Improving Short Text Classification Using Fast Semantic Expansion on Multichannel Convolutional Neural Network

Natthapat Sotthisopha[1], Peerapon Vateekul[2]
Chulalongkorn University Big Data Analytics and IoT Center (CUBIC)
Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University, Bangkok, Thailand
[1]natthapat.so@student.chula.ac.th, [2]peerapon.v@chula.ac.th

***Abstract*— Nowadays, text classification is recognized as one of crucial tools for business users to gain more insights from customers. However, textual data from customers such as comments are usually short. Thus, there are two main issues in short text categorization: (1) insufficient contextual information and (2) noisy data due to misspellings. Recently, there is a prior attempt to propose a deep learning approach for short text categorization by expanding semantic using word embeddings clustering via an algorithm based on density peaks searching. Since the number of words in word embeddings is usually large, the clustering algorithm does not scale with the size of data set, thus demanding unacceptable computational cost. In this paper, we aim to propose a fast short-text categorization framework. Rather than using a CNN with k-max pooling layer, we propose to use a faster version of Convolutional Neural Network (CNN) called "multichannel CNN." To speed up the semantic expansion process, we propose to employ mini batch K-Means++, which is considerably faster and scales well with the size of data set. Furthermore, we also introduce an additional preprocessing step to increase vocabulary coverage rate on word embeddings. We conducted experiments on four public data sets: Google Snippets, TREC, MR, and Subj. The results showed that the proposed framework does not only improve an accuracy on all data sets, but also reduces computational costs from several days to a few hours.**

Keywords— Short Texts; Natural Language Processing; Text Classification; Clustering; Convolutional Neural Network; Word Embeddings; Deep Learning

I. INTRODUCTION

As global internet usage has been tremendously increasing over the recent years due to the emergence of social networking services, users can provide feedbacks and product reviews more easily. However, most of them are usually short texts. Therefore, classifying them are an important tool for business users to gain insight into user intents, question answering and intelligent information retrieval [1]. However, representing short texts with bag-of-words (BoW) model will suffer from the sparsity of data issue and cannot capture sufficient contextual information needed for classification [2]. Consequently, classification performance on conventional models relying on BoW representations will be hindered because BoW model ignores the order of words which in turn discards the semantic relations between words. On the other hand, utilizing word embeddings for short text classification might help enrich semantic relations, but it still suffers from the insufficient contextual information due to the brevity of texts and misspellings in corpuses. Generating effective representations of short texts in order to improve classification performance has been one of the active research issues [2, 3].

Short text classification methods in the past usually involve expanding short texts using latent semantics learned by latent Dirichlet allocation (LDA) and its extension to discover topics using knowledge from external resources, because it can help enrich text representation for the classification model. Phan et al. [3] presented a framework to expand short texts by appending topic names discovered using LDA over Wikipedia. Yan et al. [4] proposed a variant of LDA called bi-term topic model (BTM) to alleviate the data sparsity issue.

Natural language processing (NLP) has seen many advances through application of deep learning based methods because of their less need for feature engineering. One on the notable contributions from them is word embeddings. They can significantly improve performance of models by representing words as dense vectors that efficiently represent semantic relationship information between words in the corpus they are trained on.

Wang et al. [5] proposed a framework to classify short texts by expanding semantic and feeding them along with the texts to CNN. The semantic expansion process involves word embeddings clustering and expanding semantic of texts by detecting semantic units. However, there is a bottleneck in computational time in the framework. The clustering algorithm used in [5] requires unacceptable computational time and resource. It also requires manual thresholds settings to discover centroids in the data. Also, the vocabulary coverage rate on data sets over pre-trained word embeddings can be increased by performing data preprocessing on corpuses. Word embeddings could be clustered by another algorithm that scales well with the size of data such as mini batch K-Means++ algorithm [6, 7].

