London Business School

Exploring Various Neural Networks

Zervaan Borok

London Business School

Introduction

The purpose of this report is to analyze the performance of various neural network models on two separate datasets. The first dataset contained customer churn data for a telecommunications company named Telco. In this problem, Telco was concerned about the number of customers that are leaving their land-line services in favor of cable competitors. The task here was to create a neural network to explain which customers are leaving and why they are leaving. The second dataset pertained to Apple's stock price for the date range December 12th, 1980 to January 31st, 2022. Here, the task was to create various neural networks using different combinations of the available features to predict the next day's closing price. We will begin by analyzing the results of the customer churn dataset before moving on to the Apple stock price dataset. Should you wish to refamiliarize yourself with how a basic neural network functions, and how it relates to the human brain, refer to Appendix 1.

Customer Churn Dataset

Data Preparation

The customer churn dataset contained 7,043 observations of 21 variables. The variables and their definitions are listed in Appendix 2, Table 1. Before any analysis could be conducted, the dataset had to be cleaned and prepared. To begin, the customer ID field was removed. This was done for two reasons. Primarily because it solely serves as a unique identifier and does not contain any predictive power. Secondarily because including it in the model would have increased the model's run time. No trend can be established with customer ID's because each one is unique (or should be) and so each will represent a unique occurrence – this is not helpful for training the model nor does it increase predictive power. In fact, including this variable might just add noise to the data and thus actually reduce model accuracy. Next, rows with missing values were removed from the dataset, which reduced the dataset by 10 observations (0.142%). This was done for the sake of model accuracy more than anything else. Had these rows not been dropped, the model may have suffered from variance understatement, distortion of data distribution, and correlation depression, but likely not to a significant degree[1] (Gill, Asefa, Kaheil and McKee, 2007). Removing these rows results in more accurate model predictions. However, the disadvantage of this approach is that the model has less observations from which to learn. Additionally, should the model ever be passed incomplete data, it's accuracy might be relatively poor if it was trained on only complete data. Removing rows also decreases model execution time. It should be noted that there are other methods of handling missing data besides removing the observations entirely. Six of these methods, and their brief descriptions, are provided below.

- **Impute Missing Values with Mean:** Use the column mean to fill the empty cells.
- **Impute Missing Values with Median:** Use the column median to fill the empty cells.
- **Impute Missing Values with Mode:** Use the column mode to fill the empty cells.
- **Last Observation Carried Forwards**: For each column, use the last valid value to fill the next missing value.
- **Use Supported Algorithms**: Some algorithms, like K-Nearest-Neighbors (KNN) and Naïve Bayes, can handle missing data.
- **Predict Missing Values**: Train the model on complete rows; use it to predict missing values[2] (Kumar, 2020).

Later, we will see how model performance changes when different imputation methods are used. For now, we will continue with the data preprocessing. Next, a logarithmic transformation on the 'Total Charges' variable was performed in order to normalize it. The scale of this variable is vastly different to that of the others in the equation and so this could reduce model accuracy if such a variable is not scaled prior to being passed to the model. This variable also has a much wider range of values than the other variables and so the logarithmic transformation also reduces its range. Following this, the entire dataset was centered and scaled, i.e., normalized. This was done so that every field in the dataset had the

same scale, and therefore, the neural network gradients would not be biased towards larger values. Finally, the dataset was split into training and testing sets in which 80% of the data was allocated to training and 20% to testing. We will now move on to the model construction.

Model Construction

The artificial neural network (ANN)* model consisted of two hidden layers, each had the units* argument set to 16, the dropout rate* set to 0.1, a uniform kernel initializer*, and a RELU activation function*. The output layer also used a uniform kernel initializer, but a sigmoid*, rather than RELU, activation function. Additionally, an Adam optimizer* and binary cross-entropy loss function* were used in this model's compilation step. Finally, the batch size* was set to 50 and the number of epochs* was set to 35. We used the sigmoid activation function in the output layer because we were dealing with binary classification. The sigmoid function ensures the model only returns values between 0 and 1. We used binary cross-entropy because, again, we were dealing with a binary response variable. By contrast, in a multiclass classification problem, we would use Softmax* as the activation function in the output layer and categorical cross-entropy* as the loss function. We would do this because Softmax converts a vector of values to a probability distribution and categorical cross-entropy accommodates any number of categories, not just two[3] (Brownlee, 2021). The model was trained with the training dataset and tested with the testing dataset. The results are shown in the table below.

Accuracy*	ROC AUC*	Precision*	Recall*	F-Measure*
0.7986	0.8437	0.6207	0.5308	0.5723

From the table above, we can see that the model performs relatively well on the testing dataset. That being said, there is certainly still some room for improvement via hyperparameter tuning, which we will explore later. For now, we will examine how the model performed under three modified datasets in which each dataset was subjected to a different imputation method.

Handling Missing Values

In the following table we compare the model outputs under three different datasets. In the first dataset, 35% of the values for the columns 'tenure', 'MonthlyCharge', and 'TotalCharge' were converted to NA values. The rows which contained these NA values were then removed. This reduced the dataset to 4,571 observations. In the second dataset, 35% of the values for the columns 'tenure', 'MonthlyCharge', and 'TotalCharge' were, again, converted to NA values. However, this time, instead of removing the rows that contained the NA values, the NA values were imputed with the respective column mean. Thus, in this case, the dataset was not reduced. The third dataset was manipulated in much the same way as the second, except this time the NA values were imputed with the respective column median. The table below illustrates the performance of the model under each of the three scenarios.

Model	Accuracy	ROC AUC	Precision	Recall	F-Measure
35% Removed	0.7978	0.8443	0.6577	0.5725	0.612
35% Mean Imputed	0.7903	0.8295	0.6519	0.5269	0.5827
35% Median Imputed	0.7846	0.8287	0.6375	0.5217	0.5738

We can see that the model performed best when the NA rows were dropped instead of mean, or median, imputed. It is also evident that the model performed better when the NA cells were imputed with their respective column median in comparison to when the NA cells were imputed with their respective column mean. We will now explore the hyperparameter tuning of the model.

