN's performance, and different hyper-parameters on TGCN model.

The code is published on GitHub[1].

[1]https://github.com/FoxerLee/TGCN

2. Related Work

2.1. Traditional Deep Learning

Traditional deep learning methods for text classification can be divided into two ways. One way of studies give more attentions on training data. These studies proved that reasonable use of word embeddings can achieve great success in deep learning methods for text classification. Wang [14] further introduced an attention framework that combined text and text label embeddings.

Besides training data, other studies focused on architecture improvement. Two common used deep learning models are CNN and RNN. In order to give more flexibility of representing sentence in these deep learning models, attention mechanism [13] was used as an internal sub-module [15].

However, these methods mainly focus on sentence in short distance, but ignore the global word relationships in a whole document.

2.2. Graph Neural Network

Graph Neural Network (GNN) is a novel model that has received a lot of attention in the field of deep learning recently. Some studies present new approaches of convolutional neural network (CNN) to work on fixed structured graphs, which makes it possible for us to generate a new graph neural network (GNN) based on CNN to solve text classification ([3], [6]). There are some studies which use GNN for text classification ([2], [10]). Kipf and Welling [6] then provided the graph convolutional network (GCN), which got state-of-art results on a number of NLP related graph datasets. to Based on GCN, Yao[16] then presented the Text Graph Convolutional Network (Text GCN) for a whole document classification. It can jointly generate the embeddings of text and text label together. Meanwhile, text GCN can catch information in the whole document, and request less data than traditional deep learning model for text classification. Our model is mainly constructed based on text GCN and we will modify it to support Chinese dataset.

2.3. Chinese text classification

Chinese text classification has attracted more research attention in recent years. Unlike English text classification, we may need alternative preprocessing method and more robust model. Tao [12] provided a novel Radical-aware Attention-based Four-Granularity (RAFG) model to take full advantages of Chinese characters, words, character-level radicals, word-level radicals simultaneously, using the feature of Chinese text that the character system of Chinese is based on hieroglyphics, which has the raw meanings. However, RAFG model needs a complex method to extract the character-level and word-level radicals of data. Our model only requires a simple word segmentation operation instead.

Another attempt for Chinese text classification is focus on training models. Zhuang [17] created a stroke-Level convolutional networks using the deeper information in the Chinese character. Li [8], on the other hand, used long short-term memory (LSTM) model to capture the sequence information.

3. Dataset

According to our plans, we plan to run our experiment on three different widely used benchmark datasets with English documents, including 20-Newsgroups (20NG), Ohsumed and R52, R8 of Reuters 21578. Meanwhile we are interested in how this model will perform on the Chinese documents. So we choose two Chinese documents dataset, which are THUCTC and fastTextData, to measure our model's robustness. We notice that the original model takes the input document word by word after well preprocessed by tokenization since the edges and vertices in the graph model depend on the documents and each single word. However, the normal Chinese documents are constructed sentence by sentence using the single Chinese characters without any space but some punctuation. Therefore, we use pkuseg [9] as a Chinese special tokenization method to preprocess the Chinese datasets we mentioned below.

- The 20NG dataset[2] (bydate version) contains 18846 documents evenly categorized into 20 different categories. We chose 20% documents from the whole dataset.
- The Ohsumed corpus[3] is from the MEDLINE database, which is a bibliographic database of important medical literature maintained by the National Library of Medicine.

[2]http://qwone.com/ jason/20Newsgroups/
[3]https://www.mat.unical.it/OlexSuite/Datasets/SampleDataSets-about.htm

- R52 and R8[4](all-terms version) are two subsets of the Reuters 21578 dataset.
- The THUCTC[5] (THU Chinese Text Classification) contains more than 740000 news from 2005 to 2011 collected by Sina News. All these news documents are evenly distributed into 14 categories, including financial, education, real estate, sports and so on. We chose 1000 news in each category to speed up training, and use pkuseg for tokenization.
- The fastTextData[6] is relatively small only containing around 24000 news data within 6 categories compared with THUCTC. While, the documents in this dataset are tokenized word by word manually. Consequently, it's much easier to implement the experiment on this dataset.