In this paper, we propose a framework for short text categorization with three improvements in which two of them can help reduce computational time and resource required. First, we cluster word embeddings using mini batch K-Means++ algorithm which can scale well to the size of data set. Second, we introduce an additional data preprocessing step to increase vocabulary coverage rate on word embeddings by matching

978-1-5386-5889-5/18/$31.00 ©2018 IEEE
SNPD 2018, June 27-29, 2018, Busan, Korea

uncovered words with the most similar words using Jaro-Winkler similarity [8]. Finally, we employ multichannel CNN that utilizes both original short texts and expanded semantic information to classify them. Thanks to these improvements, our proposed framework uses considerably less computational power and resource while manage to improve classification results.

The rest of this paper is organized as follows. In Section 2, the related works regarding text classification are reviewed. Section 3 presents the model with the proposed alternative procedures. Section 4 explains the setup of experiments and how they are conducted. Section 5 shows experiment results and the conclusion was drawn in section 6.

II. Related Works

Many widely accepted solutions to overcome the data sparsity concern has been proposed. Chen et al. [2] showed that learning multi-granularity topics can improve short text modeling. Phan et al. [3] proposed a method that relies on external Wikipedia corpus to discover hidden topics with LDA and expand short texts. Zhou et al. [9] also made use of Wikipedia to capture semantic information and improve the question similarity in concept space.

In recent years, language modeling methods have been relying on artificial neural networks (ANNs) to learn word embeddings and they have shown promising results. Mikolov et al. [10] introduced an efficient method for learning word embeddings from large corpus using the continuous Skip-gram model. Pennington et al. [11] also proposed an unsupervised learning algorithm for obtaining word vector representations called GloVe, for Global Vector, by incorporating global corpus statistics into the model.

Neural networks can also be used to learn sentence-representation of texts for classification tasks. Mikolov [12] introduced the paragraph vector which is a fixed-size vector that encodes feature representation for variable length documents. Kalchbrenner et al. [13] proposed the dynamic convolutional neural network (DCNN) for sentence modeling. It utilized k-max pooling to capture global features. Kim [14] also proposed a simple improvement that allow two channels of input word embeddings to be fed into CNN. One is kept static and the other is fine-tuned during the training phase. Zhang and Wallace [15] did a sensitivity analysis on the model proposed in [14] and give practical advices to get tune the model. We rely on this guideline to set up a baseline configuration of the multichannel CNN employed in this work.

Sang et al. [16] proposed a framework that extends features by appending similar words in pre-trained word embeddings to the original short texts and categorizes them using a k-nearest neighbors based model. Wang et al. [5] proposed a novel framework for short text classification. This is the most related work to our study. It performs semantic expansion using word embeddings clustering and CNN. It consists of four main steps. First, they employed a clustering algorithm based on density peak searching [17] to discover semantic cliques amongst word embeddings. Second, they computed multi-scale semantic units from word embeddings projected from short texts through component-wise additive composition over moving context windows with variable width as depicted in Fig. 1. Then, they obtained the nearest word embedding for each semantic unit by picking the nearest word embedding in the same semantic clique yielding least Euclidean distance between them and create expanded matrices for each short text and each window width. Finally, they trained a CNN with one convolutional layer followed by a k-max pooling layer. These matrices are fed into the model. However, there is a bottleneck in computational time in the framework. The clustering algorithm takes unacceptable computational cost and needs manual centroid selections by user. This study aims to alleviate the bottleneck in word embedding clustering by using mini batch K-Means++. We also introduce an additional preprocessing step to increase vocabulary coverage rate on pre-trained word embeddings as vocabulary in the corpus and word embeddings need to exactly match so that they can be utilized in the model.

