Efficient Character-level Document Classification by Combining Convolution and Recurrent Layers

Yijun Xiao
Center for Data Sciences,
New York University
ryjxiao@nyu.edu

Kyunghyun Cho
Courant Institute and
Center for Data Science,
New York University
kyunghyun.cho@nyu.edu

Abstract

Document classification tasks were primarily tackled at word level. Recent research that works with character-level inputs shows several benefits over word-level approaches such as natural incorporation of morphemes and better handling of rare words. We propose a neural network architecture that utilizes both convolution and recurrent layers to efficiently encode character inputs. We validate the proposed model on eight large scale document classification tasks and compare with character-level convolution-only models. It achieves comparable performances with much less parameters.

1 Introduction

Document classification is a task in natural language processing where one needs to assign a single or multiple predefined categories to a sequence of text. A conventional approach to document classification generally consists of a feature extraction stage followed by a classification stage. For instance, it is usual to use a TF-IDF vector of a given document as an input feature to a subsequent classifier.

More recently, it has become more common to use a deep neural network, which jointly performs feature extraction and classification, for document classification (Kim, 2014; Mesnil et al., 2014; Socher et al., 2013; Carrier and Cho, 2014). In most cases, an input document is represented as a sequence of words, of which each is presented as a one-hot vector.[1] Each word in the sequence is projected into a continuous vector space by being multiplied with a weight matrix, forming a sequence of dense, real-valued vectors. This sequence is then fed into a deep neural network which processes the sequence in multiple layers, resulting in a prediction probability. This whole pipeline, or a network, is tuned jointly to maximize the classification accuracy on a training set.

One important aspect of these recent approaches based on deep learning is that they often work at the level of words. Despite its recent success, the word-level approach has a number of major shortcomings. First, it is statistically inefficient, as each word token is considered separately and estimated by the same number of parameters, despite the fact that many words share common root, prefix or suffix. This can be overcome by using an external mechanism to segment each word and infer its components (root, prefix, suffix), but this is not desirable as the mechanism is highly language-dependent and is tuned independently from the target objective of document classification.

Second, the word-level approach cannot handle out-of-vocabulary words. Any word that is not present or rare in a training corpus, is mapped to an unknown word token. This is problematic, because the model cannot handle typos easily, which happens frequently in informal documents such as postings from social network sites. Also, this makes it difficult to use a trained model to a new domain, as there may be large mismatch between the domain of the training corpus and the target domain.

[1] A one-hot vector of the i-th word is a binary vector whose elements are all zeros, except for the i-th element which is set to one.

Recently this year, a number of researchers have noticed that it is not at all necessary for a deep neural network to work at the word level. As long as the document is represented as a sequence of one-hot vectors, the model works without any change, regardless of whether each one-hot vector corresponds to a word, a sub-word unit or a character. Based on this intuition, Kim et al. (Kim et al., 2015) and Ling et al. (Ling et al., 2015) proposed to use a character sequence as an alternative to the word-level one-hot vector. A similar idea was applied to dependency parsing in (Ballesteros et al., 2015). The work in this direction, most relevant to this paper, is the character-level convolutional network for document classification by Zhang et al. (Zhang et al., 2015).

The character-level convolutional net in (Zhang et al., 2015) is composed of many layers of convolution and max-pooling, similarly to the convolutional network in computer vision (see, e.g., (Krizhevsky et al., 2012).) Each layer first extracts features from small, overlapping windows of the input sequence and pools over small, non-overlapping windows by taking the maximum activations in the window. This is applied recursively (with untied weights) for many times. The final convolutional layer's activation is flattened to form a vector which is then fed into a small number of fully-connected layers followed by the classification layer.

We notice that the use of a vanilla convolutional network for character-level document classification has one shortcoming. As the receptive field of each convolutional layer is often small (7 or 3 in (Zhang et al., 2015),) the network must have many layers in order to capture long-term dependencies in an input sentence. This is likely the reason why Zhang et al. (Zhang et al., 2015) used a very deep convolutional network with six convolutional layers followed by two fully-connected layers.

