LING83600
Final Project

An attention-based BLSTM for sentiment classification

Sean Miller
smiller6@gradcenter.cuny.edu

1 Introduction

As technology and hardware have advanced, newer methods have become available in the field of natural language processing (NLP). Over the past five or six years there has been a strong shift toward using neural networks for modeling in various NLP tasks, particularly the recurrent neural network (RNN) and its many offspring, such as the Long Short-Term Memory (LSTM) network. In this paper, I will be discussing the use of an attention-based bidirectional LSTM (BLSTM) for the task of sentiment classification using the IMDb dataset [1]. While the results did not meet the state-of-the-art methods currently in practice (classic machine learning or otherwise), they do demonstrate some of the difficulties of working with neural networks, and suggest that there is plenty of room for adjustment of the model for improvement.

2 Related Work

The task of sentiment classification involves inferring a sentiment—e.g., "positive" or "negative"—from a given sequence of text. Classes are not restricted to these two labels, but rather can also target aggregated intensity or lack thereof, such as "very positive," "neutral," "very negative," etc. Many approaches have been used for the task, such as early machine learning based methods [2,3] and lexically focused classification, [4]. Recently, neural networks have been popping up more and more in the literature, such as the convolutional neural network (CNN) [5-7], recursive autoencoders [8] and, like the approach used here, LSTMs, [9-12]. LSTMs are most popular for NLP tasks because they provide some defense against exploding and vanishing gradients. RNNs and deeper networks can meet this problem when error gradients accumulate and result in large updates to the network which inhibit learning during training. Another common method to combat this is to set a threshold by which the gradients are scaled down when it is exceeded, this is known as gradient clipping. Both of these methods informed the work in this paper and have been applied in the experiments discussed below.

Despite their efficacy (by and large), one of the major drawbacks for the neural network approaches is that they lack interpretability. Because of this, many have proposed neural network architectures that include some kind of attention mechanism. Attention mechanisms come in roughly three general categories: global, local, or self-attention. Global attention refers to the entire input and could be useful for things like image captioning [17], while local attention localizes to a section of the input space as demonstrated by [18] in a machine translation task. Often in NLP tasks LSTMs are paired with self-attention to assist in capturing long-term dependencies [13-15], reporting clearer interpretability using weight visualizations, as well as improved results. That said, the understanding of "attention as explanation" has been disputed, [16]. In any case, self-attention was used for this study, due to its largely beneficial results.

3 The Data

As previously mentioned, I used the IMDb benchmark dataset provided in [1], available directly through the TorchText package from PyTorch, [19]. The dataset consists of 50,000 highly polarized movie reviews from the IMDb website and it comes with a 50-50 split for training and testing (25,000 samples each) with polarized labels of positive or negative. Additional unlabeled data is available, but it was not used here. Of the 25,000 training samples, 30% were held out in a validation set for development and

hyperparameter tuning. The full test set was used for evaluation of the model once optimal hyperparameters were established.

4 The Model

The primary model for evaluation in this paper is a bidirectional Long Short-Term Memory with self-attention (BLSTM-SA) and it is outlined more in-depth below. Similar to the model in [14] used for relation extraction, the model consists of four major components. The first layer takes the text as input and uses word embeddings to map each word into a low dimensional vector. The LSTM layer then extracts high level features in the hidden states and passes them to the attention layer to produce a weight vector. Using matrix multiplication, the weight vector is merged with word-level features from each sequence into a text-level feature vector. Finally, the output layer performs the classification.

4.1. Embeddings

The input encoder selected for this model uses pre-trained embeddings from GloVe [20]. Due to time constraints I chose not to implement fine-tuning of the embeddings during the learning process, but this could easily be added for future work. The embedding layer is of the shape (V, L), where V is the size of the vocabulary (251639 for this dataset) and L is a hyperparameter to indicate the length of the embeddings, which is set to 300 for all experiments.

4.2. LSTM

Conversely to the exploding gradient problem, the LSTM improves over a vanilla RNN architecture by using "memory" to fight the vanishing gradient. The LSTM hidden state includes a gating mechanism that decides whether or not to pass along information from the previous cell state to the next. Since text data is often sequential in nature, this means that the LSTM architecture is capable of modeling long-distance dependencies over various "time steps," which prove useful for many NLP tasks, including sentiment classification.

The LSTM takes the output from the embedding layer and models it as a sequence of over time steps corresponding to the length of the input. In order to cope with texts of varying lengths, the inputs are padded to a uniform size and data is sorted by length before batching to minimize the padding. Specific to the LSTM, the model takes optional bidirectionality, dropout, and number of hidden units as hyperparameters. Experiments in bidirectionality showed that BLSTM was far superior, so LSTM experimentation was limited. Dropout was tested at 10-50% to fight overfitting, and hidden unit values of 256, 512 and 1024 were all evaluated.

