Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)

Combining Knowledge with Deep Convolutional Neural Networks for Short Text Classification

Jin Wang[†*], Zhongyuan Wang[‡*], Dawei Zhang[♯], Jun Yan[♯]
[†] Computer Science Department, University of California, Los Angeles.
[‡] Facebook Inc.
[♯] Microsoft Research, Beijing, China.
jinwang@cs.ucla.edu; zhy@fb.com; {dawezh, junyan}@microsoft.com

Abstract

Text classification is a fundamental task in NLP applications. Most existing work relied on either explicit or implicit text representation to address this problem. While these techniques work well for sentences, they can not easily be applied to short text because of its shortness and sparsity. In this paper, we propose a framework based on convolutional neural networks that combines explicit and implicit representations of short text for classification. We first conceptualize a short text as a set of relevant concepts using a large taxonomy knowledge base. We then obtain the embedding of short text by coalescing the words and relevant concepts on top of pre-trained word vectors. We further incorporate character level features into our model to capture fine-grained subword information. Experimental results on five commonly used datasets show that our proposed method significantly outperforms state-of-the-art methods.

1 Introduction

Text classification is a crucial technology in many applications, such as web search, ads matching, and sentiment analysis. Previous studies on text classification either rely on human designed features [Wang and Manning, 2012] or use deep neural networks on distributed representation of texts [Conneau *et al.*, 2016]. Despite the impressive advances for sentences and documents, such methods still have limitations for short texts:

- Unlike paragraphs or documents, short texts do not always observe the syntax of natural language.
- Short texts lack of contexts.
- Short texts are usually rather ambiguous because they contain polysemes and typos.

Such characteristics pose major challenges in short text classification. In order to overcome them, researchers need to capture more semantic as well as syntax information from short texts. A crucial step to reach this goal is to use more advanced text representation models. According to the different ways of leveraging external sources, previous text representation models can be divided into two categories: explicit representation and implicit representation [Wang and Wang, 2016].

- **Explicit Representation** For explicit approaches, a given text is modeled following traditional NLP steps, including chunking, labeling, and syntactic parsing. Researchers create effective features from many aspects, such as knowledge base, POS tagging and dependency parsing. Although explicit models are easily understandable by human beings, it is difficult for the machine to collect useful features for disambiguation. Besides, it also suffers from the data sparsity problem. For example, when an entity is missing in a knowledge base, we cannot obtain any feature of it and thus an explicit representation will fail to work.
- **Implicit Representation** In terms of implicit representations, the text is represented using Neural Language Model (NLM) [Bengio *et al.*, 2003]. An NLM maps texts to an implicit semantic space and parameterizes them as a vector. An implicit representation model can capture richer information from context and facilitate text understanding with the help of deep neural networks. However, implicit representations also have some disadvantages: they perform poorly on new and rare words. Besides, they ignore important relations other than co-occurrence, such as IsA and IsPropertyOf. For instance, the word "angles" in text the Angles won the World Series is the name of a baseball team. However, an implicit model cannot capture the information that Angles is a baseball team and will treat it as an animal or a new word. We need additional knowledge to fill this gap.

It is ineffective to use either explicit or implicit representations independently for short text classification. For explicit models, the syntax and semantic information in short texts is too subtle for traditional NLP approaches to capture. Sometimes implicit models do not work well either. In many real applications, rare words like proper nouns occur frequently in short texts. Moreover, as is stated in the previous work [Hua *et al.*, 2015], the IsA relation, which is missing in implicit models, is crucial for short text understanding. Despite their

[*] This work is done when the authors were in MSRA.

shortcomings, explicit and implicit representations are complementary to each other as is shown in previous work [Hu *et al.*, 2016]: explicit knowledge such as logic rules can be integrated into deep neural networks to regulate the learning process. We can combine them to transform explicit knowledge into neural models.

In this work, we propose a deep neural network that makes the fusion of explicit and implicit representations of short texts. We first enrich the information of short texts with the help of an explicit knowledge base. Specifically, we associate each short text with its relevant concepts in the knowledge base. Next we combine the words and relevant concepts of the short text to generate its embedding using a pre-trained word embedding. Then we feed this word-concept embedding into a Convolutional Neural Network (CNN) to learn explicit knowledge from the joint embedding. To the best of our knowledge, this is the first work that combines explicit and implicit representation for short text understanding.