4. Method

4.1. Graph Convolutional Network

A GCN has the similar structure as the normal convolutional networks. It contains multiple layers, where each layer is constructed based on the node graph data structure. Formally, assume of graph $G = (V, E)$, where V ($|V| = n$) and E denote the vertices set and edges set. Also, we introduce an adjacency matrix $A \in \mathbb{R}^{n\times n}$ and a diagonal matrix $D \in \mathbb{R}^{n\times n}$, where $D_{ii} = \sum_j A_{ij}$. Moreover, let $X \in \mathbb{R}^{n\times k}$ be a feature matrix (map) for all the vertices in V of one layer, where k is the size of feature space and each row $x_v \in \mathbb{R}^k$ is the feature vector for v. X is the basic structure and each layer in GCN should be constructed by it. To get the new feature matrix for the $(i+1)^{th}$ layer ($L^{(i+1)}$), we can take use of the current result in i^{th} layer ($L^{(i)}$) with some matrix multiplication and activation

$$L^{(i+1)} = \rho(\tilde{A}L^{(i)}W_i) \tag{1}$$

where $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, which is same among all layers and W_i is a weight matrix and its shape depends on the second entry of $L^{(i)}$ and the first entry of $L^{(i+1)}$. ρ is a activation function, which we can use ReLu or tanh. Also, $L^{(0)} = X$, which is the initialized feature matrix. W_i could be a rectangular matrix since the size of feature space might be different inside different layers.

4.2. Text GCN

Based on the structure of the basic GCN, we build a large and heterogeneous text graph which contains word nodes and document nodes. Therefore, the size of the node set is the number of documents (corpus size) plus the number of distinct words in a corpus. In this graph, edges exist between word to word or word to document. The edge between node and document can be calculated by Tf-idf of the word in that document. As for the word to word edge, the weight can be computed using point-wise mutual information (PMI), a popular measure for word associations. By use PMI, the graph will capture the correlation among some words with higher dependency. Also, for each node itself, we need to add a self-loop on it.

After building the text graph as in [6], assume we have j layers, then the output $Z \in \mathbb{R}^{n\times F}$ of the last layer can be computed as

$$Z = \text{softmax}(\tilde{A}L^{(j-1)}W_{j-1}) \tag{2}$$

where F is the number of categories and we use softmax function rather than ReLU is we want to make predictions among a group of labels. Let $j = 2$, then the whole model can be written as

$$Z = \text{softmax}(\tilde{A}\ \text{ReLU}(\tilde{A}XW_0)W_1) \tag{3}$$

The loss function is defined using cross entropy error over all labeled documents:

$$\mathcal{L} = -\sum_{i\in\mathbf{I_D}}\sum_{f=1}^{F} Y_{if}\ln Z_{if} \tag{4}$$

[4]https://www.cs.umb.edu/~smimarog/textmining/datasets/
[5]http://thuctc.thunlp.org/
[6]https://github.com/CementMaker/cnn_lstm_for_text_classify

where $\mathbf{I_D}$ is the indices set for all documents and F is the dimension of the output feature, which is equal to the number of classes. Y is a label indicator matrix. Additionally, all the trainable parameters are coming from two huge matrix W_0 and W_1, which can be trained via gradient descent.

The architecture of our GCN training step is figure 1. The implementation of GCN is mainly based on official Text GCN[7] repository.