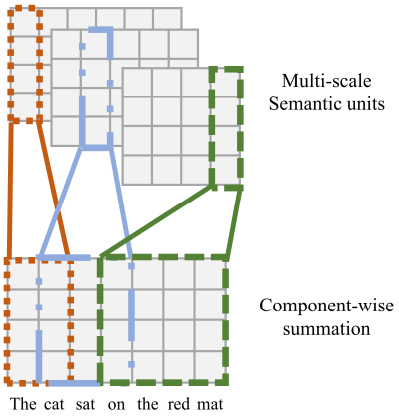

Fig. 1. Semantic unit detection over moving windows with variable widths

III. Proposed Framework

As shown in Fig. 2, the framework consists of four main components: vocabulary coverage rate improvement, word embedding clustering, expanded matrices creation, and short text classification. The pre-trained word embeddings are used as input of this framework. The output of this framework is a CNN model for classifying short texts. The details of each main component are described as follows:

A. Vocabulary Coverage Rate Improvement

To utilize word embeddings in the framework and model, each word in the corpus has to exactly match one in pre-trained word embeddings. Therefore, this step is proposed so that more word embeddings are recognized and can be used throughout the framework. It can also be viewed as an automatic method of correcting misspellings and typos.

Such task can be done by applying Jaro-Winkler similarity which is a variant of the Jaro distance metric defined as follows:

$$\mathrm{sim}_w = sim_j + (\ell p(1 - sim_j)) \quad (1)$$

where ℓ is the length of common prefix at the start of the string up to a maximum of 4 characters, p is a scaling factor for how much the score is adjusted upwards for having common

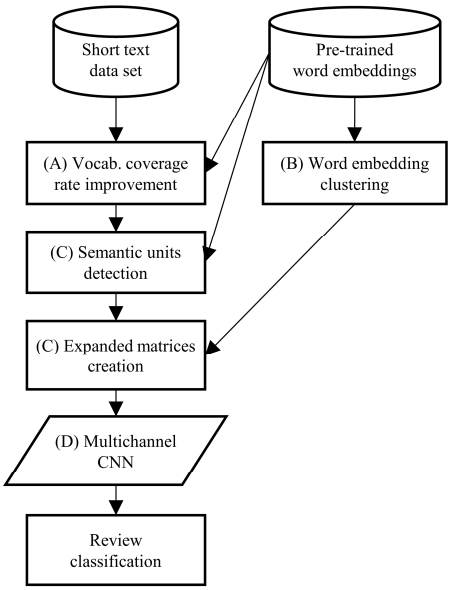

Fig. 2. Overview of our proposed framework

prefixes, and sim_j is defined as follows:

$$\mathrm{sim}_j = \begin{cases} 0 & if\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & otherwise \end{cases} \quad (2)$$

where $|s_i|$ is the length of string s_i, s_1 and s_2 are strings to be compared, m, as defined in (3), is the number of matching characters regardless of sequence order, and t is half the number of transpositions needed for matching characters between two strings. The score is normalized to between 0 and 1.

$$\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1 \quad (3)$$

Uncovered words with a length exceeding a threshold of minimum characters are selected and matched against vocabulary in word embeddings. The word that yields maximum Jaro-Winkler similarity and exceeds a minimum similarity threshold for each uncovered word is discovered and replace the uncovered words in data set.

Table 1 shows examples of closest words discovered for each uncovered word when matched against vocabulary in GloVe word embeddings. This process is mainly for automatically fixing misspellings in data set.

TABLE I. EXAMPLES OF CLOSEST WORDS DISCOVERED FOR EACH UNCOVERED WORDS.

Data set	Uncovered word	Closest word in word embeddings	Jaro-Winkler similarity
Google Snippets	podcats	podcast	0.9714
TREC	respones	response	0.9625

B. Word Embedding Clustering

Due to the number of word embeddings is usually very large, limiting the numbers of word embeddings to be used for discovering nearest word embeddings in the next step can noticeably reduce overall computational cost. As semantically related word embeddings tend to be close to each other, semantic centroids can be discovered with clustering algorithms. We propose to employ K-Means++ and mini batch K-Means algorithms which can scale well with the size of data set due to its application of random batch sampling and learning rate. The algorithm is shown in Fig. 3 and Fig. 4 respectively.

