Hindawi
Journal of Electrical and Computer Engineering
Volume 2020, Article ID 9701427, 7 pages
https://doi.org/10.1155/2020/9701427

Research Article

CharTeC-Net: An Efficient and Lightweight Character-Based Convolutional Network for Text Classification

Aboubakar Nasser Samatin Njikam and Huan Zhao

School of Information Science and Engineering, Hunan University, 057 Changsha, HN 410082, China

Correspondence should be addressed to Huan Zhao; hzhao@hnu.edu.cn

Received 23 April 2019; Revised 29 August 2019; Accepted 6 January 2020; Published 17 June 2020

Academic Editor: Ping-Feng Pai

This paper introduces an extremely lightweight (with just over around two hundred thousand parameters) and computationally efficient CNN architecture, named CharTeC-Net (Character-based Text Classification Network), for character-based text classification problems. This new architecture is composed of four building blocks for feature extraction. Each of these building blocks, except the last one, uses 1×1 pointwise convolutional layers to add more nonlinearity to the network and to increase the dimensions within each building block. In addition, shortcut connections are used in each building block to facilitate the flow of gradients over the network, but more importantly to ensure that the original signal present in the training data is shared across each building block. Experiments on eight standard large-scale text classification and sentiment analysis datasets demonstrate CharTeC-Net's superior performance over baseline methods and yields competitive accuracy compared with state-of-the-art methods, although CharTeC-Net has only between 181,427 and 225,323 parameters and weighs less than 1 megabyte.

1. Introduction

Deep neural networks have given us all kinds of new capabilities in areas such as image recognition, language translation, speech recognition, and face recognition. This is certainly because, over time, researchers have found ways to very effectively train deeper and wider neural networks. Although increasing the size of a neural network has shown, more often than not, to typically improve model accuracy, this comes at the cost of increase memory and compute requirements for training the model. One inconvenience of it is that we end up with neural network architectures that are less efficient in terms of model size and speed. In fact, for computationally limited platforms such as mobile devices, the model must be small enough to fit in memory while delivering strong performance. In addition, since in such peripherals memory access is generally much slower than computations, the number of model's parameters is very important.

This article focuses on text classification, specifically on the character-based convolutional neural network for text classification. We describe an extremely lightweight, with just over a hundred parameters, but highly efficient convolutional neural network architecture, named CharTeC-Net, which is best suited for computationally limited platforms.

The rest of this article is organized as follows. Section 2 describes the CharTeC-Net architecture. Section 3 reviews the literature on earlier work on character-based text classification. Section 4 presents experiments on six publicly available large-scale datasets for text classification and sentiment analysis. Section 5 concludes with a summary and conclusion.

2. Related Work

Text classification is a predictive modeling problem where given a text (in the form of some sequence of inputs over space or time), the task is to predict the category of the text. It has many applications in today's world, including topic labeling [1], spam filtering [2], sentiment analysis [3, 4], and web searching. As a result, this problem has attracted

considerable attention from machine learning and natural language processing (NLP) researchers and practitioners. Techniques to model this problem have evolved over time, from traditional approaches such as bag of words and ngrams to more advanced techniques such as word embedding and deep learning.

Traditional approaches to text classification such as bag of words and ngrams represent documents with sparse lexical features and then use linear models or kernel methods for classification [1, 5–7]. In particular, Joulin et al. [8] show that linear models could scale to a very large dataset rapidly with a proper rank constraint and a fast loss approximation. Though these traditional approaches ignore semantic and syntactic information about words [6], they are very popular and have long been the state-of-the-art methods for text classification.

As noted above, traditional approaches to text classification cannot capture syntactic and semantic information about words. To mitigate these limitations, word embedding was introduced [9, 10]. It is a distributed-representation technique mapping words to high-dimensional vectors. This representation can capture contextual and semantic information about words and nowadays is typically learnt using neural networks.