In order to overcome this inefficiency in modeling a character-level sequence, in this paper we propose to make a hybrid of convolutional and recurrent networks. This was motivated by recent successes of applying recurrent networks to natural languages (see, e.g., (Cho et al., 2014; Sundermeyer et al., 2015)) and from the fact that the recurrent network can efficiently capture long-term dependencies even with a single layer. The hybrid model processes an input sequence of characters with a number of convolutional layers followed by a single recurrent layer. Because the recurrent layer, consisting of either gated recurrent units (GRU, (Cho et al., 2014) or long short-term memory units (LSTM, (Hochreiter and Schmidhuber, 1997; Gers et al., 2000), can efficiently capture long-term dependencies, the proposed network only needs a very small number of convolutional layers.

We empirically validate the proposed model, to which we refer as a convolution-recurrent network, on the eight large-scale document classification tasks from (Zhang et al., 2015). We mainly compare the proposed model against the convolutional network in (Zhang et al., 2015) and show that it is indeed possible to use a much smaller model to achieve the same level of classification performance when a recurrent layer is put on top of the convolutional layers.

2 Basic Building Blocks: Neural Network Layers

In this section, we describe four basic layers in a neural network that will be used later to constitute a single network for classifying a document.

2.1 Embedding Layer

As mentioned earlier, each document is represented as a sequence of one-hot vectors. A one-hot vector of the i-th symbol in a vocabulary is a binary vector whose elements are all zeros except for the i-th element which is set to one. Therefore, each document is a sequence of T one-hot vectors $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$.

An *embedding layer* projects each of the one-hot vectors into a d-dimensional continuous vector space $\mathbb{R}^d$. This is done by simply multiplying the one-hot vector from left with a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times |V|}$, where $|V|$ is the number of unique symbols in a vocabulary:

$$\mathbf{e}_t = \mathbf{W}\mathbf{x}_t.$$

After the embedding layer, the input sequence of one-hot vectors becomes a sequence of dense, real-valued vectors $(\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_T)$.

2.2 Convolutional Layer

A *convolutional layer* consists of two stages. In the first stage, a set of d' filters of receptive field size r,

$\mathbf{F} \in \mathbb{R}^{d' \times r}$, is applied to the input sequence:

$$\mathbf{f}_t = \phi(\mathbf{F} \left[\mathbf{e}_{t-(r/2)+1}; \ldots; \mathbf{e}_t; \ldots, \mathbf{e}_{t+(r/2)}\right]),$$

where ϕ is a nonlinear activation function such as $\tanh$ or a rectifier. This is done for every time step of the input sequence, resulting in a sequence $F = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_T)$.

The resulting sequence F is *max-pooled* with size r':

$$\mathbf{f}'_t = \max\left(\mathbf{f}_{(t-1)\times r'+1}, \ldots, \mathbf{f}_{t\times r'}\right),$$

where $\max$ applies for each element of the vectors, resulting in a sequence

$$F' = (\mathbf{f}'_1, \mathbf{f}'_2, \ldots, \mathbf{f}'_{T/r'}).$$

2.3 Recurrent Layer

A *recurrent layer* consists of a recursive function f which takes as input one input vector and the previous hidden state, and returns the new hidden state:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}),$$

where $\mathbf{x}_t \in \mathbb{R}^d$ is one time step from the input sequence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$. $\mathbf{h}_0 \in \mathbb{R}^{d'}$ is often initialized as an all-zero vector.

Recursive Function The most naive recursive function is implemented as

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_x \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1}\right),$$

where $\mathbf{W}_x \in \mathbb{R}^{d' \times d}$ and $\mathbf{U}_h \in \mathbb{R}^{d' \times d'}$ are the weight matrices. This naive recursive function however is known to suffer from the problem of vanishing gradient (Bengio et al., 1994; Hochreiter et al., 2001).

More recently it is common to use a more complicated function that learns to control the flow of information so as to prevent the vanishing gradient and allows the recurrent layer to more easily capture long-term dependencies. Long short-term memory (LSTM) unit from (Hochreiter and Schmidhuber, 1997; Gers et al., 2000) is a representative example.

The LSTM unit consists of four sub-units–input, output, forget gates and candidate memory cell, which are computed by

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

Based on these, the LSTM unit first computes the memory cell:

$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1},$$

and computes the output, or activation:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t).$$

The resulting sequence from the recurrent layer is then

$$(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T),$$

where T is the length of the input sequence to the layer.

