# News Text Classification with improved CNN and LSTM

Student
The Chinese University of Hong Kong, Shenzhen
XXX@cuhk.edu.cn
&

November 15, 2019

**Abstract**

Digital journalism has increased the accessibility to news articles drastically. The sheer amount of information is a critical challenges for readers. This paper reports on a series of experiments with 4 models (including convolutional neural networks (CNN), Long short-term memory(LSTM), Stack LSTM and Conv1D-LSTM combination) trained on top of pre-trained word vectors by GloVe for text-level classification tasks. Experiment on the full data set shows that a accuracy of 90% can be achieved with a one dimensional convolutional network architecture with max pooling layers and dropout. It also shows that Conv1D-LSTM combination has good performance on news text classification and is able to decrease training time.

## 1 Introduction

While digital journalism has increased the accessibility to news articles it also provides readers with constant flow of information, most of which is of little interest to readers. Text classification technique can be used to detect news topics from a great amount of news stream, helping people organize and utilize news information according to their preference. This paper is aimed at automatically classify news articles with several neural network models based on category labels,improving recommendations for readers and trying to beat the effect of previous work.

The input to our models are text bodies of articles from 20 news group.With the development of ImageNet, many researchers can greatly improve the performance of image recognition tasks by using deep neural networks and parameters pre-trained on ImageNet (such as ResNet50, DenseNet, etc.) as an input to other image recognition tasks. Correspondingly, word embedding, a distributed representation of words pre-trained on unlabeled text, has demonstrated a strong semantic representation.Inspired by this, we improve the input layer of networks by using pre-training word vector (GloVe embeddings) obtained from large corpus and loading it into a frozen Keras Embedding layer, thus improving the performance of networks in text classification. The network ends in a softmax output over 20 categories of news groups. We compare the performance of several models: (1) a Conv1D model, (2) a LSTM model, (3) a Stack LSTM model and (4) Conv1D-LSTM combination model.

Contribution of this paper is that Conv1D-LSTM combination is proved to perform well on news text classification as well as decrease training time, which combines the advantages of both CNN and RNN models, that is, extracting position-invariant features and keeping context dependencies. The model codes can be found on https://github.com/EveDong/News-Text-Classification-with-improved-CNN-and-LSTM.

### 1.1 Related Work

In literature, the state-of-the art NLP approaches often switched between CNNs and RNNs. CNNs are good at extracting position-invariant features like key phrases while RNNs are more appropriate for context dependencies. Which architecture performs better depends mostly on the importance of understanding the whole text sequence. Vu investigate CNN and basic RNN (i.e., no gating mechanisms) for relation classification. They report higher performance of CNN than RNN and give evidence that CNN and RNN provide complementary information: while the RNN computes a weighted combination of all words in the sentence, the CNN extracts the most informative ngrams for the relation and only considers their resulting activations. Standard neural network approaches for text classification involve using recurrent neural networks like LSTMs or one-dimensional CNNs on top of pretrained word vectors.

Besides,Recent studies show that CNNs often provide equivalent information for text classification. Convolutional neural networks for sentence classification written by Kim and Yoon has proved that CNN achieves an impressive effect in the field of text classification.

Other areas of research are hybrid models of CNNs and RNNs. CNN-RNN models are used to address the limitations of the individual models, where the CNN portion reduces the number of parameters and the RNN portion captures contextual information.

Hence, motivated by works above, CNN, LSTM and stack LSTM models are trained respectively in this paper, and thus one-dimensional CNNs is combined with LSTM, aiming at improve the accuracy of text classifier.

Worth mentioning, in November 2018, Google issue state-of-the-art NLP neural network BERT. It uses main structure of Transformer encoder, which use attention models to assess which parts of the text body are critical for the classification task.The final text representation is obtained by overlapping attention layers and nonlinear layers. Compared with CNN and RNN, Transformer can consider information in two directions comprehensively and compute parallel. In the future, it should replace CNN and RNN as the mainstream method in the field of NLP. However, the structure of BERT is too complicated and computational constraints limited training, so BERT and transformer is not used in this paper.

