CSCE496 Deep Learning: Homework #4

Cale Harms, Colton Harper, and Krishnamohan Sunkara

April 4, 2018

1 QUESTION #1

1.1 INTRODUCTION

For this work, our group developed one Recurrent Neural Network (RNN) architecture that uses Long Short Term Memory (LSTM) cells and performed a hyper-parameter search to predict the next k words of a sentence, using the Penn Tree Bank (PTB) dataset [1]. In the data, each word is represented as an integer and semantic relationships between words are realized using a word embedding. Our general approach includes a single layer LSTM which is dynamically unrolled based on the time steps provided. In this report, we first describe the dataset that is used in the training process. We then describe the architecture and the sets of hyper-parameters that we have chosen to use. Next, we present the results for the architecture using different sets of hyper-parameters. Finally, we analyze our results to identify the impact of different hyper-parameters for this particular problem.

1.2 PROBLEM DESCRIPTION

1.2.1 DATASET

The dataset that our group used to train and test the RNN was the PTB dataset. The PTB dataset is a text dataset made up of about one million words from Wall Street Journal stories. This dataset often teaches RNN's to speak English, and it is a widely used dataset in Natural Language Processing (NLP) research. From the PTB dataset, we use the train and test set, which contain 929,589 and 82,430 words respectively [1]. Our custom function reads the PTB dataset and convert the data into numeric values based on a dictionary. The batch producer function takes the data and convert them to queues of training data and the test data based on the provided k-value. These queues are then used for feeding the data during the training and testing phases.

For the purpose of this homework, we only considered the training set of the

PTB dataset for training our model and used only the test data of the dataset to test out the model. No text labels were used in the model because our aim was to generate the next k words of the given sentence.

1.2.2 HYPERPARAMETERS

The architecture that we use for this problem consists of a single layer made up of LSTM cells. The outputs of the LSTM cells are provided to a dense layer consisting of *vocab_size* nodes. The inputs to the entire network are formed using $tf.nn.embedding\_lookup$ which uses an embedding matrix of dimensions $vocab\_size, embedding\_size$. The input provided to the network is $batch\_size, time\_step$ size and these inputs are provided to the network dynamically by the queues at corresponding time-steps. We train the network for 10 epochs and then test for accuracy on the test data. The accuracy of the model is higher if it predicts the exact same output as the target value which means that our accuracy checks for the number of matches in the prediction to that of the targets. Our network uses a drop out regularizer with a *keep_prob* of 0.5 and the input is also dropped based on this probability before feeding into the network. The hyper-parameters that we consider for this problem are the number of LSTM cells in a layer denoted by $LSTMsize$ and the learning rate used for optimizing the network. We also evaluate our network for different values of K.

Table 1 : Hyperparameters

LSTM Size	Learning Rate	K
100	0.001	1
100	0.0005	1
100	0.0001	1
200	0.001	1
200	0.0005	1
200	0.0001	1
500	0.001	1
500	0.0005	1
500	0.0001	1
100	0.001	5
100	0.0005	5
100	0.0001	5
200	0.001	5
200	0.0005	5
200	0.0001	5
500	0.001	5
500	0.0005	5
500	0.0001	5

1.3 RESULTS

Table 2 includes the average accuracies of our model on the test data for different sets of hyper-parameters. Our network trains on the training data and the model is saved after the training phase. The network is optimized based on the sequence-to-sequence loss between the inputs and the predicted outputs. During our testing phase, the model is re-built and the weights are restored from the saved model. The model is then given the testing data and the next k-words are predicted. These predictions are used for calculating the accuracy of the model. The entire training process is done using a drop-out regularizer and the network trains for 10 epochs.

Table 2 : Hyperparameter Search Results

LSTM Size	Learning Rate	K=1 Accuracy	K=5 Accuracy
100	0.001	0.1896	0.0716
100	0.0005	0.1816	0.0685
100	0.0001	0.1372	0.0672
200	0.001	0.2020	0.0731
200	0.0005	0.1951	0.0723
200	0.0001	0.1596	0.0706
500	0.001	0.2150	0.0771
500	0.0005	0.2096	0.0761
500	0.0001	0.1737	0.0717

1.4 EVALUATION

A higher learning rate is associated with a model that scores a higher accuracy on the test data. This increase in accuracy with an increase in the learning rate can likely be attributed to the fixed number of epochs we train our model for. It is likely that this conclusion would not necessarily hold true for the case of varying the number of epochs the model trains for.

The increase in the number of LSTM cells in a layer resulted in an increase in the model's accuracy on the test data. We think that the resultant increase in accuracy on the test data can largely be attributed to the single layer architecture that we implemented. We hypothesize that fewer cells in a single layer architecture would not be robust enough to capture the complex relationships between English words.

We also observe an inverse relationship between k and the model's accuracy; as k increases, the accuracy of the model dramatically decreases. This is likely due to the dependent nature of cells in LSTM's. Each successive word prediction is dependent upon the previously predicted words, which can lead to a quick reduction in accuracy as k increases.

1.5 CONCLUSIONS

For this homework, our group observed clear evidence that the best hyperparameter choices for our single-layer architecture with dropout was a higher learning rate, with our optimal learning rate being 0.001, and more cells in the LSTM, with our optimal number of cells in the LSTM being 500. We also observed that the smaller the value of k, the higher the accuracy was. These results were monotonic, with no outliers to the conclusions we drew. For future work, more insights into obtaining higher accuracies may be obtained by performing a similar hyperparameter search over a model that implements early stopping and a large number of epochs. Using this method in the past has typically shown that lower learning rates tend to increase the accuracy of the model because it allows the model to train for longer. We could also likely see improvements in the accuracy of the model if we added more layers to our architecture, as this too may be able to capture more information about the relationships between the words in the sentence.

ACKNOWLEDGMENT

Portions of the code used for this assignment were taken from the file given to the CSCE496 Deep Learning course for Hackathon 8. We also followed the tutorial on Recurrent Neural Networks with TensorFlow from [2]. We followed the similar structure to that described in the tutorial but implemented the core network and the training process based in Hackathon 8. The main content that we used from the tutorial is regarding the state tuples.
The following files are included in this homework submission:

- main.py
- model.py
- utils.py

[1] PTB Dataset, http://corochann.com/penn-tree-bank-ptb-dataset-introduction-1456.html

[2] Tutorial, http://adventuresinmachinelearning.com/recurrent-neural-networks-lstm-tutorial-tensorflow