Optimization of Encoder-Decoder Model and Attention Mechanism for Machine Translation

A Preprint

Spencer R. Bertsch[*]
Department of Engineering
Dartmouth College
14 Engineering Drive, Hanover NH 03755
spencer.r.bertsch.th@dartmouth.edu

March 14, 2019

Abstract

This paper presents an application of random search hyperparameter optimization and fine tuning to optimize the performance of an existing encoder-decoder network for use in machine translation. In this paper, I compare different types of optimized encoder-decoder models in the context of machine translation in order to improve the performance of a benchmark sequence to sequence network. The original model was sourced from Sean Robertson in his post on pytorch.org - "Translation with A Sequence to Sequence Network and Attention." In addition to the work performed in the original model, methods were introduced which use random search and fine-tuning optimization to reconstruct the encoder in order to find the optimal overall network topology. The resulting optimized network showed that for the encoder, a shallow yet wide network – 3 hidden GRU layers and 248 GRUs per layer – resulted in optimal model performance. Wider models were able to learn better representations from the input sequences with rounded losses of 3.2 after 1000 epochs, allowing for better translations, but at a cost of computation time. Deeper models were able to converge to a solution more quickly than shallower models, but they achieved a lower model performance with rounded losses of 3.4 after 1000 epochs.

***K*eywords** Encoder-decoder · GRU · Attention · Machine translation · Hyperparameter optimization · Fine tuning

1 Introduction

Unlike fully connected and convolutional neural networks, recurrent neural networks have allowed models to learn from sequence or time series data. Sequence to sequence models, also referred to as encoder-decoder models or seq2seq models are a type of recurrent neural network which use two networks: an encoder network which encodes an input and a decoder network with produces output. Although words in a phrase, sentence, or paragraph are not an example of time series data, they can be encoded and treated as sequential data because the ordering of the words in a sentence is crucial to the sentence meaning. Encoder decoder models, therefor, have been a popular model for use in natural language processing applications in recent years. More recently, modeling sequence data using attention has grown in popularity because of the benefits added by allowing the models to learn which parts of the sequence are the most important or influential.

Gated RNNs and encoder-decoder models have become the networks of choice in NLP applications for several reasons. First, T^n or the length of some sequence T is often too large for a vanilla RNN to maintain the gradients across all time steps and produce a satisfactory result. This problem is solved, however, using both gated units such as LSTM or GRUs in the hidden layers of the RNN and some type of clipping regime such as norm clipping to keep the gradients from getting too large and exploding. If the gradients being computed get too small or too large, the learned parameters in the network will not reflect the true structure of the data being learned.

[*]github.com/spencerbertsch1

Second, $T^n \neq T^{n'}$ or the length of a learned sequence is often not the length of the output sequence. Encoder-decoder models solve this problem by allowing the decoder to develop its own results sequentially. The length of the input has no bearing on the length of the output; in the case of machine translation, only the context is learned by the encoder so the decoder can choose the best words from the dictionary to choose to reconstruct the meaning of the input phrase.

This paper outlines an experiment using gated recurrent neural networks which make use of GRUs (Gated Recurrent Units) with an attention mechanism on the decoder network. The network is benchmarked, then hyperparameters are optimized using random search and additional fine tuning. The performance of the network is tested using the loss achieved after some standard number of epochs which varies during testing between 1,000 and 10,000. The dataset used in this project is the Tatoeba English-French sentence dataset which is pruned for simpler testing to a corpus of 4,345 French words, 2,803 English words, and a total of 10,599 word phrases which are split and used for training and testing.

Hypothesis: Shallow and narrow encoder-decoder models will yield initially limited results suffering from repetitive outputs or simply losing the context of the input phrase. The networks ability to learn the context of the input sequence will depend on the model's ability to learn from the data and more gated units will allow a better learned model.

