11-747 Assignment 1: Text Classifier

Artidoro Pagnoni
apagnoni@cs.cmu.edu

February 11, 2020

1 Introduction

Text classification is a classic topic in natural language processing, and consists of assigning pre-defined labels to a series of variable-length texts. In this project we compare the performance of several varieties of text classifiers. We explore neural network approaches using continuous bag-of-word (Mikolov et al. (2013)) with embeddings, convolutional neural networks (Kim (2014)) as well as long short-term memory (LSTM) networks (Hochreiter and Schmidhuber (1997)), with and without self-attention. We also use meta learning techniques such as ensembling to improve prediction results.

2 Problem Description

The problem of interest in the present work is predicting the topic of a given text. This is a multiclass classification task with labels with 16 labels. Some examples of the labels are: "Sports and recreation", "Music", and "History". In the dataset we are given input sentences $\boldsymbol{X}$ and labels y. The input sentences have varying lengths and are composed of words from a vocabulary $\mathcal{V}$ containing 120634 words in total. The label set of the language can only take 16 values $\mathcal{T} = \{i\}_{i=1}^{16}$. Our task is to use the training data $\{(\boldsymbol{X}, y)\}^n$ to predict the labels of unseen sentences.

We propose 7 models for this task that we optimize through Maximum Likelihood Estimation MLE. We have experimented both with fastText and Glove embeddings, finally opting for fastText which gave better validation results. We use an embedding matrix $E \in \mathbb{R}^{\|\mathcal{V} \times d\|}$, where d is the embedding vector length for each word. We use the pre-trained word embeddings for all models, with embedding dimension $d = 300$.

3 Models and Algorithms

We explored with various models for this task. For all the model we used `CrossEntropyLoss`, which combines a *LogSoftmax* layer and the *NLLLoss* in one single class (This approach proved to be more numerically stable than using the two in separate layers).

3.1 Continuous Bag-of-Words

We use a variant of the CBOW technique proposed in Mikolov et al. (2013), based on the intuition that words can have multiple degrees of similarity that can be captured in continuously distributed, d-dimensional vectors. Instead of using CBOW to predict a current word, as described in the original paper, we simply leverage the pre-trained embeddings, summing over the sentence's word vectors and feeding the result through a final fully connected layer W_{fc}.

$$P(y|\boldsymbol{x}) = \text{softmax}\left(W_{fc}^{T}\sum_{t=1}^{T} E(x_t)\right)$$

Where E is the function that maps a word x_t to it's embedding $\boldsymbol{e}_{x_t}$.

3.2 Convolutional Neural Network (1, 2 layers)

CNN 1, our single layer CNN, is an attempt to replicate the results presented in Kim (2014). As in the CNN-multichannel presented in the paper, we maintain two sets of 300 dimensional word vectors, only one of which gets gradients backpropagated through during training. The convolutional layer consists of 100 channels each of filters of kernel sizes 3, 4, 5. We use max pooling over time and ReLU activations. This is followed by a single fully-connected layer and a softmax output. For regularization, we also employ dropout on the final fully-connected layer, a cap of 3 on the weights of that layer, and early stopping. Differences include initializing word vectors with fastText as opposed to word2vec, training with mini-batch size of 64, and using Adamax as opposed to Adadelta.

CNN 2, our two layer CNN, is an extension of CNN 1. The motivation behind this extension was that having deeper convolutional layers would widen the receptive field for learnable features in the convolutional layers. Intuitively, the hope is that the first convolutional layer followed by pooling would aggregate information among groups of contiguous words, and the second convolutional layer may learn higher level information about the sentence for the text classification task. This model maintains the multichannel property (two sets word vectors, only one of which is trained). The first convolutional layer contains 32 channels each of filters of kernel sizes 3, 4, 5, followed by a max pooling layer of kernel size 3 and ReLU activation. The second convolutional layer contains 64 channels each of filters of kernel size 3, and is followed by ReLU and max pooling over time. Finally, we again have a fully connected layer and softmax output.

3.3 Long Short-Term Memory Network

Following the motivation behind the use of CNN in Kim (2014), we experimented with LSTM models. Word embeddings are able to capture word specific characteristics and provide great representations for the words, however they lack the ability to harness the context in which they appear in the sentence. Our objective using CNNs and LSTMs was to produce word representations that were able to take into account the greater context of the sentence, and have a large receptive field. LSTMs have been used in many language applications such as speech recognition, and language modeling, and have demonstrated that they can produce representations that are dependent on their context. Others have successfully experimented with the use of LSTM in the context of text classification as in Qian et al. (2017) which further supports our intuition in using this model.