London Business School

Hyperparameter Tuning

In order to tune the hyperparameters, a grid search of sorts was performed on the dropout rate and units parameters. The results of this search indicated the optimal value for the first dropout rate was 0.001 while the optimal value for the second dropout rate was 0.1. The optimal value for the number of units was 16 for both hidden layers. For comparison, a KNN machine learning algorithm was also constructed and passed the same dataset. The table below shows the difference in performance between the untuned neural network, tuned neural network, and KNN models. The table below compares the performance of the untuned ANN, tuned ANN, and KNN models.

Model	Accuracy	ROC AUC	Precision	Recall	F-Measure
Untuned ANN	0.7896	0.8437	0.6207	0.5308	0.5723
Tuned ANN	0.7946	0.8443	0.6391	0.5174	0.5719
KNN	0.7711	0.8016	0.5416	0.5722	0.5565

As evidenced by the table above, the ANN models outperform the KNN model. The tuned ANN is marginally better than the untuned ANN, but not significantly so. The concern of overfitting the neural networks can be alleviated by the knowledge that the values in the table above were derived by passing the unseen test dataset to the models. Additionally, convergence plots of the untuned and tuned ANN models can be found in Appendix 4. It is possible that the tuned ANN model would significantly outperform the untuned one if the range of the hyperparameter grid search had been larger.

Apple Stock Price Dataset

Data Preparation

The Apple stock dataset contained the opening price, closing price, adjusted closing price, high of the day, low of the day, and the volume for the years 1980-2022. This dataset did not contain any missing values, which is not atypical when working with financial data. The only issue was that the 'volume' field had an entry of 0 for one of the days – whether this is an error or not is somewhat irrelevant as this pertains to only one record out of the 10,372 in the dataset. However, in order for some linear transformations to work, such as the logarithmic transformation, fields that are set to be transformed must contain non-zero values. Thus, for this single instance, the value of 0 was replaced by the column median. The base dataset was then asymmetrically trisected; the training dataset was allocated the first 9,000 observations, the validation dataset was allocated the next 1,000 observations, and the testing dataset was allocated the remaining 1,372 observations – this same split was done for each model in the analysis that follows (we cannot do random allocations here because we are working with time series data). After the data had been prepared, model construction could begin.

LSTM Model Construction

The first model constructed was a very rudimentary one in which only the three previous days' closing prices were used to predict the next day's closing price. This model had a batch size* of 10, one long-short-term-memory (LSTM)* layer in the neural network, and 32 units* within that one layer. The number of epochs* was set to 100 and the callback function was enabled to stop the model from training if there was no learning for 5 epochs. Additionally, the gradient descent optimizer* and the mean squared error loss function* were specified within the model. The kernel initializer*, optimizer*, activation function*, and dropout rate* parameters were not specified. The next five models were also passed these same parameter values, but each utilized different features to predict the next day's closing price. Each feature in each model used its values from the three previous days for said prediction. In the model performance comparison table below, the

name of each model indicates which features were used – the 'Full Model' contained all the available features. Evidently, the 'Full Model' and the 'Close Price & Volume' model performed the best out of the six created.

Model	R-Squared*	RMSE*
Close Price & Low	0.9266405	42.66085
Close Price & Adj Close Price	0.9438863	40.26519
Close Price Only	0.9574976	39.02544
Close Price & High	0.9578958	37.54735
Full Model	0.9783324	46.44875
Close Price & Volume	0.9868369	39.16758

Simple RNN Model Construction

Following this, four simple recursive neural network (RNN)* models were created to compare the RNN and LSTM methods. All of these models were passed the same parameters as the six models in the table above and used the same aforementioned train-test-validation split. Only the method and features differed. In the model performance comparison table below, the name of each model indicates which features were used. Again, it is apparent that the last two models performed the best out of the four that were created, even though all four have very good R-squared values.

Model	R-Squared	RMSE
Close Price, Adj Close Price, & Low	0.9131433	42.60290
Close Price Only	0.9247131	34.49151
Close Price, Adj Close Price, & Volume	0.9961583	13.98663
Full Model	0.9962308	16.11142

Hyperparameter Tuning of LSTM & Simple RNN Models

Next, the hyperparameters of the 'Full Model' under both the LSTM method and simple RNN method were fine tuned. Utilizing logic along with some trial and error resulted in the following hyperparameter changes. The batch size was reduced to 5 from 10, the number of units in the LSTM / (RNN) layer of the model was increased from 32 to 256, the callback function was not implemented, and the number of epochs was set to 250. The relatively optimal hyperparameter values listed above applied to both the simple RNN model and the LSTM model. The table below shows the difference in performance between the untuned and tuned 'Full Models', while the respective plots can be found in Appendix 5.

Model	R-Squared	RMSE
Untuned LSTM	0.9783324	46.44875
Tuned LSTM	0.9977706	10.51505
Untuned RNN	0.9962308	16.111422
Tuned RNN	0.9985557	1.777648

Even though all of these models exhibit impressive accuracy, it is clear that the tuned models outperform their untuned counterparts by a considerable margin, as evidence by the large differences in RMSE values. The table also illustrates that the tuned RNN model outperformed all of the other models by a considerable margin. While the differences in R-squared values between the four models are not very significant, the same cannot be said for the RMSE values. It should be noted that it is possible the tuned RNN model's accuracy could be increased even more by further reducing the batch size, increasing the number of epochs, and increasing the number of units in the simple RNN layer.