The ability to correctly classify or categorize text from raw data has been greatly revolutionized in recent years using deep learning techniques [11, 13] with state-of-the-art performances on several small datasets [14]. However, these models are still largely word-based. This makes it difficult to capture user-generated content with typos and new vocabulary [15] and yield models that are language-specific. To tackle these limitations, character-level information for various NLP tasks was explored. For instance, a hybrid approach combining character-level convolutional neural networks (CNNs) with word embedding was used for part-of-speech tagging [16] and named entity recognition [17]. Similarly, character-level CNNs were combined with ngrams method for information retrieval [18]. But it was the work of [15] that enabled neural networks to be trained end-to-end on texts without any preprocessing, not even tokenization. They introduced a convolutional operator on characters that can automatically learn the concepts of words and sentences.

Inspired by this seminal work, Zhang et al. [19] proposed the first deep CNN for text that operates at the character level. Their model takes as input a sequence of encoded characters and has 6 convolutional layers and 3 fully connected classification layers. The configuration of these layers is task-specific. For instance, 256 filters and 1024 units are used for relatively large datasets, while for larger datasets, 1024 filters and 2048 units are used. Thus, the structure of this architecture is complex, task-specific, and therefore difficult to extend and generalize to other datasets. Another major contribution of Zhang et al.'s work is the introduction of several large-scale datasets for text classification, which we use in this paper. Building on the top of Zhang et al.'s seminal work and inspired by recent advancement of convolutional architectures for computer vision [20, 22], presented a simpler but very deep model with 29 convolutional layers. This certainly prompted He et al.[23] to study on the importance of depth in convolutional models for text classification.

The combination of convolutional and recurrent networks for sentence classification was explored as well. Xiao and Cho [24] used a convolutional network with up to 5 layers to learn high-level features, which serve as input for an LSTM. Kim et al. [14] introduced a character-aware neural language model by combining a CNN on character embeddings with a highway LSTM on subsequent layers. Radford et al. [25] also explored a multiplicative LSTM (mLSTM) on character embeddings and found that a basic logistic regression learnt on this representation can achieve state-of-the-art result on the Sentiment Tree Bank dataset [26] with only a few hundred labeled examples.

3. CharTeC-Net Architecture

The main hypothesis behind character-based text classification is that a multilayered neural network can examine text one character at a time, using successive layers to build a hierarchical understanding of words, then phrases, and eventually whole documents.

In this section, we first describe how to preprocess the text one character at a time as this will serve as a basis for understanding the core of our network design choices. We then describe the overall architecture of our network and discuss our design choices along the way.

3.1. Input Sequence. Following [19], all preprocessing is done one character at a time, which is the atomic representation of a word, identical to the pixels for the images. The idea is to convert each input character into a one-hot encoding vector. To do this, we need a vocabulary or alphabet set. There are only 68 possibilities in English, and that includes 26 lowercase English letters, 10 Arabic numerals, 31 punctuation marks, and the nonspace characters: "\n," "!," "#," "‘," "%," "$," "’," "&," ")," "(," "+," "*," "–," "‚", "/," ".," "1," "0," "3," "2," "5," "4," "7," "6," "9," "8," ";," ":," "=," "<," "?," ">," "@," "[", "], "\\," "_," "‚" "a," "`," "c," "b," "e," "d," "g," "f," "i," "h," "k," "j," "m," "l," "o," "n," "q," "p," "s," "r," "u," "t," "w," "v," "y," "x," "{", "z", "}", "|," "~." To this, we add a special character in the alphabet set to represent all characters we might encounter in our texts, but are not part of the 68 English character set. Each character is then represented as a 69 long one-hot encoding vector, which means that each character has 69 dimensions with a single bit turned on in the position (relative to the vocabulary) corresponding to that character. Next, since texts can have different lengths, we have to make sure that all the input text has the same length so that the CNN can handle the batch data. To do this, we set the maximum length of the sentence to 1014 following [19]; smaller texts are padded with 0, while larger ones are truncated. All texts will therefore remain the same length.

3.2. Pointwise Layer. CharTeC-Net architecture uses pointwise layers. Initially, pointwise layers, also known as 1×1 convolutions, were proposed in network-in-network [27]. Typically, it is a standard convolution that uses a 1×1 kernel or a kernel that iterates through every single pixel of the input feature. This kernel has a depth of whatever many channels the input feature has. This type of convolution is dual purpose: not only can it be used to reduce or increase the channel dimension of the feature map but also by adding a nonlinear activation to them (like ReLU) You are increasing model capacity [28].

