Exploring Neural Models for Text Classification

Alexander Shypula
Carnegie Mellon University
Language Technologies Intitute
ashypula@cs.cmu.edu

February 10, 2020

Abstract

Text classification has long remained an important task in natural language processing (NLP). In the following study, convolutional neural network (CNN), recurrent neural network (RNN), attention-based models, as well as ensembling methods are explored to address a text classification task. After exploring over 70 models, a multi-filter convolutional neural network architecture based on [2] Kim 2014 was shown to be the most superior architecture with an accuracy of **87.01%** on the validation set. An ensemble of 9 different models further increased accuracy on the validation set to **88.49%**.

1 Introduction

Text classification has long been a traditional task in the field of machine learning, for example classifying e-mails as spam is typically introduced to students when teaching the naïve bayes classifiation algorithm. Neural network architectures have shown a tremendous amount of success in addressing the complicated nature of natural language, from tasks ranging from language modeling, named entity recognition, semantic role labeling, machine translation, to sentiment analysis, etc. The success of neural networks for natural language processing has stemmed both from the increase in the processing power of modern computers, the increase in amounts of training data, as well as novel methods of providing neural networks with effective inductive bias.

Core to these innovations have been the use of unsupervised word embeddings ([1] Collobert et al. 2011) and ([11] Mikolov et al. 2013), convolutional filters ([12] Waibal et al. 1989), recurrent network gating functions ([7] Hochreiter and Schmidhuber 1997), attention ([6] Bahdanau et al. 2014), and the transformer ([8] Vaswani et al. 2017). The following study utilizes pre-trained word embeddings with all the following methods for the task of text classification.

2 Related Work on Text Classification

The area of text classification within NLP has been an active area of research. In [2] Kim 2014, superior results were achieved by utilizing a modified architecture based on [1] Collobert et al. 2011 in which multiple convolutional filters were passed over pre-trained word embeddings, max-pooled over time, and then passed through a fully-connected network. The authors also utilized other regularization methods such as constricting the L2 norm of the weights and utilizing dropout ([13] Srivastava et al. 2014). [10] Kowsari et al. 2018 introduce Random Multimodal Deep Learning for Classification (RMDL) which ensembles multiple random CNN, RNN, and multi-layer perceptron networks using majority voting for superior classification results. [9] Yao et al. 2018 take a different approach, treating text within a corpus as a graph with document and word nodes, and then training a graph convolutaional neural network for text classificaiton tasks. Lastly, [15] Yang et al. 2020 explore XLNet, a transformer-based model yielding state of the art results.

3 Method

3.1 Pre-Trained Word Embeddings

In all of the following experiments pre-trained fastText word embeddings ([4] Mikolov et al. 2017) were used to represent words as vectors in 300 dimensional space. The word embeddings capture distributional properties of words, in this case, through the CBOW model. The CBOW model minimizes the following loss function in which $s(w, C)$ is a scoring function between an individual word w_t the CBOW model is trying to predict and the remaining context C within the sliding window. The left-hand side is the positive sample, while all examples $n \in N$ are negative samples of n, C pairs.

$$\sum_{t=1}^{T} \left[\log(1 + e^{-s(w_t, C)}) + \sum_{n \in N} \log(1 + e^{s(n, C)}) \right]$$

The scoring function is defined as the average dot-product between each of the predicted words in the context window $w' \in C$ represented as $u_{w'}$ and the vector representation of the context v_w.

$$s(w, C) = \frac{1}{|C|} \sum_{w' \in C} u_{w'}^T v_w$$

The vector representation of the context for word w referrred to as v_w is calculated as a sum of haddamard products between position vectors and the corresponding word embeddings. Each position in the sliding context window has its own vector d_p for all positions $p \in P$ within the sliding window excluding the target word.