While implicit and explicit representation models can provide rich semantic features for short text understanding, they still miss some semantic information. For example, given a short text buy apple iPhone7, as "iPhone7" is a new word, neither the knowledge base nor the pre-trained word embedding will recognize it. Inspired by the character level language model [Kim *et al.*, 2016], we combine the input word-concept embedding with a character-level embedding to obtain more semantic features. We use a separate CNN with only character embeddings as the input and concatenate its outputs to that of the main network in the fully-connected layer as the feature vector for the output layer. In this way, we can acquire more subword information such as morphemes that is missing in word-level embedding. In the above example, the word "iPhone7" will propose characteristic features similar to "iPhone". As "i-Phone" is included in knowledge base, we can capture the meaning of the new word "iPhone7" in this way.

We use a separate Convolutional Neural Network with only character embedding as the input and concatenate its outputs to those of the main network in the fully-connected layer as the feature vector for the output layer. In this way, we can acquire more subword information such as morphemes that is missing in word-level embedding.

The main contributions of this paper are summarized as follows:

- We associate relevant concepts with short texts by leveraging explicit knowledge and generating the implicit representation of the short text. To capture fine-grained semantic information, we also integrate the character level features into the joint embedding.
- We build a jointly model using Convolutional Neural Network to learn the coalesced embedding and to perform classification.
- We conduct extensive experiments on five commonly used datasets. The results show that our model significantly outperforms state-of-the-art methods.

2 Related Work

2.1 Short Text Understanding

Short Text Understanding has become a hot topic in recent years. The most crucial step for understanding short text is conceptualization. Previous studies rely on either external knowledge bases [Song *et al.*, 2011; Wang *et al.*, 2015] or lexical information [Hua *et al.*, 2015] to get the concepts associated with the short text. To understand short text, another important task is evaluating the similarity between two short texts. This problem has been solved with either explicit representation [Li *et al.*, 2013] or implicit representation [Kenter and de Rijke, 2015].

2.2 Text Classification

Traditional text classification methods rely on human-designed features. The most widely used feature is to represent text as a vector of terms, namely "Bag-of-Words". Other studies mainly focus on generating more complex features, such as POS tagging and tree kernel [Post and Bergsma, 2013]. The classifiers can be built using machine learning algorithms such as Naive Bayes and Support Vector Machine [Wang and Manning, 2012]. For short text classification task, previous studies focus on feature expanding [Shen *et al.*, 2006] by leveraging context information from search engines. However, such methods have a serious problem of data sparsity.

2.3 Neural Language Model

With the rapid development of distributed word representation and deep neural networks, many new ideas for traditional NLP tasks have been proposed. The Neural Language Model solves the data sparsity problem by representing words with dense vectors [Bengio *et al.*, 2003; Pennington *et al.*, 2014]. The embedding vectors obtained through NLMs can capture meaningful syntactic and semantic information and map semantically similar words close in the induced vector space. Recently there have been many studies on learning the representation of texts with different granulates: [Hill *et al.*, 2016] focus on learning the embedding of a phrase, while [Palangi *et al.*, 2016],[Kalchbrenner *et al.*, 2014] and [Le and Mikolov, 2014] focuses on learning the embedding of sentences. The text embedding enables us to measure the similarity between different texts by simply computing the distance between embedding vectors.

With fixed-length embedding vectors as input, using deep neural networks for various types of NLP tasks has gradually become popular. [Socher *et al.*, 2011] introduced an auto-encoder using Recurrent Neural Network to capture the syntax information of a sentence. [Socher *et al.*, 2013] proposed the Recursive Neural Tensor Network for sentiment analysis. [Collobert *et al.*, 2011] first use CNN with pre-trained word embedding for text classification. [Kim, 2014] further improves the performance by using multi-channel embedding. [Zhang *et al.*, 2015] and [Conneau *et al.*, 2016] proposed very deep neural networks with only character level information as input. While such methods work well for large documents, they perform poorly on short texts due to the limited information provided by them.

