

# Shallow Network with Rich Features for Text Classification

Duy-Tin Vo, Yue Zhang, and Xiaodan Zhu

<sup>1</sup> Université Laval

2325 Rue de l'Université, Ville de Québec, Québec, Canada G1V0A6

duytinvo@gmail.com

<sup>2</sup> Singapore University of Technology Design

8 Somapah Road, Singapore 487372

yue.zhang@sutd.edu.sg

<sup>3</sup> Queen's University

99 University Ave, Kingston, Ontario, Canada K7L3N6

zhu2048@gmail.com

**Abstract.** Very deep neural models with many layers have been investigated for document classification, achieving the best results on large datasets in a trade-off of high computation. While deep models can increase the expressiveness of the neural network, their strength can also come from enlarged model parameters. In this paper, we investigate a different approach to extend the parameter space, exploiting rich features instead of deep architectures. Central to our idea is a different view on model composition—instead of stacking multiple neural layers in depth, we juxtapose heterogeneous component neural networks in breadth. Interestingly, under standard text classification evaluation on eight different datasets, the shallow model efficiently yields best performances on small-scale benchmarks and competitive results on large-scale benchmarks.

## 1 Introduction

Neural network models have emerged as cutting-edge approaches to text classification, with recursive neural network (ReNN) [1], convolutional neural network (CNN) [2, 3], and long short-term memory (LSTM) [4, 5] having been shown to give state-of-the-art accuracies for classifying short text along different semantic dimensions. For longer text, such as documents of multiple sentences as shown in Figure 1, *composite* neural network models appear to be promising in rendering better expressiveness, outperforming the aforementioned shallow neural architectures [6, 7].

Hierarchical models [8–10] have been a typical approach to representing document features bottom-up, where a neural network is deployed to learn sentence representation from word embeddings, and then a separate network is applied to obtain document representation from sentence vectors in turn. Deep model without differentiating sentences and documents have also been utilized. As depicted in Figure 2a, Zhang et al. [6] have recently developed a deep composite network, which leverage a 9-layer neural model (i.e., 6 CNN layers and 3 fully-connected layers) with one-hot-character inputs. Conneau et al. [7] extended the method of Zhang et al. [6] in constructing a

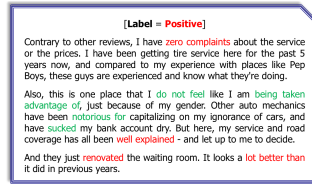


Fig. 1: Document-level text classification.

very deep CNN model (i.e. up to 49 CNN layers and 3 fully-connected layers) with character-embedding inputs, yielding the best performance when the sizes of training data is sufficiently large.

Zhang et al. [6] made experimental comparisons between the deep CNN models using one-hot characters and pre-trained word embeddings. In addition, they compared deep CNNs with several traditional models for text classification including LSTM and discrete models (e.g. bag-of-words, bag-of-ngrams, and TFIDF) on eight datasets of varying scales of sizes. They found that character-level deep CNN models are superior on large datasets, while discrete ngram models give better results on datasets that contain *less than 600,000 documents*. In addition, LSTM works significantly better than discrete models on longer sentences. The authors concluded that “there is not a single machine learning model that can work for all kinds of datasets”. Conneau et al. [7] further extended the method of Zhang et al. [6], using character embeddings for very deep CNNs on the same datasets. They reported similar findings, showing that very deep models gave the state-of-the-art results when trained on sufficiently large data, under particular settings, and the performance of a neural network is not proportional to its depth.

Our model follows this observation of Zhang et al. [6], making use of different types of sub-networks (e.g. CNN and LSTM) for better accuracies. Similar to the model of Zhang et al. [6], our model has a composite neural network structure. On the other hand, instead of stacking multiple network units in *depth*, we collocate them in *breadth*, where each component neural network unit is considered a different feature extractor. With respect to model parameters, both Zhang et al. [6] and our model enlarge the number of parameters by having multiple network components. With respect to features, Zhang et al. [6] learn a stack of features of increasingly more abstract levels of information by depth in space, while we learn a range of different shallow feature abstractions that are known to compliment each other.

As shown in Figure 2b, we first make use of two vectorization approaches (i.e. one-hot encoding and embeddings) at two different basic input levels (namely characters and words, respectively), through which we map a document into two representation matrices. Second, a set of variable-size CNNs are applied to each input matrix to extract local context features. In addition, we employ bidirectional LSTMs (BiLSTMs) on the word embedding level in order to capture global dependency features. These features are then fed into a fully connected layer to learn a document representation, before being passed to a *softmax* layer for prediction.

