REGION EMBEDDING VIA LOCAL CONTEXT KERNEL FOR TEXT CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper proposed two novel text classification model architectures with effective extracting of local predictive structures which can be represented as low-dimensional vector for small text regions, i.e., region embeddings. To produce the region embeddings we proposed the local context kernels, which are word specific parameter matrices. The local context kernels can be learned to capture the specific relationships between words and their local context. With the kernels, we proposed two methods to produce the region embeddings from two different perspectives. For text classification, we simply sum the region embeddings to represent the text, and use an upper fully connected layer to achieve the classification task.The local context kernels and word embeddings are trained as part of the models. Experimental results show that the proposed methods can achieve new state of art results on several benchmark text classification tasks with identical hyperparameters setting. We provide visualizations and analysis showing that the learned kernels can capture syntactic and semantic resemblance between words.

1 INTRODUCTION

Text classification is an important task in natural language processing, which has been studied for years with applications such as topic categorization, web search, sentiment classification. A simple yet effective approach for text classification is to represent documents as bag-of-words, and train a linear classifier, e.g., a logistic regression, a support vector machines (Joachims, 1998; Fan et al., 2008) or a naive bayes (McCallum et al., 1998). Although bag-of-words methods are efficient and popular, they lose the local ordering information of words which has been proved useful particularly on sentiment classification (Pang et al., 2002).

To benefit from local word order for text classification, n-grams which is a simple statistics of ordered word combinations usually performs good (Pang et al., 2002; Wang & Manning, 2012; Joulin et al., 2016). With a n-gram vocabulary, Wang & Manning (2012) simply use one-hot vectors as discrete embedding of n-grams. Recently, FastText (Joulin et al., 2016) directly learns distributed embeddings of n-grams and words for the task of interest. In FastText, learning the embedding of n-gram can be regarded as learning low-dimensional vector representation of a small fixed size region, i.e, region embedding. Although the n-grams was widely used, two caveats come with n-grams: 1) the vocabulary size increases exponentially with n, and yielding a high-dimensional feature space, which makes it difficult to apply large region size(e.g., $n > 4$). 2) A look up layer with an unique id is used to generate representation for each n-gram, thus the data sparsity problem cannot be avoided.

In this article, we focus on learning task specific region embedding for text classification, while Johnson & Zhang (2015) learn the task independent region embedding from unlabeled data. Instead of using handcrafted n-grams features, we directly learn the region embedding based on word sequence inputs, which solves the data sparsity problem. An additional advantage is that our model is flexible for region sizes, which makes is easily to learn the embedding of large size region. (Intuitively, given a word sequence in a fixed size region, both words and words' relative position are the key informations to generate the semantic of the region. ~~Noticed that semantics of regions are composed of words in each region, nevertheless each word has a specific influence to its local context, which means semantics of regions are composed differently, we proposed the word specific local context kernels to capture the influences from words to their local context.In our approach, each word has an additional local context kernel besides the original word embedding~~). To utilize the key infor-

mations, we proposed an additional local context kernel for each word besides word embeddings. The local context kernel of a word is a parameter matrix whose columns can be learned to address the different influence to the context words in different relative position. With local context kernels we can implicitly learn the region embeddings by ordered words inputs. Benefited from the density of word embedding and the proposed context kernel, our method can apply larger region size and better generalization than n-grams.

The convolutional networks(CNN) were proposed to make use of internal structure (word order) implicitly for text classification (Kim, 2014) (Johnson & Zhang, 2014). The essence of CNN is to learn embeddings for small fixed size regions(e.g., "not good" in a document), and each kernel of the convolutional layer tries to capture a specific semantic or structural feature. The kernel of CNN is shared globally, thus commonly several kernels are needed to preserve task related features, while the proposed local context kernel is word specific and forced to capture the particular semantic of the word to its local context. Moreover, the convolution kernels extract the predictive feature by applying convolution operation on the words vector sequence, while the local context kernels are used as distinct linear projection functions on the context words in different relative position.