3 Model Design

In this section, we present a joint model called Knowledge Powered Convolutional Neural Network (KPCNN), using two sub-networks to extract the word-concept and character features. We first introduce the way to conceptualize short texts with the help of a knowledge base. Then we describe the model and show how to learn the features from the embeddings incorporating information from the word, concept and characters.

3.1 Short Text Conceptualization

The first step of our work is short text conceptualization. We reach this goal using an existing knowledge base, such as DBpedia, Yago, Freebase and Probase [Wu *et al.*, 2012]. We will use Probase in this paper[1]. Because compared with other knowledge bases, Probase has a much broader coverage of concepts about wordly facts. Besides, Probase contains probabilistic information with which we can quantify many measurements of short texts, such as popularity, typicality and categorization. By leveraging the large number of IsA relations in Probase, we can get a list of concepts as well as their relevance to the short text.

Here we denote the concept vector as $\mathcal{C} = \{< c_1, w_1 > , < c_2, w_2 >, ..., < c_k, w_k >\}$, where c_i is a concept in the knowledge base, and w_i is a weight to represent the relevance of the short text associated with c_i. Given a short text, we can obtain its concept vector using the conceptualization API provided by Probase. It computes the concept vector of a given short text using a novel knowledge-intensive approach [Hua *et al.*, 2015]. In Probase, the number of k is set to 10. If there are more than 10 concepts, the top-10 results will be returned. For example, given a short text "CNOOC Signed a PSC with ROC", we can get its concept vector as {<client,0.9>, <channel,0.6>, <mythological creature,0.6>, <international well-known enterprise,0.2>, <chinese oil major,0.2>}.

3.2 Overall Architecture of the Model

After we get the results of conceptualization, we can combine the knowledge of concepts with the embedding of short texts. For the word and concept embedding, we use pre-trained word embedding and keep them static. But as there is no pre-trained embeddings for the character level, we should allow the embedding of characters to be modified during the training process. In this way, the character embedding can be well-tuned while training the model. To this end, we propose a two-branch model as shown in Figure 1. It has two components: the upper sub-network is for the word and concept embedding of the short text, and the lower sub-network is for the character embedding. Both of them are CNNs. With such a model, we can learn rich features from both the word level and the character level, respectively.

The upper component consists of seven layers: one input layer, two convolution layers, two pooling layers and two hidden layers.

[1]The data is now public available as part of the Microsoft Concept Graph: https://concept.msra.cn/

Input Layer. The input layer transforms the short text into a matrix of embedding, denoted as $\mathbf{W} \in R^{(k+n)\times m}$ as the input of the network, where n and k is the the maximum number of words and concepts, respectively. And m is the dimension of word embedding. We obtain $\mathbf{W}$ by concatenating the embedding of words and concepts together: $\mathbf{W} = \mathbf{W}_w \oplus \mathbf{W}_c$.

Here $\mathbf{W}_w$ and $\mathbf{W}_c$ are the embedding of the words and concepts, respectively. And $\oplus$ is the concatenation operation. The way to construct $\mathbf{W}_w$ is rather straightforward: suppose the short text consists of n words, and $\mathbf{v}_i^w \in R^m$ is an m-dimensional vector of the i^{th} word in the short text. We can get $\mathbf{W}_w$ by simply concatenating them:

$$\mathbf{W}_w = \mathbf{v}_1^w \oplus \mathbf{v}_2^w \oplus ... \oplus \mathbf{v}_n^w \tag{1}$$

To get the representation of $\mathbf{W}_c$, we also need to consider the weight of concepts at the same time. For each embedding vector $\mathbf{v}_i^c \in R^m$ of concept c_i, we multiply it by the constant w_i to denote the weight of a given concept. Then we have:

$$\mathbf{W}_c = w_1\mathbf{v}_1^c \oplus w_2\mathbf{v}_2^c \oplus ... \oplus w_k\mathbf{v}_k^c \tag{2}$$

If the short text or concept vector is not long enough, we will use 0 as padding. We get the embeddings $\mathbf{v}_i^w$ and $\mathbf{v}_i^c$ by looking up the pre-trained word embedding.