With these clusters, the number of word embeddings needed for discovering nearest word embedding of each semantic unit is limited to only word embeddings that are semantically related to each semantic unit.

Given: k, data set X
Initialize set C to store the first centroid c_1 picked uniformly at random from X.
Let D(x) denote the shortest distance from a data point to the closest center already chosen.
for i ← 2 to k do
  c_i ← choosing x ∈ X with probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$

Fig. 3. K-Means++ algorithm

Given: k, mini-batch size b, data set X
Let f(C, x) denote the nearest centroid to x.
Use initial centroids C picked with K-Means++ algorithm.
v ← 0
for i ← 1 to t do
  M ← b examples picked randomly from X
  for x ∈ M do d[x] ← f(C,x) end
  for x ∈ M do
    c ← d[x] // Get cached center for x
    v[c] ← v[c] + 1 // Update per-center counts
    η ← $\frac{1}{v[c]}$ // Get per-center learning rate
    c ← (1 − η)c + ηx // Take gradient step to update center
  end
end

Fig. 4. Mini batch K-Means algorithm

C. Semantic Unit Detection and Expanded Matrices Creation

Semantic units are useful for capturing important local semantic information from short text. As shown in Fig. 1, they are detected by performing component-wise summation over word embeddings being convolved by moving context windows with varying width under an observation that meaningful phrases can appear at any position in a short text. Therefore, sentence-wide semantic units containing useful information are discovered.

In order to detect semantic units, a moving context window $\mathrm{W}_{win} \in R^{m \times d}$, where d is the dimensionality of word embeddings and m is window width, is used to convolve with the projected matrix $\mathrm{PM} \in R^{N \times d}$ obtained from a short text S with N tokens over word embeddings. Its weights are fixed to one. This operation is defined as follows:

$$[seu_1,\ seu_2, \ldots,\ seu_{l-m+1}] = PM \otimes W_{win} \quad (4)$$

where $seu_i \in R^d$ is the i[th] semantic unit, l is the length of input short text, and seu_i is defined as follows:

$$seu_i = \sum_{j=1}^{|PM^{win,i}|} PM_j^{win,i} \quad (5)$$

$PM_j^{win,i}$ is the j[th] row from the sub-matrix $PM^{win,i}$ which is windowed on the projected matrix PM by W_{win} with the i[th] time striding.

Finally, expanded matrices are created for each corresponding short text as depicted in Fig. 5. As semantically related words are often close to each other and form clusters in vector spaces, meaningful semantic units should have at least one close neighboring word embedding that can represent it. This is done by predicting word embeddings cluster of semantic units detected in each short text, and then the nearest word embedding within the same cluster yielding the highest cosine similarity exceeding a minimum similarity threshold is selected and used to form expand matrices EMs for each context window width.