Two Day Prediction

Finally, an attempt was made to construct a model capable of predicting the next two days' closing prices using the same historical data. However, instead of using the three previous days' data, the seven previous days' data was used. For this analysis, the full model under the simple RNN method was employed. Here, the batch size was set to 10, the number of units in the simple RNN layer was set to 128, the callback function was not utilized, and the number of epochs was set to 200. The table below depicts the model's performance on predicting the closing price one day into the future and two days into the future; the associated plot can be found in Appendix 6. As we can see, the model's performance does deteriorate when asked to predict the closing price two days into the future as compared to one day into the future, but only by a very small margin. The model is still very accurate with regards to the two-day predictions.

Model	R-Squared	RMSE
Predicted Price 1 Day	0.9982128	5.9494194
Predicted Price 2 Day	0.9978712	7.5271178

Conclusion

The analysis of the customer churn data revealed some interesting findings regarding neural networks. First, they are very powerful in predicting the outcome of binary classification problems and tend to out-perform traditional machine learning classification algorithms such as the K-Nearest-Neighbors algorithm and Support Vector Machine algorithm. Second, they continue to perform well even when the data has been subjected to an imputation method. Finally, the results from the analysis indicate that observations with missing values should be removed, rather than imputed, so long as a sufficient number of complete observations remain. The Apple stock analysis also provided some key takeaways with respect to utilizing neural networks in practice. Even though the models constructed for the Apple stock analysis performed very well, there are still some limitations that ought to be addressed. The most notable challenge stems from the inherent characteristics of the stock market itself. For example, in a factory, deep learning can help improve efficiency by optimizing certain procedures that are subject to fixed logistical business constraints. By contrast, there are no pre-defined boundaries or procedures that the stock market obeys or follows. Additionally, the stock market is never stationary. Furthermore, there are events capable of dramatically affecting the stock market which neither humans nor machines can even hope to predict. These events can take the form of social unrest, political instability, natural disasters, and war, amongst others. As an illustration of the impact such events can have on the stock market, we will examine two that have occurred in recent history. The first event, the COVID-19 pandemic, began in early 2020; and the second event, the Russian-Ukrainian war, is currently ongoing. With respect to the COVID-19 pandemic, logic would dictate that such an event ought to cripple the entire stock market; however, this did not happen. While the entire market did fall when the pandemic first broke out, it rebounded within a few months. In fact, some industries, such as the technology sector, actually experienced unprecedented growth during this time period[21] (James, 2020). Due to the fact that the war in the Ukraine only began very recently, we have very limited data which we can analyze in order to assess the impact it has had on the stock market. When the war first broke out, markets tumbled quite steadily, but were far from crashing. Following the announcements of global sanctions against Russia shortly after the war commenced, many indices experienced a sharp rebound. Considering Russia is one of the five global superpowers on the United Nations Security Council and armed with a vast expanse of nuclear warheads, the markets should have fallen much more drastically than they have, but only time will tell if this atypical trend continues. Deep learning algorithms are also unable to account for slippage* and liquidity* in the stock market. These two factors can drastically impact profits, especially when one is trading in a very narrow time frame, such as the intraday one. In short, deep learning algorithms are very powerful tools capable of revealing patterns in the data that would otherwise remain hidden, but their use in the stock market should be exercised with due caution as there are some incredibly influential external factors that are impossible to predict or account for.

London Business School

Reference List

1. Gill, M., Asefa, T., Kaheil, Y. and McKee, M. (2007) 'Effect of Missing Data on Performance of Learning Algorithms for Hydrologic Predictions: Implications to an Imputation Technique', *Water Resources Research*, 43, pp. 1-2. doi: 10.1029/2006WR005298

2. Kumar, S. (2020) *7 Ways to Handle Missing Values in Machine Learning*. Available at: https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e (Accessed: 26 February 2022).

3. Brownlee, J. (2021) *Multi-Class Classification Tutorial with the Keras Deep Learning Library*. Available at: https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/ (Accessed: 26 February 2022).

4. Kang, N. (2017) *Introducing Deep Learning and Neural Networks — Deep Learning for Rookies (1)*. Available at: https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf5883 (Accessed: 26 February 2022).

5. Techopedia. (2021) *Artificial Neural Network*. Available at: https://www.techopedia.com/definition/5967/artificial-neural-network-ann (Accessed: 26 February 2022).

6. DeepAI. *Recurrent Neural Network*. Available at: https://deepai.org/machine-learning-glossary-and-terms/recurrent-neural-network (Accessed: 26 February 2022).

7. Brownlee, J. (2021) *A Gentle Introduction to Long Short-Term Memory Networks by the Experts*. Available at: https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/ (Accessed: 26 February 2022).

8. Wang, J. (2019) *What is "units" in LSTM layer of Keras?* Available at: https://zhuanlan.zhihu.com/p/58854907 (Accessed: 26 February 2022).

9. Brownlee, J. (2020) *How to Reduce Overfitting with Dropout Regularization in Keras*. Available at: https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/ (Accessed: 26 February 2022).

10. Keras. *Layer Weight Initializers*. Available at: https://keras.io/api/layers/initializers/#randomnormal (Accessed: 26 February 2022).

11. Chen, B. (2021) *7 Popular Activation Functions You Should Know in Deep Learning and How to Use Them with Keras and TensorFlow 2*. Available at: https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6 (Accessed: 26 February 2022).

12. Radečić, D. (2020) *Softmax Activation Function Explained*. Available at: https://medium.com/towards-data-science/softmax-activation-function-explained-a7e1bc3ad60 (Accessed 26: February 2022).

13. Maithani, M. (2021) *Guide to Tensorflow Keras Optimizers*. Available at: https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/ (Accessed: 26 February 2022).

14. Baeldung. (2020) *The Difference Between Epoch and Iteration in Neural Networks*. Available at: https://www.baeldung.com/cs/neural-networks-epoch-vs-iteration#3-batch (Accessed: 26 February 2022).