Convolution Layer. The function of convolution layers is to extract higher level features from the input matrix. To get different kinds of features, we apply filters with different sizes. Similar to many previous works, we fix the width of each filter as m and treat the height h of it as a hyper parameter. Given a filter $\omega \in R^{h\times m}$, a feature s_i is generated from a window of words and concepts $[\mathbf{v}_i : \mathbf{v}_{i+h-1}]$ by:

$$s_i = g(\omega \cdot [\mathbf{v}_i : \mathbf{v}_{i+h-1}] + b) \tag{3}$$

Here $b \in R$ is a bias term. And g is a non-linear function. In this work we use ReLU as the non-linear function for convolution layers. The filter is applied to all possible windows of words and concepts in $\mathbf{W}$ to produce a feature map $\mathbf{s} \in R^{n+k-h+1}$. This process can be repeated for various filters with different heights to increase the feature coverage of the model.

Pooling Layer. The function of a pooling layer is to further abstract the features generated from convolution layer by aggregating the scores for each filter. In this work, we apply a max-over-time pooling operation over each feature map. The idea is to choose the highest value on each dimension of vector to capture the most important feature. With pooling layers, we can induce a fixed-length vector from feature maps.

Hidden Layer. In order to make full use of rich features obtained from the pooling layers, we use a non-linear hidden layer to combine different pooling features. We use tanh as the activation function here in our work. In this layer, we can also apply dropout [Hinton *et al.*, 2012] as a mean of regularization by randomly setting to zero a proportion of elements of the feature vector.

Similarly, the lower subnetwork also consists of seven layers: one input layer, two convolution layers, two pooling layers and two hidden layers. The input of this subnetwork is a sequence of encoded characters in the short text. The encoding is done by first generating an alphabet of all the characters