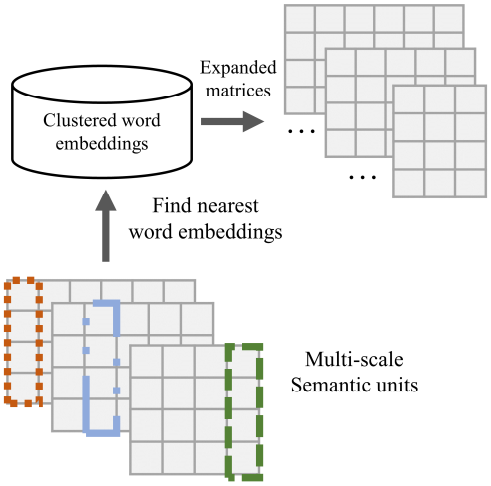

Fig. 5. Creation of expanded matrices from semantic units

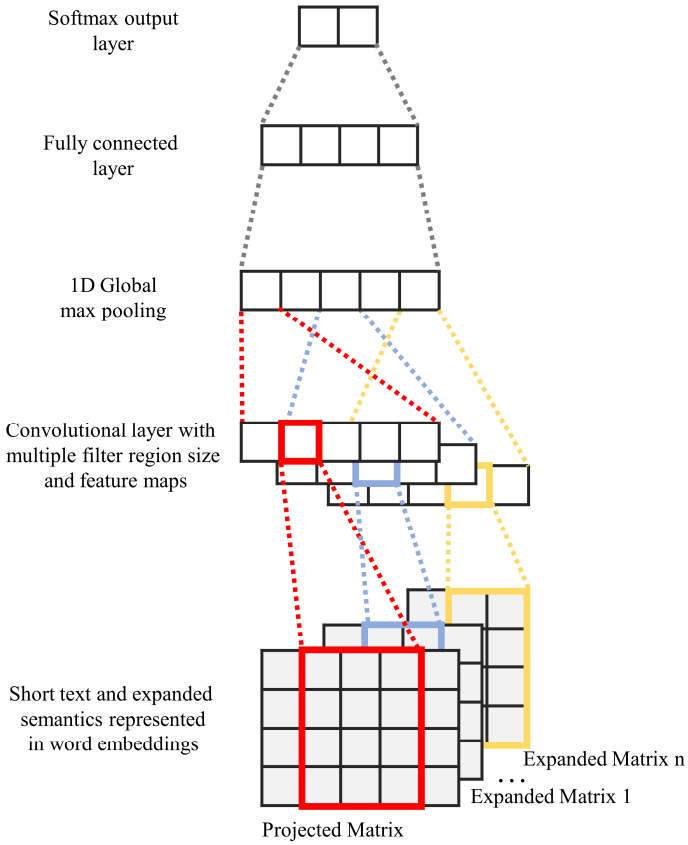

Fig. 6. Architecture of the multichannel CNN

D. *Multichannel CNN*

The multichannel CNN adapted from [14] is used to extract local features and classify short texts. Its architecture is shown in Fig. 6. PM and EMs from the last process are fed into the model in parallel. They are denoted as A matrix in (6). The dimensionality of both types of matrices is $N \times d$. We use $A[i{:}j]$ to denote the sub-matrix of or an EM from row i to j. This convolutional layer will produce output sequence $o \in R^{s-h+1}$ by applying the filter on sub-matrices of A described in (6).

$$o_i = f \cdot A[i{:}i+w-1] \tag{6}$$

where i $\in [1, N-w+1]$. Then, we calculate feature map $c \in R^{N-W+1}$ for this filter by adding a bias term b to output sequence and applying an activation function f as follows:

$$c_i = f(o_i + b) \tag{7}$$
$$c = [c_1, c_2, \dots, c_{N-w+1}] \tag{8}$$

Multiple filters for the same region width may be used so that complementary features can be learned from the same regions. Moreover, filters with variable region sizes may also be specified to learn features from n-grams. Then, as suggested in the guideline [15], one-dimensional global max pooling [18] is applied to each feature map and concatenated into a fixed-length feature vector $\hat{f}$. It is fed to a fully-connected layer with dropout and a softmax layer with weights W to predict the final class. In the last layer, a linear transformation is first performed on a short text s_i as follows:

$$\varphi(s_i, W) = W\hat{f} \tag{9}$$

Then, the output from (9) which is a vector with dimension of $|C|$, where C is labels of the data set. Each element of the vector can be interpreted as a possibility score for each corresponding label. Next, a softmax function in (10) is applied on the output vector from (9) to transform the score vector into a distribution of probability of each label. Finally, the class c_j with maximum $p(c_j|s_i, W)$ is selected as the predicted label for short text s_i as shown in (11).