3.3. Shortcut Connection. The power of shortcut connections came to light in a network architecture called ResNet, introduced in [21]. Shortcut connections are used from higher-level layers to lower-level layers. This way, data not only flow through each layer in a sequential order but also follow shortcuts from higher layers to lower layers. This technique has proved to be extremely powerful to mitigate the vanishing gradient descent problem and therefore making it possible to train very deep neural networks. Precisely, this design allowed the authors of the ResNet architecture to build and train much deeper neural networks than before. For instance, instead of the 16 to 19 layers in VGG networks [20], ResNets typically can have up to 150 layers.

The CharTeC-Net architecture makes use of shortcut connections, not to build deeper neural networks, but to ensure that the original signal in the training data is shared across each building blocks of the network.

3.4. Designing the Network. The overall architecture of our network is illustrated in Figure 1. Our model consists of 10 layers arranged in 4 building blocks for feature extractions and a block for text classification. Each of these blocks consists of two layers. The input to our model is a 3D tensor of size (*Batch_size*, *seq_length*, and *vocab_size*) containing the embeddings of the characters; *seq_length* is set to 1014, which is the length of the input sequence. *vocab_size* is the size of our vocabulary or alphabet set (69 in this case) and can be interpreted as the "channel" dimension (similar to rgb channels in images) of the input text (see Appendix). For the feature extraction, each block in the model, except the last, starts with 192 pointwise convolutions, followed by 69 convolutions of kernel size 3. For the last building block of our feature extraction, we replace the pointwise convolutions used in the previous building blocks by a maxpooling layer of size 2 and stride 1. We then construct shortcut connections between the 3D input tensor and the output from each block, as depicted in Figure 2. The full CharTeC-Net architecture, then, consists of 4 of these building blocks in a row. This is followed by a regular average pooling layer (of kernel size 19 and strides 19) and a fully connected classification layer with softmax activation.

3.4.1. Implementation Details. Except for the pointwise layers and the average pooling layer, the output from every other layers is padded. The rectified linear unit activation is used in all the convolution layers, and the softmax activation is applied to the fully connected layer.

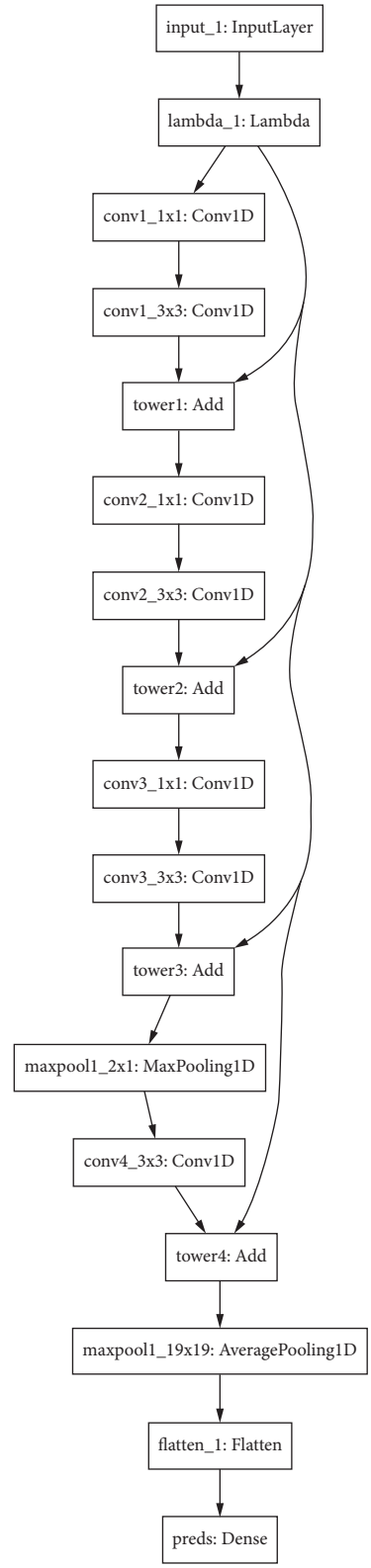

Figure 1: The CharTeC-Net architecture.

