University of New Brunswick

Time Series Analysis
(EE 6563)

Assignment #4

Professor:
Erik Scheme
Electrical and Computer Engineering

Author:
Saeed Kazemi
(3713280)

April 16, 2021

1. **Explore the attached dataset and find additional information from the resources listed below. Note that this is the same dataset as was used in assignment 2.**
   (a) The dataset comprises NY Stock Exchange with several additional predictors, as explained in the following paper (especially sections 5 and 6): (Source).
   (b) The original dataset was obtained from: (This link).
   (c) Although the original dataset includes 5 different files, we will only use the ”NYSE.csv” file, which includes values from 2010 to 2017.

2. **Hold out the last 3 months of 2017 for out-of-sample prediction and implement the following:**

    (a) **Begin by forecasting with a single LSTM-layer and optimize its performance by varying the various hyper-parameters, for example: LSTM units, no. of layers, batch size, learning rate, etc. Plot and analyze the model performance (accuracy and loss) vs. the number of epochs. You should also test different techniques to avoid model overfitting. Be sure to include and contrast examples of the different approaches – don't only show "the best". Plot the predictions and report the forecasting error using appropriate metrics.**

    (b) **Repeat the forecasting using a 1D CNN-based forecasting model. Evaluate and optimize its performance similar to point part (a). This may require you to go beyond course material, but there are plenty of resources online that you can leverage.**

    (c) **Now, implement a hybrid ConvLSTM forecasting model by combining the CNNs and LSTMs together. Evaluate and optimize its performance in comparison to part (a) and (b). Plot the predictions and report the forecasting error using appropriate metrics in comparison to part (a) and (b).**

Figure 1 depicts my framework for all three parts of this assignment. For each part, we used the same procedure but with different architectures for DNN. In the following, I will discuss each step.

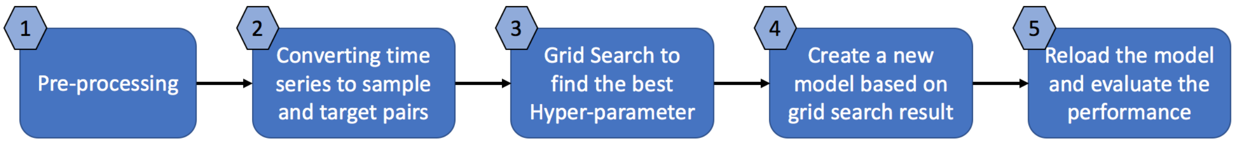

1
Pre-processing
2
Converting time series to sample and target pairs
3
Grid Search to find the best Hyper-parameter
4
Create a new model based on grid search result
5
Reload the model and evaluate the performance

Figure 1: The framework that was used for this question.

In the pre-processing step, we used the last 50 samples for the test and others for the training set. This part is equal to the last three months. Then the MinMax function was used to change the input scaling between zero and one due to the fact that the signal range was large. for this purpose, I used the training set for finding min and max values and then transformed both training and test set based on these two values. Figure 2 shows the raw data used for this project.