$$p(c_j|s_i, W) = \frac{\exp(\varphi_j(s_i, W))}{\sum_{j=1}^{|C|} \exp(\varphi_j(s_i, W))} \tag{10}$$
$$\hat{c}_j = \underset{c_j}{\operatorname{argmax}}\, p(c_j|s_i, W) \tag{11}$$

This CNN is trained with the objective to minimize the cross-entropy of the predicted and actual distributions for all training samples with the categorical cross-entropy function. In the training phase, the set of parameters $\theta = [k, W]$, where k is the kernel weights of the filters in convolutional layer, are optimized. We also employ L2 regularization term to help prevent overfitting problem over the parameter set θ and back propagation is performed so that the parameter set θ is updated with Adaptive Moment Estimation (Adam) algorithm [19].

IV. Experimental Setup

A. Data sets

The proposed framework is tested on four data sets: (1) Google Snippets [3]: Search snippets data set, (2) TREC [20]: Question classification data set, (3) MR [21]: Sentence polarity data set, (4) Subj [21]: Subjectivity data set. Their summary statistics are shown in Table 2.

TABLE II. Summary statistics of the experimental data sets. CV refers to "10-fold cross-validation."

Data sets	# of classes	Avg. length	Train size	Test size	Vocab. Size
Google Snippets	8	18	10,060	2,280	30,643
TREC	6	10	5,452	500	8,981
MR	2	19	10,000	CV	18,544
Subj	2	22	10,662	CV	21,056

The minimum Jaro-Winkler similarity threshold for improving vocabulary coverage rate is 0.94 when the minimum length of uncovered words to be processed is 4 characters. Expanded matrices for each data sets are obtained by gathering nearest word embeddings yielding the highest cosine similarity more than a minimum threshold of 0.75.

B. Hyperparameters and Training

We describe the hyperparameters used in multichannel CNN in Table 3 which is based on the guideline in [15]. GloVe word embeddings containing 400,000 words are used in the framework. Out of vocabulary words in original short texts are randomly initialized from a uniform distribution of [-0.1,0.1]. They will be fine-tuned during training. Multiple expanded matrices may be fed into the model along with the projected matrix. For example, if n expanded matrices are fed along with the projected matrix, these expanded matrices are obtained from context windows with widths ranging from 2 to $n + 1$.

TABLE III. Hyperparameters of multichannel CNN per Channel.

Description	Values
Input word embeddings	100-dimension GloVe vectors
Filter region sizes	3, 4, 5
Feature maps	200
Activation function	Rectified Linear Unit
Pooling strategy	1D global max pooling
Dropout rate	0.5
l2 norm constraint	3

C. Baseline Methods

There are three popular methods as baselines as follows:

- TF-IDF+SVM: Texts are represented in the form of term frequency-inverse document frequency (TF-IDF) and classified by linear support vector machine (SVM)
- LSTM: Each word in texts are matched with word embeddings and fed to a single LSTM layer [22] followed by an average pooling and a softmax layer.
- Non-static CNN: This model in [14] is a special case of Multichannel-CNN. It only takes matrix of word embeddings as input. Word embeddings are fine-tuned during training.

V. Experiment Results

A. Improvement in Vocabulary Coverage Rate

This experiment shows the effect of the improvement in vocabulary coverage rate performed by the proposed data pre-processing step. Uncovered words containing more than four characters are selected. Then, each of them is used to search for the most likely similar word in word embeddings which yield the highest Jaro-Winkler similarity exceeding a minimum threshold of 0.94.

Table 4 shows changes in vocabulary coverage rate on 4 data sets. After data pre-processing, coverage rates on every data set unanimously increase by an average of 3.34%. Only Google Snippets has low coverage rate, and it benefits the most from this process by an increment in coverage rate of 5.52%.

TABLE IV. Comparison of vocabulary coverage rate on word embeddings before and after data preprocessing (%).