For text classification, we simply use bag of region embeddings to represent the document, and apply an additional non-linear activation function and a fully connected layer. The word embeddings and local context kernels are jointly trained on the classification task.

2 Related Works

Text classification has been studied for years, traditional approaches focused on feature engineering and using different types of machine learning algorithms. For feature engineering, bag-of-words features are efficient and popular. In addition, the hand-crafted n-grams or phrases are added to make use of word order in text data, which has been shown effectiveness in Wang & Manning (2012). For machine learning algorithms, linear classifiers are widely used, such as naive bayes (McCallum et al., 1998), logistic regression and support vector machines (Joachims, 1998; Fan et al., 2008). Howerve, these models commonly suffer the data sparsity problem.

Recently, several neural models have been proposed, the pre-trained word embeddings of *word2vec* (Mikolov et al., 2013) have been widely used as the inputs to deep neural models such as recursive tensor networks (Socher et al., 2013). On the other hand, models that can directly learn the word embeddings specifically for the task have been proposed recently, such as FastText (Joulin et al., 2016) and CNN (Kim, 2014), which both have been successfully applied on text classification tasks. In the rest of this section, we briefly introduce FastText and CNN, the two approaches that are most similar to ours.

FastText FastText simply averages the word embeddings to represent the document, and uses a full connected linear layer as the classifier. The word embeddings are trained for each task specifically. To utilize the local word order of small regions, FastText adds hand-crafted n-grams features. With the simple model architecture, FastText has been proved to be effective and highly efficient on text classification tasks. Similarly, our models use bag of region embeddings to represent the document, and followed by the same full-connected layer. Differently, our models directly learn the semantics of regions based on words, hand-crafted features is not required.

CNN CNN is a feed-forward network with convolution layers interleaved with pooling layers, which are originally used for image processing tasks. For natural language processing, words are commonly converted to vectors. CNN directly applies convolutional layer on word vectors, both word vectors and the shared(word independent) kernles are the parameters of CNN, which can be learned to capture the predictive structures of small regions. Our purpose is similar with CNN, which tries to learn task specific representations for regions. We apply local context kernels on word vectors, unlike CNN, our kernels are word dependent and the sepcific kernel is learned for each word.

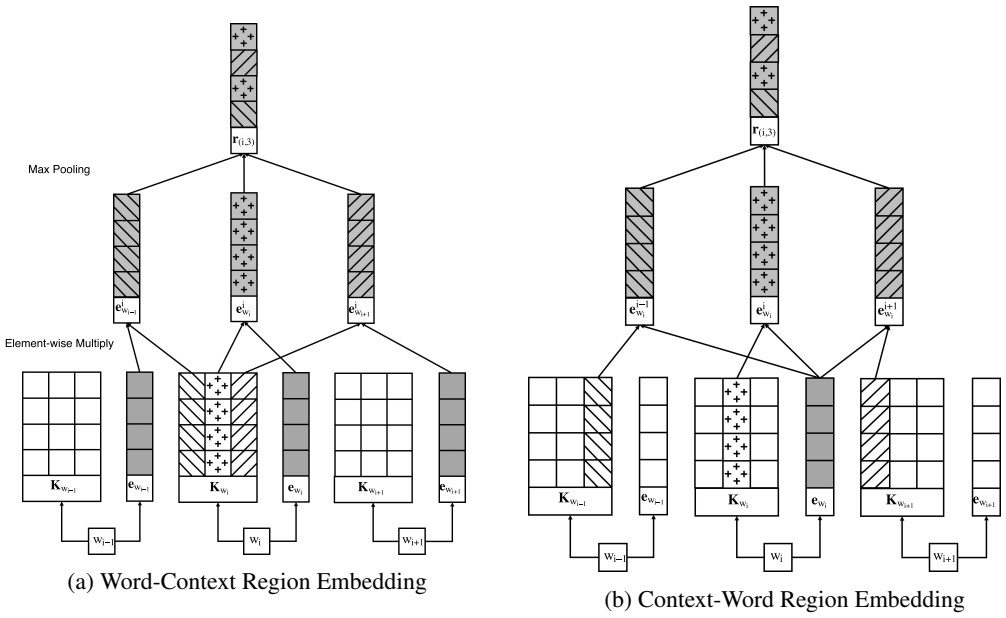
3 METHOD

Several previous works have been proposed to show the effectiveness of word order on text classification task, such as statistical n-gram features, or applying CNNs to learn an embedding of a fixed size region.