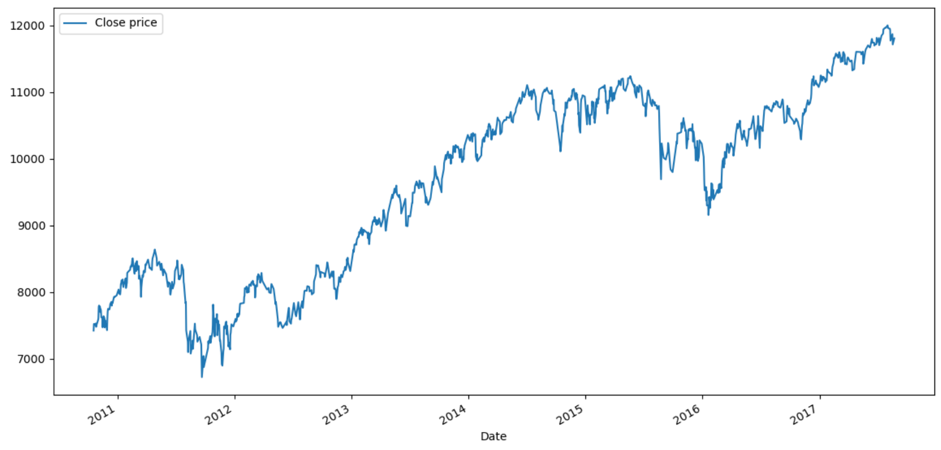
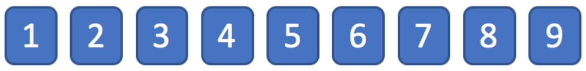
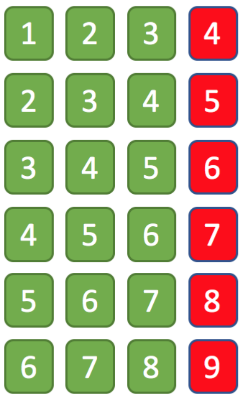


Figure 2: The raw signal.

In the next step, I have converted normalized time-series data to pairs samples and output, or in other words, the supervised data. For doing this, I have used the create_dataset() function from the tutorial code. This function needs two arguments time-series signal and window size and returns a sequence of a signal based on window size as input, and then the next value as output. The below image depicts this converting with the window size of 3. For this question, the window size is set to 15.



(a) The time-series signal.



(b) Samples sequence with corresponding output values.

Figure 3: Converting time-series signal to the supervised data. For this example, window size is equal to 3.

To optimize our model, I have used grid search to find the best hyperparameters. Since grid search is a function on the Scikit-learn library, we needed to convert the Keras model to a Scikit-learn model. So that I have imported KerasClassifier from "keras.wrappers.scikit_learn". Then a GridSearchCV instance was made based on our model, the search space, and a 5-fold cross-validation object. Tables 1 to 3 show the search space of grid search as well as the best hyper-parameters for our model for all three parts of this question.

Table 1: The search space of GridSearchCV method as well as the best hyper-parameters for part (a).

```
param_grid=
    {
        'LSTM_units':     [5, 15, 10],
        'batch_size':     [16, 32, 64],
        'learning_rate':  [0.0001, 0.001, 0.01]
    }

Best hyper-parameters=
    {
        'LSTM_units':     10,
        'batch_size':     16,
        'learning_rate':  0.01
     }
```

Table 2: The search space of GridSearchCV method as well as the best hyper-parameters for part (b).

```
param_grid=
    {
        'batch_size':     [16, 32, 64],
        'learning_rate':  [0.0001, 0.001, 0.01]
    }

Best hyper-parameters=
    {
        'batch_size':     16,
        'learning_rate':  0.01
     }
```

Table 3: The search space of GridSearchCV method as well as the best hyper-parameters for part (c).

```
param_grid=
    {
        'LSTM_units':     [5, 15, 10],
        'batch_size':     [16, 32, 64],
        'learning_rate':  [0.0001, 0.001, 0.01]
    }

Best hyper-parameters=
    {
        'LSTM_units':     5,
        'batch_size':     16,
        'learning_rate':  0.001
     }
```

After optimizing hyperparameters, a new model has been created according to the best hyperparameters of the last step. Then the model was trained with 100 epochs. Also, validation_split was set to 0.2 which means 20 percent of the training data to be used as validation data. Validation data was used for avoiding the overfitting problem. Moreover, a callback method was used to track the loss function on the training and validation set. The callback is called only in the training phase with the model.fit() function to save the best model in a checkpoint file, so the model could be loaded later. Tables 4 to 6 indicate the results of overtrained and trained models on the test and train set.

Table 4: The results of overtrained and trained models on the test and train set for part (a).

	RMSE
Overtrained model on Train set:	105.94
Trained model on Train set:	106.13
Overtrained model on Test set:	142.18
Trained model on Test set:	82.33

Table 5: The results of overtrained and trained models on the test and train set for part (b).