## 2 The Proposed Algorithm

### 2.1 Theory of CNN1D Model

Here is a breif introduction to CNN1D. For input Layer, Sequence x contains n tokens. Each token is represented by a d-dimensional dense vector; thus the input x is represented as a feature map of dimensionality $d \times n$. Convolution Layer is used for representation learning from sliding windows. For an input sequence with n tokens: $x_1, x_2, \ldots, x_n$, let vector $\mathbf{c}_i \in \mathbb{R}^{wd}$ be the concatenated embeddings of w tokens $x_{i-w+1}, \ldots, x_i$ where w is the filter width and $0 < i < s + w$. Embeddings for $x_i, i < 1$ or $i > n$, are zero padded. We then generate the representation $\mathbf{p}_i \in \mathbb{R}^d$ for the window $x_{i-w+1}, \ldots, x_i$ using the convolution weights $\mathbf{W} \in \mathbb{R}^{d \times wd}$ : $\mathbf{p}_i = Relu\left(\mathbf{W} \cdot \mathbf{c}_i + \mathbf{b}\right)$ where bias $\mathbf{b} \in \mathbb{R}^d$. For maxpooling, all window representations $\mathbf{p}_i (i = 1 \cdots s + w - 1)$ are used to generate the representation of input sequence x by maxpooling: $\mathbf{x}_j = \max\left(\mathbf{p}_{1,j}, \mathbf{p}_{2,j}, \cdots\right) (j = 1, \cdots, d)$.

### 2.2 Theory of LSTM Model

LSTM models the word sequence x as follows:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{x}_t \mathbf{U}^i + \mathbf{h}_{t-1} \mathbf{W}^i + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{x}_t \mathbf{U}^f + \mathbf{h}_{t-1} \mathbf{W}^f + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{x}_t \mathbf{U}^o + \mathbf{h}_{t-1} \mathbf{W}^o + \mathbf{b}_o\right) \\
\mathbf{q}_t &= \tanh\left(\mathbf{x}_t \mathbf{U}^q + \mathbf{h}_{t-1} \mathbf{W}^q + \mathbf{b}_q\right) \\
\mathbf{p}_t &= \mathbf{f}_t * \mathbf{p}_{t-1} + \mathbf{i}_t * \mathbf{q}_t \\
\mathbf{h}_t &= \mathbf{o}_t * \tanh\left(\mathbf{p}_t\right)
\end{aligned}$$

LSTM has three gates: input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$ and output gate $\mathbf{o}_t$. All gates are generated by a sigmoid function over the ensemble of input $\mathbf{x}_t$ and the preceding hidden state $\mathbf{h}_{t-1}$. In order to generate the hidden state at current step $t$, it first generates a temporary result $\mathbf{q}_t$ by a tanh nonlinearity over the ensemble of input $\mathbf{x}_t$ and the preceding hidden state $\mathbf{h}_{t-1}$, then combines this temporary result $\mathbf{q}_t$ with history $\mathbf{p}_{t-1}$ by input gate it and forget gate $\mathbf{f}_t$ respectively to get an updated history $\mathbf{p}_t$, finally uses output gate ot over this updated history $\mathbf{p}_t$ to get the final hidden state $\mathbf{h}_t$.

## 3 Experiments

### 3.1 Dataset Overview

We conduct experiments on two datasets: Wikipedia GloVe dataset Jeffrey Pennington et al(2014) and 20 newsgroup dataset Ken Lang(1997). 20 newsgroup dataset is a collection of 20,000 messages, collected from 20 different net news newsgroups. One thousand messages from each of the twenty newsgroups were chosen at random and partitioned by newsgroup name. 20000 news articles are labeled with 20 categoriesWe split the dataset, 0.8 for training and 0.2 for validating. The Wikipedia GloVe dataset contains 400K uncased vocabulary and each vocabulary is represented by a 100d vector. GloVe stands for "Global Vectors for Word Representation",which is a popular embedding technique based on factorizing a matrix of word co-occurence statistics. We use the GloVe dataset to impose a GloVe embedding.