Bidirectional Recurrent Layer One property of the recurrent layer is that there is imbalance in the amount of information seen by the hidden states at different time steps. The earlier hidden states only observe a few vectors from the lower layer, while the later ones are computed based on the most of the lower-layer vectors. This can be easily alleviated by having a bidirectional recurrent layer which is composed of two recurrent layers working in opposite directions. This layer will return two sequences of hidden states from the forward and reverse recurrent layers, respectively.

2.4 Classification Layer

A *classification layer* is in essence a logistic regression classifier. Given a fixed-dimensional input from the lower layer, the classification layer affine-transforms it followed by a *softmax* activation function (Bridle, 1990) to compute the predictive probabilities for all the categories. This is done by

$$p(y = k|X) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x} + b_k)}{\sum_{k'=1}^{K} \exp(\mathbf{w}_{k'}^\top \mathbf{x} + b_{k'})},$$

where $\mathbf{w}_k$'s and b_k's are the weight and bias vectors. We assume there are K categories.

It is worth noting that this classification layer takes as input a *fixed-dimensional* vector, while the recurrent layer or convolutional layer returns a variable-length sequence of vectors (the length determined by the input sequence). This can be addressed by either simply max-pooling the vectors (Kim, 2014) over the time dimension (for both convolutional and recurrent layers), taking the last hidden state (for recurrent layers) or taking the last hidden states of the forward and reverse recurrent networks (for bidirectional recurrent layers.)

3 Character-Level Convolutional-Recurrent Network

In this section, we propose a hybrid of convolutional and recurrent networks for character-level document classification.

3.1 Motivation

One basic motivation for using the convolutional layer is that it learns to extract higher-level features that are invariant to local translation. By stacking multiple convolutional layers, the network can extract higher-level, abstract, (locally) translation-invariant features from the input sequence, in this case the document, efficiently.

Despite this advantage, we noticed that it requires many layers of convolution to capture long-term dependencies, due to the locality of the convolution and pooling (see Sec. 2.2.) This becomes more severe as the length of the input sequence grows, and in the case of character-level modeling, it is usual for a document to be a sequence of hundreds or thousands of characters. Ultimately, this leads to the need for a very deep network having many convolutional layers.

Contrary to the convolutional layer, the recurrent layer from Sec. 2.3 is able to capture long-term dependencies even when there is only a single layer. This is especially true in the case of a bidirectional recurrent layer, because each hidden state is computed based on the whole input sequence. However, the recurrent layer is computationally more expensive. The computational complexity grows linearly with respect to the length of the input sequence, and most of the computations need to be done sequentially. This is in contrast to the convolutional layer for which computations can be efficiently done in parallel.

Based on these observations, we propose to combine the convolutional and recurrent layers into a single model so that this network can capture long-term dependencies in the document more efficiently for the task of classification.

3.2 Model Description

The proposed model, to which we refer as a convolution-recurrent network (ConvRec), starts with a one-hot sequence input

$$X = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T).$$

This input sequence is turned into a sequence of dense, real-valued vectors

$$E = (\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_T)$$

using the *embedding layer* from Sec. 2.1.

We apply multiple *convolutional layers* (Sec. 2.2) to E to get a shorter sequence of feature vectors:

$$F = (\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_{T'}).$$

This feature vector is then fed into a *bidirectional recurrent layer* (Sec. 2.3), resulting in two sequences

$$H_{\text{forward}} = (\overrightarrow{\mathbf{h}}_1, \overrightarrow{\mathbf{h}}_2, \dots, \overrightarrow{\mathbf{h}}_{T'}),$$
$$H_{\text{reverse}} = (\overleftarrow{\mathbf{h}}_1, \overleftarrow{\mathbf{h}}_2, \dots, \overleftarrow{\mathbf{h}}_{T'}).$$

We take the last hidden states of both directions and concatenate them to form a fixed-dimensional vector:

$$\mathbf{h} = \left[\overrightarrow{\mathbf{h}}_{T'}; \overleftarrow{\mathbf{h}}_1\right].$$

Finally, the fixed-dimensional vector $\mathbf{h}$ is fed into the *classification layer* to compute the predictive probabilities $p(y = k|X)$ of all the categories $k = 1, \dots, K$ given the input sequence X.

See Fig. 1 (b) for the graphical illustration of the proposed model.