In this article we also focused on learning the representations of small text regions which preserve the local internal structure information for text classification. The regions in text can be defined as fixed length substrings of the document. More specifically, with w_i stands the i-th word of the document, we use $region(i, c)$ to denote the regtion with center word w_i, where c is the region size. For instance, given a sentence such as *The food is not very good in this hotel*, $region(3, 5)$ means the substring *food is not very good*.

Considering the semantic influences from words to their context are diverse from each other, we take a different way to learning effective region representations on text classification. Instead of using n-grams or global convolutional kernels, we proposed the local context kernels to address the word specific influences to their context. In our approach, besides the original word embedding, each word has a context kernel to interact with the context. The details of the local context kernels will be introduced later. In the ability to capture the influences through the local context kernels, we proposed two methods to perform the regional embeddings on text classification from different views.

In the rest of this section, we will introduce the local context kernels firstly, and two new architectures to produce the region embeddings using local context kernels will be introduced, finally we will introduce how we use the region embeddings on text classification.



(a) Word-Context Region Embedding

(b) Context-Word Region Embedding

Figure 1: Architectures of region embedding using local context kernels in different perspectives

3.1 LOCAL CONTEXT KERNELS

Words are commonly converted to low dimensional dense vectors when apply neural network model on natural language processing tasks. More formally, the embedding $\mathbf{e}_{w_i}$ of word w_i is represented by a column in a matrix $\mathbf{E} \in \mathcal{R}^{h \times v}$ with a look up layer, where v is the size of vocabulary, h is the embedding size. The columns of $\mathbf{E}$ are indexed by positions of the words in the vocabulary.

Under the consideration of that the occurrence of a word influences the semantics of context words differently, we design an additional specific local context kernel for each word. We define the local

context kernel $\mathbf{K}_{w_i} \in \mathcal{R}^{h \times c}$ of w_i as a matrix which can be obtained through a loop up layer in $\mathbf{U} \in \mathcal{R}^{h \times c \times v}$ by the index of w_i in the vocabulary.

We hope that each column in the kernel matrix of a given word can address the particular semantic and syntactic influence to the words in a given relative position. The columns in the local context kernel of can be regarded as diverse linear projection functions on the embeddings of words in the local region. The parameters of these projection functions(i.e., columns of the kernel matrix) can be learned to capture the semantic and syntactic influence from the word to its context. We call the output of these projection functions projected word embeddings, which represented the new semantics of words in the local region under the influence of the center word.

Formally, given the kernel $\mathbf{K}_{w_i}$ of w_i and the embedding $\mathbf{e}_{w_{i+t}}$ of w_{i+t}, the $(\frac{c}{2}+t)$-th column in $\mathbf{K}_{w_i}$ is denoted by $\mathbf{K}_{w_i,t}$, which can be regarded as a linear projection function to on $\mathbf{e}_{w_{i+t}}$, where $-\frac{c}{2} <= t <= \frac{c}{2}$. We apply a element-wise multiply(denoted by $\odot$) to compute the projected word embedding $\mathbf{e}^i_{w_{i+t}}$ of w_{i+t} as:

$$\mathbf{e}^i_{w_{i+t}} = \mathbf{K}_{w_i,t} \odot \mathbf{e}_{w_{i+t}}$$

Since word in different relative position is projected by different linear function, the proposed local context kernels can be regarded as utilized the word order in a different way.