Data sets	Before	After	% change
Google Snippets	67.81	73.33	+5.52
TREC	95.68	97.08	+1.40
MR	95.44	98.95	+3.51
Subj	95.50	98.43	+2.93
Average	**88.61**	**91.95**	**+3.34**

The results in Table 5 show that this process can improve accuracy most of the time in every data set. For example, non-static CNN yields 87.24% test accuracy after data pre-processing which is 0.31% more. However, the effect is more subtle on data sets with already very high coverage rate before preprocessing. Note that accuracy from LSTM model on Subj after data preprocessing slightly decrease due to noises introduced by this process. Further adjustments in relevant thresholds for this process may help alleviate this issue.

TABLE V. Comparison of classification accuracy before and after data preprocessing (%). Boldface performs better.

	Google Snippets		TREC		MR		Subj	
Methods	***Bef.***	***Aft.***	***Bef.***	***Aft.***	***Bef.***	***Aft.***	***Bef.***	***Aft.***
TFIDF+SVM	69.38	**69.52**	85.60	85.60	74.13	**74.18**	86.27	**86.63**
LSTM	85.04	**85.88**	91.00	**91.40**	77.21	**77.30**	**92.89**	92.85
Non-static CNN	86.93	**87.24**	92.40	92.40	78.98	**79.36**	93.00	**93.10**

B. Classification Performance of Multichannel CNN

We evaluate classification accuracy of our framework by varying number of expanded matrices fed to CNN from 0 to 3.

Table 6 shows that our proposed framework outperforms baseline methods in term of accuracy on every data set. It achieves 87.72%, 93.40%, 79.63%, and 93.40% test accuracies on Google Snippets, TREC, MR, and Subj respectively. However, the best performance of our framework on each data set varies with the number of expanded matrices utilized. For instance, our framework yields the best test accuracy on TREC when 3 expanded matrices are fed into multichannel CNN. Note that accuracy may drop when windows with too large or too small size are used because they may generate noise or lose important information needed for phrase disambiguation.

TABLE VI. CLASSIFICATION ACCURACY OF OUR PROPOSED FRAMEWORK AGAINST BASELINE METHODS (%). BOLDFACE IS THE WINNER.

Data sets	TF-IDF+ SVM	LSTM	Multichannel CNN (Ours)			
			PM	*PM + 1 EM*	*PM + 2 EMs*	*PM + 3 EMs*
Google Snippets	69.52	85.88	87.24	87.41	**87.72**	86.00
TREC	85.60	91.40	92.20	92.80	92.60	**93.40**
MR	74.18	77.30	79.36	79.41	**79.63**	79.42
Subj	86.36	92.85	93.10	93.38	93.37	**93.40**

PM: Projected Matrix comprising of word embeddings of original short texts,
EM: Expanded Matrix comprising of nearest word embeddings of semantic units.

C. Discussion on Computational Time

We compare computational times taken in word embedding clustering and model training time per epoch in the framework in [5] and ours as shown in Table 7. The measurements are done on a machine running on AMD Ryzen 5 1600x and Nvidia GTX 1080. First, we compare the time taken to cluster 100-dimension GloVe word embeddings using density peak clustering and mini batch K-means++. We implement the algorithm that performs exhaustive calculation parallelly because caching distances between word embeddings in memory is infeasible. It took almost 5 days to complete with 8 processes, while mini batch K-means++ only takes around 4 minutes. Most of the time was spent on processing intermediary values needed for centroid discovery.

Finally, we implement the CCNN model proposed by [5] using PyTorch [23] in order to measure and compare the number of trainable parameters and training time per epoch with our multichannel CNN. The training times per epoch reported in Table 7 is averaged over 30 epochs when 3 EMs are fed into both model along with PM. CCNN has 112 thousand trainable parameters (when k in k-max pooling layer is 5 and 50 feature maps are produced in each input matrix) as opposed to 9.5 million parameters in our model. The dramatic difference in number of trainable parameters is because CCNN keeps its word embeddings static while our model fine-tunes them during training. However, our model takes only 10 seconds to train each epoch which is approximately 41.8% faster than CCNN thanks to the utilization of One-dimensional convolutional layer and One-dimensional global max pooling which are considerably simpler than the corresponding ones employed in CCNN.