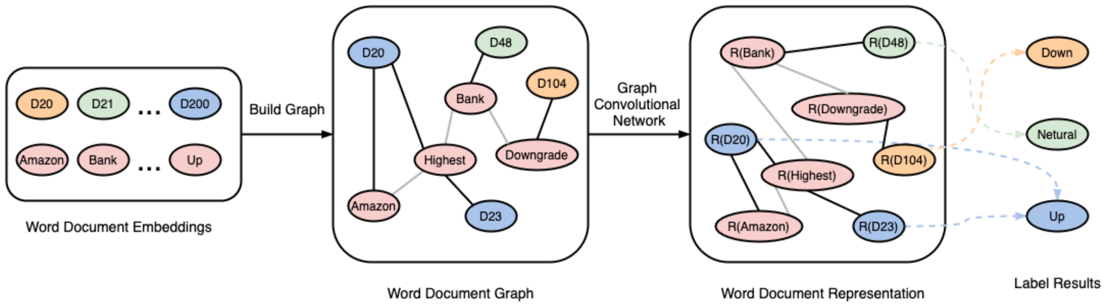


Figure 1: Steps for Training GCN. ”D” means they are document nodes, which is a piece of title data for each. Others are word nodes. There are two edges: black edges are document-word edges and gray edges are word-word edges. $R(x)$ means the representation of x. Different colors are for different labels.

5. Experiment

In this section, we evaluate tf2-based TGCN model on two different experimental tasks. Specifically, in order to make the model robust and general, we want to determine: (1) Can our new model also achieve satisfactory results on English text dataset, or higher accuracy than the original model? (2) Can our new model achieve satisfactory results on two selected Chinese text dataset? The following part will introduce baselines we compared with our model, settings of our model and performances of baselines and our model.

5.1. Baselines

We compare the TGCN model with multiple state-of-the-art text classification methods below:

- **Tf-Idf + LR:** Term frequency–inverse document frequency vectors are used as document embeddings. The classifier is Logistic Regression.
- **fastText:** It is a simple and efficient text classification method proposed by Juolin [5]. The document embeddings are represented by the average of word or n-grams embeddings. The classifier is a linear classifier. Two baselines (with and without bigrams) are evaluated.
- **LEAM:** Label-embedding atteentive models which is introduced in [14]. It embedded each label in the same space with the word vectors. So this model could maitain the interpretability of word embeddings.
- **SWEM:** Simple word embedding models which is introduced in [11]. It provided two polling strategies for word embeddings, (1) max-pooling for interpretability and (2) hierarchical pooling for spatial (n-gram) information.

5.2. Settings

The embedding size of the first convolutional layer is 200, and the learning rate is 0.01. In order to improve our model's robustness, we add dropout process in each convolutional layer with dropout rate 0.5. The L_2 loss weight is set as $5 * 10^{-4}$. The maximum of training epochs is 200 with Adam. We randomly chose 10% data in training dataset as validation dataset for early stopping. The early stopping is set as 5 epochs that if there are no more loss decrease of validation dataset. The setting of parameters is mainly based on [6] and [16]. For baseline models, the parameters are following their original papers or the default settings.

[7]https://github.com/yao8839836/text_gcn

Model	R8	R52	20NG	Ohsumed	THUCTC	fastTextData
Tf-Idf + LR	0.9488	0.8742	0.8216	0.5486	0.9367	0.9575
fastText	0.9658	0.8871	0.7672	0.5045	0.9217	0.9583
fastText (bigrams)	0.9621	0.8563	0.7583	0.4829	0.8950	0.9427
LEAM	0.9389	0.9102	0.8321	0.5503	–	–
SWEM	0.9488	0.9301	0.8527	0.6323	–	–
TGCN	**0.9694**	**0.9361**	**0.8702**	**0.6884**	**0.9512**	**0.9637**

Table 1: Test Accuracy on different dataset. They are mean results for 10 times. TGCN receives highest accuracy in all datasets. LEAM and SWEM don't support Chinese datasets, so there are no results for them on THUCTC and fastTextData datasets.