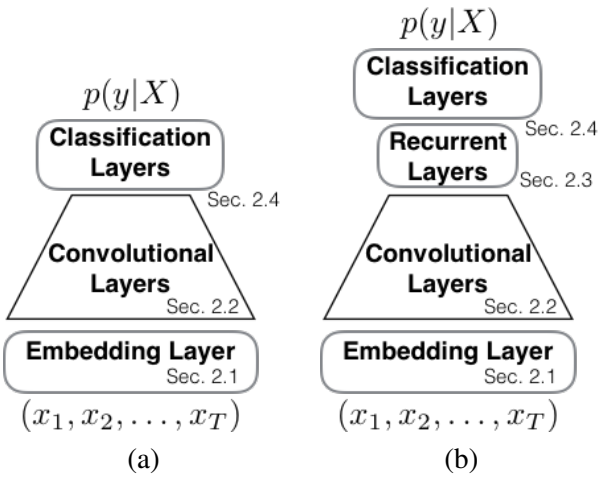


Figure 1: Graphical illustration of (a) the convolutional network and (b) the proposed convolution-recurrent network for character-level document classification.

Data set	Classes	Task	Training size	Test size
AG's news	4	news categorization	120,000	7,600
Sogou news	5	news categorization	450,000	60,000
DBPedia	14	ontology classification	560,000	70,000
Yelp review polarity	2	sentiment analysis	560,000	38,000
Yelp review full	5	sentiment analysis	650,000	50,000
Yahoo! Answers	10	question type classification	1,400,000	60,000
Amazon review polarity	2	sentiment analysis	3,600,000	400,000
Amazon review full	5	sentiment analysis	3,000,000	650,000

Table 1: Data sets summary.

3.3 Related Work

Convolutional network for document classification The convolutional networks for document classification, proposed earlier in (Kim, 2014; Zhang et al., 2015) and illustrated in Fig. 1 (a), is almost identical to the proposed model. One major difference is the lack of the recurrent layer in their models. Their model consists of the embedding layer, a number of convolutional layers followed by the classification layer only.

Recurrent network for document classification Carrier and Cho in (Carrier and Cho, 2014) give a tutorial on using a recurrent neural network for sentiment analysis which is one type of document classification. Unlike the convolution-recurrent network proposed in this paper, they do not use any convolutional layer in their model. Their model starts with the embedding layer followed by the recurrent layer. The hidden states from the recurrent layer are then averaged and fed into the classification layer.

Hybrid model: Conv-GRNN Perhaps the most related work is the convolution-gated recurrent neural net (Conv-GRNN) from (Tang et al., 2015). They proposed a hierarchical processing of a document. In their model, either a convolutional network or a recurrent network is used to extract a feature vector from each sentence, and another (bidirectional) recurrent network is used to extract a feature vector of the document by reading the sequence of sentence vectors. This document vector is used by the classification layer.

The major difference between their approach and the proposed ConvRec is in the purpose of combining the convolutional network and recurrent network. In their model, the convolutional network is strictly constrained to model each sentence, and the recurrent network to model inter-sentence structures. On the other hand, the proposed ConvRec network uses a recurrent layer in order to assist the convolutional layers to capture long-term dependencies (across the whole document) more efficiently. These are orthogonal to each other, and it is possible to plug in the proposed ConvRec as a sentence feature extraction module in the Conv-GRNN from (Tang et al., 2015). Similarly, it is possible to use the proposed ConvRec as a composition function for the sequence of sentence vectors to make computation more efficient, especially when the input document consists of many sentences.

Recursive Neural Networks A recursive neural network has been applied to sentence classification earlier (see, e.g., (Socher et al., 2013).) In this approach, a composition function is defined and recursively applied at each node of the parse tree of an input sentence to eventually extract a feature vector of the sentence. This model family is heavily dependent on an external parser, unlike all the other models such as the ConvRec proposed here as well as other related models described above. It is also not trivial to apply the recursive neural network to documents which consist of multiple sentences. We do not consider this family of recursive neural networks directly related to the proposed model.

4 Experiment Settings

4.1 Task Description

We validate the proposed model on eight large-scale document classification tasks from (Zhang et al.,

Model	Embedding Layer Sec. 2.1		Convolutional Layer Sec. 2.2				Recurrent Layer Sec. 2.3
	$\|V\|$	d	d'	r	r'	ϕ	d'
C2R1DD	96	8	D	5,3	2,2	ReLU	D
C3R1DD				5,5,3	2,2,2		
C4R1DD				5,5,3,3	2,2,2,2		
C5R1DD				5,5,3,3,3	2,2,2,1,2		

Table 2: Different architectures tested in this paper.