3.2 Word-context region embedding

In the first proposed architecture, we focus on addressing the center word's influence on the words in its local context, e.g., in the sentence *The food is not very good in this hotel*, the occurrence of word *not* might bring a semantical reservation on the local region. Hence we use the local context kernel of the center word and word embeddings of all words in the region to compose the region embedding, and call this model word-context region embedding. Formally, for a $region(i, c)$, given the kernel $\mathbf{K}_{w_i}$ of center word w_i and the embeddings of words in the region, the projected embeddings can be computed as the following matrix:

$$[\mathbf{e}^i_{w_{i-\frac{c}{2}}}; \mathbf{e}^i_{w_{i-\frac{c}{2}+1}}; ...; \mathbf{e}^i_{w_{i+\frac{c}{2}-1}}; \mathbf{e}^i_{w_{i+\frac{c}{2}}}]$$

To compose the representation of a region $region(i, c)$ noted as $\mathbf{r}_{i,c}$, we simply max pooling on the column dimension of the projected embedding matrix. Figure 1a shows the details of the first model architecture. For instance, in the sentence *The food is not very good in this hotel*, the projected word embeddings in the $region(3, 5)$ are produced by the element-wise multiplies between columns in the local context kernel of *not* and word embeddings of words *food*, *is*, *not*, *very* and *good*, and the embedding $\mathbf{r}_{3,5}$ of $region(3, 5)$ can be obtain by max pooling on these projected word embeddings.

3.3 Context-word region embedding

The second architecture goes as a different view, which address the local context words' influence on the center word in the region, and we call this model context-word region embedding. Similarly, for a $region(i, c)$, the projected embeddings can be represent as the following matrix:

$$[\mathbf{e}^{i-\frac{c}{2}}_{w_i}; \mathbf{e}^{i-\frac{c}{2}+1}_{w_i}; ...; \mathbf{e}^{i+\frac{c}{2}-1}_{w_i}; \mathbf{e}^{i+\frac{c}{2}}_{w_i}]$$

Since the kernels of words in the local context may contains semantic information as well as the word embeddings, the region embedding $\mathbf{r}_{i,c}$ can be composed with a max pooling on the column dimension of the projected embedding matrix. Figure 1b shows the details of the second model architecture.

3.4 Region embedding for text classification

The documents on text classification tasks commonly are of variable-length, to achieve the classification task, the documents need to be represented as fixed size vectors. In order to show the effectiveness of our proposed region embedding models, we simply sum the embeddings of all regions in one document with an additional non-linear activation function to represent the document. We use a Full-Connected linear layer as the classifier for text classification.

Formally, the model can be represented as following:

$$f(\mathbf{x}; \mathbf{E}, \mathbf{U}, \mathbf{W}, \mathbf{b}) = g(\mathbf{W}^\top \sigma(\sum_{i=0}^{n} \mathbf{r}_{i,c}) + \mathbf{b})$$

where $\mathbf{x}$ is the input text sequence, $\mathbf{W}$ and $\mathbf{b}$ denote the weight matrix and bias of the fully connected layer respectively, g denotes the *softmax* function of the output layer, σ denotes the *softsign* function and n denotes the number of regions in the document, $\mathbf{r}$ is the region embedding which can be computed by the two proposed methods. $\mathbf{E}$, $\mathbf{U}$, $\mathbf{W}$ and $\mathbf{b}$ can be updated in the training period.

4 EXPERIMENTS

We report the experiments with proposed models in comparison with previous models. The Code is publicly available on the Internet.

4.1 DATASETS

We use publicly available datasets from Zhang et al. (2015) to evaluate our models. There are total 8 text classification datasets, corresponding to sentiment analysis, news topics, question-answer, ontology extraction, respectively. Table 1 shows the descriptive statistics of datasets used in our experiments. To guarantee comparable indications, same evaluation protocol of Zhang et al. (2015) is employed.