5.3. Performance

Table 1 shows test accuracy of our TGCN and baseline models on each dataset. As shown, TGCN receives best accuracy on all dataset and outperforms all baseline models on both English and Chinese datasets. Unlike fastText, which performs better than Tf-Idf + LR on short documents data (R8, R52), but worse on longer documents data (20NG, THUCTC), the results of TGCN prove that TGCN will not be affected by the length of document.

For the results of baseline models, we notice that Tf-Idf + LR receives a good performance on long text datasets, like 20NG and THUCTC with random initialization word embeddings. LEAM and SWEM also perform well in all English datasets, which suggest that label embeddings and pooling strategies have a positive influence on this task. However, the code provided by the authors of LEAM and SWAM doesn't support Chinese datasets. An interesting point is that the accuracy on Ohsumed dataset of each model is not so high compared with other dataset. The reason might be there are lots of uncommon used words in this corpus, which leading to more difficulty on generating the embeddings. So, besides the length of the document, the accuracy will also be influenced by the sentences inside the corpus.

6. Analysis

There are several possible reasons for TGCN's high performance. As mentioned in paper [7], the features of a node are computed with the weighted average of itself and its second order neighbors. In this way, the word nodes can captured the label information of its neighbor document nodes, then transfer to other word and document node. Finally, label information is covered to every node in the whole graph. Word nodes can be regarded as pathways in training. Based on the above analysis, we can explain the reason why TGCN doesn't outperform baseline models much on short text datasets (R8, R52) than long text datasets (THUCTC, 20NG). A short text dataset can only generate fewer nodes and edges than other text datasets. The few word-document edges limit the label information passing among the nodes.

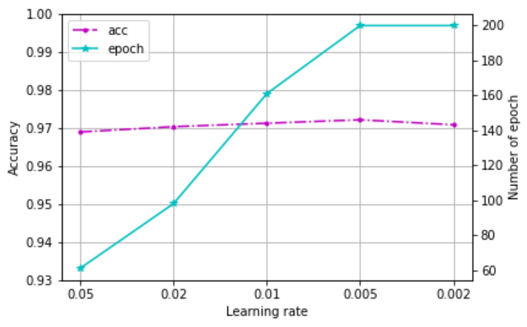
More detail analysis about parameters and visualization are provides below:

6.1. Learning rate

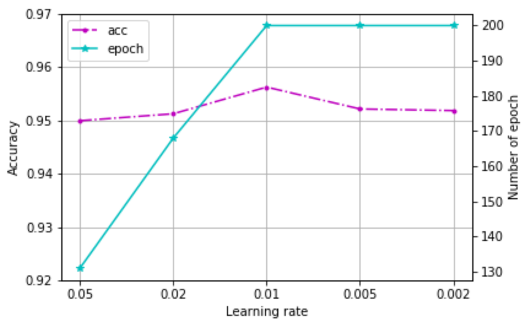
Figure 2 shows test accuracy and number of epoch when training is finished for different learning rate. We find that larger learning rate can significantly reduce the training time with a small loss of accuracy. For instance, $lr = 0.05$ only needs 61 epochs to receive 0.9689 accuracy, comparing $lr = 0.01$ with 160 epochs and 0.9712 accuracy on R8 dataset. The experimental results match our perception: a large learning rate can speed up the convergence of our model. Although it may cause the fluctuation of loss function, the effect on accuracy is limited.

6.2. Size of training data

Following [16], we compare test accuracy using partial training data of TGCN and baseline models. The results of 5%, 10%, 15%, 20% and 25% are shown in figure 3. The results show that TGCN will not be affected by size of training data, unlike baseline models. For instance, TGCN reaches 0.9063 accuracy on R8 dataset with 5% training data, comparing fastText only reaches 0.4100 on R8 dataset with 5% training data. On THUCTC dataset, TGCN reaches 0.9242 accuracy with 15% training data, which is higher than fastText' accuracy with the whole training dataset. It suggests that the label information is covered in the whole word document graph, so we don't need to use a large data with repeated label information. This makes TGCN easier to train.