2015). The sizes of the data sets range from 200,000 to 4,000,000 documents. These tasks include sentiment analysis (Yelp reviews, Amazon reviews), ontology classification (DBPedia), question type classification (Yahoo! Answers), and news categorization (AG's news, Sogou news).

Data Sets A summary of the statistics for each data set is listed in Table 1. There are equal number of examples in each class for both training and test sets. DBPedia data set, for example, has 40,000 training and 5,000 test examples per class. For more detailed information on the data set construction process, see (Zhang et al., 2015).

4.2 Model Settings

Referring to Sec. 2.1, the vocabulary V for our experiments consists of 96 characters including all upper-case and lower-case letters, digits, common punctuation marks, and spaces. Character embedding size d is set to 8.

As described in Sec. 3.1, we believe by adding recurrent layers, one can effectively reduce the number of convolutional layers needed in order to capture long-term dependencies. Thus for each data set, we consider models with two to five convolutional layers. Following notations in Sec. 2.2, each layer has $d' = 128$ filters. For AG's news and Yahoo! Answers, we also experiment larger models with 1,024 filters in the convolutional layers. Receptive field size r is either five or three depending on the depth. Max pooling size r' is set to 2. Rectified linear units (ReLUs, (Glorot et al., 2011)) are used as activation functions in the convolutional layers. The recurrent layer (Sec. 2.3) is fixed to a single layer of bidirectional LSTM for all models. Hidden states dimension d' is set to 128. More detailed setups are described in Table 2.

Dropout (Srivastava et al., 2014) is an effective way to regularize deep neural networks. We apply dropout after the last convolutional layer as well as after the recurrent layer. Without dropout, the inputs to the recurrent layer $\mathbf{x}_t$'s are

$$\mathbf{x}_t = \mathbf{f}'_t$$

where $\mathbf{f}'_t$ is the t-th output from the last convolutional layer defined in Sec. 2.2. After adding dropout, we have

$$r_t^i \sim \text{Bernoulli}(p)$$

$$\mathbf{x}_t = \mathbf{r}_t \odot \mathbf{f}'_t$$

p is the dropout probability which we set to 0.5; r_t^i is the i-th component of the binary vector $\mathbf{r}_t \in \mathbb{R}^{d'}$.

4.3 Training and Validation

For each of the data sets, we randomly split the full training examples into training and validation. The validation size is the same as the corresponding test size and is balanced in each class.

The models are trained by minimizing the following regularized negative log-likelihood or cross entropy loss. X's and y's are document character sequences and their corresponding observed class assignments in the training set D. $\mathbf{w}$ is the collection of model weights. Weight decay is applied with $\lambda = 5 \times 10^{-4}$.

$$l = -\sum_{X,y \in D} \log(p(y|X)) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

We train our models using AdaDelta (Zeiler, 2012) with $\rho = 0.95$, $\epsilon = 10^{-5}$ and a batch size of 128. Examples are padded to the longest sequence in each batch and masks are generated to help identify the padded region. The corresponding masks of

Data set	# Ex.	# Cl.	Our Model Network	# Params	Error (%)	(Zhang et al., 2015) Network	# Params	Error (%)
AG	120k	4	C2R1D1024	20M	8.39/**8.64**	C6F2D1024	27M	-/9.85
Sogou	450k	5	C3R1D128	.4M	4.82/**4.83**	C6F2D1024⋆	27M	-/4.88
DBPedia	560k	14	C2R1D128	.3M	1.46/**1.43**	C6F2D1024	27M	-/1.66
Yelp P.	560k	2	C2R1D128	.3M	5.50/5.51	C6F2D1024	27M	-/**5.25**
Yelp F.	650k	5	C2R1D128	.3M	38.00/**38.18**	C6F2D1024	27M	-/38.40
Yahoo A.	1.4M	10	C2R1D1024	20M	28.62/**28.26**	C6F2D1024⋆	27M	-/29.55
Amazon P.	3.6M	2	C3R1D128	.4M	5.64/5.87	C6F2D256⋆	2.7M	-/**5.50**
Amazon F.	3.0M	5	C3R1D128	.4M	40.30/40.77	C6F2D256⋆	2.7M	-/**40.53**