The context is the sum of the haddamard products of each position vector with the corresponding word embedding $u_{p'}$ thus creating the continuous bag of words (CBOW).

$$v_w = \sum_{p \in P} d_p \odot u_{p'}$$

Regarding the specific implementation details, each of the following networks in this study relied on converting text into a 2D matrix of word embeddings over time. Each batch was zero padded to the length of the maximum sequence in the batch. Word vectors not in the fastText vocabulary were initialized to zero. The embedding matrix was formed by choosing all words within both the train, validation, and test corpus: this is justified by the fact that the purpose of word embeddings is to learn representations for a large vocabulary that may generalize to words in the validation and test sets that do not appear in the train corpus. Had the validation and test sets been masked from these experimental setup, the same results could be reproduced by utilizing the entire 2 million fastText embeddings as an embedding layer lookup table; however, such an architecture would have been inconvenient for this experimental setup. Regardless, no supervised training or unsupervised training occurred on either the validation or test sets. Subword information fastText vectors were not used for the following experiments.

3.2 Convolutional Neural Networks

Following an architecture inspired by [1] Collobert et al. 2011, [2] Kim 2014 shows a multi-filter CNN followed by max-pooling can provide superior text classification. Some different choices in utilizing CNNs for text classification include deciding to make word embeddings static, adding batch-normalization, utilizing dropout, varying the width of the convolutional filters, varying the amount of CNN output channels, and utilizing convolutional filters of different size in parallel. In addition to these more traditional methods, convolutional layers can be stacked upon one-another wherein residual skip-connections ([14] He et al. 2015) may be added to stymie the exploding and vanishing gradient problem characteristic of deep neural networks.

The first set of experiments consisted of a convolutional neural network with only one filter size. Experiments were run with differing numbers of layers, differing numbers of output channels, and differing values for dropout. In this set of experiments the most superior results came from the combination of a filter of width 2 with 64 output channels, batch-norm, and no dropout. Adding multiple layers and more than 64 output channels did not improve classification performance; however, had dropout been used for these multi-layer and higher-capacity networks, it is plausible their performance may have been better.

The convolutional layers were applied with a ReLU activation and max-pooled over time. If batch-normalization was utilized, it was applied after the convolutional operation before the non-linearity was applied. Afterwards, dropout

was applied to the output of the max-pool and then fed into one fully-connected layer with 16 output units.

Model List				
Model	Filter Width	Channels	Dropout	Accuracy
Conv1d - static	2	64	0.0	81.95%
Conv1d - static, bn	2	64	0.0	**85.22%**
Conv1d - static, bn	2	64	0.1	84.29%
Conv1d - static, bn	2	64	0.2	84.60%
Conv1d - static, bn	2	96	0.0	84.45%
Conv1d - static, bn	2	128	0.0	85.07%
Conv1d - static, bn	3	64	0.0	83.82%
Conv1d - static, bn, 2 layer	2	64	0.0	84.44%
Conv1d - static, bn, 2 layer w/ skip-connection	2	64	0.0	82.73%
Conv1d - static, bn, 3 layer w/ skip-connection	2	64	0.0	83.20%

Another set of experiments were run to test the effectiveness of non-static embeddings. Because non-static embeddings seemed to cause over-fitting, another technique was added provided by [3] Goldberg 2017 in which the embeddings $\mathbf{E}$ were kept constant, but another matrix with identical dimension to the embeddings matrix was created $\mathbf{\Delta}$ where the final embedding used in the classification task was $\mathbf{E}' = \mathbf{E} + \mathbf{\Delta}$. To prevent further overfitting to the training data, the cross-entropy loss function was modified to penalize the weights of $\mathbf{\Delta}$, where λ is a parameter used to control the level of normalization.

$$\mathcal{L} = \sum_i p_i \log \hat{p_i} + \lambda \left\| \Delta \right\|_2$$

It was shown; however in experiments that adding the matrix $\mathbf{\Delta}$ did not improve results. All the following models were based off a CNN model with a filter-width of 2 with 64 output channels, batch-normalization, and no dropout. It could be justified that the non-static training caused the embeddings in the train set to over-fit on the classification task; and as a result, what was learned did not generalize to the remaining words the validation set not present in the training set.