15. Keras. *Losses*. Available at: https://keras.io/api/losses/ (Accessed: 26 February 2022).

16. Great Learning Team. (2020) *Understanding ROC (Receiver Operating Characteristic) Curve | What is ROC?* Available at: https://www.mygreatlearning.com/blog/roc-curve/#performanceassessment (Accessed: 26 February 2022).

17. Brownlee, J. (2020) *How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification*. Available at: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/#:~:text=Precision%20quantifies%20the%20number%20of,and%20recall%20in%20one%20number. (Accessed: 26 February 2022).

18. Glen, S. (2021) *RMSE: Root Mean Square Error*. Available at: https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/ (Accessed: 26 February 2022).

19. Fernando, J. (2021) *R-Squared*. Available at: https://www.investopedia.com/terms/r/r-squared.asp (Accessed: 26 February 2022).

20. Brownlee, J. (2020) *A Gentle Introduction to Cross-Entropy for Machine Learning*. Available at: https://machinelearningmastery.com/cross-entropy-for-machine-learning/ (Accessed: 26 February 2022).

21. James, M. (2020) *Can Machine Learning Models Accurately Predict the Stock Market?* Available at: https://www.smartdatacollective.com/can-machine-learning-models-accurately-predict-stock-market/ (Accessed: 26 February 2022).

22. Hayes, A. (2021) *Slippage*. Available at: https://www.investopedia.com/terms/s/slippage.asp (Accessed: 26 February 2022).

23. Hayes, A. (2021) *Liquidity*. Available at: https://www.investopedia.com/terms/l/liquidity.asp (Accessed: 26 February 2022).

24. R Core Team (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

25. JJ Allaire and François Chollet (2022). keras: R Interface to 'Keras'. R package version 2.7.0.9000. https://keras.rstudio.com

26. JJ Allaire and Yuan Tang (2022). tensorflow: R Interface to 'TensorFlow'. R package version 2.7.0.9000. https://github.com/rstudio/tensorflow

27. Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

28. Julia Silge, Fanny Chow, Max Kuhn and Hadley Wickham (2021). rsample: General Resampling Infrastructure. R package version 0.1.0. https://CRAN.R-project.org/package=rsample

29. Max Kuhn and Hadley Wickham (2021). recipes: Preprocessing Tools to Create Design Matrices. R package version 0.1.16. https://CRAN.R-project.org/package=recipes

30. Max Kuhn and Davis Vaughan (2021). yardstick: Tidy Characterizations of Model Performance. R package version 0.0.9. https://CRAN.R-project.org/package=yardstick

31. Max Kuhn, Simon Jackson and Jorge Cimentada (2020). corrr: Correlations in R. R package version 0.4.3. https://CRAN.R-project.org/package=corrr

32. Hadley Wickham and Jim Hester (2021). readr: Read Rectangular Text Data. R package version 2.0.1. https://CRAN.R-project.org/package=readr

33. JJ Allaire (2021). tfruns: Training Run Tools for 'TensorFlow'. R package version 1.5.0. https://CRAN.R-project.org/package=tfruns

34. Max Kuhn (2021). caret: Classification and Regression Training. R package version 6.0-88. https://CRAN.R-project.org/package=caret

35. H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

36. Jim Hester (2020). glue: Interpreted String Literals. R package version 1.4.2. https://CRAN.R-project.org/package=glue

37. Hadley Wickham (2021). forcats: Tools for Working with Categorical Variables (Factors). R package version 0.5.1. https://CRAN.R-project.org/package=forcats

38. Matt Dancho and Davis Vaughan (2022). timetk: A Tool Kit for Working with Time Series in R. R package version 2.7.0. https://CRAN.R-project.org/package=timetk

39. Matt Dancho and Davis Vaughan (2021). tidyquant: Tidy Quantitative Financial Analysis. R package version 1.0.3. https://CRAN.R-project.org/package=tidyquant

40. Davis Vaughan and Matt Dancho (2020). tibbletime: Time Aware Tibbles. R package version 0.1.6. https://CRAN.R-project.org/package=tibbletime

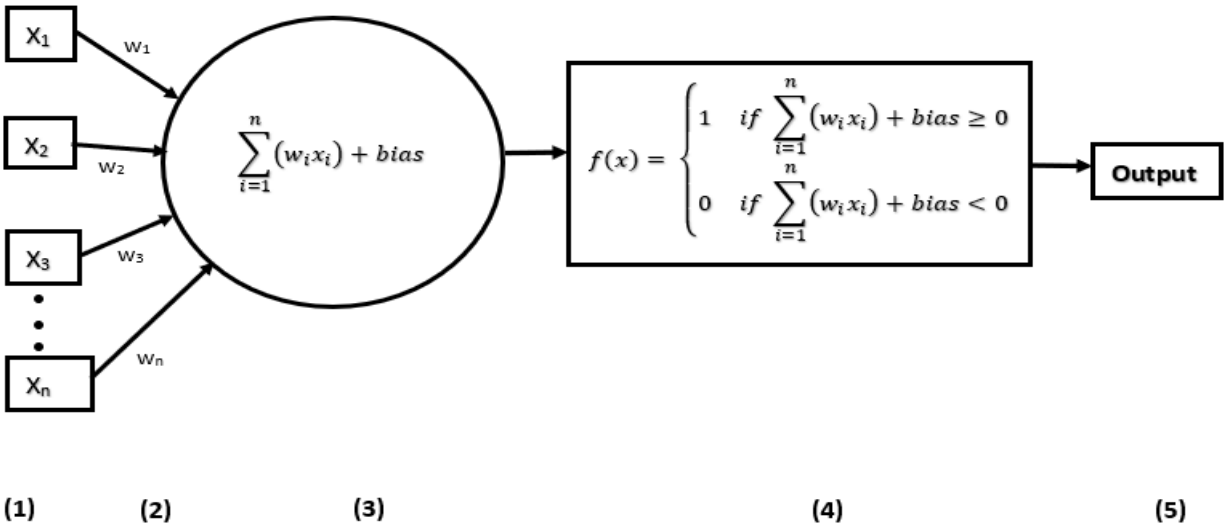
41. Claus O. Wilke (2020). cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'. R package version 1.1.1. https://CRAN.R-project.org/package=cowplot