Table 3: Results on character-level document classification. CCRRFFDD refers to a network with C convolutional layers, R recurrent layers, F fully-connected layers and D dimensional feature vectors. $\star$ denotes a model which does not distinguish between lower-case and upper-case letters. We only considered the character-level models without using Thesaraus-based data augmentation. We report both the validation and test errors. In our case, the network architecture for each dataset was selected based on the validation errors. The numbers of parameters are approximate.

the outputs from convolutional layers can be computed analytically and are used by the recurrent layer to properly ignore padded inputs. The gradient of the cost function is computed with backpropagation through time (BPTT, (Werbos, 1990)). If the gradient has an L2 norm larger than 5, we rescale the gradient by a factor of $\frac{5}{\|\mathbf{g}\|_2}$. i.e.

$$\mathbf{g}_c = \mathbf{g} \cdot \min\left(1, \frac{5}{\|\mathbf{g}\|_2}\right)$$

where $\mathbf{g} = \frac{\mathrm{d}l}{\mathrm{d}\mathbf{w}}$ and $\mathbf{g}_c$ is the clipped gradient.

Early stopping strategy is employed to prevent overfitting. Before training, we set an initial $patience$ value. At each epoch, we calculate and record the validation loss. If it is lower than the current lowest validation loss by 0.5%, we extend $patience$ by two. Training stops when the number of epochs is larger than $patience$. We report the test error rate evaluated using the model with the lowest validation error.

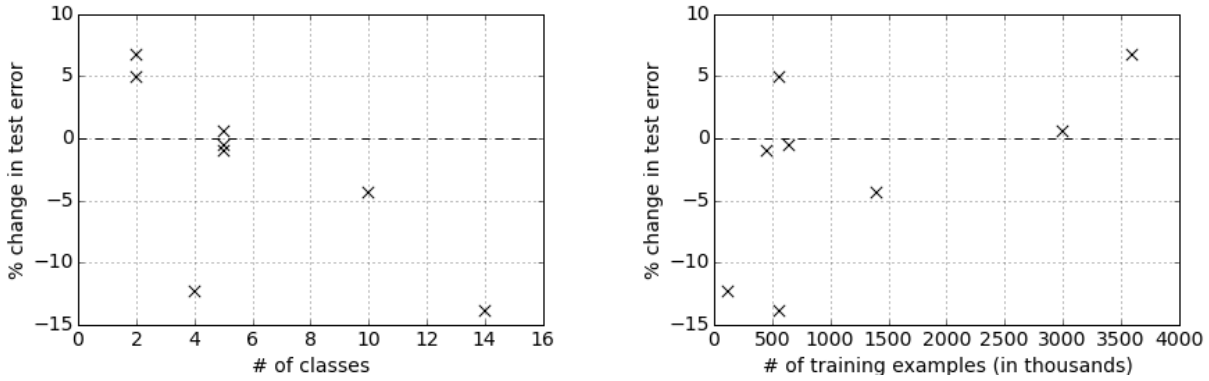
5 Results and Analysis

Experimental results are listed in Table 3. We compare to the best character-level convolutional model without data augmentation from (Zhang et al., 2015) on each data set. Our model achieves comparable performances for all the eight data sets with significantly less parameters. Specifically, it performs better on AG's news, Sogou news, DBPedia, Yelp review full, and Yahoo! Answers data sets.

Number of classes Fig. 2 (a) shows how relative performance of our model changes with respect to the number of classes. It is worth noting that as the number of classes increases, our model achieves better results compared to convolution-only models. For example, our model has a much lower test error on DBPedia which has 14 classes, but it scores worse on Yelp review polarity and Amazon review polarity both of which have only two classes. Our conjecture is that more detailed and complete information needs to be preserved from the input text for the model to assign one of many classes to it. The convolution-only model likely loses detailed local features because it has more pooling layers. On the other hand, the proposed model with less pooling layers can better maintain the detailed information and hence performs better when such needs exist.

Number of training examples Although it is less significant, Fig. 2 (b) shows that the proposed model generally works better compared to the convolution-only model when the data size is small. Considering the difference in the number of parameters, we suspect that because the proposed model is more compact, it is less prone to overfitting. Therefore it generalizes better when the training size is limited.