Table 1: Statistics of Datasets

Dataset	Classes	Train Samples	Test Samples	Tasks
Yelp Review Polarity	2	560,000	38,000	Sentiment Analysis
Yelp Review Full	5	650,000	50,000	
Amazon Review Polarity	2	3,000,000	650,000	
Amazon Review Full	5	3,600,000	400,000	
AG's News	4	120,000	7,600	News Topics
Sogou News	5	450,000	60,000	
Yahoo! Answers	10	1,400,000	60,000	Question Answer
DBPedia	14	560,000	70,000	Ontology Extraction

4.2 BASELINES

Our models are compared with several widely used supervised text classification models. We report the n-grams and TFIDF baselines from Zhang et al. (2015), as well as the character level convolutional model (char-CNN) of Zhang & LeCun (2015), the word level convolutional model (word-CNN) of Zhang & LeCun (2015), the character based convolution recurrent network (char-CRNN) of Xiao & Cho (2016), the very deep convolutional network (VDCNN) of Conneau et al. (2016), the Discriminative LSTM (D-LSTM) of Yogatama et al. (2017) and the bigram Fasttext (bigram-Fasttext) of Joulin et al. (2016).

4.3 IMPLEMENTATION DETAILS

For data preprocessing, all the texts of datasets are tokenized by Stanford tokenizer and all words are converted to lower case. Words that appear only once are treated as out-of-vocabulary (OOV) items, and any stop words or symbols are kept. Additionally, padding lenght of $\frac{c}{2}$ are added to both the head and tail of each document.

Optimal hyperparameters are tuned with 10% of the training set on Yelp Review Full dataset, and identical hyperparameters are applied to all datasets: the dimension of word embedding is 128, the region size is 7 which means the shape of local context kernel matrix of each word is 128×7, the learning rate is set 1×10^{-4}, and the batch size is 16. For optimization, the embeddings of words and kernels are randomly initialized with Gaussian Distribution. Adam (Kingma & Ba, 2014) are used

as the optimizer, Cross-entropy loss with softmax is applied meanwhile. We do not use any extra regularization methods, such as L2 normalization or dropout. Algorithms are entirely implemented with TensorFlow and trained on NVIDIA Tesla P40 GPUs.

4.4 RESULTS

Table 2: Test Set Accuracy [%] Compared to other Methods on several Datasets

Model	Yelp P.	Yelp F.	Amz. P.	Amz. F.	AG	Sogou	Yah. A.	DBP
BoW	92.2	58.0	90.4	54.6	88.8	92.9	68.9	96.6
ngrams	95.6	56.3	92.0	54.3	92.0	97.1	68.5	98.6
ngrams TFIDF	95.4	54.8	91.5	52.4	92.4	97.2	68.5	98.7
char-CNN	94.7	62.0	94.5	59.6	87.2	95.1	71.2	98.3
char-CRNN	94.5	61.8	94.1	59.2	91.4	95.2	71.7	98.6
bigram-FastText	95.7	63.9	94.6	60.2	92.5	96.8	72.3	98.6
VDCNN	95.7	64.7	**95.7**	**63.0**	91.3	96.8	73.4	98.7
D-LSTM	92.6	59.6	-	-	92.1	94.9	73.7	98.7
W.C.region.emb	**96.4**	**64.9**	95.1	60.9	**92.8**	**97.6**	**73.7**	**98.9**
C.W.region.emb	96.2	64.5	95.3	60.8	**92.8**	97.3	73.4	**98.9**

Table 2 is a performace summary of compared methods on listed datasets. We use underscores to represent the best known published results, and use the bold to indicate the new records. On six datasets of eight, our models achieve new state-of-art or equally best with a performamce gain highest to 0.7(%). We beat all the previous models on all datasets except VDCNN, while the latter performes almost best on all classification tasks before the proposing of our methods. As a result, we slightly win VDCNN on six datasets and lost in two of Amazon datasets. Although the slight advantage, our methods are remarkable on account of its simplicity along with divergence. Concretly, The upper layer structure of our models only uses a summing up operation, which is more concise and robust than any other deep or complex models. On the other hand, different with the fashionable convolutional kernels, which are shared across all the sequence, we use the self context-aided kernel for each word. To our best known, we are the first to introduce word specific kernels techniques.