42. Hadley Wickham and Dana Seidel (2020). scales: Scale Functions for Visualization. R package version 1.1.1. https://CRAN.R-project.org/package=scales

Appendix 1

Neural Networks & Deep Learning

In essence, deep learning is a subfield of machine learning, and machine learning is a subfield of artificial intelligence. Artificial intelligence came into existence in the 1950's, followed by machine learning in the 1980's and deep learning in the 2010's. The reason deep learning only recently gained traction is because historically, computers did not possess the power to properly implement them. That being said, the first instance of a neural network was technically created in 1958 when an American psychologist by the name of Frank Rosenblatt constructed a machine with the intended ability of sensing, recognizing, remembering, and responding to stimulus, or inputs, in a fashion comparable to that of a human brain. This machine, and others that followed, suffered from poor performance until back propagation was introduced in the 1970's. Perhaps unsurprisingly, the way in which neural networks function is inspired by the way neurons in our brain function. Put simply, neurons in our brain work as follows: each nerve cell has extensions, called dendrites, which receive signals and then relay them to the body of the cell. The nerve cell body processes these signals and subsequently decides whether to send signals to other nerve cells or not. Should the nerve cell decide to send signals to other cells, axons, which are another type of cell body extension, will release chemical communications to other cells instructing them to relay these signals. The structure of a neural network is best laid out in diagram format, as shown below.



From left to right, the neural network is structured as follows:

1. On the leftmost side of the diagram, there are four small rectangles. These are the neurons which contain the input data.

2. Next, we have the arrows, called synapses, which apply a neuron-specific weight to each of the inputs.

3. Following this, we have the large singular circle. In this step, each input is multiplied by its corresponding weight and the resulting products are then summed. A pre-determined value representing bias is then added to the result of this summation.

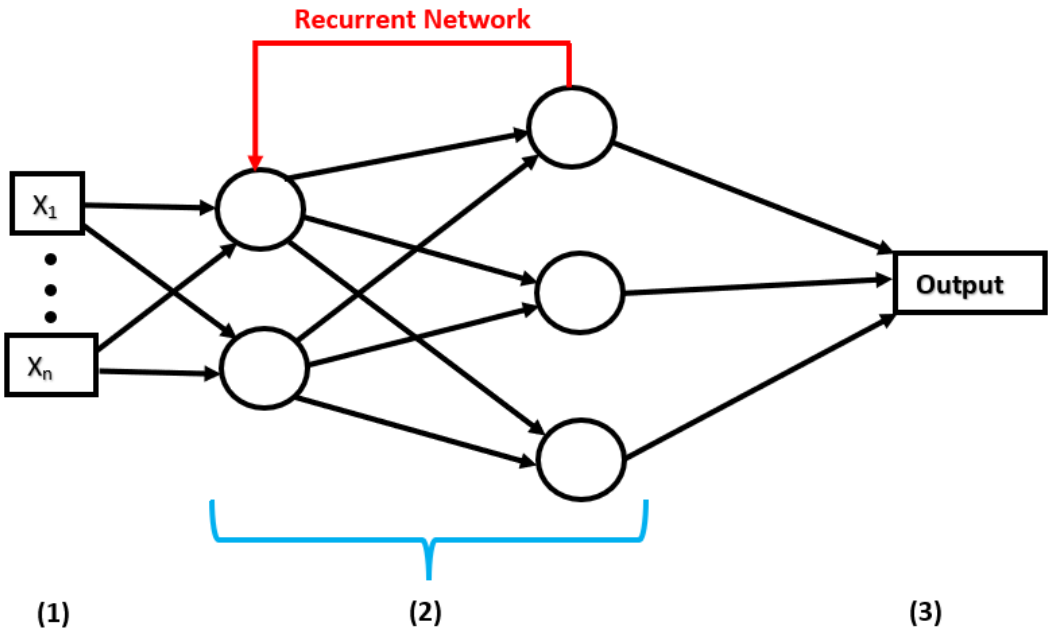
4. The formulas in the rectangle represent a step function (this is very basic and more advanced activation functions, such as gradient descent, are available) which takes the result from Step (3) and assigns an output value of either 0 or 1. If the result from Step (3) is greater than, or equal to, zero, then an output value of 1 is returned. Conversely, if the result from Step (3) is less than zero, then an output value of 0 is returned.

5. Here we have the final output, which takes on the value of either 0 or 1.

The weights for each input are then tuned to their optimal values by using the activation function specified in the model[4] (Kang, 2017).

Recursive Neural Network

Pictured below is a rudimentary diagram of a basic recursive neural network.



From left to right, the neural network is structured as follows:

1. On the leftmost side of the diagram, there are two rectangles, which are the inputs.
    a. Not shown here are the weights, which would be illustrated along the arrows as in the previous diagram.

2. In the middle of the diagram, there are five circles arranged into two columns. Each column of circles represents a hidden layer; thus, there are two hidden layers in the diagram. A model which has more than one hidden layer is considered to be a 'deep' model.