4.3. Self-attention

The output from the LSTM layer is fed through the self-attention weighting mechanism. Outputs from the hidden units of the LSTM are represented as matrix consisting of timestep vectors [h_1, h_2, …, h_N], where N is the length of the input text. The mechanism first calculates a weighted sum of these output vectors by applying tanh to the matrix, followed by a matrix multiplication with a transpose of the learned weight parameter vector. A new representation of the hidden layer output is then obtained using matrix multiplication of the original LSTM output with the softmax of the weighted outputs before passing it along to the model output for classification.

4.4. Output

Since this is a binary classification task, the output layer is a single unit with tanh activation. The labels are encoded as 0 for "negative" sentiment and 1 for "positive" sentiment. The model could easily be

converted to accommodate a larger label set with more output units and a SoftMax activation for other datasets.

In order to train the best model with limited resources, I used Google Collab with a GPU runtime for all experiments. As previously mentioned, experiments included modifications to various parts of the model, as well as the optimizers. Stochastic gradient descent, Adam and RMSprop were chosen for experimentation, and various configurations of learning rates, momentum, and epochs were tested. In addition to fighting overfitting with the dropout parameter, early stopping was implemented to ensure an optimal model would be evaluated for each experiment. Gradient clipping was given minor experimentation, but did not seem to be very effective beyond 0.1. All code for the data, model, training and testing is available on GitHub[1] and is based on existing implementations from [14] and [21] with original modifications and reformatting.

5 Results & Discussion

After a several days of hyperparameter tuning, the optimal model was a BLSTM-SA with 512 hidden units and 30% dropout after the LSTM layer. The best optimizer for the task was RMSprop with learning rate set to 0.0001, momentum at 0.9, gradient clipping at 0.1, and training took no longer that 20 epochs with a minibatch size of 32 (about 1 minute per epoch using the GPU, orders more for CPU). RMSprop was the best and fastest optimizer, getting higher accuracies and slightly lower loss values, followed by Adam and SGD with Nesterov momentum. As for training metrics, Figures 1 and 2 show that even with RMSprop, loss values were hard to manage despite the improvements in accuracy. This was consistent across all experiments, regardless of hyperparameter configuration.

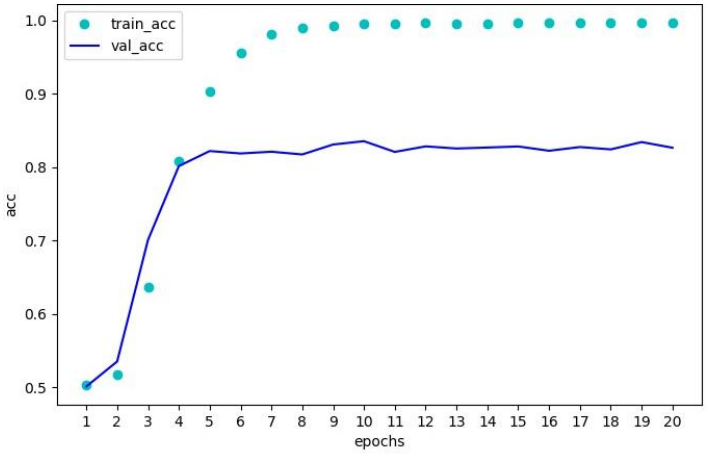

Figure 1. Training & validation accuracy, BLSTM-SA

1 https://github.com/m-sean/AttnBiLSTM-Sentiment-Detection

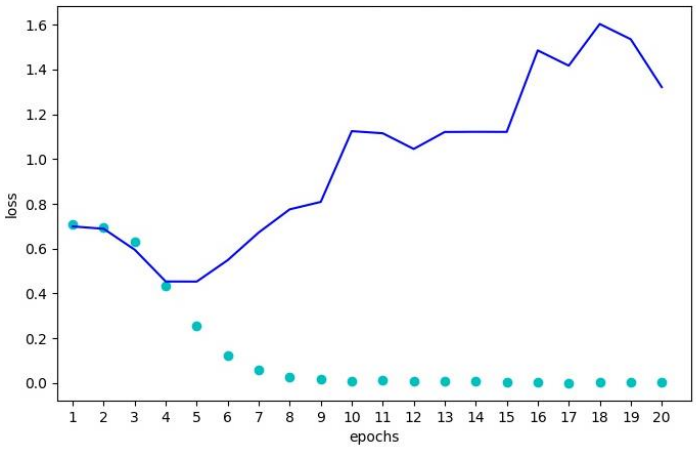

Figure 2. Training & validation loss, BLSTM-SA

Almost all training accuracies reached over 97-99%, but validation scores topped out at 76.49% (using SGD with Nesterov) to 83.55% (using RMSprop). Additionally, most learning tended to happen early on, until a plateau in accuracy which would occasionally producing a significantly better result. Loss typically decreased within the first 5 epochs, and then begin to grow erratically. Referring back to the Figures above, it would seem that the model generally makes good predictions, but the disparity between loss and accuracy suggests that the model either becomes less confident about those predictions and/or, the bad predictions get worse. The latter could arise in cases like sarcasm, where the text doesn't match the sentiment at face value, or in cases where negation flips sentiment but it's not caught (although this should be unlikely due the bidirectionality of the LSTM layer).
Running the model on the held-out test set showed some consistency with the training results, coming in with accuracy of 81.41% and loss of 1.45. State of the art for this data starts in the high 80s, [22] used a CNN+LSTM classifier to achieve 88.9% accuracy.