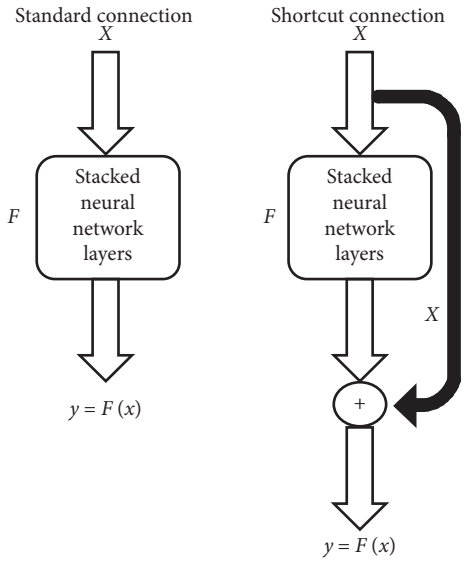

Figure 2: Difference between a standard connection and a shortcut/skip connection.

4. Experimental Evaluation

4.1. Datasets. We evaluate the effectiveness of our model on the same eight publicly available large-scale datasets used in [19] and summarized in Table 1. They span several text classification tasks such as sentiment analysis, topic classification, or news categorization. The size of these datasets ranges from 120k up to 3.6M samples in the training set. The number of classes ranges from 2 to 14. AG's news is the smallest dataset, while Amazon Review Full is the largest. The statistics of the datasets are summarized in Table 1.

4.2. Hyperparameters and Training. For all experiments, we train our model's parameters with the Adam Optimizer [29] with an initial learning rate of 0.001, and a minibatch size of 80. The model is implemented using Keras with Tensorflow as backend and trained using a single NVidia GTX GPU. The hyperparameters are chosen following [14, 19], which are described below. We do not use "thesaurus data augmentation" or any other preprocessing, except lowercasing. On average, it takes about 10 epochs to converge.

4.3. Results and Discussions. Table 2 details the accuracy obtained with our models (CharTeC-Net) and compares them with state-of-the-art results on 8 corpus and text classification tasks. The models from the literature we compare to are

(i) bag of words: the BOW model is based on the most frequent words from the training data [19]

(ii) ngrams: the bag-of-ngrams model exploits the most frequent word ngrams from the training data [19]

(iii) ngrams TFIDF: same as the ngrams model but uses the words TFIDF (term-frequency inverse-document frequency) as features [19]

(iv) fastText: a linear word-level model with a rank constraint and fast loss approximation

(v) char-CNN: character-level convolutional network with 6 hand-designed CNN layers [19]

(vi) char-CRNN: recurrent layer added on the top of a character convolutional network [24].

(vii) VDCNN: very deep character-level model with 29 convolutional layers

(viii) Naive Bayes: a simple count-based word unigram language model based on the Naive Bayes assumption

(ix) Kneser–Ney Bayes: A more sophisticated word count-based language model that uses trigrams and Kneser–Ney smoothing

(x) MLP Naive Bayes: an extension of the Naive Bayes word-level baseline using a two-layer feedforward neural network

(xi) Discriminative LSTM: word-level model with logistic regression on the top of a traditional LSTM

(xii) Generative LSTM-independent comp.: a class-based word language model with no shared parameters across classes

(xiii) Generative LSTM-shared comp.: a class-based word language model with shared components across classes visually compares the performances of 3 character-level models with 2 word-level models. Character-level models include our shallow-and-wide CNN model

(xiv) Shallow-and-wide CNN: a model for both character-based and word-based CNN with 3 convolutional layers. A global max-pooling is then applied to the whole sequence on each filter. Finally, the outputs of each kernel are concatenated to a unique vector and fed to a fully connected layer. The kernel sizes of the convolutional layers for character-level (char shallow-and-wide CNN) are 15, 20, and 25 with 700 filters each. For word-level (word shallow-and-wide CNN), kernels size are 3, 4, and 5 with 100 filters for each convolutional layer.

Table 3 shows the number of parameters and model size of the CharTeC-Net architecture based on the number of classes present in each dataset. CharTeC-Net is extremely lightweight, making it ideal for computationally limited platforms such as mobile devices and other internet-connected devices. In comparison, the typical fastText model [8] uses a few gigabytes of memory.