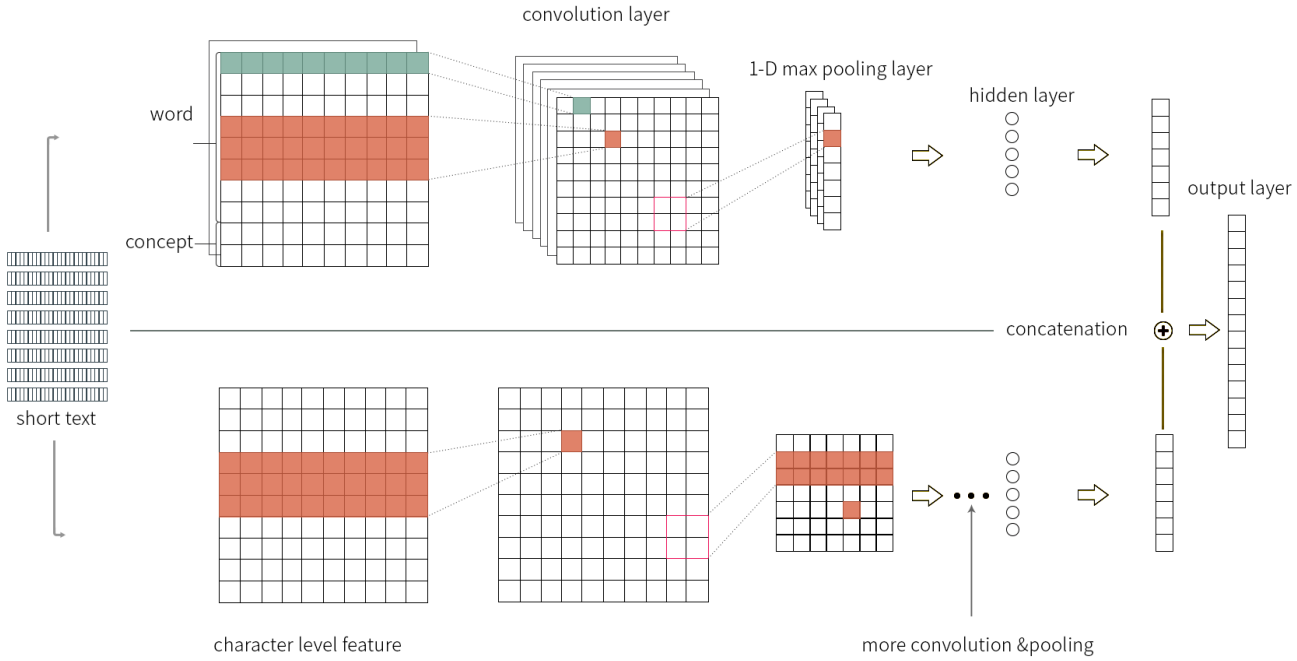


Figure 1: Architecture of the Overall Model

in the dataset and then randomly initializing the embedding of each character with m_c dimensions. Then the sequence of characters is transformed into a matrix $\mathbf{W}_c \in R^{L \times m_c}$. Here L is a hyper parameter that limits the maximum size of the sequence. Any character exceeding length L is ignored. In this work we set the value of L to be 256.

Finally, we combine the output vectors of the two subnetworks by concatenating them. Then we apply an output layer on the joint vector to convert the output numbers into probabilities for classification.

3.3 Training

We define all the parameters to be trained as a set Θ. Here we denote the set of training data as $\mathcal{X}$ and the set of class label as $\mathcal{Y}$. For each $x \in \mathcal{X}$, the network computes a score $s(y; x, \Theta)$ for each class $y \in \mathcal{Y}$. To transform the scores into a conditional probability distribution in the output layer, we use a softmax operation over the scores for all $y \in \mathcal{Y}$:

$$p(y|x, \Theta) = \frac{\exp(s(y; x, \Theta))}{\sum_{\forall \tau \in \mathcal{Y}} \exp(s(\tau; x, \Theta))} \quad (4)$$

The training target of the model is to maximize the log-likelihood over the training set with respect to Θ:

$$\Theta \mapsto \sum_{x \in \mathcal{X}} \log p(y|x, \Theta) \quad (5)$$

We use Adagrad [Duchi *et al.*, 2011] to optimize the training process. At the t^{th} epoch, the parameters are updated as:

$$\Theta_t = \Theta_{t-1} - \frac{\alpha}{\sqrt{\sum_{i=1}^{t} g_i}} g_t \quad (6)$$

where α is the learning rate and g_t is the gradient at epoch t. All the parameters are initialized from a uniform distribution, we follow many previous studies to make such settings.

Table 1: A Summary of Datasets

Datasets	#class	Training/Test set	Avg. Len
TREC	6	5952/500	10
Twitter	3	8,204/3,005	19
AG News	4	120,000/7,600	7
Bing	4	31,383/3,488	8
Movie Review	2	8,530/2,132	20

Table 2: Hyper Parameters

Parameter	Values
filter sizes	upper:[3,4,5,6] lower:[6]
dropout rate	0.5
hidden layers dimension	upper:100 lower: 50
embedding dimension	$m = 300$ $m_c = 100$
learning rate	$\alpha = 0.01$

4 Evaluation

4.1 Experiment Setup

To show the effectiveness of our model, we conduct experiments on five widely used datasets: TREC, Twitter, AG news, Bing and Movie Review. The details of each dataset are listed in Table 1.

TREC. This is a question answering dataset [2]. It involves 6 different types of questions, such as whether the question is about a location, about person or numeric information.

Twitter. This is a set of tweets with 3 kinds of sentiments: positive, neutral and negative. The labels are added

[2] http://cogcomp.cs.illinois.edu/Data/QA/QC/

Table 3: Accuracy of Composed Models on Different Datasets

Model	TREC	Twitter	AG news	Bing	Movie Review
WC + LR	52.8	57.57	61.56	74.48	60.44
BoW + SVM	85.66	56.23	72.7	80.33	77.52
CNN	89.33	57.24	86.11	96.2	81.52
CharCNN	76	44.96	78.27	85.64	77.01
WCCNN	91.21	57.87	85.57	96.22	**83.77**
KPCNN	**93.46**	**59.84**	**88.36**	**99.17**	83.25

by the original author of the dataset [3]. We have preprocessed this dataset for the ease of use in this work: We removed the spams and quotes in the tweets. For tweets written in other languages, we translate them into English using Google Translator. We also remove the # hashtags that are not located at the end of the message.