Experiments show that our conceptually simple shallow model consistently yields competitive results on eight different text classification datasets, which range from 120 thousand to 3 million documents with size of documents between 38 words and 575 words. In addition, it achieves better results compared to those of more traditional models on small-scale datasets, on which the very deep models [6, 7] gave inferior results. It has been well understood that a neural network of three layers or more can learn more functions compared to a neural network of one or two layers [11]. On the other hand, the power of very deep models have been shown mostly empirically [12] in a trade-off of high computational complexity. Our results might suggest further research questions to answer of the relative strengths of very deep networks and wide extension of model parameterization while still mitigating the model complexity. Our code is publicly available at <http://XXXXXX.XXX>.

## 2 Related Work

In its broad sense, text classification can be performed on inputs ranging from short text (e.g., sentences) to much longer documents. The tasks have been traditionally tackled with various statistical models using bag-of-word features [13–15]. More recently, neural networks have achieved the state of the art on many text classification tasks.

At the sentence level, numerous attempts have been made to model sentential meaning for different tasks such as sentiment classification. For example, Socher et al. [1] propose a recursive neural network to learn sentence representations over tree structures in a bottom-up fashion. Kim [2] applies a shallow two-channel CNN to extract local features from sentences for text classification. Kalchbrenner et al. [3] use a dynamic multi-layer CNN to extract non-local features. Wang et al. [16] employ LSTM with pre-trained embeddings for sentiment analysis of short text such as Tweets. Most other existing models are variants of these architectures, such as adding gate structures into ReNN [17], augmenting character embeddings in CNN [18], and employing tree-structure for LSTM [19, 5]. However, these approaches result in weaker performances when directly applied to long text [6].

At the document level, several composite neural networks have been proposed for achieving better performances, by rendering hierarchical representation for documents. For example, Tang et al. [8] attempt to hierarchically model a document by using CNN or LSTM with word embedding inputs to learn sentence compositions, before feeding them to a Gated Recurrent Unit (GRU) for representing a document. Zhang et al. [9] utilize one LSTM to learn sentence representations from word embeddings and another LSTM to documents of sentences from sentence vectors, before using a CNN to extract document features from LSTM hidden layers. Similarly, Yang et al. [10] propose two attention-enriched GRUs for hierarchical word-sentence-document representation. One limitation of such methods, however, is the modelling documents from sentences, which is subject to errors from sentence segmentation and cannot be applied directly also for short text classification.

Zhang et al. [6] build a deep 9-layer model to tackle both short and long text classification. Without differentiating documents and sentences, they employ 70 common characters and one-hot-encoding to map a span of text into a fixed-size matrix (e.g.

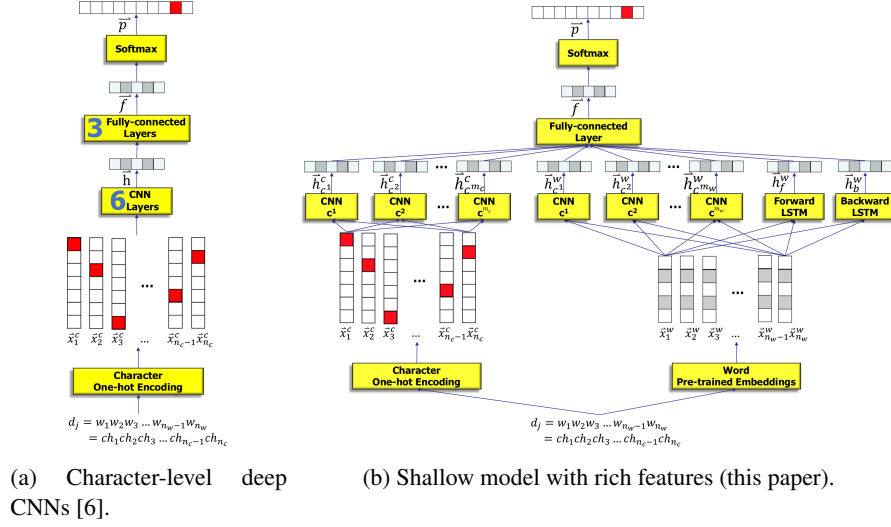


Fig. 2: Baseline and our model.

$70 \times 1024$ ). Then a deep network of 6 CNN layers and 3 fully connected layers is employed to classify the fixed-size matrix. Their model is similar to previous methods in stacking basic network units into a composite network for better inferring implicit features. Conneau et al. [7] extend the work of Zhang et al. [6] by deepening CNN model to enhance the performance. Conneau et al. [7] employ embeddings of the 70 common characters as inputs of a very deep CNN model, which consists of up to 49 CNN layers and 3 fully-connected layers. These models [6, 7] yield the best performances in large-scale datasets. For small-scale data sets, however, traditional approaches are still better than their neural network.