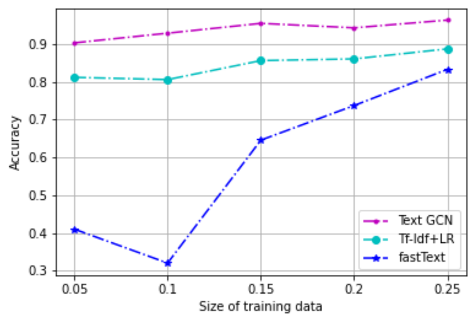


(a) R8

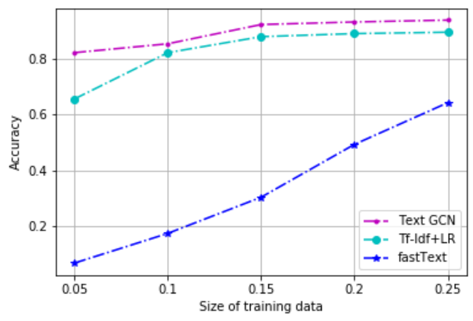


(b) THUCTC

Figure 2: Test accuracy and training epochs for different learning rate. A larger learning rate will need a smaller number of epoch to finish training.



(a) R8



(b) THUCTC

Figure 3: Test accuracy by partial training data. A small amount of data does not reduce the accuracy of TGCN too much.

6.3. Embedding size

Following [16], we check whether different embedding sizes will influence test accuracy. The results are shown in figure 4. Unlike results in [16], we find a higher embedding size can receive a better test accuracy, although the improvement is small (0.0055 higher from 100 to 250). And the embedding size of 10 can only reaches 0.8750 accuracy. It suggests that a low embedding size, however, may not be able to transfer label information well.

6.4. Visualization

We also visualize the word embeddings using t-SNE [4]. The doc embeddings of R8 dataset in different layers with 200 dimensions learned by TGCN is shown in figure 5. Comparing the embeddings in first layer, the embeddings for label **earn** in second layer has a more obvious distance between embeddings for label **acq**, and less misclassification results. It can be seen that the vectors obtained after training have a high degree of discrimination, so a good classification accuracy rate can be finally obtained.

7. Limitations

During the implementation, the main limitation we are facing currently is the lackness of the powerful hardware and training platform. Since we need to do experiments on two very long-sentence dataset (1000+ length for each sentence in THUCTC and 20NG) and one big-size corpus (10000+ documents in Oshumed), we need a huge GPU memory space and a very long training time. To tackle this challenge, we tried to use HPC from New York University. However, it was not stable sometimes. At the same time, Colab platform provides enough GPU memory but a limited CPU. We still need a large

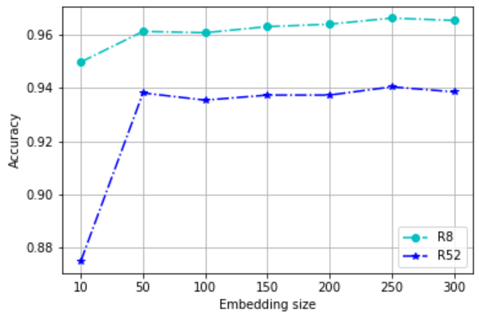
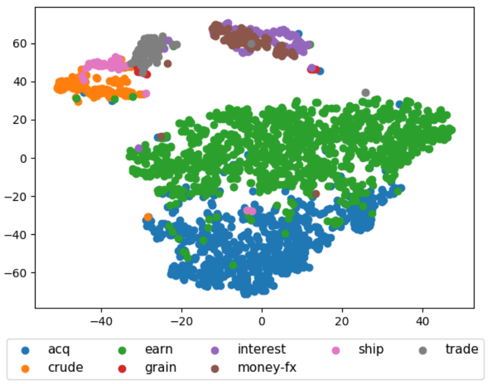
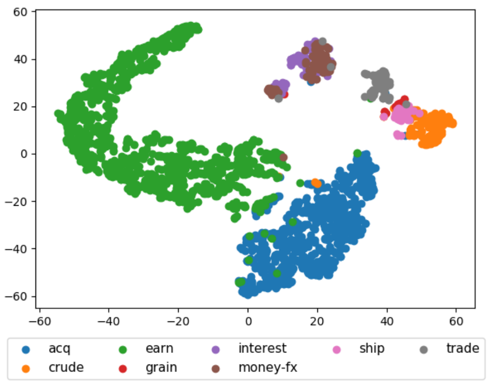