AG news. This dataset is adopted from [Zhang *et al.*, 2015]. The original data consists of both articles and descriptions of AG's corpus of news. In order to test for short texts, we remove the articles and only use the titles in our experiment.

Bing. This is a dataset of query logs adopted from [Wang *et al.*, 2014]. They are divided into 4 categories according to their contents.

Movie Review. This dataset consists of one sentence per comment on movies. Classification involves detecting positive/negative reviews [Pang and Lee, 2005]. For this dataset, we randomly split 80% as the training set and the remaining 20% as test set. In this process, we keep a balance number of items with each label in the training set.

There are several hyper parameters in our model. We set them empirically. The details are in Table 2. We use the tool word2vec [4] to train word and concept embedding. If a word is unknown, we will randomly initialize its embedding.

4.2 Baseline Methods

We compared our method with several state-of-the-art approaches: two feature-based methods and two deep learning based methods. The metric for evaluating each model is the accuracy of prediction.

Word-Concept Embedding + LR. This baseline uses the weighted word embedding as well as concept embedding to represent each short text. For the weighted word embedding $\mathbf{V}_w \in R^m$, we use the tf-idf value of each word as the weight. Given the concept vector $\mathcal{C}$, the concept embedding $\mathbf{V}_c \in R^m$ is the weighted average of embedding of each concept:

$$\mathbf{V}_c = \frac{\sum_{i=0}^{k} w_i \mathbf{v}_i^c}{\sum_{i=0}^{k} w_i} \tag{7}$$

And the overall embedding is the average of $\mathbf{V}_w$ and $\mathbf{V}_c$. Then we use Logistic Regression over such embedding to perform classification. A similar idea has been proposed in [Huang *et al.*, 2012] to capture global context for classification.

BoW + SVM. This baseline is proposed by [Wang and Manning, 2012]. The basic idea is to use the traditional SVM algorithm to build a classifier. We use the unigrams as the feature for short texts. The weight of each feature is the frequency of each unigram.

CNN. This method uses a one-layer CNN for text classification proposed by [Kim, 2014]. It uses a multi-channel architecture for text embedding. We obtain its source code from the author [5] and use its default settings for hyper parameters.

CharCNN. We also compared our work with a recently proposed method [Zhang *et al.*, 2015]. It uses a 12-layer convolutional neural network with only character level features as the input. We obtain the source code from the author [6].

WCCNN. This baseline is proposed by us. We use only the upper subnetwork of our proposed model. And the embedding is the concatenation of word embedding and concept embedding matrices.

4.3 Discussion of Results

The results on all the datasets are shown in Table 3. We can see that our model KPCNN significantly outperforms state-of-the-art methods. In all the five datasets, our proposed model outperforms the best baselines by 2% to 5% in accuracy. Even without the character level information, the WCCNN model still outperforms most state-of-the-art methods. The reason is that the neural network can effectively represent the semantic content of a short text. With the help of the concept embedding, we can acquire richer features from the short text and feed them into our model.

By comparing our model with the proposed baseline WCCNN, we can see that the character level information can help improve the performance of the model. With the help of the character features, we have improvement in accuracy on four out of five datasets.