Our method is consistent with the above methods in leveraging composite neural network structure for document classification. Similar to the method of Zhang et al. [6] and Conneau et al. [7] and different from Tang et al. [8] and Yang et al. [10], we make no differentiation between short and long texts, by disregarding sentence boundaries. Multiple neural networks are integrated, rather than building a two-layer sentence/document model. Stacking component networks in depth helps to learn more abstract features. However, it can make training challenging in back-propagating errors. In contrast, shallow networks are easier to train and also relatively more interpretable. For example, CNN has been shown to be closely related to ngram features [2, 3] and the mechanism of LSTMs for classification have been analyzed heavily [16, 4]. Our method is also in line with prior research on leveraging ensemble of neural networks for language modelling [20], and solving NLP tasking more efficiently by using rich shallow features [21], as opposed to using deep architectures.

### 3 Wide Text Classification Network

Given an input document  $\mathbf{d}$ , which consists of  $n_w$  words ( $\mathbf{d} = w_1 w_2 \dots w_{n_w}$ ) and  $n_c$  characters ( $\mathbf{d} = ch_1 ch_2 \dots ch_{n_c}$ , padded if necessary), we aim to classify  $\mathbf{d}$  to one of  $C$  classes. As shown in Figure 2b, we first pass the document to an input neural layer for vectorization. Then, we apply a set of neural network units for modelling local and global contexts. The context features are then supplied to one fully connected layer for feature integration, from which the output layer is finally constructed and used for prediction.

#### 3.1 Input Layer

**One-hot-encoding matrix:** We follow Zhang et al. [6] in leveraging a one-hot-encoding algorithm and  $d_c$  common characters to map a character  $ch_t$  to a sparse one-hot vector  $\mathbf{x}_t^c \in \mathbb{R}^{d_c}$ . Given a document  $\mathbf{d}$  containing  $n_c$  characters, we transform it into a fixed-size matrix:

$$\mathbf{X}^c = \mathbf{x}_1^c \mathbf{x}_2^c \dots \mathbf{x}_{n_c}^c \in \mathbb{R}^{d_c \times n_c} \quad (1)$$

**Word embedding matrix:** In addition to the one-hot character matrix, a word embedding matrix is used as our second way to represent a document. Each word  $w_t$  in  $\mathbf{d}$  is mapped to a dense embedding vector  $\mathbf{x}_t^w$  with a dimension  $d_w$  via a look-up table, so as to obtain an input matrix:

$$\mathbf{X}^w = \mathbf{x}_1^w \mathbf{x}_2^w \dots \mathbf{x}_{n_w}^w \in \mathbb{R}^{d_w \times n_w}, \quad (2)$$

where word embeddings can be either randomly initialized or pre-trained, using Word2vec [22] or Glove [23].

Given a document  $\mathbf{d}$ , the two input matrices  $\mathbf{X}^c$  and  $\mathbf{X}^w$  are fed into a set of neural network units for extracting rich features.

#### 3.2 Modelling Local Context Using CNNs

CNN has been shown effective for extracting local features from raw data, for both image processing [24] and NLP [25]. A convolution operation can be regarded as a sum over element-wise multiplication between a filter with a region size  $l$ , parameterized by a weight matrix  $\mathbf{W}_j^l \in \mathbb{R}^{d_c \times l}$ , and a sequence of  $l$  input columns in  $\mathbf{X}^c$ , represented by a sub-matrix  $\mathbf{X}^c[k : k + l - 1] \in \mathbb{R}^{d_c \times l}$ ,  $k = 1 \dots n - l + 1$ .

$$c_{kj}^l = g(\mathbf{W}_j^l \cdot \mathbf{X}^c[k : k + l - 1] + b_j) \quad (3)$$

In the Equation 3,  $j$  is the number of feature maps,  $b_j \in \mathbb{R}$  is a bias term, and  $g$  is a non-linear activation function. Performing the convolution operation with a region size  $l$  and  $d_l^c$  feature maps over the input matrix  $\mathbf{X}^c$ , we obtain a convolutional output matrix:

$$\mathbf{C}^1 = \mathbf{c}_1^1, \mathbf{c}_2^1, \dots, \mathbf{c}_{n-l+1}^1 \in \mathbb{R}^{d_l^c \times (n-l+1)} \quad (4)$$

Element-wise max-pooling is then used to extract the  $d_l^c$  most salience features  $\mathbf{h}_{c^1}^c$  for the succeeding layers.

$$\mathbf{h}_{c^1}^c = \text{pooling}_{max}(\mathbf{C}^1) \in \mathbb{R}^{d_l^c}, \quad (5)$$

Table 1: Dataset statistics.