Sentiment	Precision	Recall	F1 score
Negative	0.80622	0.82087	0.81348
Positive	0.82398	0.80953	0.81669

Table 1. BLSTM-SA Precision, Recall, and F1 score

Returning to a more positive note, Table 1 shows that there is consistency between the F1 scores for each sentiment label. Precision is almost 2% higher for positive, which suggests that the model could be slightly more likely to predict negative sentiment in general, and therefore mislabel some positive reviews as negative. This is supported by the fact that recall is higher for negative, showing that more samples relevant to the negative label are correctly identified.

6 Conclusion

Although the results above are not up to par with state of the art methods, there is plenty of room for improvement. Alternative attention mechanisms and gated RNN architectures could be explored or

hybridized with other neural net architectures. The model also stands to benefit from a much more in depth development stage to find the optimal hyperparameters, reduce loss, and increase accuracy. Perhaps variations on the inputs could also be tested, such as n-gram features instead of word features.

While there are many possible options for enhancing performance, this shouldn't be the only focus. The BLSTM-SA studies in this paper is quite expensive to train as is, so in addition to improving the modeling process, logistics of training and memory costs should be taken into account as well—simplifying and reducing wherever possible. Matrix multiplication is far easier to use on GPUs, but such resources are not widely available. The bi-directional block self-attention network (Bi-BloSAN) proposed in [13] shows that reducing memory costs and training time doesn't necessarily require a tradeoff in performance. When training neural networks for any task, such factors should be always considered. For tasks like sentiment analysis, various methods are in already in deployment in industry, and it's necessary to maintain responsible and accessible practices that are not just able to produce results, but are also easily replicable, verifiable, and generalizable.

7 References

[1] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

[2] Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede. (2011). Lexicon-based methods for sentiment analysis. Computational linguistics 37(2):267–307.

[3] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. (2002). Thumbs up?: sentiment classification using machine learning techniques. In ACL. pages 79–86.

[4] Bo Pang and Lillian Lee. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In ACL. pages 115–124.

[5] Yoon Kim. (2014). Convolutional neural networks for sentence classification. In EMNLP. pages 1746–1751.

[6] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. (2014). A convolutional neural network for modelling sentences. In ACL. pages 655–665.

[7] Tao Lei, Regina Barzilay, and Tommi Jaakkola. (2015). Molding CNNs for text: non-linear, non-consecutive convolutions. ACL.

[8] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In EMNLP. pages 1631–1642.

[9] Tomáš Mikolov. (2012). Statistical language models based on neural networks. Presentation at Google, Mountain View, 2nd April.

[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[11] Kai Sheng Tai, Richard Socher, and Christopher D Manning. (2015). Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075.

[12] Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. (2015). Long short-term memory over recursive structures. In ICML. pages 1604–1612.

[13] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. (2018). Bi-directional block self-attention for fast and memory-efficient sequence modeling. ICLR.

[14] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. (2016). Attention-based bidirectional long short-term memory networks for relation classification. In The 54th Annual Meeting of the Association for Computational Linguistics. pages 207–213.

[15] Letarte, G., Paradis, F., Gig`u, P., Laviolette, F. (2018). Importance of Self-Attention for Sentiment Analysis. Tech. https://www.aclweb.org/anthology/W18-5429

[16] Sarthak Jain and Byron C. Wallace. (2019). Attention is not explanation. CoRR, abs/1902.10186.

[17] Xu, K. et al. (2015). Show, attend and tell: Neural image caption generation with visual attention. In Proc. International Conference on Learning Representations http://arxiv.org/abs/1502.03044.

[18] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. (2015). Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025.

[19] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. (2017). Automatic differentiation in PyTorch. In NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques, Long Beach, CA, US, December 9, 2017.

[20] Jeffrey Pennington, Richard Socher, and Christopher Manning. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543.

[21] Prakash Pandey. (2018). Text classification using deep learning models in Pytorch, https://github.com/prakashpandey9/Text-Classification-Pytorch .

[22] Jose Camacho-Collados and Mohammad Taher Pilehvar. (2018). On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. In Proc. of Blackbox EMNLP Workshop.