	RMSE
Overtrained model on Train set:	125.88
Trained model on Train set:	110.47
Overtrained model on Test set:	314.87
Trained model on Test set:	117.64

Table 6: The results of overtrained and trained models on the test and train set for part (c).

	RMSE
Overtrained model on Train set:	121.36
Trained model on Train set:	117.86
Overtrained model on Test set:	600.56
Trained model on Test set:	318.32

For part (a), an LSTM model was built. Figure 4 indicates the layers of this model. Figures 5 and 6 show the predicted output and Training and Validation Loss, respectively.

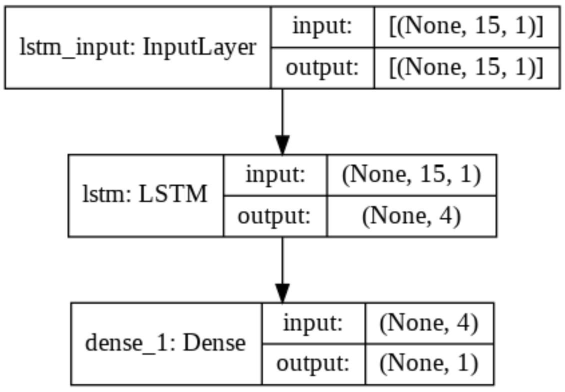


Figure 4: The graph of model used in part (a).

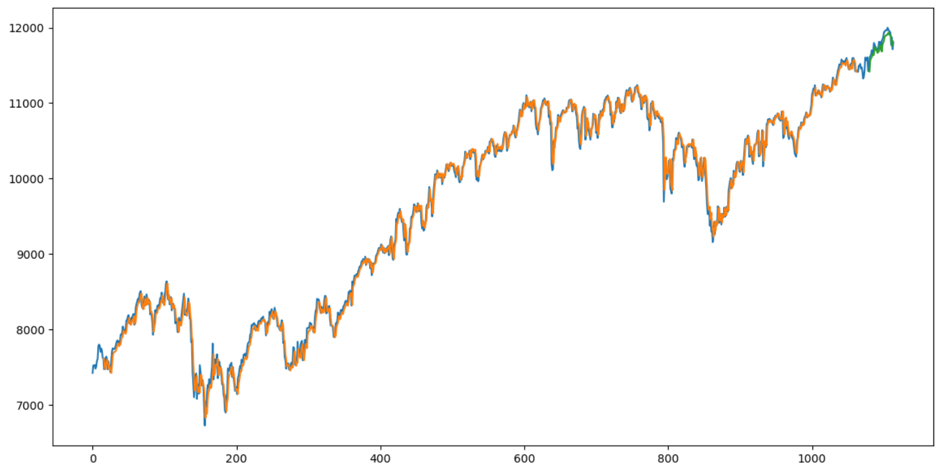


Figure 5: The original signal and predicted value of model (a).

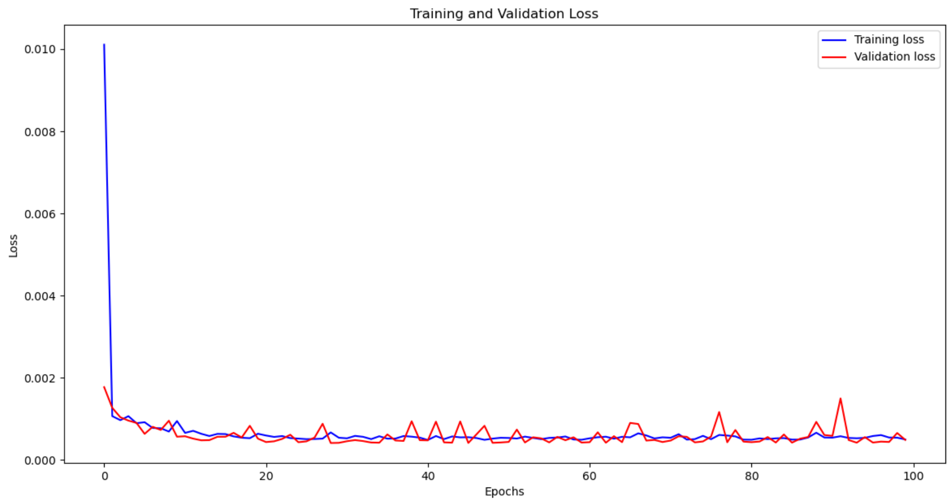


Figure 6: Training and Validation Loss of model (a) versus epoch.

