Hyperparameters of RNN Architectures for POS Tagging using Surface-Level BERT Embeddings

Vilém Zouhar
vzouhar@lsv.uni-saarland.de

Abstract

Contextual embeddings from pretrained BERT models have been useful in a variety of natural language processing tasks. This paper focuses on one of the basic of those tasks, Part of Speech tagging, and compares several simple recurrent neural network models (vanilla RNN, GRU, LSTM) and their hyperparameters (hidden state size, recurrent layers and dropout, bidirectional, dense layers).

While stacking recurrent layers or densely connected output layers negatively affects the performance, adding bidirectionality and increasing hidden state size improve it significantly.

1 Introduction

In comparison to one-hot word embeddings, more complex word embeddings offer significant boosts in performance. One of the main disadvantages of one-hot word embeddings is their high dimensionality and sparsity. Trained word embeddings are able to capture word meaning based on its collocations. The main families of word embeddings are (1) non-contextual, e.g. word2vec (Mikolov et al., 2013) and (2) contextual, e.g. BERT (Devlin et al., 2018). While the first assigns one word the same embedding regardless of the context, the second one produces different ones based on the context. This improvement comes at the cost of significantly higher training requirements. In this experiment, we make use of pretrained $\text{BERT}_{\text{BASE}}$-Cased made available by Wolf et al. (2020).

Part of Speech tagging is the task of assigning part of speech tags to every word in a sentence. We make use of the OntoNotes 4 corpus (Weischedel et al., 2011) of 70k English sentences with POS annotations (49 POS classes).

The goal of this paper is to document the effect of basic recurrent neural network architectures on task performance. The used models: vanilla RNN (Elman, 1990) - henceforth only "RNN", LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) are combined with various architecture hyperparameter choices: hidden state size, recurrent layers and dropout, bidirectionality (Schuster and Paliwal, 1997) and dense layers.

2 Methods

The corpus was split on sentence boundaries to 80% training, 10% development and 10% test sets. The whole batch was used to compute the gradient at each step. The optimizer was Adam (Kingma and Ba, 2014) with the learning rate specified with every architecture type. The number of epochs was fixed to 100, though almost all models would saturate before epoch 25. Finally, the output of every model is processed by softmax to get a distribution.

Model state with the best performance on the development set was used to evaluate on the test set. We report model performance (by class accuracy) on the test set as well as on the training set (with dropout off). The motivation for the latter is to observe overfitting.

Embeddings Vector embeddings for every subword unit were extracted from the top-level (last) layer (768-dimensional vector). Further performance improvements could be made by, e.g. concatenating the top four layers or averaging them. Token embeddings are computed as the average of corresponding subword embeddings.[1]

Models We consider four baselines based on densely connected layers to gain an insight into the performance of models based on single token embedding. The purpose of these configurations is to offer a fast, viable baseline in comparison to more complex models.

[1]The embeddings were stored with 16bit float precision to reduce memory footprint.

Although the embeddings provided by BERT are contextualized, they are frozen in this experiment and may not contain enough information about the rest of the sentence. Recurrent neural networks may be used to alleviate this issue specifically. We experiment with three basic recurrent cells: RNN, LSTM, GRU. For each of this cell, we also explore the following six architecture hyperparameters.

Due to computational limitations, the search for optimal hyperparameters is *not* performed using gridsearch. For every parameter type, we modify only this dimension and keep the rest from the baseline (type+activation, hidden state size, recurrent layers + dropout, bidirectional, dense layers) of (LSTM, 128, 1+0.0, no, 1). We make a conscious faulty assumption regarding the relative independence of these parameters. Despite that, the exploration can still provide us with insight with respect to the generally good architecture hyperparameters. For brevity, we describe every model and hyperparameters in the respective part of the results.

3 Results

Baselines There are four model configurations that utilize densely connected layers (some with dropout added). The activation function for nonfinal layers is Tanh. Densely connected layer with this activation function, n output size and dropout of probability p is denoted as L_n^p. We define model versions with an without dropout. The output is then fed through softmax to get a distribution. Batch size is fixed to 2048 *tokens*; learning rate is 0.001. For completeness, we also list the majority class classifier to provide the lowest possible, uninformed baseline.

Model	Train	Test
Majority	13.7%	13.7%
L_{49}†	91.7%	91.4%
$L_{128} \times L_{49}$	91.5%	90.7%
$L_{128}^{0.1} \times L_{49}$	91.3%	90.6%
$L_{256} \times L_{256} \times L_{49}$	91.3%	90.7%
$L_{256}^{0.1} \times L_{256}^{0.1} \times L_{49}$	91.3%	90.8%

Table 1: Performance of densely connected baselines

Baseline results can be found in Table 1. The best model (on test) is marked with †. Overfitting does not seem to be an issue, though adding dropout shrinks the gap between train and test performance. Despite that, deeper networks perform slightly worse than just logistic regression. This may be the result of choices of batch size, learning rate and the activation function.[2] Furthermore, taking BERT embeddings from, e.g. the top four layers may benefit deeper networks. Even logistic regression provides a very performant baseline, also considering the uninformed majority classifier.