By propagating the original signal present in the training data over the network via residual connections, associated with the use of a large pool and strides at the end of the network, CharTeC-Net does not only require no regularization (e.g., dropout, weight-decay, and early-

Table 1: Large-scale text classification datasets used in our experiments.

Dataset	#train	#test	#classes	#classification task
AG's news	120 k	7.6 k	4	English news categorization
Sogou news	450 k	60 k	5	Chinese news categorization
DBPedia	560 k	70 k	14	Ontology classification
Yelp review polarity	560 k	38 k	2	Sentiment analysis
Yelp review full	650 k	50 k	5	Sentiment analysis
Yahoo! Answers	1,4 M	60 k	10	Topic classification
Amazon Review Full	3,0 M	650 k	5	Sentiment analysis
Amazon Review Polarity	3,6 M	400 k	2	Sentiment analysis

Note: see [19] for detailed description.

Table 2: Accuracy results for all the models. Numbers are in percentage.

Models	AG's news	Sogou news	DBPedia	Yelp review polarity	Yelp review full	Yahoo! answers	Amazon review full	Amazon review polarity
Bag of words [19]	88.8	92.8	96.6	92.2	58.0	68.9	54.6	90.4
ngrams [19]	92.0	97.0	98.6	95.6	56.3	68.5	54.3	92.0
ngrams TFIDF [19]	92.4	97.2	98.7	95.4	54.8	68.5	52.4	91.5
Char-CNN [19]	87.2	95.1	98.3	94.7	62.0	71.2	58.7	94.5
fastText [8]	92.5	96.8	98.6	95.7	63.9	72.3	60.2	94.6
Char-CRNN [24]	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN [22]	91.3	96.8	98.7	95.7	64.7	73.4	63.0	95.7
Naive Bayes [30]	90.0	86.3	96.0	86.0	51.4	68.7	—	—
Kneser-Ney Bayes [30]	89.3	94.6	95.4	81.8	41.7	69.3	—	—
MLP Naive Bayes [30]	89.9	76.1	87.2	73.6	40.4	60.6	—	—
Discriminative LSTM [30]	92.1	94.9	98.7	92.6	59.6	73.7	—	—
Generative LSTM-independent comp. [30]	90.7	93.5	94.8	90.0	51.9	70.5	—	—
Generative LSTM-shared comp. [30]	90.6	90.3	95.4	88.2	52.7	69.3	—	—
Char shallow-and-wide CNN [23]	90.7	—	98.0	94.4	60.3	70.2	—	—
Word shallow-and-wide CNN [23]	92.2	—	98.7	95.9	64.9	73.0	—	—
CharTeC-Net (our model)	91.6	96.4	98.5	95.83	63.6	67.8	61.5	94.0

Table 3: Number of parameters and model size of CharTeC-Net based on the number of classes.

#classes	#parameters	Model size (in megabyte)
2	181,427	0.76
4	188,743	0.79
5	192,401	0.80
10	210,691	0.88
14	225,323	0.94

stopping) but also demonstrates competitive performances with minimal parameters and extremely lightweight model size.

Our results are comparable to those of other systems. However, CharTeC-Net combines the very nice properties of character-based CNN models while being highly optimized for model size and memory footprint. You may want to use a character-based CNN text classifier over a word-based text classifier if:

(i) Your data are noisy: character-based models are particularly attractive for user-generated content with typos and new vocabulary, as well as for a set of novel tasks such as replacing regex rules for ad blocking purposes.

(ii) Your data are large: character-based models require large datasets, from several thousands up to millions of samples.

Additionally, character-based models find patterns within the character streams and thus do not suffer from having a limited vocabulary. In summary, if you are looking for a compact deep learning architecture for text classification so that the complete model fits in a limited amount of