Figure 4: Test accuracy with different embedding size.



(a) Doc embeddings of R8 in first gcn layer



(b) Doc embeddings of R8 in second gcn layer

Figure 5: Doc embeddings of R8 in different layers. Embeddings are reduced to 2 dimensions using t-SNE [4].

CPU memory running TGCN model on 20NG dataset to generate word graph. Finally, we found Floydhub platform which providing enough CPU and solve our problems. Currently, the two platforms we are using are: (1) colab, with Tesla K80 GPU 12GB and CPU 12GB; (2) Floydhub, with Tesla K80 GPU 12GB and CPU 61GB.

The remaining work is mainly on results collection and analysis. As mentioned in section 5, there are five parts we have shown and want to make more detail analysis. The results are mainly based on English datasets. We plan to run same experiments on Chinese datasets. Besides the mentioned parts, we will also compare the different methods of Chinese segmentation, training time of different models, etc. And the final code will be collected and available on GitHub.

8. Conclusion

In this work, we tried to reimplement the TGCN model solving the text classification task on several dataset both in English and Chinese. As for English text, following the same setting from Yao [16], we run TGCN multiple times on these dataset and got performing results compared with the original paper; while, for the Chinese dataset, we first took use of tokenization tool from pkuseg and a slight different data cleaning tools to preprocess the dataset. After that, we run several experiment on them and also got the performing results. Based the results, we proved that TGCN model is robust because it can perform better than all the baselines not only on dataset with different sentence length (corpus size) but also on dataset with different language.

Furthermore, we made some analysis on our results mainly focusing on Learning rate, size of training set, embedding set etc. Specifically, TGCN model does not need the whole corpus or a large embedding size to get the higher accuracy observably. Additionally, the learning rate is not a significant aspect for the better performance.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

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

[4] G. E. Hinton and S. Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:857–864, 2002.

[5] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[6] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[7] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*, 2018.

[8] Y. Li, X. Wang, and P. Xu. Chinese text classification model based on deep learning. *Future Internet*, 10(11):113, 2018.

[9] R. Luo, J. Xu, Y. Zhang, X. Ren, and X. Sun. Pkuseg: A toolkit for multi-domain chinese word segmentation. *CoRR*, abs/1906.11455, 2019.

[10] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 World Wide Web Conference*, pages 1063–1072, 2018.

[11] D. Shen, G. Wang, W. Wang, M. R. Min, Q. Su, Y. Zhang, C. Li, R. Henao, and L. Carin. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. *arXiv preprint arXiv:1805.09843*, 2018.

[12] H. Tao, S. Tong, H. Zhao, T. Xu, B. Jin, and Q. Liu. A radical-aware attention-based model for chinese text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5125–5132, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. 2017.

[14] G. Wang, C. Li, W. Wang, Y. Zhang, D. Shen, X. Zhang, R. Henao, and L. Carin. Joint embedding of words and labels for text classification. *arXiv preprint arXiv:1805.04174*, 2018.

[15] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

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

[17] H. Zhuang, C. Wang, C. Li, Q. Wang, and X. Zhou. Natural language processing service based on stroke-level convolutional networks for chinese text classification. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 404–411. IEEE, 2017.