2 Analysis

2.1 Machine Translation

Encoder-decoder networks have become the primary model of choice for machine translation for several reasons, the main being that the encoder doesn't simply encode each word individually, but rather produces an encoded vector representation of the "context" of an input phrase. This can also be a powerful tool when applied to other natural language processing applications such as content summarization and short text generation. In this paper I discuss the changes being made to an encoder which produces encoded output, which is subsequently fed as input to an attention decoder which then reconstructs the input phrase in another language. One of the unique advantages of this model is derrived from the model's ability to learn from teacher forcing, a method in which the ground truth y^t corresponding to each input x^t can be fed directly into the first hidden layer at node h^{t+1}. [4]

2.2 Network Optimization

Both manual search and grid search methods are often employed when optimizing network parameters. Instead, random search optimization methods were used in lieu of grid search or manual search because random search hyperparameter optimization is more efficient than both other commonly used methods. [5]

By far the most complete solution to network optimization would be grid search. Knowledge of a parameter's impact on model performance comes at the cost of computation time, which renders grid search an unrealistic solution to the problem of parameter optimization. This is specially true when confronted with the task of optimizing a large number of n parameters, causing the solution space and computation time to grow very quickly. This begs the question: how to we efficiently generate configurations of hyperparameters for all solutions? The answer lies in defining an upper and lower boundary for each parameter and allowing a random search to select the parameters for testing. This is the method which was used in this application. Only two hyperparameters were chosen for optimization due to the heavy computational cost of training an encoder-decoder system sequentially across sets of parameters.

In addition to simply optimizing the encoder and decoder width and depth, fine tuning was also applied to further improve network performance. "Fine tuning" in a deep learning context often refers to using an existing model and training it on a specific dataset related to a an application at hand instead of starting from scratch.[1] In this case, however, I'm referring to finding the optimal area of n dimensional parameter space which contains the parameter set which yields the best model performance. This can be found simply by finding the result from the first optimization pass with the lowest loss value, then reinitializing the random search optimization using that point as the center point for a new parameter space. In order to optimize the encoder-decoder network in this project, upper and lower bounds of 1 and 25 were chosen for the network depth, and upper and lower bounds of 25 and 250 were chosen for the number of gated units in each hidden layer.

3 Results

The first pass of the optimization method showed that the shallowest model yielded the best results, so the bounds were converged around the point [4, 238] and the optimization methodology was repeated with new, tighter boundaries. On each pass of the optimizer, a random uniform distribution produced five samples, each of which corresponding with one set of hidden layers and number of GRUs per layer.

Regularization methods were employed through the use of dropout layers in the decoder. By plotting the testing loss we can see that the slope never increases above zero, so there is no indication that other regularization methods such as L1, L2 or early stopping are needed. Further experiments could involve reducing the probability of dropout in the decoder network and testing for an improvement in the encoder-decoder's performance before overfitting became detrimental.

Table 1: Initial Optimization Results

Number of Hidden GRU Layers	Hidden State Units	Loss
11	124	3.267
4	238	3.264
17	63	3.321
22	136	3.301
10	248	3.386

In Table 1 we see that the optimal network resulting from the first pass contained four hidden GRU layers and 238 GRUs per layer, achieving a loss of 3.264 after 1000 epochs. After the bounds for the optimization were updated, the method was run again with five new sets of hyperparameters. See Table 2 for results of fine tuning; the highest performing network with three hidden layers and 248 GRUs per layer achieved a loss of 3.211 after 1000 epochs. Note that epochs were reduced to 1000 in order to manually decrease computation time for this exercise.

Table 2: Fine Tuning Results

Number of Hidden GRU Layers	Hidden State Units	Loss
3	231	3.352
3	248	3.211
2	227	3.375
6	249	3.259
6	236	3.284

We can also see from Figure 1 that encoder-decoder models with deeper networks converge more quickly but fail to reach a high level of model performance. Wider, shallow networks need to be trained for longer periods, but after training they are able to achieve better results. These results contradict the hypotheses because shallower networks outperformed the deeper networks, but wider encoder networks were able to learn a better representation of the data as expected.