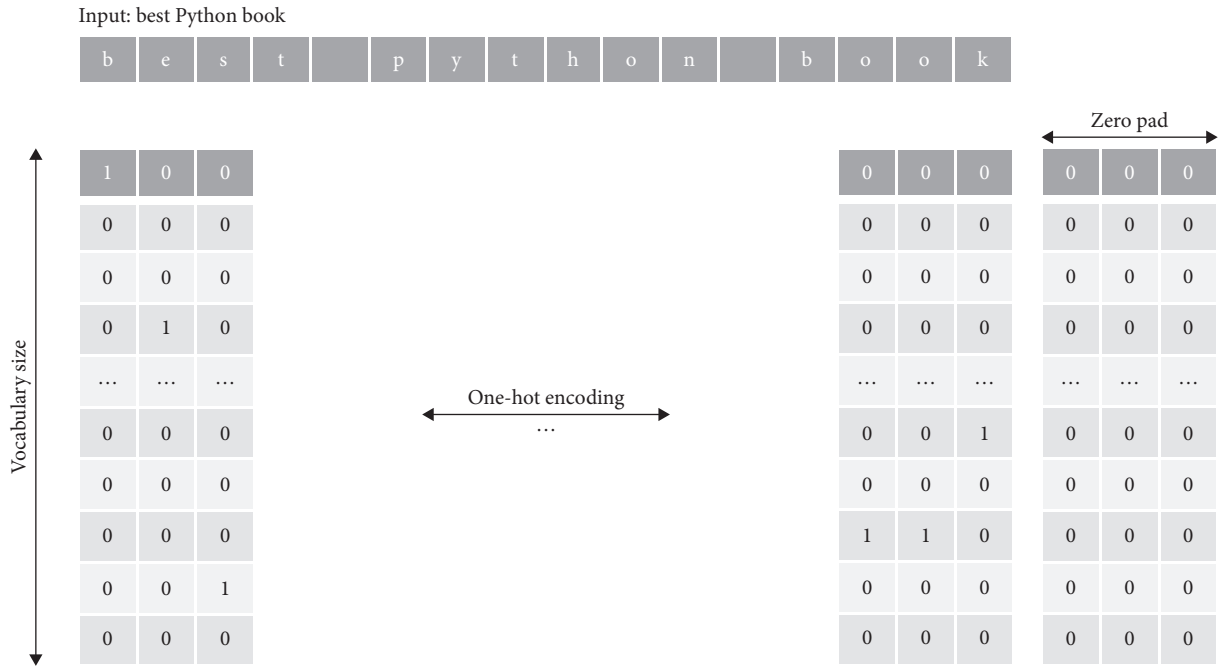

Figure 3: Character-level text encoding.

memory without loss in accuracy, CharTeC-Net is currently the most suitable solution.

5. Conclusion

We introduced a new model architecture called CharTeC-Net that operates at the lowest atomic representation of texts, i.e., characters. CharTeC-Net was designed with model size and model accuracy in mind. To demonstrate the effectiveness of this approach, this architecture was evaluated on eight freely available large-scale datasets, and we were able to show that its performance is on par with state-of-the-arts. Our models are much smaller in terms of number of parameters than other approaches. And to the best of our knowledge, this is the lightest model available for NLP.

Appendix

When we hear about convolutional neural network (CNNs), we generally think about applications in computer vision such as image classification, object detection, or semantic segmentation. However, just like images, texts can be processed as a raw combination of bare units that are not pixels but characters. In this section, we go into the details of how CNN architecture is used for text classification at the character level while drawing the analogy with CNN on image data.

Suppose we have a set of Amazon reviews that we wish to classify as either positive or negative. In other to prepare these data, there are two things we have to define: a vocabulary and a fixed length for each review. The vocabulary is a set of all characters that our data may contain. For instance, for English, we may consider the 26 lowercase English letters, the 10 Arabic numerals, the 31 punctuation marks, and the nonspace character to form our vocabulary. Generally, a neural network needs to have the same input size for all of its data. That is why, a predefined fixed length for our Amazon review data is chosen. For instance, if our fixed input length is set to 1014 characters, then for each review we will check its length and pad out to 1014 if too short or truncate to 1014 if too long.

Next, we cast each review in lowercase, split it into a list of individual characters, and turn each character into a one-hot vector. As you can see in Figure 3, the one-hot vector represents each character by a sparse vector where the entry that corresponds to the position of the character in the vocabulary has its value turned on (set to 1).

This process allows to turn every text (amazon review in this example) into a matrix of 0s and 1s. If we choose our fixed length for the input text data to be 1014 and define a vocabulary of size 69, then every text will be turned into a matrix of size 1014×69. Now, our text data are ready to be fed into a convolutional neural network.