TABLE VII. COMPARISON OF COMPUTATIONAL TIME TAKEN IN PROCESSES IDENTIFIED TO BE BOTTLENECKS.

Process to compare	CCNN [5]	Our framework
Word embeddings clustering	5 days	4 minutes
Average model training time per epoch	17.17 seconds	10.0 seconds

VI. CONCLUSION

In this paper, we propose a fast short-text categorization framework with three improvements. Two of them are proposed to reduce computational time and resource, and the other is an additional preprocessing step to increase vocabulary coverage rate on word embeddings and improve accuracy. We show that our framework takes less computational time and still outperforms the baseline methods. With the application of this framework, semantic expansion process for short text categorization can be performed noticeably faster.

For future works, introducing Recurrent Neural Network and attention mechanism into classification model may further improve classification accuracy.

REFERENCES

[1] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas, "Short Text Classification in Twitter to Improve Information Filtering," *Proc. 33rd Int. ACM SIGIR Conf. Res. Dev. Inf. Retr. SE - SIGIR '10*, no. July, pp. 841–842, 2010.

[2] M. Chen, X. Jin, and D. Shen, "Short text classification improved by learning multi-granularity topics," in *IJCAI*, 2011, pp. 1776–1781.

[3] X.-H. Phan, L.-M. Nguyen, and S. Horiguchi, "Learning to classify short and sparse text & web with hidden topics from large-scale data collections," in *Proceeding of the 17th international conference on World Wide Web - WWW '08*, 2008, p. 91.

[4] X. Yan, J. Guo, Y. Lan, and X. Cheng, "A biterm topic model for short texts," *WWW '13 Proc. 22nd Int. Conf. World Wide Web*, pp. 1445–1456, 2013.

[5] P. Wang, B. Xu, J. Xu, G. Tian, C. L. Liu, and H. Hao, "Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification," *Neurocomputing*, vol. 174, pp. 806–814, 2016.

[6] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1025.

[7] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web - WWW '10*, 2010, p. 1177.

[8] W. E. Winkler, "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.," 1990.

[9] G. Zhou, Y. Liu, F. Liu, D. Zeng, and J. Zhao, "Improving question retrieval in community question answering using world knowledge," in *IJCAI*, 2013, pp. 2239–2245.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 1–9.

[11] J. Pennington, R. Socher, and C. Manning, "Glove: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[12] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *ICML*, 2014, pp. 1188–1196.

[13] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," in *Proceedings of the 52nd Annual Meeting of the ACL (Volume 1: Long Papers)*, 2014, pp. 655–665.

[14] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv Prepr. arXiv1408.5882*, 2014.

[15] Y. Zhang and B. Wallace, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification," pp. 253–263, 2015.

[16] L. Sang, F. Xie, X. Liu, and X. Wu, "WEFEST : Word Embedding Feature Extension for Short Text Classification," 2016.

[17] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science (80-.).*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[18] Y.-L. Boureau, J. Ponce, and Y. Lecun, "A Theoretical Analysis of Feature Pooling in Visual Recognition," *Proc. 27th Int. Conf. Mach. Learn.*, pp. 111–118, 2010.

[19] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization," *Int. Conf. Learn. Represent. 2015*, pp. 1–15, 2015.

[20] X. Li and D. Roth, "Learning question classifiers: The role of semantic information," *Nat. Lang. Eng.*, vol. 12, no. 3, pp. 229–249, 2006.

[21] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd annual meeting on ACL*, 2005, pp. 115–124.

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] A. Paszke *et al.*, "Automatic differentiation in PyTorch," 2017.