3. On the rightmost side of the diagram, we again have a rectangle representing the output of the model[6] (DeepAI).

Appendix 2

Table 1

Variable	Definition
Customer ID	Unique customer identifier
Gender	Gender of customer
Senior Citizen	Binary variable indicating whether or not the customer is a senior citizen
Partner	Binary variable indicating whether or not the customer has a partner.
Dependents	Binary variable indicating whether or not the customer has dependents
Tenure	Length of time the customer has been/was with the company
Phone Service	Binary variable indicating whether or not the customer has phone service
Multiple Lines	Factor variable indicating whether or not the customer has multiple lines
Internet Service	Type of internet service the customer receives (fiber, DSL, etc.)
Online Security	Binary variable indicating whether or not the customer has online security
Online Backup	Binary variable indicating whether or not the customer has online backup
Device Protection	Binary variable indicating whether or not the customer has device protection
Technical Support	Binary variable indicating whether or not the customer has technical support
Streaming TV	Binary variable indicating whether or not the customer streams TV shows
Streaming Movies	Binary variable indicating whether or not the customer streams movies
Contract	Type of contract customer has with company (monthly, yearly, etc.)
Paperless Billing	Binary variable indicating whether or not the customer has paperless billing
Payment Method	Method with which the customer pays their bill (bank transfer, check, etc.)
Monthly Charges	Monthly charge for the customer
Total Charges	Cumulative charges for the customer
Churn	Binary variables indicating whether the customer is still with the company

London Business School

Appendix 3

Neural Network Definitions

1. **Artificial Neural Network (ANN):** A computational model whose logic imitates the way nerve cells operate within the human brain[5] (Techopedia, 2021).

2. **Recurrent Neural Network (RNN):** A type of neural network in which logic from prior observations is used to inform the forthcoming occurrences[6] (DeepAI).

3. **Long-Short-Term-Memory (LSTM) Neural Network:** A type of recurrent neural network with the ability to learn order dependence in sequential estimation problems[7] (Brownlee, 2021).

4. **Units Parameter:** Dictates to the model the number of units in the layer[8] (Wang, 2019).

5. **Dropout Rate Parameter:** Dictates the dropout rate value to the model. The dropout rate function probabilistically removes inputs, which can take the form of input variables in the data sample or activation functions from a preceding layer, and thus prevents them from being passed to the subsequent layer. This effectively simulates a vast quantity of neural networks with differing structures which, in general, increases the robustness of the network's nodes to the inputs[9] (Brownlee, 2020).

6. **Kernel Initializer Parameter:** Indicates to the model which statistical distribution should be used to initialize the input weights.
    a. **Uniform:** This kernel initializer corresponds to the uniform statistical distribution[10] (Keras).

7. **Activation Function Parameter:** Dictates which activation function the model should use. These functions transform the sum of the weighted inputs before passing them to the output function.

    a. **Rectified Linear Unit (RELU):** $\begin{cases} 0 & if \quad x \leq 0 \\ x & if \quad x > 0 \end{cases}$

    b. **Sigmoid**[11] (Chen, 2021): $\sigma(x) = \frac{1}{1+e^{-x}}$

    c. **Softmax**[12] (Radečić, 2020): $Softmax(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j^{(i)}}}$

8. **Optimizer Parameter:** Dictates which optimizer the model should use. Optimizers modify the neural network attributes, such as the weights and learning rate, to reduce losses.
    a. **Adaptive Moment Estimation (Adam):** This optimizer employs the momentum concept via the addition of previous gradients' fractions to the current one.
    b. **Gradient Descent (SGD):** This is the most popular optimizer and many others, such as Adam, have been built on top of it. SGD works by calculating the changes in every weight parameter that impacts the loss function before tuning each individual weight based on its respective gradient. These steps are repeated sequentially until the loss function arrives at its minimum[13] (Maithani, 2021).

9. **Epoch Parameter:** Dictates the number of epochs the model should use. An epoch is defined as training the neural network model for one cycle using all of the training data. A single epoch constitutes one forward pass and one backward pass through the model[14] (Baeldung, 2020).

10. **Batch Size Parameter:** Dictates the batch size value the model should use. Batch size is defined as the number of training observations in a single iteration[14] (Baeldung, 2020).

11. **Loss Function Parameter:** Indicates to the model which quantity it should seek to minimize in the training process.
    a. **Binary Cross-Entropy:** Calculates the cross-entropy loss between the predicted labels and the true labels.
    b. **Categorical Cross-Entropy:** Functions in a similar fashion to binary cross-entropy but accommodates more than two labels.
    c. **Mean Squared Error (MSE):** Calculates the mean squared errors between predicted and actual labels[15] (Keras).

Measurement Metrics & Financial Terms Definitions

1. **Accuracy:** Proportion of true, and false, observations correctly predicted by the model.

2. **ROC AUC:** The receiver operating characteristic (ROC) is a probability curve while the area under the curve (AUC) represents the degree to which the model is able to distinguish between classes; the higher the AUC, the better the model is at distinguishing between class types[16] (Great Learning Team, 2020).

3. **Precision:** This metric computes the number of correct positive predictions that the model made by dividing the number of correctly predicted positive observations by the total number of positive observations that were predicted by the model[17] (Brownlee, 2020).

4. **Recall:** This metric computes the number of correct positive predictions that the model made out of all the possible positive predictions that the model could have made. The computation entails dividing the number of true positives by the sum of the number of true positives and false negatives[17] (Brownlee, 2020).

5. **F-Measure:** This metric combines precision and recall into a single measurement that captures the properties of the two separate metrics. This measurement is calculated using the formula shown below[17] (Brownlee, 2020).

$$\frac{(2 * \text{Recall} * \text{Precision})}{(\text{Recall} + \text{Precision})}$$

6. **Root Mean Squared Error (RMSE):** Defined as the standard deviation of the prediction errors (i.e., residuals). The formula for performing this calculation is as shown below[18] (Glen, 2021).

$$RMSE_{prediction_0} = \sqrt{\left[\sum_{i=1}^{N} \frac{(x_{prediction_i} - x_{observed_i})^2}{N}\right]}$$

7. **R-Squared:** Defined as a measure which quantifies how much of the variation in the data can be explained by the model. The formula for performing this calculation is as shown below [19] (Fernando, 2021).

$$R^2 = 1 - \left(\frac{Unexplained\ Variation}{Total\ Variation}\right)$$

8. **Cross-Entropy:** Defined as a measure of how much two probability distributions differ for a given set of events or a given random variable[20] (Brownlee, 2020).