Effect of Non-Static Emebeddings			
Model	λ	Dropout	Accuracy
Conv1d - differentiable **E**	-	0.0	**83.51%**
Conv1d - differentiable **E**	-	0.2	**83.51%**
Conv1d - differentiable $\mathbf{\Delta}$	0.0	0.0	79.32%
Conv1d - differentiable $\mathbf{\Delta}$	0.0001	0.0	77.76%
Conv1d - differentiable $\mathbf{\Delta}$	0.001	0.0	77.14%
Conv1d - differentiable $\mathbf{\Delta}$	0.01	0.0	76.83%
Conv1d - differentiable $\mathbf{\Delta}$	0.1	0.0	81.80%
Conv1d - differentiable $\mathbf{\Delta}$	1.0	0.0	82.27%

Another set of experiments were run in which filters of different width were used simultaneously similar to the architecture used in [2] Kim 2014. Adding batch-normalization did not seem to help in the classification task; however, adding filters of different size with a high amount of network capacity yielded the best results of any individual model tried in the remainder of this study. The number of channels reported is the sum of all output channels from all of the filters.

Model List with Multiple Filters				
Model	Filter Widths	Channels	Dropout	Accuracy
Conv1d - static	2,3,4,5	64	0.0	85.38%
Conv1d - static	2,3,4,5	128	0.0	85.07%
Conv1d - static	2,3,4,5	512	0.0	86.31%
Conv1d - static	2,3,4,5	512	0.3	86.78%
Conv1d - static	2,3,4,5	512	0.5	**87.09%**
Conv1d - static, bn	2,3,4,5	512	0.3	86.63%
Conv1d - static	2,3,4,5,6,7	516	0.5	86.46%

3.3 Recurrent Neural Networks

[5] Sutskever et al. 2014 show that a multi-layer LSTM recurrent neural network with an acceptor architecture was able to obtain strong results for machine translation, especially when reversing the input sentence, using a large beam size, and ensembling multiple LSTM networks together.

In the following experiments, a bi-directional LSTM acceptor was used to encode the input text. Where h_L is the last hidden state of the left-encoder in the penumtilate LSTM layer and h_R is the last hidden state of the right-encoder in the penumtilate LSTM layer the concatenated vector $[h_L, h_R]$ was then fed as the input to a final fully-connected layer. Another experiment was also ran to test if adding a non-linear transformation to the sentence encoding $[h_L, h_R]$ via the tanh function would improve results. Although fewer experiments were run compared to other types of models, the bi-directional LSTM acceptor models surprisingly

did not vary much in their classification performance on the validation set and usually performed quite well.

Bi-Directional LSTM Acceptor Results				
Model	Hidden Size	Number Layers	Dropout	Accuracy
LSTM - static	256	1	0.0	85.38%
LSTM - static	256	1	0.3	85.38%
LSTM - static w/ non-linearity	256	1	0.0	85.07%
LSTM - static w/ non-linearity	256	1	0.3	85.07%
LSTM - static	256	2	0.0	85.07%
LSTM - static	512	1	0.0	85.38%
LSTM - static	512	1	0.3	**86.31%**
LSTM - static	512	1	0.5	85.54%

3.4 Attention-Based and Transformer-Encoder Based Networks

Attention ([6] Bahdanau et al. 2014) has been shown to produce superior results in numerous tasks in NLP ([15] Yang et al. 2020). One key innovation in attention has been the use of attention-only models, such as the transformer ([8] Vaswani et al. 2017).

In the following experiments, a self-attention based MLP architecture as well as the transformer encoder from [8] Vaswani et al. 2017 were used to create representations of the text. In both models, the word embeddings were passed through one or more layers of attention / transformer blocks. The output embeddings were either max-pooled over time, weight-pooled using the attention weights, or both were concatenated before being fed into a fully-connected layer.