Name	#Classes	Task	#Training	#Testing
AG's news	4	News categorization	120k	7.6k
Sogou news	5	Chinese news categorization	450k	60k
DBPedia	14	Ontology classification	560k	70k
Yelp review polarity	2	Sentiment analysis	560k	38k
Yelp review full	5	Sentiment analysis	650k	50k
Yahoo! Answers	10	Question type classification	1.4m	60k
Amazon review polarity	2	Sentiment analysis	3.6m	400k
Amazon review full	5	Sentiment analysis	3m	650k

**Character-level CNNs:** We apply a sequence of  $m_c$  ( $m_c \leq n_c$ ) variable-size convolution operations to extract local context features at the character level resulting in  $m_c$  representations of the document:

$$\mathbf{h}_{c1}^c, \mathbf{h}_{c2}^c, \dots, \mathbf{h}_{cm_c}^c$$

**Word-level CNNs:** Another set of variable-size CNNs with a sequence of  $m_w$  filters is applied to the embedding matrix  $\mathbf{X}^w$  to extract local context features at the word level from the document:

$$\mathbf{h}_{c1}^w, \mathbf{h}_{c2}^w, \dots, \mathbf{h}_{cm_w}^w$$

### 3.3 Modelling Global Context Using BiLSTMs

In addition to the set of  $C^{m_c} + C^{m_w}$  CNNs, we employ two LSTMs to further enhance feature diversity. LSTM [26] takes a recurrent process to encode the entire document  $s$  sequentially left to right. It has been shown to efficiently capture long-range dependencies. A LSTM cell block consists of an input gate  $\mathbf{i}_t$ , a memory cell  $\mathbf{c}_t$ , a forget gate  $\mathbf{f}_t$  and an output gate  $\mathbf{o}_t$ . For each embedding  $\mathbf{x}_t$ , the recurrent model outputs a corresponding hidden state  $\mathbf{h}_t$  by controlling information flow between the inputs  $\mathbf{x}_1 \dots \mathbf{x}_t$  and the history  $\mathbf{h}_1 \dots \mathbf{h}_{t-1}$ . We choose a LSTM variation [27] that computes the hidden state  $\mathbf{h}_t$  as:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_{\tilde{c}} \mathbf{x}_t + \mathbf{U}_{\tilde{c}} \mathbf{h}_{t-1} + \mathbf{b}_{\tilde{c}}) \\
\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

where  $\sigma$  is the *sigmoid* function,  $\odot$  denotes element-wise multiplication, and  $\mathbf{W}_k, \mathbf{U}_k, \mathbf{b}_k$  ( $k \in \{i, o, f, \tilde{c}\}$ ) are model parameters. The above operation on  $\mathbf{x}_1^w \mathbf{x}_2^w \dots \mathbf{x}_{n_w}^w$  results in a sequence of hidden state  $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_{n_w}$  ( $\mathbf{h}_j \in \mathbb{R}^{d_b}$ ).

We employ a bidirectional extension of LSTM (BiLSTMs) [28] at the word embedding level to model the global context of a document  $\mathbf{d}$ . In particular, a history feature

$\mathbf{h}_f^w$  is learned in the forward direction from left to right ( $\mathbf{h}_f^w = \mathbf{h}_{n_w}^f$ ), while a future context feature  $\mathbf{h}_b^w$  is obtained in the backward direction from right to left ( $\mathbf{h}_b^w = \mathbf{h}_1^b$ ), as shown in Figure 2b.  $\mathbf{h}_{n_w}^f$  and  $\mathbf{h}_1^b$  are obtained from two different sets of parameters  $\mathbf{W}_k, \mathbf{U}_k, \mathbf{b}_k$  ( $k \in \{i^f, o^f, f^f, \tilde{c}^f\}$ ) and  $\mathbf{W}_k, \mathbf{U}_k, \mathbf{b}_k$  ( $k \in \{i^b, o^b, f^b, \tilde{c}^b\}$ ), respectively.