Number of convolutional layers An interesting observation from our experiments is that the model accuracy does not always increase with the number of convolutional layers. Performances peak at two or three convolutional layers and decrease if we add



(a) (b)

Figure 2: Relative test performance of the proposed model compared to the convolution-only model w.r.t. (a) the number of classes and (b) the size of training set. Lower is better.

more to the model. As more convolutional layers produce longer character n-grams, this indicates that there is an optimal level of local features to be fed into the recurrent layer. Also, as discussed above, more pooling layers likely lead to the lost of detailed information which in turn affects the ability of the recurrent layer to capture long-term dependencies.

Number of filters We experiment large models with 1,024 filters on AG's news and Yahoo! Answers data sets. Although adding more filters in the convolutional layers does help with the model performances on these two data sets, the gains are limited compared to the increased number of parameters. Validation error improves from 8.75% to 8.39% for AG's news and from 29.48% to 28.62% for Yahoo! Answers at the cost of a 70 times increase in the number of model parameters.

Note that in our model we set the number of filters in the convolutional layers to be the same as the dimension of the hidden states in the recurrent layer. It is possible to use more filters in the convolutional layers while keeping the recurrent layer dimension the same to potentially get better performances with less sacrifice of the number of parameters.

6 Conclusion

In this paper, we proposed a hybrid model that processes an input sequence of characters with a number of convolutional layers followed by a single recurrent layer. The proposed model is able to encode documents from character level capturing sub-word information.

We validated the proposed model on eight large scale document classification tasks. The model achieved comparable results with much less convolutional layers compared to the convolution-only architecture. We further discussed several aspects that affect the model performance. The proposed model generally performs better when number of classes is large, training size is small, and when the number of convolutional layers is set to two or three.

The proposed model is a general encoding architecture that is not limited to document classification tasks or natural language inputs. For example, (Chen et al., 2015; Visin et al., 2015) combined convolution and recurrent layers to tackle image segmentation tasks; (Sainath et al., 2015) applied a similar model to do speech recognition. It will be interesting to see future research on applying the architecture to other applications such as machine translation and music information retrieval. Using recurrent layers as substitutes for pooling layers to potentially reduce the lost of detailed local information is also a direction that worth exploring.

Acknowledgments

This work is done as a part of the course DS-GA 1010-001 Independent Study in Data Science at the Center for Data Science, New York University.

References

[Ballesteros et al.2015] Miguel Ballesteros, Chris Dyer, and Noah A Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. *arXiv preprint arXiv:1508.00657*.

[Bengio et al.1994] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

[Bridle1990] John S Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer.

[Carrier and Cho2014] Pierre Luc Carrier and Kyunghyun Cho. 2014. LSTM networks for sentiment analysis. Deep Learning Tutorials.

[Chen et al.2015] Liang-Chieh Chen, Jonathan T. Barron, George Papandreou, Kevin Murphy, and Alan L. Yuille. 2015. Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform. *CoRR*, abs/1511.03328.

[Cho et al.2014] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

[Gers et al.2000] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

[Glorot et al.2011] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hochreiter et al.2001] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jfirgen Schmidhuber. 2001. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*, volume 1. IEEE.

[Kim et al.2015] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2015. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*.

[Kim2014] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[Krizhevsky et al.2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Ling et al.2015] Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

[Mesnil et al.2014] Grégoire Mesnil, Marc'Aurelio Ranzato, Tomas Mikolov, and Yoshua Bengio. 2014. Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. *arXiv preprint arXiv:1412.5335*.

[Sainath et al.2015] T.N. Sainath, O. Vinyals, A. Senior, and H. Sak. 2015. Convolutional, long short-term memory, fully connected deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4580–4584, April.

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

[Srivastava et al.2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

[Sundermeyer et al.2015] Martin Sundermeyer, Hermann Ney, and Ralf Schluter. 2015. From feedforward to recurrent lstm neural networks for language modeling. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 23(3):517–529.

[Tang et al.2015] Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432.

[Visin et al.2015] Francesco Visin, Kyle Kastner, Aaron C. Courville, Yoshua Bengio, Matteo Matteucci, and KyungHyun Cho. 2015. Reseg: A recurrent neural network for object segmentation. *CoRR*, abs/1511.07053.

[Werbos1990] P. Werbos. 1990. Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550–1560.

[Zeiler2012] Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

[Zhang et al.2015] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advanced in Neural Information Processing Systems (NIPS 2015)*, volume 28.