9. **Slippage:** In the context of the stock market, slippage is defined as the difference between the trade's expected price and executed price (i.e., the difference between the price shown on the screen when one clicks 'buy' and the price said buy order is actually filled at). Slippage can happen at any point in time, but typically occurs during periods of extreme volatility, which are usually characterized by the usage of market orders instead of limit orders. Slippage can also take place upon the execution of a relatively large order if there is insufficient volume to sustain the current bid-ask spread at the selected entry, or exit, price[22] (Hayes, 2021).

10. **Liquidity:** In finance, liquidity is defined as the ease with which a security, or an asset, can be converted into raw cash without having an impact on its market value. Put simply, if one were to attempt to convert an illiquid asset to cash, the price one would receive would likely be well below the market value of the asset. In the case of a very liquid asset, the received price would likely be very close, if not equal, to the market value of the asset[23] (Hayes, 2021).

Appendix 4

Figure 1: Untuned ANN Convergence Plot

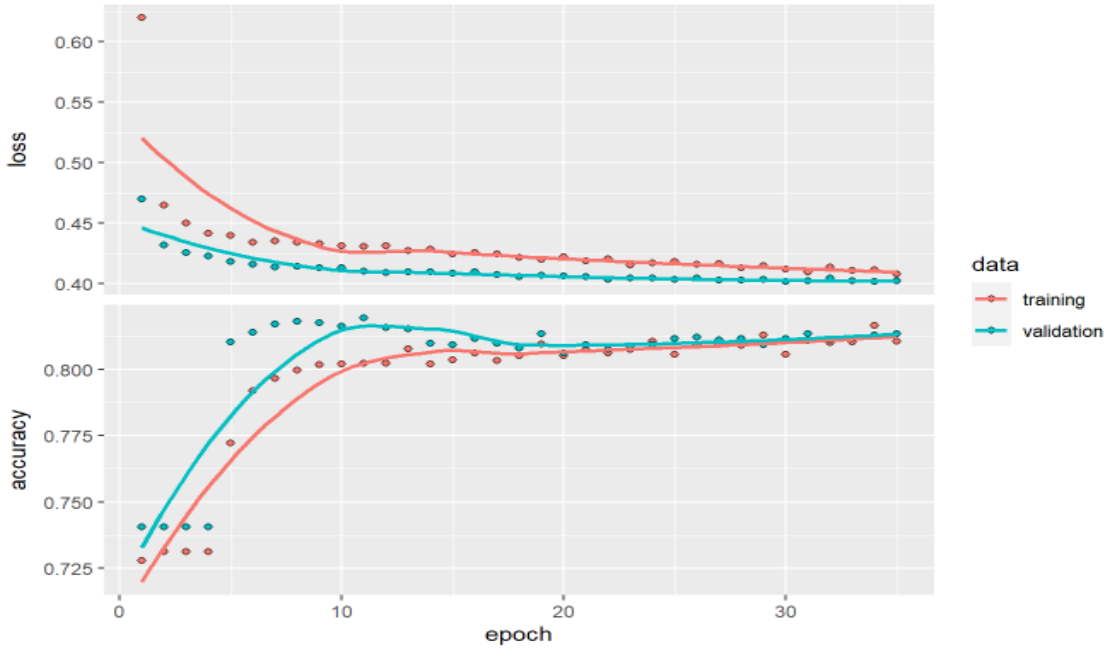
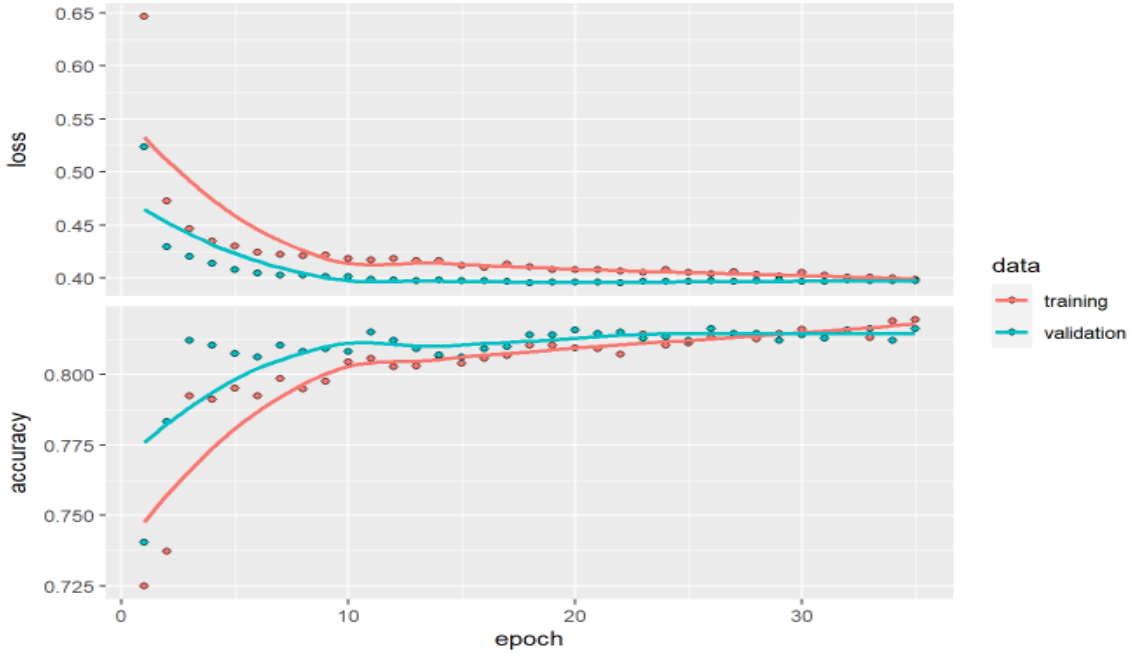