### 3.4 Hidden Layer for Feature Integration

The local features  $\mathbf{h}_{c1}^j, \mathbf{h}_{c2}^j, \dots, \mathbf{h}_{cm_j}^j, j \in \{c, w\}$  at the character and word levels, and the global context features  $\mathbf{h}_f^w, \mathbf{h}_b^w$  are concatenated into a rich feature vector representation

$$\mathbf{h} = \mathbf{h}_{c1}^c \oplus \dots \oplus \mathbf{h}_{cm_c}^c \oplus \mathbf{h}_{c1}^w \oplus \dots \oplus \mathbf{h}_{cm_w}^w \oplus \mathbf{h}_f^w \oplus \mathbf{h}_b^w,$$

where  $\oplus$  denotes the concatenation operation and  $\mathbf{h} \in \mathbb{R}^{d_h}$  ( $d_h = m_c d_l^c \times m_w d_l^w \times 2d_b$ ). As shown in Figure 2b, we recruit a fully connected layer to automatically model the document as:

$$\mathbf{f} = g(\mathbf{W}_{hd} \cdot \mathbf{h} + \mathbf{b}_{hd}), \quad (6)$$

where  $\mathbf{f} \in \mathbb{R}^{d_{hd}}$  is the vector document representation,  $g$  indicates a non-linear activation function.  $\mathbf{W}_{hd} \in \mathbb{R}^{d_h \times d_{hd}}$  and  $\mathbf{b}_{hd} \in \mathbb{R}^{d_{hd}}$  are the model parameters. By adding one hidden layer, we hypothesize that the model can automatically select appropriate features from  $\mathbf{h}_{c1}^c, \mathbf{h}_{c1}^w, \mathbf{h}_f^w$  and  $\mathbf{h}_b^w$ , which encode input levels and different contextual features.

### 3.5 Output layer

The feature vector  $\mathbf{f}$  is fed to a *softmax* layer for classification, where the probability of the output classes are computed as a vector

$$\mathbf{p} = \text{softmax}(\mathbf{W} \cdot \mathbf{f} + \mathbf{b}), \quad (7)$$

where  $\mathbf{W} = \mathbf{w}_1 \dots \mathbf{w}_C \in \mathbb{R}^{d_{hd} \times C}$  denotes the weight parameters, and  $\mathbf{b} \in \mathbb{R}^C$  is the bias term.

### 3.6 Training

Given a set of labeled documents, the cross-entropy loss is used as the training objective:

$$\text{loss}(\theta) = -\frac{1}{N} \sum_{i=1}^N \mathbf{y}^{(i)} \cdot \log(\mathbf{p}^{(i)}), \quad (8)$$

where  $\theta$  indicates the set of parameters,  $N$  is the number of training data, and  $\mathbf{y}^{(i)}$  represents the one-hot labeled vector of  $i^{th}$  output. We tune the parameters  $\theta$ , which include word embeddings, per mini-batch using ADAM [29].

Table 2: Hyper-parameters.

Layer	Parameters and Values
One-hot	$d^c = 70, n_c = 1014$
Embeddings	$d^w = 25, p_{drop} = 0.2$
VSCNNs	$d_l^c = d_l^w = 50, m_c = m_w = 5$
BiLSTMs	$d_b^b = d_f^b = 50$
Hidden	$d^{hd} = 300, p_{drop} = 0.5$
Others	$batchsize = 50, maxepochs = 50$

## 4 Experiments

### 4.1 Experimental Setup

**Datasets:** We conduct experiments on the eight large-scale datasets introduced by Zhang et al. [6], which include a variety of classification tasks such as sentiment analysis, news content categorization, ontology classification, and question type classification. The statistics of the datasets are shown in Table 1. Volumes of the training data range from hundreds of thousands samples to millions, which are ideal for empirically comparing complicated neural network architectures. We randomly picked 5% of the training data as validation sets.

**Pre-processing:** At the word level, we convert all words to lower-case and use space tokenization. Words with frequencies less than three times are deemed *unknown* tokens. We also insert  $m - 1$  *padding* tokens at the beginning and  $n - len - m + 1$  *padding* tokens at the end of a text to ensure that variable-size CNNs can slide over the entire text, where  $len$  is the actual length of a text, and  $n = len_{max} + 2(m - 1)$ . The Sogou news dataset, which is in Chinese and is translated into Pinyin (a phonetic romanization of Chinese), contains some documents with more than 40,000 tokens. For these documents, we consider only the first 3,000 words to improve computational efficiency<sup>4</sup>.

At the character level, Zhang et al. [6] defined a set of 70 most common characters mapping each input document into an array of 1014 characters in the set, with each input document being truncated after the 1014<sup>th</sup> character. Characters not belonging to common characters are encoded to be a zero vector. Zhang et al. [6] show that this method works empirically effectively while giving reasonable efficiency. We follow their method and set  $n_c = 1014$ ,  $d_c = 70$ , making zero-vector paddings when the number of characters in a document is less than 1014.