It should be noted that since texts, unlike images, are one-dimensional data, it makes sense to use 1D convolutions. So, instead of having 2D kernels of depth 3 (for rgb images) that slide horizontally and vertically over the pixels, we will have one dimensional kernel of depth the size of our vocabulary that slides horizontally over the characters. This is because each character has been transformed into a one-hot vector of a length equal to the size of our vocabulary. The input channel of our model must therefore be equal to the size of our vocabulary to which the depth of the kernel must correspond.

Data Availability

The datasets used during the current study are publicly available at https://drive.google.com/drive/u/0/folders/0Bz8a_Dbh9Qhbfll6bVpmNUtUcFdjYmF2SEpmZUZUcVNiMUw1TWN6RDV3a0JHT3kxLVhVR2M.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was sponsored by the National Nature Science Foundation of China (61173106). Their funding, by providing machines equipped with GPUs, made this work possible.

References

[1] S. Wang and C. D. Manning, "Baselines and bigrams: simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, ACL '12*, vol. 2, pp. 90–94, Jeju Island, Korea, July 2012.

[2] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A bayseian approach to filtering junk e-mail," in *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, Madison, WI, USA, July 1998.

[3] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Portland, OR, USA, June 2011.

[4] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008.

[5] R.-En Fan, K.-W. Chang, C.-J. Hsieh, X.R. Wang, and C.-J. Lin, "Liblinear: a library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[6] T. Joachims, "Text categorization with support vector machines: learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning ECML '98*, pp. 137–142, Chemnitz, Germany, April 1998.

[7] A. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," in *Proceedings of the Workshop on Learning for Text Categorization, AAAI'98*, pp. 41–48, Madison, WI, USA, March 1998.

[8] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," 2016, https://arxiv.org/abs/1607.01759.

[9] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, https://arxiv.org/abs/1301.3781.

[11] R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine learning (ICML 2008)*, Helsinki, Finland, July 2008.

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[13] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," 2014, https://arxiv.org/abs/1404.2188.

[14] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," in *Proceedings of the AAAI*, AAAI Press, Phoenix, AZ, USA, pp. 2741–2749, February 2016.

[15] A. Severyn and A. Moschitti, "UNITN: training deep convolutional neural network for twitter sentiment classification," in *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pp. 464–469, Denver, Colorado, June 2015.

[16] C. D. Santos and B. Zadrozny, "Learning character-level representations for part-of-speech tagging," in *Proceedings of the 31st International Conference on Machine Learning (JMLR: W&CP)*, vol. 32, Beijing, China, June 2014.

[17] C. D. Santos and V. Guimaraes, "Boosting named entity recognition with neural character embeddings," 2015, https://arxiv.org/abs/1505.05008.

[18] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, Shanghai, China, November 2014.

[19] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS'15*, MIT Press, Cambridge, MA, USA, pp. 649–657, December 2015.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, https://arxiv.org/abs/1409.1556.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, July 2016.

[22] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," 2016, https://arxiv.org/abs/1606.01781.

[23] H. T. Le, C. Cerisara, and A. Denis, "Do convolutional networks need to be deep for text classification?" in *Proceedings of the Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI Publications, New Orleans, LA, USA, February 2018.

[24] Y. Xiao and K. Cho, "Efficient character-level document classification by combining convolution and recurrent layers," 2016, https://arxiv.org/abs/1602.00367.

[25] A. Radford, R. Jozefowicz, and I. Sutskever, "Learning to generate reviews and discovering sentiment," 2017, https://arxiv.org/pdf/1704.01444.

[26] R. Socher, A. Perelygin, J. Wu et al., "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing Association for Computational Linguistics*, pp. 1631–1642, Seattle, WA, USA, 2013.

[27] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, https://arxiv.org/abs/1312.4400.

[28] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, June 2015.

[29] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," 2014, https://arxiv.org/pdf/1412.6980.pdf.

[30] D. Yogatama, C. Dyer, W. Ling, and P. Blunsom, "Generative and discriminative text classification with recurrent neural networks," 2017, https://arxiv.org/abs/1703.01898.