For part (b), Fully Convolutional Networks (FCN) [1] has been implemented. This model is an end-to-end convolutional network with three blocks. Each block has a convolutional layer followed by batch normalization and ReLU activation layer (see figure 7). The output of the third block is averaged over the *time dimension which corresponds to the global average pooling (GAP) layer. Finally, a dense layer is fully connected to the GAP layer's output to produce the final prediction. Figures 8 and 9 show the predicted output and Training and Validation Loss, respectively.*

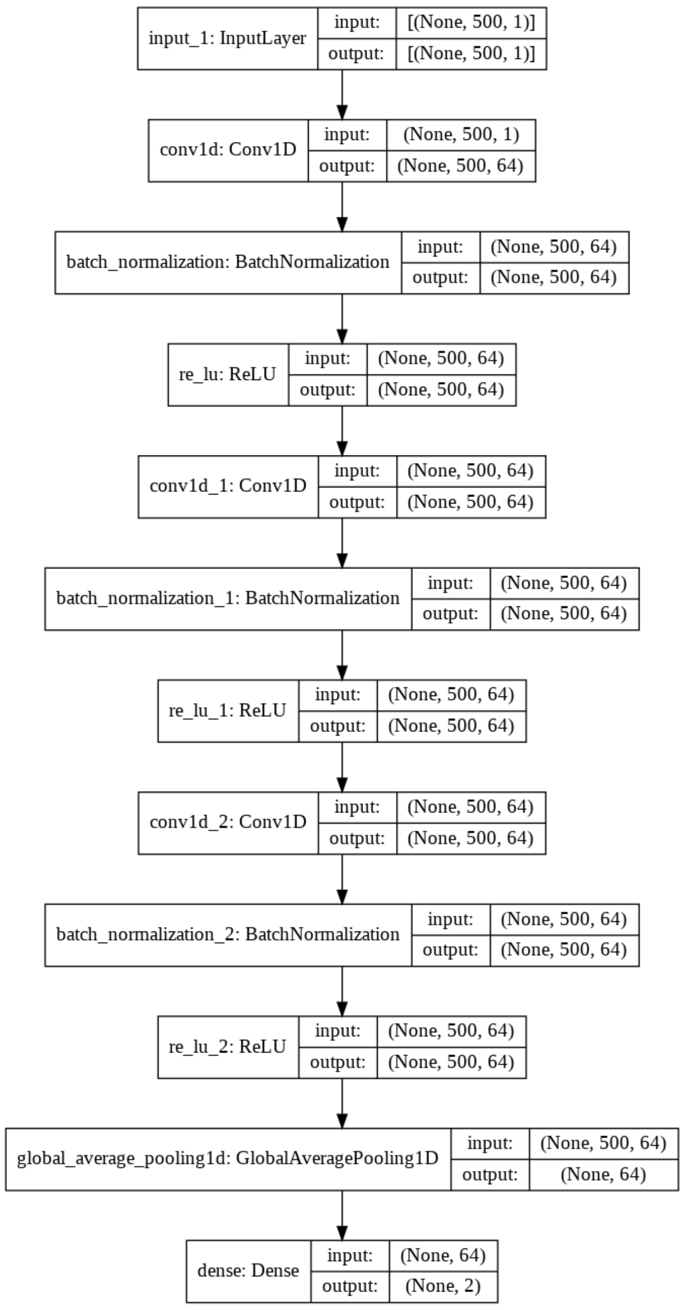

input_1: InputLayer | input: [(None, 500, 1)] | output: [(None, 500, 1)]
conv1d: Conv1D | input: (None, 500, 1) | output: (None, 500, 64)
batch_normalization: BatchNormalization | input: (None, 500, 64) | output: (None, 500, 64)
re_lu: ReLU | input: (None, 500, 64) | output: (None, 500, 64)