**Hyper-parameters:** We merge all the training samples except Sogou news, and use word2vec [22] to obtain pre-trained word embeddings with a vector size of 25, which is used to initialize the word look-up table. We randomly initialize Pinyin word embeddings for Sogou news. Word embeddings are fine-tuned during training. We apply

<sup>4</sup> This implies that the remaining words of a long document in Pinyin are not considered for classification. Such efficiency consideration follows Zhang et al. [6], who truncate input documents by considering only the first 1014 characters, which is significantly less than 3,000 words.



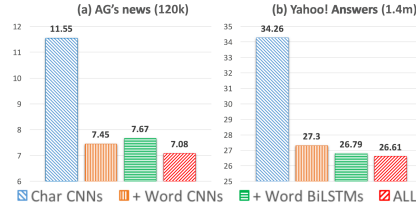


Fig. 3: Ablation experiments.

dropout [30] at the embedding layer and the last layer. All non-linear activation functions  $g$  used in Section 3 are *ReLU* (rectified linear unit). Early stopping is employed if there is no improvement on validation data after three consecutive epoches. Detailed configurations are shown in Table 2. Hyper-parameters are chosen according to common values in the literature, and we did not perform grid search due to the large scales of some datasets.

**Computational efficiency:** We run all experiments on GeForce GTX1080 GPU. Thanks to the shallow structure, our model is far more efficient than very deep models in terms of computations. Particularly, for training, the model took 300 seconds/epoch on 120k samples. However, it only took a couple of seconds (7 seconds) for testing on 7.6k samples.

## 4.2 Development Experiments

**Ablation Developments:** We study the effectiveness of each standard neural network unit by doing ablation experiments on one small-scale data (*AG's news* with an average length of 38 words) and one large-scale data (*Yahoo! Answers* with an average length of 96 words). Four models are compared:

- **char CNNs:** using local context features extracted by character-level CNN units to represent documents;
- **+ word CNNs:** utilizing both local context features of character-level and word-level CNN units;
- **+ word BiLSTMs:** employing the features of **char CNNs** and global context features obtained by BiLSTM units;
- **ALL:** combining all the features mentioned above.

As shown in Figure 3, the error rates of **char CNNs** on *AG's news* and *Yahoo! Answers* are 11.55% and 34.26%, respectively. By adding word-level neural network units (i.e. CNNs and BiLSTM), the performance of **ALL** is increased significantly on both datasets, with an absolute error reduction of 4.47% and 7.65%, respectively. There is a doubled improvement rate in the absolute error on large-scale labeled data in comparison with small-scale labeled data.

In addition, augmenting character-level CNN with word-level CNNs (**+ word CNNs**) gives a better performance (a 7.45% error rate) in comparison with adding word-level BiLSTMs (**+ word BiLSTMs**) (7.67% error rate) for short text classification. The opposite trend is observed for long text classification, where LSTM outperforms CNN.

Table 3: Effectiveness of hidden layer, where  $^{\ddagger}$  denotes a p-value below  $10^{-3}$  by pairwise t-test.

Model	AG's news	Yahoo! Answers
ALL <sup>-</sup>	7.16	26.74
ALL	<b>7.08<sup>‡</sup></b>	<b>26.61<sup>‡</sup></b>

These results are explainable by the characteristics of CNN in extracting local features and LSTM in capturing long-range-dependency features. The ablation experiments justify our hypothesis that juxtaposing different standard neural network units enriches distinct features for our system.

**Effectiveness of hidden layer:** We exploit the capability of the feature integration layer on *AG's news* and *Yahoo! Answers* datasets by comparing:

- **ALL<sup>-</sup>**: directly feeding all local and global features to the *softmax* layer for prediction;
- **ALL**: our model in Figure 2b.

The results are shown in Table 3. Without using a hidden layer (**ALL<sup>-</sup>**), the error rates on the two datasets are 7.16% and 26.74%, respectively. By adding one hidden layer for feature integration, the performance of our system increases significantly to 7.08% and 26.61%, respectively. We further compute a pairwise t-test between two models and show that the use of a hidden layer is significant (with  $p < 10^{-3}$ ). However, when inserting one more hidden layer, we do not observe consistently improved accuracies. We therefore use one hidden layer for feature integration.

