QRNNs for IMDb sentiment classification task

Pavol Vidlička

pavol.vidlicka@gmail.com

May 20, 2018

Abstract

This report presents results of our TensorFlow implementation of quasi-recurrent layer and its performance on the IMDb review sentiment classification task. We compare the results with the results obtained using LSTM layers. We also discuss performance and possible improvements to the implemetation.

I. Introduction

Quasi-recurrent neural networks were intorduced in the paper [Bradbury et al., 2016] as an alternative of a regular RNNs that are able to make a better use of parallelism and thus improve train and test times.

We present our own implementation of the QRNN concept and results that support the claim of the original paper that QRNNs are a more parallel alternative of LSTMs with at least the same perfomance in terms of model accuracy.

II. Dataset

We are using the well known IMDb Large Movie Review Dataset [Maas et al., 2011]. The dataset has been processed in the following way: it has been tokenized[1] and the tokens were encoded by integers. The values '0' and '1' were reserved for '<EMPTY>' and '<UNKNOWN>' tokens respectively. The rest of the tokens are sorted and given an index based on their count in the train set. Only the ones with three or more occurences are included. This procedure yields a vocabulary size of 44387.

We use the GloVe ([Pennington et al., 2014]) vector space embedding for the words. More specifically, the 300 dimensional embedding trained on a 840B tokens Common Crawl corpus.

[1]using 'simple_tokenize' from 'gensim.utils'

The GloVe embedding included word vectors for 43851 tokens out of our vocabulary. The rest (including '<EMPTY>' and '<UNKNOWN>') is mapped to zero vectors.

The IMDb reviews were encoded using our vocabulary. To reduce noise, the tokens with index 50 or lower were filtered out. The averare number of tokens after the filtering in the test set is 138.7. The lenght distribution is however very skewed. We homogenized the length to 150 tokens shortening or padding with zeros when needed. Around 70% of the train set reviews had a length of 150 or less.

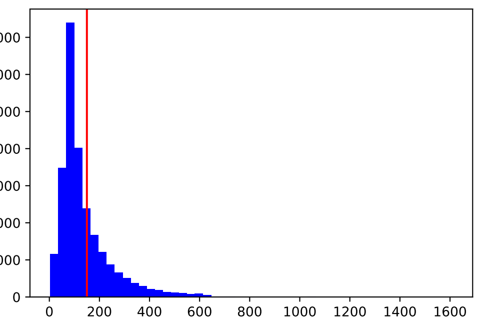

Figure 1: *Processed reviews lenghth in train set*

The test set has been processed using encoding obtained from the train set. However, if we were to apply our model to a different corpus, we could extend the vocabulary since the model was trained on word vectors. The extension should conform to the fact that we filtered out uncommon words and most common words for the training of the mode.

III. QR layer implementation

Our QR layer implementation uses tensorflow's 'tf.layers.conv2d' internally for the convolutional part. The pooling is realized using higher order function 'tf.scan'.

We have also experimented with a different implementation (refered to as 'alternative'), where the pooling is realized using the 'RNNCell' class present in tensorflow. However, it did not yield a better perfomance.

IV. Experiments

All the experiments were conducted using a Nvidia 930M mobile GPU. The modest performance of the card influenced the decision to work a computationally less expensive model than in the original QRNN paper. We have also not conducted a extensive hyperparateter optimization, so the accuracy reported does not have to be indicative of best possible performance.

i. Model architecture

We have used a densely connected two layer recurrent network with 256 units in each recurrent layer. That is, the output of a recurrent layer is concatenated with the input and passed on. The the final output is passed to a dense layer with a sigmoid activation function.

The recurrent layer is either a QR layer or a LSTM layer. We wanted to explore if the QR layer can be used in place of a LSTM layer.

For the LSTM layers we have used the Keras 'CuDNNLSTM' layer. The tensorflow's 'CudnnLSTM' is faster, but we did not manage to get it to converge properly by tweaking the parameters using the provided API.

ii. Train time

The train times per epoch are shown in Table 1.

Recurrent Layer	Time / Epoch (s)
CuDNNLSTM	195
LSTM	655
QRNN	315
QRNN alternative	435

Table 1: *Train times per epoch*

Our implementation of QRNN is not faster than CuDNNLSTM. The probable cause is that the implementation uses a high-level API. The chainer implementation used for the experiments of the original paper[2] includes custom C code written for the pooling step.

Another possible cause of the perfomance difference is the GPU that has been used. The Nvidia 930M may not have enough resources to provide paralellism needed for the QRNN to beat the LSTM network.

However, the QRNN is faster than a network using Keras' LSTM layer.

Implementing the pooling at a lower level could help to improve the speed.

iii. Validation accuracy

We compare validation accuracy over 10 epochs of the QRNN and LSTM network. The 'test' part of the IMDb dataset has been used for validation. A L^2 regularization of 10^{-6} was used for convolutions in the QR layers and the filter width has been set to $k = 2$.

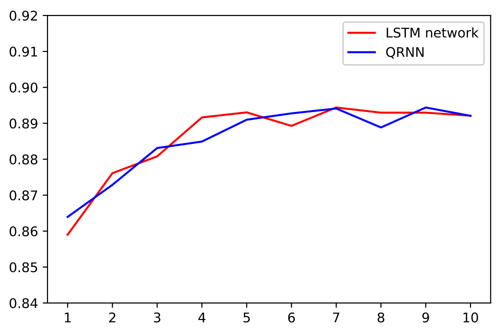

Figure 2: *Validation accuracy by epoch*

The achieved accuracy (see Figure 2) is nearly

[2]https://gist.github.com/jekbradbury/a3a5ae890328db49d8093c1a5bdc8a1efile-qrnn-py

identical. Training for more than 10 epoch did not prove to be benefitial for either network.

V. Conclusion

We have implemented the quasi-recurrent network and benchmarked it again an equivalent LSTM network. We have discussed some ofthe performance aspects and identified possible causes of worse performance than reported in the original paper.

References

[Bradbury et al., 2016] Bradbury, J., Merity, S., Xiong, C., and Socher, R. (2016). Quasi-recurrent neural networks. *CoRR*, abs/1611.01576.

[Maas et al., 2011] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.