### 3.2 Data Preprocessing

The 20 newsgroup data came in various formats and required different levels of preprocessing. We applied the following cleaning tokenizing steps: 1. Go through the whole texts in the 20 newsgroup dataset and build a dictionary based on word frequency. 2. Remove all the punctuation,tabs and line breaks, and turn the texts into space-separated sequences of words (words maybe include the ' character). These sequences are then split into lists of tokens. We set only the most common 20,000 words are kept based on word frequency. 3. Vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary). 0 is a reserved index that won't be assigned to any word. 4. Pads sequences to the same length. We transform a list of 20000 sequences (lists of integers) into a 2D Numpy array of shape (20000, 1000). 1000 is the length of the longest sequence. Sequences that are shorter than 1000 are padded with value at the end. Sequences longer than 1000 are truncated. Then we vectorize the labels by one-hot code and we can thus converts a class integers to binary class matrix(20000*20dim).After that, we load pre-trained word embeddings into an Embedding layer: 1. Build the embedding matrix. Search the most common 20000 words in the GloVe dataset. Words found will be replaced by the corresponding 100-d vectors. Words not found will be all-zero vectors. 2. We set 'trainable = False' so as to keep the embeddings fixed and turns positive integers (indexes) into dense vectors of fixed size.At last, we modify a sequence in-place by shuffling its contents and split the data into a training set(0.8) and a validation set(0.2).

### 3.3 Baseline

For our baseline we used a simple softmax regression. An average ($\mathbf{d}_{avg}$) was found across all word vectors m an article to get a smgle 50 dimensional vector representmg that article, $\mathbf{d}_{avg} = \frac{1}{N}\sum i = 1w_i$, where N is total number of words and $w_i$, the GloVe vector for word ı. This vector was then fed into an output layer with a softmax activation function to obtain predictions. All models introduced in this section were trained using categorical cross-entropy loss with rmsprop optimizer and evaluated with an accuracy metric (% of correctly classified articles). The reasons behind choices, advantages and limitations of models are described in Chapter 4. Illustrations of model architectures can be found in Figure 2-5 in Appendix.

### 3.4 Conv1D Convolutional Neural Network

Our Conv1D model consisted of 3 one-dimensional convolutional layers, each with 128 filters (size=5, stride=1). To reduce the dimension of the parameter space, a max-pooling layer followed each convolutional layer (size=5, stride=1). The output was then flattened and fed into a fully connected layer of 128 hidden units with ReLu activation, followed by an output layer with softmax activation for final predictions. Dropout was used to address overfitting. The hyperparameters we trained for this model were learning rate, sequence length, filters and dropout rate. For the best model we used a learning rate of 0.001and dropout rate of 0.5.

### 3.5 LSTM Recurrent Neural Network

The LSTM model was a 2 layer model where each LSTM cell had a hidden size of 128. The LSTM model was trained to predict the class of a sequence of 1000 words from articles. The output from the LSTM model was fed into a fully connected layer with a softmax activation to obtain predictions. To prevent overfitting, we used dropout for regularization. The hyperparameters we trained for this model were learning rate, size of LSTM cells, sequence length and dropout rate. For the best model we used a learning rate of 0.001 and dropout rate of 0.5.

### 3.6 Stacked LSTM

In the stacked LSTM model, we stack 3 LSTM layers with on top of each other, making the model capable of learning higher-level temporal representations. We set each LSTM cell had a hidden size of 128. The first two LSTMs return their full output sequences, but the last one only returns the last step in its output sequence, thus dropping the temporal dimension (i.e. converting the input sequence into a single vector). The output from the LSTM model was fed into a fully connected layer with a softmax activation to obtain predictions. To prevent overfitting, we used dropout for regularization. The hyperparameters we trained for this model were learning rate, sequence length, size of LSTM cells and dropout rate. For the best model we used a learning rate of 0.001 and dropout rate of 0.5.

### 3.7 Conv1D-LSTM Combination

In order to combat limitations of LSTM and Conv1D we also considered a combination of the two models. The Conv1D-LSTM contained 2 one-dimensional convolution layers followed by max-pooling layers. The convolutional layers had 128 filters of size 5 and the max-pooling layers were of size 5 with stride of 1. The output from the convolutional part was fed into 1 layer of LSTM cells (of size 128) followed by an output layer with softmax activation for final predictions. Dropout was used to address overfitting. The hyperparameters we trained for this model were learning rate, sequence length, filters and dropout rate. For the best model we used a learning rate of 0.001 and dropout rate of 0.5.

## 4 Results and Discussion

### 4.1 Hyperparameter Tuning

An essential part in deep learning is it to find a set of hyperparameters that maximize the chosen metric (in our case accuracy). For our models we generally tuned the hyperparameters in the following sequence: learning rate, sequence length, architecture (#hidden units, #layers, #filters) and dropout rate. The best hyperparameters for each model are reported when describing the model architectures in Chapter 3. Overall the most sensitive hyperparameters were the learning rate and sequence length.