The self-attention based MLP was based on code from the torch-nlp repository. Unfortunately, no reference paper for this attention implementation could be found, but the equations for computing the attention block are below. In the following equations d represents the number of dimensions of the input embeddings, l represents the length of the text sequence, $\mathbf{E}$ represents the word embeddings, and W_q and W_{out} are learned parameters:

$$\mathbf{E} \in \mathbb{R}^{lxd}$$

$$W_q \in \mathbb{R}^{dxd}$$

$$W_{out} \in \mathbb{R}^{lx2d}$$

$$Q = W_q \cdot E^T$$

$$W_{\widetilde{A}} = Q \cdot E^T$$

$$W_{A_{i,j}} = \frac{e^{\widetilde{A}_{i,j}}}{\sum_j e^{\widetilde{A}_{i,j}}}$$

$$A = W_A \cdot Q$$

$$Out = \tanh\left(W_{out} \cdot [A, Q]\right)$$

Compared the the RNN networks, the self-attention based methods varied far more in their performance. Additionally, pooling the attention encodings using the attention weights themselves performed well; however, concatenating the attention pooling output with the max-pooling output seemed to have no positive impact on the classification performance. In these experiments, a high amount of dropout was usually included, as the models had a large amount of capacity and were able to over-fit on the train data. While adding a second self-attention block led to the best performance; utilizing three and four blocks actually degraded performance despite the use of residual connections between attention block layers.

Self-Attention Experiments			
Model	Attention Blocks	Dropout	Accuracy
Self-Attention, max-pool	1	0.0	84.60%
Self-Attention, max-pool	1	0.3	84.29%
Self-Attention, max-pool	1	0.5	84.60%
Self-Attention, max-pool	2	0.3	83.35%
Self-Attention, max-pool	2	0.5	**86.00%**
Self-Attention, max-pool	3	0.5	84.60%
Self-Attention, max-pool	4	0.5	82.89%
Self-Attention, attn pool	2	0.3	85.38%
Self-Attention, attn pool	2	0.5	84.60%
Self-Attention, attn pool & max-pool	2	0.3	84.60%
Self-Attention, attn pool & max-pool	2	0.5	84.29%

In addition to the self-attention experiments shown above, numerous experiments were run with the attention encoder from [8] Vaswani et al. 2017. Similar to the self-attention experiments, a large amout of droupout was often used as the transformer models had high capacity and were capable of overfitting the training data.

Transformer(Tfmr) Encoder Results					
Model	Attn Heads	Tfmr Width	Tfmr Dropout	Penultimate Dropout	Acc
1 Tfmr Block, max-pool	4	512	0.1	0.5	85.38%
1 Tfmr Block, max-pool	4	1024	0.1	0.2	84.76%
1 Tfmr Block, max-pool	4	1024	0.1	0.5	**85.84%**
1 Tfmr Block, max-pool	4	1024	0.3	0.5	84.60%
1 Tfmr Block, max-pool	4	2048	0.1	0.5	84.91%
1 Tfmr Block, max-pool	10	2048	0.1	0.5	85.36%
2 Tfmr Blocks, max-pool	10	2048	0.1	0.5	82.89%

Three additional experiments were run feeding the transformer output into a CNN w/ filter sizes of 2,3,4,5 with a total of 512 output channels as well as in parallel with the same CNN wherein the max-pooled transformer output was concatenated with the max-pooled CNN output. It was interesting to see that while excessive layers in the self-attention and transformer experiments led to degraded performance; additional convolutional operations on top of the transformer encoding actually marginally increased performance.

Transformer(Tfmr) Encoder + CNN Results					
Model	Attn Heads	Tfmr Width	Tfmr Dropout	Penultimate Dropout	Acc
1 Tfmr Block + (2,3,4,5) conv1d filters on top	4	1024	0.1	0.5	**86.00**%
1 Tfmr Block, max-pool & (2,3,4,5) conv1d filters in parallel	4	1024	0.1	0.5	84.91%
1 Tfmr Block, max-pool & (2,3,4,5) conv1d filters in parallel	4	1024	0.1	0.8	85.38%

3.5 Ensembling Results

[10] Kowsari et al. 2018 show that ensembling models from CNNs, RNNs, and MLPs together using simple majority voting can lead to superior performance on text classification tasks without having to introduce a great deal of complexity. In the following experiments, MLPs and interpolation of predicted results were used to ensemble CNN, RNN, and attention-based models. It was shown that all MLP-based ensembles performed even worse than the baseline CNN model (87.09%). The MLP-based ensembles performed better when the text encodings (before penultimate fully-connected layer) were used instead of the logit outputs. Nonetheless, all interpolation methods performed above the baselines. Of all ensembles, a log-linear interpolation ensemble of 9 highly-performing models achieved the highest classification accuracy of 88.49%.