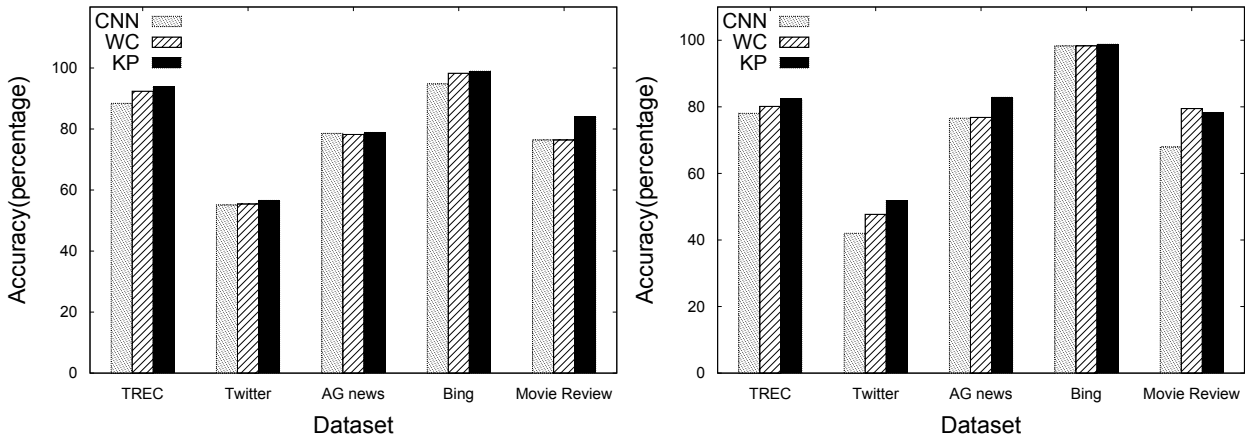
While character-level features are helpful, it is worth noting that the CharCNN model [Zhang *et al.*, 2015] does not perform well in our experiment. The reason is that due to the shortness and sparsity of short texts, CharCNN is unable to capture enough features with only character level information. Therefore, although CharCNN works well for document-level datasets, it is not effective in solving the problem of short text classification. Moreover, as CharCNN has a fixed alphabet, it fails to take many unknown characters in the

[3] https://www.cs.york.ac.uk/semeval-2013/task2/

[4] https://code.google.com/archive/p/word2vec/

[5] https://github.com/yoonkim/CNN_sentence

[6] https://github.com/zhangxiangxiao/Crepe



(a) Glove embedding (b) Word-Concept embedding

Figure 2: Results with different embeddings

Table 4: The number of unknown words in each embedding

Embedding	TREC	Twitter	AG news	Bing	Movie Review
vocabulary size	9592	26840	51299	15011	18762
Word2vec	467	5133	5096	1350	2314
Word-Concept	3891	17029	35185	9709	2302
Glove	3715	16111	34729	9660	1153

Twitter dataset. So the performance on that dataset is rather poor.

From the above results, we can also observe that compared with traditional methods, convolutional neural network approaches achieve better results. The main reason is that CNN is able to capture richer features with various filter sizes in the convolution layer and select more discriminative features from pooling layers.

Our model can be integrated with other word embedding, too. Here we perform some experiments using two different word embeddings: Word-Concept [Cheng *et al.*, 2015] and Glove [Pennington *et al.*, 2014]. In order to show the contribution of different features, we test with three methods. CNN is the method that uses only the embedding of words in short text. WC is the WCCNN model with only upper subnetwork. And KP is our KPCNN model. The results are shown in Figure 2.

We can see that the overall results using Glove and Word-Concept embedding are not as good as using word2vec embedding. The main reason is that compared with word2vec, these two embedding methods have too many unknown words. The number of unknown words in each dataset is shown in Table 4. We can see that for all the datasets, there are fewer unknown words in word2vec. In the TREC dataset, the Word-Concept embedding fails to recognize more than one third of the words. In this case, the embedding vectors of most words are initialized randomly and the quality of the short text representation will be seriously influenced.

From the results in Figure 2, we can further conclude that the conceptualization and character level information does help improve the overall performance. For the Word-Concept embedding, the accuracy of KP can be up to 12% higher than that of CNN. This is because Word-Concept makes use of additional knowledge source to learn contextual word representation. And the implicit representation of words involves a combination of its intrinsic vector and the most context-appropriate concept vector. Therefore, the KP model can make full use of the joint representation of short text and achieve better accuracy than the CNN model.

Finally, we can see from Figure 2 that KPCNN can have up to 7% accuracy gaining than the other two models. This is because word-level embedding can not handle out-of-vocabulary words. This phenomenon is more significant as the number of unknown words in Glove and Word-Concept is much larger. With the help of character level information, we can easily address this problem.

