Ashwin R. Rao
CSC578-710 (Online)
Final Project Documentation

Walk-Through: https://youtu.be/D6i254GzW6w
Kaggle Username: Ashwin Rao
Rank: 31

The competition model that was submitted to Kaggle consisted of the following hyper-parameters and network structure. An LSTM input layer with 16 nodes, return_sequences set to True, a dropout layer next with 50% of previous activations set to be dropped, a second LSTM layer that also had 16 nodes and return_sequences set to True, and a final Dense layer with one node to produce a singular prediction for the model. The optimizer used was the Adam optimizer. I used a batch_size of 75 training samples, and ran the original model that I submitted to Kaggle for 40 epochs. When re-running the model to generate the graphs of loss and accuracy (for the notebook), I only ran the model for 10 epochs but it did not seem to greatly affect the accuracy or loss of the model. The mean absolute error for this model was 3.11, per my calculations and 2.99 according to Kaggle.

The second model that I produced, after I submitted to Kaggle consisted of the following hyperparameters and network structure. An LSTM layer with 128 nodes, return_sequences set to True, a dropout layer with 50% of activations set to be dropped, another LSTM layer with 64 nodes without returning sequences, a second dropout layer with the same percentage as the first, a Dense layer with 32 nodes and a ReLU activation function applied, followed by the final Dense layer with a single node for a singular output. The optimizer used was stochastic gradient descent, batch_size was 450, and the model was run for 10 epochs. Per my calculations, the mean absolute error for this model was 2.784, which is lower than the Kaggle-submitted model.

Both models, however, failed in one regard. Their accuracy in both training and test sets remained the same throughout training and were exactly the same for both the Kaggle-submitted model and the second attempt. This, coupled with the high MAE for both models, indicates that the models were simply a poor fit to the data. The fact that the accuracy for both training and test sets did not change during training (evidenced by flat lines on the accuracy vs. epochs graphs in my notebook) indicates that there was something wrong with either the hyperparameters/parameters or the input data itself.

After running my first model, I tweaked all the hyperparameters and parameters in all the ways I could think of. I switched optimizers from Adam to SGD, specifying various learning rates, momentum rates and decay rates. That did not improve performance. I thought it could be batch size. Changing that had no effect. I went through other tests like attempting to alter the network structure itself. But I ran into a number of problems with the network not accepting the dimensions of my input matrices and target vectors. So I remained with the general structure of my network that you see in the Kaggle-submitted model. I finally settled on the notion that my data may be incorrectly generated. So I went back and rewrote the function I

had created to time shift the data. I did not want to rely on the pandas shift() method, even though I understood how it could be used to generate time shifted data for this exact application. When I tried running shift() on a loop ranging from 0 to 24 (early on in the project), the computer took so long to execute that line that I had to restart the kernel each time. I ended up simply converting the entire dataframe to numpy array and iterating over that to generate my time-shifted data. I tried applying the same time shifting function to the target variable as well. The instructions indicate that the target vectors should be flat, 2D arrays of shape (n,1) but as a last resort I tried time-shifting them. This did not have any impact when running the model, aside from producing a model with a lower MAE than what I had been getting. This was the model that I ended up submitting to Kaggle.

After submitting, I went back and redid the network structure and hyperparameter tuning using the original 2D vector of target values. I had a repeat performance of the Kaggle-submitted model whereby my accuracy for training and test sets were fixed, but my loss was changing. Hyperparameter tuning and making changes to the network did not seem to help. I did end up with a network that achieved a lower MAE than the Kaggle submission, but not by much. And that could have easily been due to different random weight initializations during training of that particular model at that particular time. With no fluctuations in training and validation accuracy, I found myself unable to improve the performance of my model and assume still that there must be some issue with my input data or the way I have shaped it.

This assignment was extremely challenging, but also rather rewarding. Though I failed in some regards to increase the performance of my model, I triumphed in gaining a more sound understanding of how time series data is generated and utilized for predicting future events from regular patterns in the past. The data preparation stage took by far the most time for me to complete, as I was not sure how to prepare my feature and target matrices in order to make my model begin learning on the data. I revisited this stage many times and am regretful that I was not able to figure out if the input data was indeed the problem or if there was something else at play that caused my low model performance. In any case, I learned a lot about neural networks this quarter and deeply enjoyed all the programming assignments no matter how technical and challenging.