Ensemble Results		
Number Models Used	Method	Accuracy
3	1 layer MLP on logits	81.26%
3	3 layer MLP on logits	83.83%
3	3 layer MLP on logits w/ 0.3 dropout	82.42%
3	3 layer MLP on text encoding w/ 0.2 dropout	85.54%
3	3 layer MLP on text encoding w/ 0.5 dropout	86.93%
3	5 layer MLP on text encoding w/ 0.5 dropout	86.63%
3	linear interpolation of model logits	88.02%
3	linear probability interpolation	87.87%
3	log-linear probability interpolation	88.02%
9	linear probability interpolation	87.87%
9	log-linear probability interpolation	**88.49**%

4 Conclusion

In this study, CNNs, RNNs, attention-based and transformer-encoder based networks were explored for a text classification task. While CNNs ultimately outperformed all other models (87.09%), all three classes of models exhibited strong performance. Additionally, numerous ensembling techniques were also explored wherein all MLP-based ensembles underperformed the individual model baseline; however, all interpolation methods improved upon the the baseline where the best ensemble had an accuracy of 88.49% on the validation set.

References

[1] Collobert, R. & Jason W. & Bottou L. et al. (2011) Natural Language Processing (Almost) from Scratch. In *Journal of Machine Learning Research 12 (2011)*, pp. 2493-2537.

[2] Kim Y. (2014) Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* , pp. 1746–1751.

[3] Goldberg Y. (2017) Neural Network Methods for Natural Language Processing. In *Synthesis Lectures on Human Language Technologies (2017).*

[4] Mikolov T. & Grave E. et al. (2017) Advances in Pre-Training Distributed Word Representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)* .

[5] Sutskever I. & Vinyals O. & Le Q. (2014) Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27 (NIPS 2014).*

[6] Bahdanau D. & Cho K. & Bengio Y. (2014) Neural Machine Translation by Jointly Learning to Align and Translate. In *Second International Conference on Learning Representations (ICLR 2014).*

[7] Hochreiter S. & Schmidhuber J. (1997) Long Short-Term Memory. In *Neural Computation 9(8)*, pp. 1735-1780.

[8] Vaswani A. & Shazeer N. et al. (2017) Attention is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017)*, pp. 6000–6010.

[9] Yao L. & Mao C. & Luo Y. (2018) Graph Convolutional Neural Networks for Text Classification. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*.

[10] Kowsari K. & Heidarysafa M. et al. (2018) RMDL: Random Multimodal Deep Learning for Classification. In *Proceedings of the 2nd International Conference on Information System and Data Mining (CISDM 2018)*, pp. 19-28.

[11] Mikolov T. & Sutskever I. et al. (2013) Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS 2013)*, pp. 3111–3119.

[12] Waibal A. & Hanazawa T. & Hinton G. et al. (1989) Phoneme Recognition Using Time-Delay Neural Networks. In *IEEE Transactions on Acoustics, Speech, and Signal Processing vol. 37 no. 3*, pp. 328–339.

[13] Srivastava N. & Hinton G. & Krizhevsky A. & Sutskever I. & Salakhutdinov R. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In *Journal of Machine Learning Research 15 (2014)*, pp. 19291958.

[14] Kaiming H. & Zhang X. & Ren S. & Sun J. (2015) Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*.

[15] Yang Z. & Dai Z. & Yang Y. & Carbonell J. & Salakhutdinov R. & Le Q. (2020) XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv Preprint: arXiv:1906.08237*.