5 Conclusion

In this paper, we propose a novel model that takes advantage of both explicit and implicit representations for short text classification. We enrich the features of short texts by conceptualizing them with the help of a well known knowledge base. We combine the associated concepts with the words to generate the embedding of short text. We also utilize the character level information to enhance the embedding of short text. With such embedding as the input, we build a joint model on the basis of the CNN to perform classification. Experiments on real data show that our method achieves significant improvement over state-of-the-art methods for short text classification.

References

[Bengio *et al.*, 2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural proba-

bilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[Cheng *et al.*, 2015] Jianpeng Cheng, Zhongyuan Wang, Ji-Rong Wen, Jun Yan, and Zheng Chen. Contextual text understanding in distributional semantic space. In *CIKM*, pages 133–142, 2015.

[Collobert *et al.*, 2011] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[Conneau *et al.*, 2016] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016.

[Duchi *et al.*, 2011] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[Hill *et al.*, 2016] Felix Hill, KyungHyun Cho, Anna Korhonen, and Yoshua Bengio. Learning to understand phrases by embedding the dictionary. *TACL*, 4:17–30, 2016.

[Hinton *et al.*, 2012] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[Hu *et al.*, 2016] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *ACL*, 2016.

[Hua *et al.*, 2015] Wen Hua, Zhongyuan Wang, Haixun Wang, Kai Zheng, and Xiaofang Zhou. Short text understanding through lexical-semantic analysis. In *ICDE*, pages 495–506, 2015.

[Huang *et al.*, 2012] Eric H. Huang, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Improving word representations via global context and multiple word prototypes. In *ACL*, pages 873–882, 2012.

[Kalchbrenner *et al.*, 2014] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *ACL*, pages 655–665, 2014.

[Kenter and de Rijke, 2015] Tom Kenter and Maarten de Rijke. Short text similarity with word embeddings. In *CIKM*, pages 1411–1420, 2015.

[Kim *et al.*, 2016] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[Kim, 2014] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751, 2014.

[Le and Mikolov, 2014] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, pages 1188–1196, 2014.

[Li *et al.*, 2013] Pei-Pei Li, Haixun Wang, Kenny Q. Zhu, Zhongyuan Wang, and Xindong Wu. Computing term similarity by large probabilistic isa knowledge. In *CIKM*, pages 1401–1410, 2013.

[Palangi *et al.*, 2016] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab K. Ward. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Trans. Audio, Speech & Language Processing*, 24(4):694–707, 2016.

[Pang and Lee, 2005] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*, 2005.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[Post and Bergsma, 2013] Matt Post and Shane Bergsma. Explicit and implicit syntactic features for text classification. In *ACL*, pages 866–872, 2013.

[Shen *et al.*, 2006] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Query enrichment for web-query classification. *ACM Trans. Inf. Syst.*, 24(3):320–352, 2006.

[Socher *et al.*, 2011] Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*, pages 801–809, 2011.

[Socher *et al.*, 2013] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642. Citeseer, 2013.

[Song *et al.*, 2011] Yangqiu Song, Haixun Wang, Zhongyuan Wang, Hongsong Li, and Weizhu Chen. Short text conceptualization using a probabilistic knowledgebase. In *IJCAI*, pages 2330–2336, 2011.

[Wang and Manning, 2012] Sida I. Wang and Christopher D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL*, pages 90–94, 2012.

[Wang and Wang, 2016] Zhongyuan Wang and Haixun Wang. Understanding short texts(tutorial). In *ACL*, 2016.

[Wang *et al.*, 2014] Fang Wang, Zhongyuan Wang, Zhoujun Li, and Ji-Rong Wen. Concept-based short text classification and ranking. In *CIKM*, pages 1069–1078, 2014.

[Wang *et al.*, 2015] Zhongyuan Wang, Kejun Zhao, Haixun Wang, Xiaofeng Meng, and Ji-Rong Wen. Query understanding through knowledge-based conceptualization. In *IJCAI*, pages 3264–3270, 2015.

[Wu *et al.*, 2012] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Qili Zhu. Probase: a probabilistic taxonomy for text understanding. In *SIGMOD*, pages 481–492, 2012.

[Zhang *et al.*, 2015] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NIPS*, pages 649–657, 2015.