### 4.2 Performance of Models

From the table above we can see that, the baseline was able to predict with 56.7% accuracy, which is little better than random guessing. Because of the sequential text structure of news articles, our first attempt to improve the accuracy of the classification task was to implement a LSTM recurrent network. LSTM work well for inputs with contextual dependencies but take a very long time to train. We achieved an accuracy of 74.1%. In order to speed up training, we next considered a convolutional network (Conv1D). The Conv1D model took a short time to train, allowing us to hyperparameter tune it, and gave surprisingly high accuracy of 70.9%.

| Model | Training set | | Validation | | Epoch | Time |
|---|---|---|---|---|---|---|
| | Loss | Accuracy | Loss | Accuracy | | |
| Baseline | 0.7628 | 0.5406 | 1.7406 | 0.5674 | 100 | 5394.78s |
| Conv1D | 0.2489 | 0.9160 | 1.7905 | 0.7094 | 100 | 15848.26s |
| LSTM | 0.4162 | 0.8482 | 0.9692 | 0.7409 | 100 | 20314.72s |
| Stacked LSTM | 0.3541 | 0.8819 | 1.2440 | 0.6809 | 100 | 40026.06s |
| Conv1D+LSTM | 0.2735 | 0.9015 | 1.0055 | 0.7972 | 100 | 15497.79s |

After that we stack the LSTM to make the model capable of learning higher-level temporal representations. But we find that the accuracy is lower, just 68.1% and the training time is extremely long, reaching twice as long as LSTM. The fact that the Conv1D and the LSTM have similar accuracy can be accounted to the fact that the LSTM only only considers context in the vicinity but cannot establish contest over the course of thousands of words. Lastly, we explored if we could combine the best from both models, by reducing the input of the LSTM through a Conv1D architecture. The combined hypertuned architecture showed a higher performance of 80% and the traing time is similar to Conv1D, which is far less than LSTM and stacked LSTM.

## 5 Reference

[1] 20 newsgroup Dataset. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html

[2] Wikipedia GloVe Dataset. http://dumps.wikimedia.org/enwiki/20140102/

[3] Pennington J, Socher R, Manning C. Glove: Global vectors for word representation[C]. //Proceedings of the 2014 conference on empirical methpre-trained GdoVinembedding as inputs to our models.20-2273.

[4] Lai S, Xu L, Liu K, et al. Recurrent Convolutional Neural Networks for Text Classification[C]. //AAAI. 2015, 333: 2267-2273.

[5] Yin W, Kann K, Yu M, et al. Comparative study of cnn and rnn for natural language processing[J]. arXiv preprint arXiv:1702.01923, 2017.

[6] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in Neural Information Processing Systems. 2017: 5998-6008.

[7] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[8] Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised Sequence Learning. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15). MIT Press, Cambridge, MA, USA, 3079–3087.

[9] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic Compositionality through Recursive Matrix-Vector Spaces. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics, Jeju Island, Korea, 120111211.

[10] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Doha, Qatar, 1746–1751.

## Appendix

pipeline.png

Figure 1: Pipeline

Conv1D.png

Figure 2: Conv1D Convolutional Neural Network

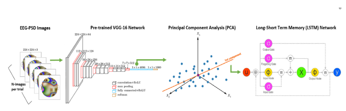

Fig. 3. Network architecture for EEG-PSD trend based deep-learning method.

Figure 3: LSTM Recurrent Neural Network

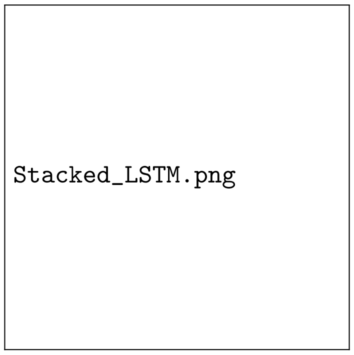

Figure 4: Stacked LSTM

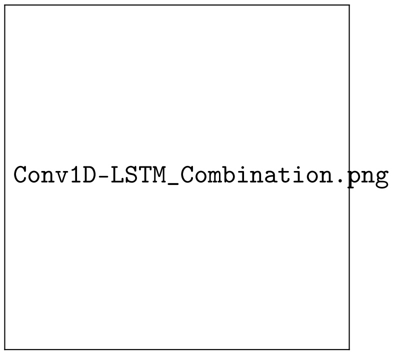

Figure 5: Conv1D-LSTM Combination