conv1d_1: Conv1D | input: (None, 500, 64) | output: (None, 500, 64)
batch_normalization_1: BatchNormalization | input: (None, 500, 64) | output: (None, 500, 64)
re_lu_1: ReLU | input: (None, 500, 64) | output: (None, 500, 64)
conv1d_2: Conv1D | input: (None, 500, 64) | output: (None, 500, 64)
batch_normalization_2: BatchNormalization | input: (None, 500, 64) | output: (None, 500, 64)
re_lu_2: ReLU | input: (None, 500, 64) | output: (None, 500, 64)
global_average_pooling1d: GlobalAveragePooling1D | input: (None, 500, 64) | output: (None, 64)
dense: Dense | input: (None, 64) | output: (None, 2)

Figure 7: The graph of model used in part (b).

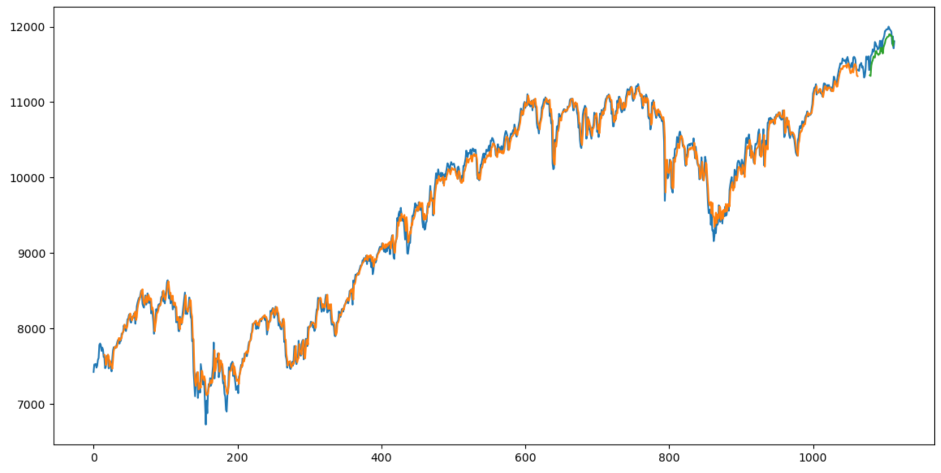


Figure 8: The original signal and predicted output of model (b).

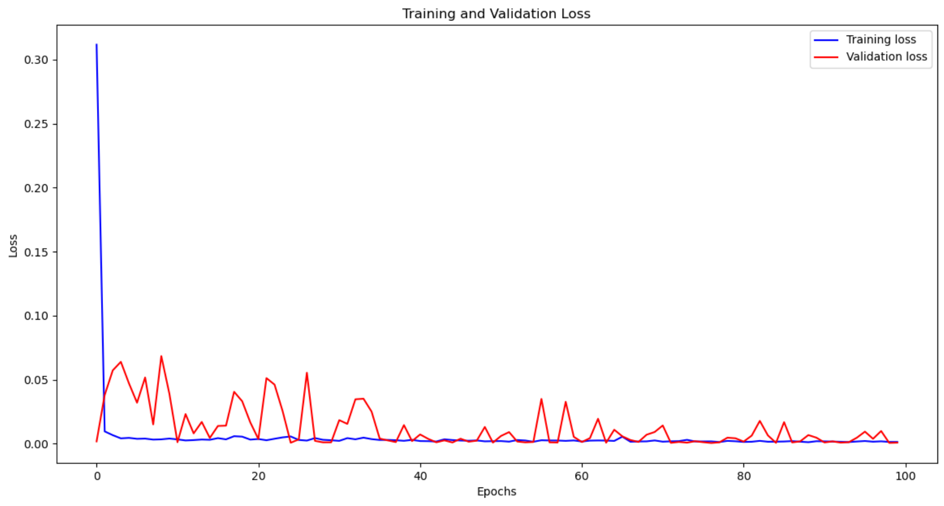


Figure 9: Training and Validation Loss of model (b) versus epoch.