### 4.3 Final Results

For each dataset, we compare the performance of our model with those of the previous best methods<sup>5</sup>, namely:

- **ngrams**: multinomial logistic regression using features of the 500,000 most frequent 5-grams.
- **ngram TFIDF**: multinomial logistic regression with features based on the TF/IDF of ngrams.
- **word LSTM**: a one-layer LSTM classifier; the input is pre-trained word embeddings [22] with a dimension of 300.
- **word deep-CNNs**: a 9-layer CNN model with a large structure (i.e. the number of feature maps is 1024), which consist of 6 CNN layers and 3 fully connected layers; the input is pre-trained word embeddings [22] with dimension of 300.
- **char deep-CNNs<sup>+</sup> 1**: the “Lg. Conv. Th.” model of Zhang et al. [6], which uses a large CNN with the number of feature maps being 1024 and data augmentation technique.
- **char deep-CNNs<sup>+</sup> 2**: the “Sm. Conv. Th.” model of Zhang et al. [6], which uses a small CNN with the number of feature maps being 256 and data augmentation technique.

<sup>5</sup> We do not compare our method to the method of Yang et al. [10], because they employ only three datasets among eight datasets with different settings in the number of training data.

Table 4: Testing errors, where \*\* denotes the second best results.

Model	AG	Sogou	DBPedia	Yelp P	Yelp F	Yahoo	Amazon P	Amazon F
ngrams	7.96	2.92	1.37	4.36	43.74	31.53	45.73	7.98
ngrams TFIDF	7.64**	2.81**	1.37	4.56	45.20	31.49	47.56	8.46
word LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
word deep-CNNs	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
word deep-CNNs <sup>+</sup>	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
char deep-CNNs 1	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
char deep-CNNs 2	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
char deep-CNNs 3	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
char deep-CNNs <sup>+</sup> 1	13.39	-	1.60	5.82	39.30	28.80	40.45	4.93
char deep-CNNs <sup>+</sup> 2	14.80	-	1.85	6.49	40.16	29.84	40.43	5.67
char deep-CNNs <sup>+</sup> 3	10.89	-	1.69	5.42	37.95	29.90	40.53	5.66
char very deep-CNNs 1	9.36	3.61	1.36	4.35	<b>35.28</b>	27.17	37.58	<b>4.28</b>
char very deep-CNNs 2	8.67	3.18	1.41	4.63	37.00	27.16	38.39	4.94
char very deep-CNNs 3	8.73	3.36	1.29**	4.28**	35.74	<b>26.57</b>	<b>37.00</b>	4.31
Our model (ALL)	<b>7.08</b>	<b>2.71</b>	<b>1.24</b>	<b>4.27</b>	36.84**	26.61**	39.43**	4.73**

- **char deep-CNNs<sup>+</sup> 3**: the “Sm. Full Conv. Th.” model of Zhang et al. [6] with distinctions between lower and upper characters.
- **char very deep-CNNs 1**: the 29-CNN-layer model of Conneau et al. [7], which employ “convolution” operation as a down-sampling method between two convolutional blocks.
- **char very deep-CNNs 2**: the 29-CNN-layer model of Conneau et al. [7], which utilize “k-max pooling” as a down-sampling method between two convolutional blocks.
- **char very deep-CNNs 3**: the 29-CNN-layer model of Conneau et al. [7] which use “max pooling” as a down-sampling method between two convolutional blocks.

**Classification:** The results are shown in Table 4. Our model yields the competitive performance on all the eight datasets. In particular, on small-scale data such as *AG’s news* and *DBPedia*, where the accuracies of very deep neural network models have not surpassed traditional statistical methods, our model produces the lowest error rates of 7.08% and 1.24%, respectively. For large-scale data, such as *Yahoo! Answers*, our model produces competitive result (e.g. 26.61 on *Yahoo! Answers*) compared to the performance of the very deep methods **char very deep-CNNs 3** (e.g. 26.57 on *Yahoo! Answers*).

**Sentiment Analysis:** On small-scale data such as the *Yelp review*, our combined model outperforms all the best approaches with an error rate of 4.27%. Similar to topic classification, the performance of our model (i.e. 4.27%) is superior to that of traditional methods (i.e. 4.36%) on sentiment polarity classification, which were achieved by the **ngrams** model. On large-scale data, namely *Amazon Review* in both polarity and fine-grained classification, our model gives competitive results to the very deep model of Conneau et al. [7] and better performances in comparison with the deep method of Zhang et al. [6] by 1% and 0.2%, respectively, which employed data augmentation tech-

nique. It is worth noting that when trained on small-scale data (i.e. less than 600,000 labeled samples), our model obtains a new state of the art.