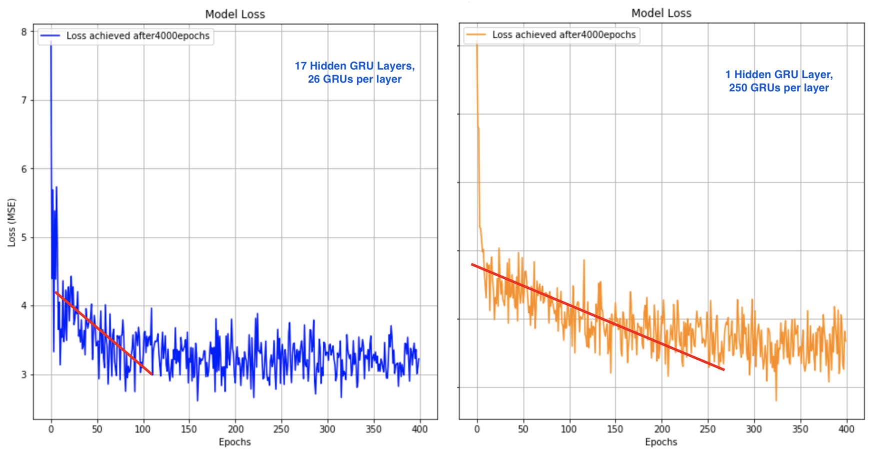

Figure 1: Learning curves for deep (17 GRU layers) and shallow (1 GRU layer) networks.

There are certain limitations inherent in this methodology; the learning rate in this example is constant so there may be small crevasses which were never entered in the solution space. If this is true, the minima found by the initial random search and optimization would not reveal the true parameters which yield the best model results. Additionally, this methodology would benefit from a larger range for both the number of hidden layers in the network and also the number of GRU units in each layer. Examining a larger parameter space would also increase confidence that the solution which was derived represents the true parameter values which optimize the network for this application.

Lastly, in order to reduce training time but preserve comparability of the networks being tested, I reduced the number of total epochs each network used for training. Although this saved time during experimentation, it could lead to inaccuracies in the case that there are nonlinearities present during network training. Deeper networks, for example, converge more quickly than shallow networks; if the deep and shallow networks were tested over several ranges of epochs, more insight could be gathered into which truly converges more quickly and to what extent they lead to model performance.

4 Conclusion

In this paper, I presented an application of random search hyperparameter optimization and fine tuning to optimize the performance of an existing encoder-decoder network for use in machine translation. Wider, shallower models generally outperformed deeper, narrower models yielding rounded losses of 3.2 and 3.4 respectively. It was also shown that deeper models converge more quickly than shallower models, but are inevitably outperformed by shallower, wider models.

Future work could involve testing the network with different gated units such as LSTM units, or by simply adding other hyperparameters to the random search optimization method. As deep learning becomes more popular and network optimization becomes more commonplace, libraries dedicated exclusively to attaining the correct network topology and hyperparameters are being introduced. Future work could include applying a library designed for network optimization such as autokeras to this encoder-decoder network and note any improvements made beyond those found in this paper.

5 Acknowledgements

I gratefully acknowledge Professor Eugene Santos at the Thayer School of Engineering at Dartmouth College for overseeing this course and for helping me to achieve a better technical and conceptual understanding of deep learning.

References

[1] Felix Yu. A Comprehensive Guide to Fine-Tuning Deep Learning Models in Keras (Part I). In *flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html*.

[2] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In *arXiv:1508.04025v5, 2015 Sep 20th* Arxiv, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385v1*, 2015.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning http://www.deeplearningbook.org *Goodfellow-et-al-2016*.

[5] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. In *Journal of Machine Learning Research 13 (2012) 281-305, 2012 Feb 11th.*