For part (c), since Keras library does not support the ConvLSTM1D, a CNN-LSTM model was implemented (I talked about it with Pradeep). For building this model, I have used Fully Convolutional Networks (FCN) in a Sequence and then replaced the GAP layer with an LSTM layer (see figure 10). Fig-

ures 11 and 12 show the predicted output and Training and Validation Loss, respectively.

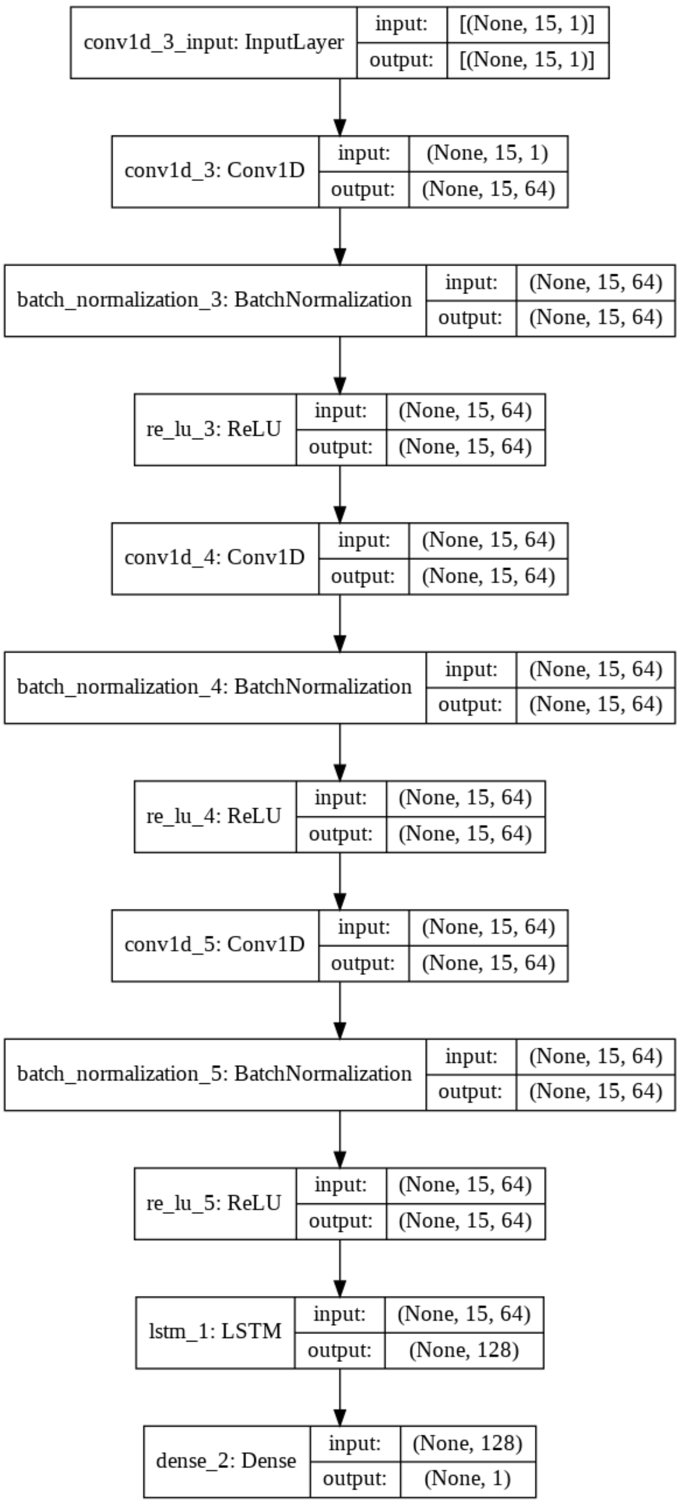


Figure 10: The graph of model used in part (c).