We propose as single layer LSTM. To obtain dependence from both past and future words in the sentence we employ bidirectional LSTMs outputting 200 dimensional hidden representations of the words $\boldsymbol{h}_i$. The variable number of word representations $\boldsymbol{h}_i$ is then combined into a single 200 dimensional vector $\hat{\boldsymbol{h}}_i$ with a max pooling overtime layer and ReLU activation. This is again followed by a single fully-connected layer and a softmax output. For regularization, we follow Kim (2014) and use dropout on the final fully-connected layer, a cap of 3 on the weights of that layer, and early stopping. Training is done with Adamax.

3.4 LSTM-CNN

The LSTM-CNN model is a combination of our basic single layer LSTM model and our CNN model. Intuitively, convolutions captures local contiguous group structure while LSTMs seem to maintain a more global memory. We combine these in hopes of capturing the advantages of both. LSTMs and CNNs have different approaches to increasing the receptive fields of the word representations, and we explored whether combining these approaches also brought improved representations. The model is composed of a single convolution layer consisting of 64 channels and three different kernel sizes of 3,4,5 on top of a bidirectional LSTM layer with hidden state size 64. This is followed by max pooling over time, a fully connected layer, and softmax output. We use similar training and regularization methods as above.

3.5 LSTM-ATTN

The LSTM-ATTN model enhances our basic bidirectional LSTM model with self attention. The intuition for using self attention is to capture the contextual meaning of each word. This can also be achieved through the LSTM alone, however, using self-attention the signal is more direct since we take the linear combination of word embeddings, and use the hidden state of the LSTM as the keys and queries. We use dot product attention between the LSTM outputs. We the take the linear combination of the embeddings to get the contextualized representation of each word. For word x_j, the attention coefficient with word x_i is given by:

$$\alpha_{ij} = \boldsymbol{h}_i^T \boldsymbol{h}_j$$

Where $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are the outputs of the LSTM for words x_i and x_j respectively. The contextual representation of word x_i is then given by linear combination of the embedding vectors $\boldsymbol{e}_j$ for all words in the input text:

$$\boldsymbol{c}_i = \sum_{j=1}^{T} \alpha_j \boldsymbol{e}_j$$

The contextual representations $\boldsymbol{c}_i$ are then combined into a single vector using a max pooling over time. Then we use a fully connected layer, and softmax to get probabilities over the labels. We use similar training and regularization methods as above.

3.6 Training Tricks

To handle out-of-vocabulary words, we reduce the vocabulary to only contain the words that appear at least twice in the training corpus. Furthermore, we set the embedding vector for the out-of-vocabulary words <unk> to the average of the pretrained fastText embeddings.

In terms of regularization, for all our models we use dropout with probability 0.5 and normalize the norm of the last fully connected layer to 3.

Our learning rate is 0.001 and batch size is 64. We employ a learning rate scheduler with patience 3 for the optimizer Adamax which reduced by half the learning rate if the validation loss does not decrease for three consecutive epochs.

3.7 Ensemble

The final submission was an ensemble of the CNN 1, CNN-2, LSTM-ATTN, and LSTM-CNN models, taking the average of the probabilities predicted by those models. We chose one model from each class in the hope that the different approaches with which these models are able to capture contextual information will be combined. Ensembling helps to reduce variance and combines the benefits of the various models involved. This achieved validation accuracy of 0.86158.

4 Experiments

Table 1 shows the training, validation, and test accuracies of the aforementioned models.

Model	Train Acc.	Validation Acc.
CBOW	0.81359	0.81804
LSTM	0.84621	0.81648
LSTM-ATTN	0.84115	0.82270
LSTM-CNN	0.82753	0.84136
CNN-2	0.87357	0.84914
CNN-1	0.85122	0.85536
ENSEMBLE	0.85723	**0.86158**

Table 1: Table with the main results.

5 Conclusion

This work involved implementing and experimenting with various models for text classification. We find that the CNN basline performs best. It seems clear that the CNN is a good inductive bias for capturing contextual information that is useful for text classification. All models that included the CNN performed better than the ones that did not. A notable challenge was managing overfitting in the larger neural network models. Following Kim (2014), we employed a combination of dropout, early stopping, and capping the norm of some parameter weights. Additionally, in

our extensions, we reduced hidden layer sizes as we made our models deeper and observed clear overfitting with a few training epochs.
Potential extensions include trying Transformer Encoders and further hyperparameter tuning. We tuned the hyperparameters to give the best performance with the baseline CNN model and did not do a comprehensive search for the other models.

References

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Qian, Q., Huang, M., Lei, J., and Zhu, X. (2017). Linguistically regularized LSTM for sentiment classification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1679–1689.