In fact, we list the comparation of our two proposed models. Both of them perform fairly well. In general, Word-Context model is slightly better than Context-Word model. The difference is that Context-Word model uses the same embedding of center word to get the projected word embeddings while Word-Context model uses different embeddings of words in one region, so the latter may use more decision units in each pooling operation than the former.

4.5 EXPLORATORY EXPERIMENTS

In this subsection we are going to do a set of exploratory experiments to study the relative effect of each component in our model. Typical cases will be analyzed to validate properties of various aspects of our models. Without loss of generality, we only analyzed the Word-Context region embedding model in our exploratory experiments.

4.5.1 EFFECT OF REGION SIZE AND EMBEDDING SIZE

Our method uses a fixed size of region as contextual information just like CNN. So the selection of region size really matters. A small region may lose of some long-distance patterns, whereas large regions will lead into more noise. Luckily, our models seem to be fairly insensitive towards kinds of datasets. Actually, we just select consistent configuration of size 7 for entire datasets and it has been able to outperform best published results ever.

Figure 2 describes the performance in Yelp Review Full with different region sizes, and when the size equals to 1, the result is quite close to unigram Fasttext(Acc 60.7%), but still get 0.6%+ promotion. Intuitively, the central word can not influence other words except itself when the size equals to 1. The performance increases with the growth of region size up to 7, followed by a drop.

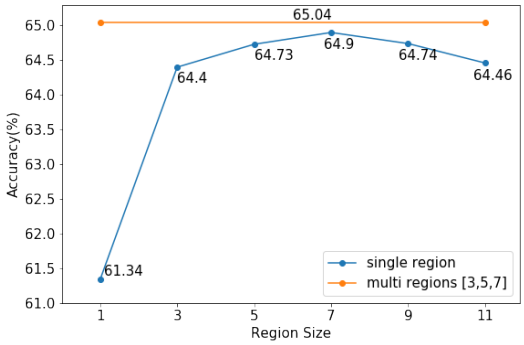


Figure 2: Effect of the hyperparameters ($region\ size$) on Yelp Review Full dataset.Shows the comparison of single fixed region size 7 and multi sizes combination [3,5,7]

To further study the effectiveness of region sizes, we experimented our models with the combination of multi region sizes. Figure 2 shows the comparisons between single region sizes and the multi-size combination of *3,5,7*, which shows that the combination of multi region sizes is slightly better than the best single region size 7. The effectiveness of multi-size combination can be explained by the difference of influence ranges between words, such as *very* only emphasizes the next word while *however* may lay stress on a wide range of the following words.

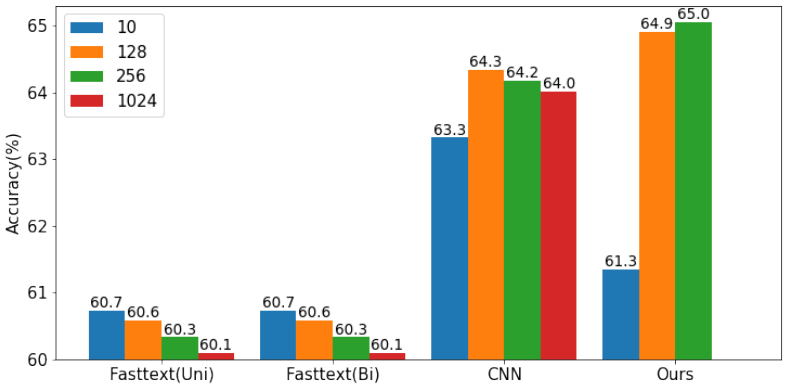


Figure 3: Effect of the hyperparameters ($embedding\ size$) on Yelp Review Full dataset. Shows the effect of different settings of embedding size among three kinds of models, unigram Fasttext, CNN and ours. We unified use region size 7 for CNN and ours.