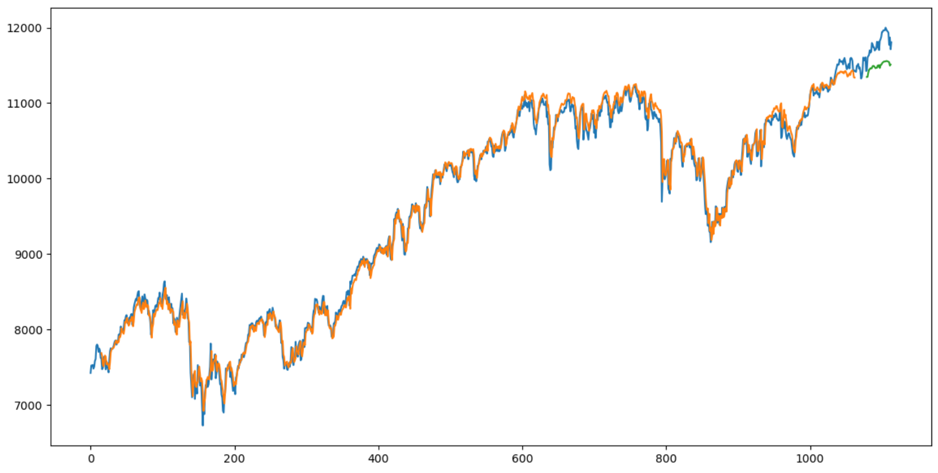


Figure 11: The original signal and predicted output of model (b).

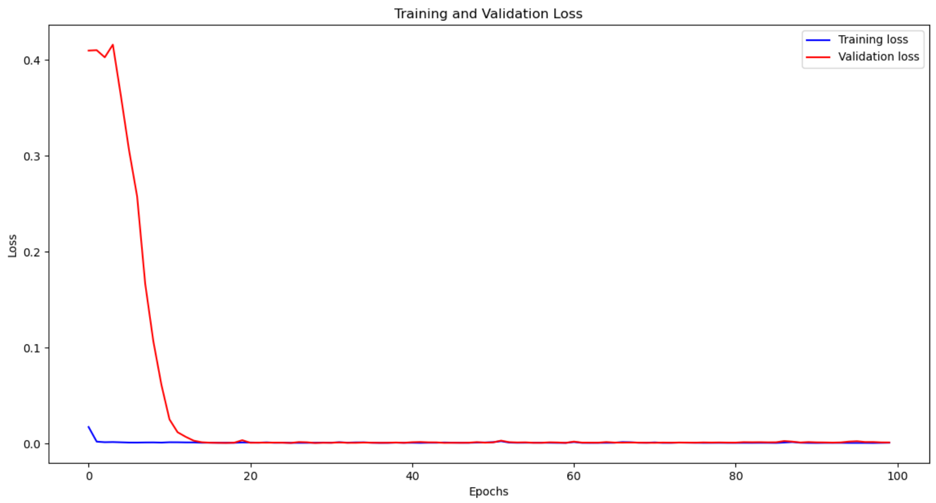


Figure 12: Training and Validation Loss of model (c) versus epoch..

3. **Finally, compare your best model performance with the best statistical model found in Assignment 2 (ARIMA, SARIMA, VAR, etc.). Report and discuss your findings why certain models would outperform the others?**

   In assignment 2, The RMS error of VAR and ARIMAX were 117.565 and 256.297, respectively. However, the best RMS error for deep networks was 82.33, which is lower than the previous assignment. Both results of the statistical models were achieved by window rolling approaches. Furthermore, both VAR and ARIMAX models have leveraged additional information as exogenous variables. However, deep networks have used only the main variable for prediction. Also, after the training phase, the deep networks would not be updated.

REFERENCES

[1] Z. Wang, W. Yan, and T. Oates, “Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline,” Tech. Rep. [Online]. Available: https://github.com/cauchyturing/.