#### 4.4 Discussion

The main reasons of the above improvements is the modelling of a document from different perspectives at different levels. With respect to the length of texts, our model mitigates the lack of semantic information when classifying short text by activating variable-size CNNs for extracting a set of discrete local features. When applied on long text, BiLSTMs are leveraged to extract long-term dependencies over the document. These different neural network units are automatically and favorably learned and selected depending on the input length. Regarding the scale of the datasets, the model takes advantage of multiple input levels and various component neural networks in breadth when learning over large-scale data. On small-scale data our model still works effectively thanks to the use of two different vectorization methods, namely character one-hot encoding and word embedding. This mechanism permits the system to extract enriched features from two different perspectives, which are deficient in traditional deep neural networks.

### 5 Conclusion

We experimented with a shallow neural network with rich CNN and LSTM features for text classification, comparing it to a deep network with many stacked CNN components. By jointly employing different levels of textual representations and wiring multiple neural network units in breadth, our model achieved competitive performances for classifying documents on different datasets with varying lengths and classification objectives. Interestingly, on small-scale datasets, where very deep models are shown inferior to traditional methods, our composite approach performed significantly better than previous best deep models and traditional discrete methods. These results might inspire further investigation of the strengths and weaknesses of very deep neural network models. We release our code and models at <http://XXXXXX.XXX>.

### References

1. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, D.C., Ng, A., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of EMNLP. (2013) 1631–1642
2. Kim, Y.: Convolutional neural networks for sentence classification. In: Proceedings of EMNLP. (2014) 1746–1751
3. Kalchbrenner, N., Grefenstette, E., Blunsom, P.: A convolutional neural network for modelling sentences. In: Proceedings of ACL. (2014)
4. Li, J., Luong, T., Jurafsky, D., Hovy, E.: When are tree structures necessary for deep learning of representations? In: Proceedings of EMNLP. (2015) 2304–2314
5. Zhu, X., Sobhani, P., Guo, H.: Long short-term memory over recursive structures. In: Proceedings of ICML. (2015)

6. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Proceedings of NIPS. (2015)
7. Conneau, A., Schwenk, H., Barrault, L., Lecun, Y.: Very deep convolutional networks for text classification. In: Proceedings of EACL. (2017) 1107–1116
8. Tang, D., Qin, B., Liu, T.: Document modeling with gated recurrent neural network for sentiment classification. In: Proceedings of EMNLP. (2015) 1422–1432
9. Zhang, R., Lee, H., Radev, D.: Dependency sensitive convolutional neural networks for modeling sentences and documents. In: Proceedings of NAACL-HLT. (2016)
10. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., Hovy, E.: Hierarchical attention networks for document classification. In: Proceedings of NAACL-HLT. (2016) 1480–1489
11. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al.: Greedy layer-wise training of deep networks. In Proceedings of NIPS (2007)
12. Zhou, J., Cao, Y., Wang, X., Li, P., Xu, W.: Deep recurrent models with fast-forward connections for neural machine translation. Journal of TACL (2016)
13. Yang, Y., Pedersen, J.O.: A comparative study on feature selection in text categorization. In: Proceedings of ICML. (1997) 412–420
14. Joachims, T.: Transductive inference for text classification using support vector machines. In: ICML. (1999) 200–209
15. Pang, B., Lee, L., Vaithyanathan, S.: Thumbs up?: sentiment classification using machine learning techniques. In: Proceedings of ACL. (2002) 79–86
16. Wang, X., Liu, Y., SUN, C., Wang, B., Wang, X.: Predicting polarities of tweets by composing word embeddings with long short-term memory. In: Proceedings of ACL-IJCNLP. (2015) 1343–1353
17. Chen, X., Qiu, X., Zhu, C., Wu, S., Huang, X.: Sentence modeling with gated recursive neural network. In: Proceedings of EMNLP. (2015) 793–798
18. dos Santos, C., Gatti, M.: Deep convolutional neural networks for sentiment analysis of short texts. In: Proceedings of COLING. (2014)
19. Tai, S.K., Socher, R., Manning, D.C.: Improved semantic representations from tree-structured long short-term memory networks. In: Proceedings of ACL-IJCNLP. (2015)
20. Le, P., Dymetman, M., Renders, J.M.: LSTM-based mixture-of-experts for knowledge-aware dialogues. In: Proceedings of RepL4NLP. (2016) 94–99
21. Zhang, Y., Nivre, J.: Transition-based dependency parsing with rich non-local features. In: Proceedings of ACL:HLT. (2011) 188–193
22. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of NIPS. (2013)
23. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of EMNLP. (2014) 1532–1543
24. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998) 2278–2324
25. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. JMLR (2011)
26. Hochreiter, S., Schmidhuber, J.: Long short-term memory. JNC **9** (1997) 1735–1780
27. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with lstm. JNC (2000)
28. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: Proceedings of ICASSP. (2013)
29. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
30. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. JMLR (2014)