In addition to the analysis of region sizes, we further studied the influence of word embedding dimensions. Figure 3 lists the comparative results on Yelp Review Full with different embedding dimensions. The results show that our model is less prone to be overfitting than Fasttext and CNN with the word embedding dimension increasing.

In fact, the amount of parameters in our models is relatively large. Since we learn a specific kernel for each word, under the same word embedding dimension, our parameter size has been expanded by *region size* times. Notice that the sizes of parameters are relatively consistent among *1024 in Fasttext*, *1024 in CNN* and *128 in ours*.

4.5.2 EFFECT OF KERNELS

In this section, we explored some comparative experiments to show the effectiveness of our proposed word kernels. The experiments are employed based on unigram Fastext baseline, which has similar upper layer structure with our models. Table 3 illustrated the interrelated results.

Firstly, we remove the entire kernels from our model, which means it is just a variant version of unigram fasttext, we call it Fasttext(Win-pool). The difference is that Fasttext sums up the word

Table 3: Comparative decomposition results on Yelp Review Full dataset. For Fasttext(Unigram), embedding dimension is 10. For Fasttext(Win-pool), C.W.region.emb(Scalar) and C.W.region.emb(our model), region size is 7 and embedding dimension is 128

.

Decomposition	Performance(%)
Fasttext(Unigram)	60.73
Fasttext(Win-pool)	61.01(+**0.28**)
C.W.region.emb(Scalar)	63.18(+**2.45**)
C.W.region.emb(Our model)	64.9(+**4.17**)

embeddings directly while Fasttext(Win-pool) sums up the window pooled embeddings in a stride of 1. It yields a slightly accuracy gain of 0.28% than unigram fasttext.

Secondly, we apply a simplified scalar version of kernel to Fasttext(Win-pool). Unlike our proposed kernels, the kernel of w_i has the shape with $1 \times c$, hence the kernel can be regarded as scalling on corresponding word embeddings in its local context. We call this method C.W.region.emb(Scalar). Compared to our models, it yields a huge parameter size reduction, however, it already yileds a significant gain of 2.45% compared to unigram Fastext.

Futermore, C.W.region.emb(our model) is the variant version of C.W.region.emb(Scalar) that the scala kernel is expanded to a dense vector, which the kernel of w_i has shape with $h \times c$. Adding the low dimensional dense kernels improves the performance by 4.17% compared to unigram Fasttext and 1.72% to C.W.region.emb(Scalar). The results proved that our proposed word kernels can considerably improves the performance compared to the unigram fasttext even with a simpler scalar version.

In order to further analyze what kernels actually capture, Figure 4 presents the heat maps of word kernels for chosen word samples, such as *however*, *very*, e.g. each row represent a word kernel of the center word, r_i denoted as i-th columns L2-norm on the right in the word kernel matrix and similar as l_i. Qualitative but not fully rigorous, a normalized L2-norm is used to render the shade, and the intensity of r_i reflects the degree of the center word's influence on the relative i-th word on the right.

What the figure reflects keeps consistent with intuitive priors of human beings. In the perspective of *however*, right contexts play the key role for classification polarity because of the emotional turnaround, the color is indeed deeper in r_i than l_i, so does *but*. For word *very*, r_1 is more prominent than the rest of all, which capture some modified patterns like *very happy* or *very sad*. For word *good*, tendencies will be completely different for pattern *not good*, *very good* and *not that good*, which are intensive negative, intensive positive and slightly hesitated, separately, the position of l_1 will be strengthened as a result, so does word *bad*.