Recurrent Units We explore the capabilities of LSTM and GRU units. Furthermore, we may choose different activation functions for the RNN: ReLU and Tanh. Batch size is fixed to 16 *sentences*, corresponding on average to 348 tokens; learning rate is 0.0005. When reporting performance, the base parameter setup is marked with *.

Recurrent Unit	Train	Test
RNN+ReLU	94.2%	92.9%
RNN+Tanh	96.3%	94.8%
LSTM*†	97.1%	95.2%
GRU	96.7%	95.0%

Table 2: Performance of models recurrent neural networks with different recurrent units

Table 2 shows the performance of different RNN units. All of them outperformed the baseline by a significant margin. Although they all share similar performance, RNN with ReLU activation performed the worst, while LSTM showed to be the most useful for this task.

Hidden State Size A common parameter to modify recurrent neural network capacity is the size of the hidden state vector passed between steps.

Hidden state size	Train	Test
16	93.3%	92.4%
32	94.9%	93.6%
64	95.5%	93.8%
128*	97.1%	95.2%
256	97.6%	95.5%
512	97.7%	95.6%
1024†	98.1%	95.9%

Table 3: Performance of recurrent neural networks with different hidden state size

The steady increase in performance on both train and test data sets suggest that the model is able to make use of the larger capacity. It is also interesting

[2] Leaky ReLU with $\alpha = 0.01$ performed even worse.

to see that even with a hidden state size of 16, the recurrent model is able to outperform the baseline.

Number of stacked Recurrent Layers Recurrent neural networks can be stacked together. When larger than 1, the output of the previous layer is processed by another recurrent network layer. It is also possible to introduce a dropout layer in between these stacked recurrent layers. We also show the results with dropout probability 0.1.

Number of layers	Train	Test
1 layer*†	97.1%	95.2%
2 layers	95.5%	93.9%
2 layers + dropout	95.5%	94.0%
3 layers	95.0%	93.7%
3 layers + dropout	95.2%	94.0%

Table 4: Performance of recurrent neural networks with different number of recurrent layers stacked

As documented in Table 4, stacking multiple recurrent neural layers worsens the performance in this case. This might be the result of choosing the base hidden state size of 128 and could perhaps lead to improvements if larger values were used.

Bidirectionality If bidirectionality is added, the sentence is processed left to right by one recurrent network and right to left by another one. The final vectors are then concatenated, resulting in twice as large a final vector for every token.

Bidirectional	Train	Test
No*	97.1%	95.2%
Yes†	98.4%	96.7%

Table 5: Performance of recurrent neural networks with either bidirectional processing on or off

Adding bidirectional processing (shown in Table 5) vastly improves the performance compared to other architecture hyperparameter choices. A fair comparison would be to a model with a hidden state vector of size 256 since bidirectional processing uses two hidden state vectors sized 128 each. Bidirectionality helps even in this case.

Dense Layers The output for every token is the hidden state vector at that specific timestep. This can be further processed by deeper, densely connected layers. We again use Tanh as the activation function.

Output dense	Train	Test
L_{49}*†	97.1%	95.2%
$L_{128} \times L_{49}$	95.8%	94.2%
$L_{256} \times L_{256} \times L_{49}$	94.7%	93.5%

Table 6: Performance of recurrent neural networks with different shapes of output densely connected layers

Results in Table 6 document, that similarly to densely connected networks, deeper architectures only decrease the performance. Different activation functions could lead to different results.

Combination of Best Parameters Finally, we train a system with the (individually) best-performing parameters: LSTM, hidden state size 512,[3] one recurrent network layer, bidirectionality, single output layer. This resulted in an accuracy of 98.9% and 97.1% on training and test sets, respectively. Keeping bidirectionality and increasing hidden state size to 1024 leads to accuracies of 98.8% and 97.2%, respectively.

4 Conclusion

In this paper we focused on the six most important hyperparameter choices of simple recurrent neural networks in the task of POS tagging with top-layer BERT embedding.

Out of the four basic recurrent units, LSTM performed the best, although only by a small margin over GRU. We find that deeper architectures are not helpful on the given corpus, not even for the baselines. The two most important hyperparameters which positively affected the performance were bidirectionality and greater hidden state size.

4.1 Future Work

This paper made a conscious faulty independence assumption of the inspected parameters. To make this research formally sound, a full gridsearch or a more informed search should be performed.

Furthermore, this paper did not touch upon the batch size, optimizer parameters and different BERT embedding extraction, which also interact with other hyperparameters.

The length limit of this paper also did not allow for error analysis, which could reveal low performance on specific classes and could suggest improvements in this regard.

[3] Combined hidden state vectors are 1024-dimensional.

References

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, et al. 2011. Ontonotes release 4.0. *LDC2011T03, Philadelphia, Penn.: Linguistic Data Consortium*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.