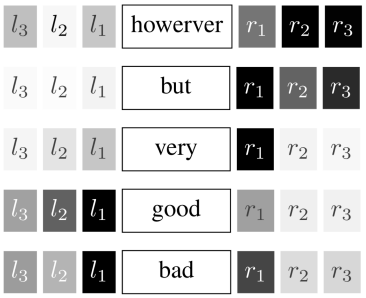


Figure 4: Heat maps of choose words trained on Yelp Review Polarity, which belongs to a binary classification task of sentiment analysis. Each row represent the kernel of the center word. Region size is 7 and embedding size is 128.

Actually, from the motivation of word specified kernels, we would like to believe this feature help capture syntactic and semantic influence on the relative position of its contexts. It can be regarded as a way to make use of word order in kernel-covered regions.

4.5.3 VISUALIZATION

In this subsection, we will try to visualized evaluate the contribution of each word(or selected region) to the classification performance. Since matrix has the law of association, the contribution can be truthfully divided into each pooled region owing to the simple summing up layer of our model. Furthermore, window-inner pooling is indeed explained by corresponding word's certain dimension via argument max function. Overlaying will be considered because covered word can be pooled by different windows. Through such a trivial backward, we can calculate each word's proportion to the final category logits. Notice that not the self embedding acts here, but the projected embedding of self one on surrounding words' kernel.

Table 4: Visualization of chosen samples on Yelp Review Polarity dataset. Green denotes positive contribution while read denotes negative. Two methods are compared about no kernel and with kernel.

Method	Sentence Samples											Phrase
No kernel	get	your	wallet	ready	,	the	prices	are	crazy	high	!	crazy high
With kernel	get	your	wallet	ready	,	the	prices	are	crazy	high	!	crazy high
No kernel	nothing	remarkable	,	but	not	bad	either					not bad either
With kernel	nothing	remarkable	,	but	not	bad	either					not bad either

For clarity, we choose a binary classification task of sentiment analysis. In Table 4, we list two real case samples in Yelp Review Polarity dataset, in which our model performance as expected. Words or artificially selected phrases are highlighted green if they are positive factors, red if they are negative. The intensity of the color indicates the extends of the polarity.

We show the visualization of the two dispose on samples, no kernel(Fasttext) and with kernel. For sentence *get your wallet ready, the prices are crazy high!*, word colors will be only the embodiment of their own embedding if no kernel adopted. The word *crazy* is colored positive, even the exclamation mark behaves slightly green. The word *high* is colored negative, oppositely. Because the intensity of *crazy* is higher than *high*, the entire polarity of phrase *crazy high* is totally positive, which is a complete mistake. But with kernel, things have changed quite a bit. Because of the exist of *high* as the context, the positive polarity of *crazy* and *!* vanishes. The negative polarity of *high* is stronger, phrase *crazy high* performs positive overall. For another case *nothing remarkable, but not bad either*, things seems more interesting. If no kernel, *remarkable* will be positive, while *nothing, not, bad* perform negative, respectively. But with kernel, the polarity of the part ahead of *but* is reduced, meanwhile the polarity of *not* and *bad* flip attractively. As a result, phrase *not bad either* performs positive overall.

5 CONCLUSION

This paper proposed two novel architectures for text classification that learns embeddings of small text regions by introduced local context kernels. *Without aid of additional resources*, the embeddings and kernels are trained specifically for the task of interest. Benefiting the simplicity of our models, we achieved state-of-the-art performances on six benchmark text classification datasets with *identical hyperparameters*.

REFERENCES

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*, 2016.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

Thorsten Joachims. Making large-scale svm learning practical. Technical report, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, 1998.

Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058*, 2014.

Rie Johnson and Tong Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. In *Advances in neural information processing systems*, pp. 919–927, 2015.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pp. 41–48. Madison, WI, 1998.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 79–86. Association for Computational Linguistics, 2002.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pp. 90–94. Association for Computational Linguistics, 2012.

Yijun Xiao and Kyunghyun Cho. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*, 2016.

Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. Generative and discriminative text classification with recurrent neural networks. *stat*, 1050:6, 2017.

Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pp. 649–657, 2015.