Figure 2: Tuned ANN Convergence Plot



Appendix 5

Figure 1: Untuned LSTM Predicted VS. Actual Plot

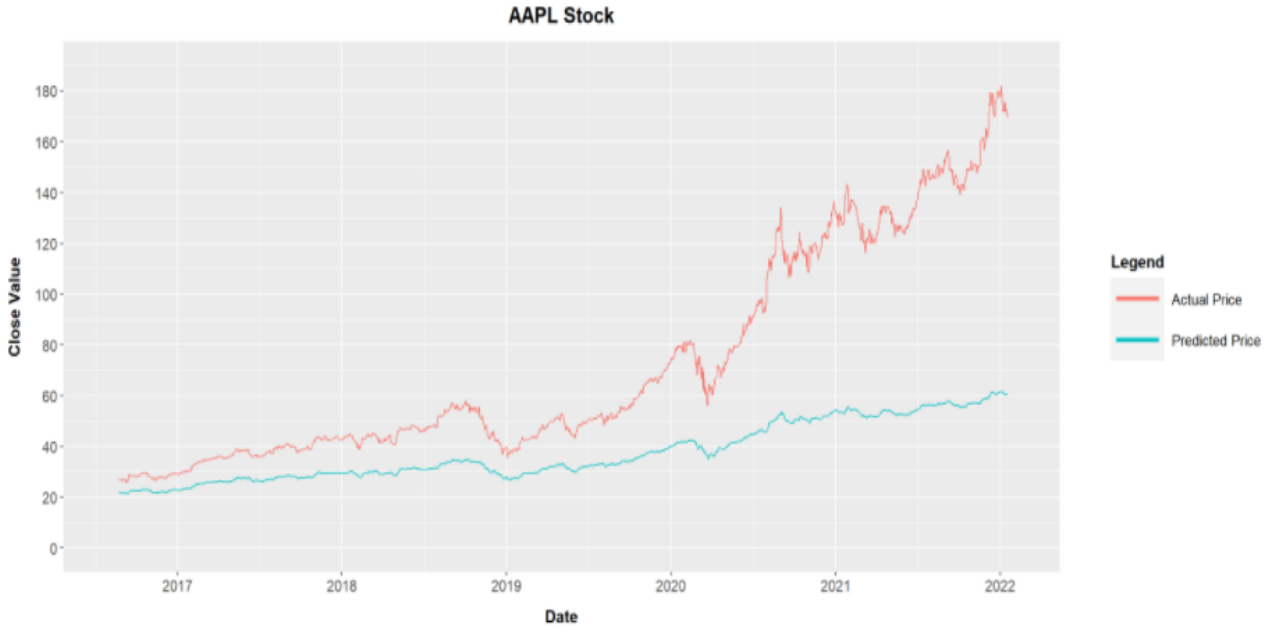
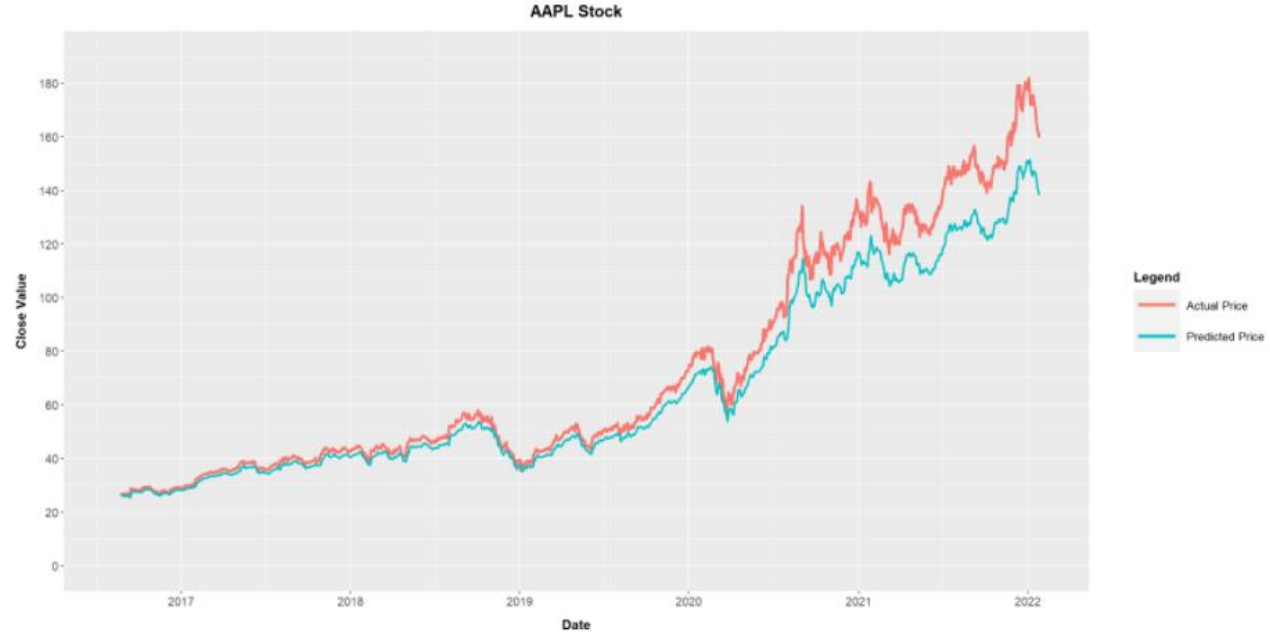


Figure 2: Tuned LSTM Predicted VS. Actual Plot



London Business School

Figure 3: Untuned RNN Predicted VS. Actual Plot

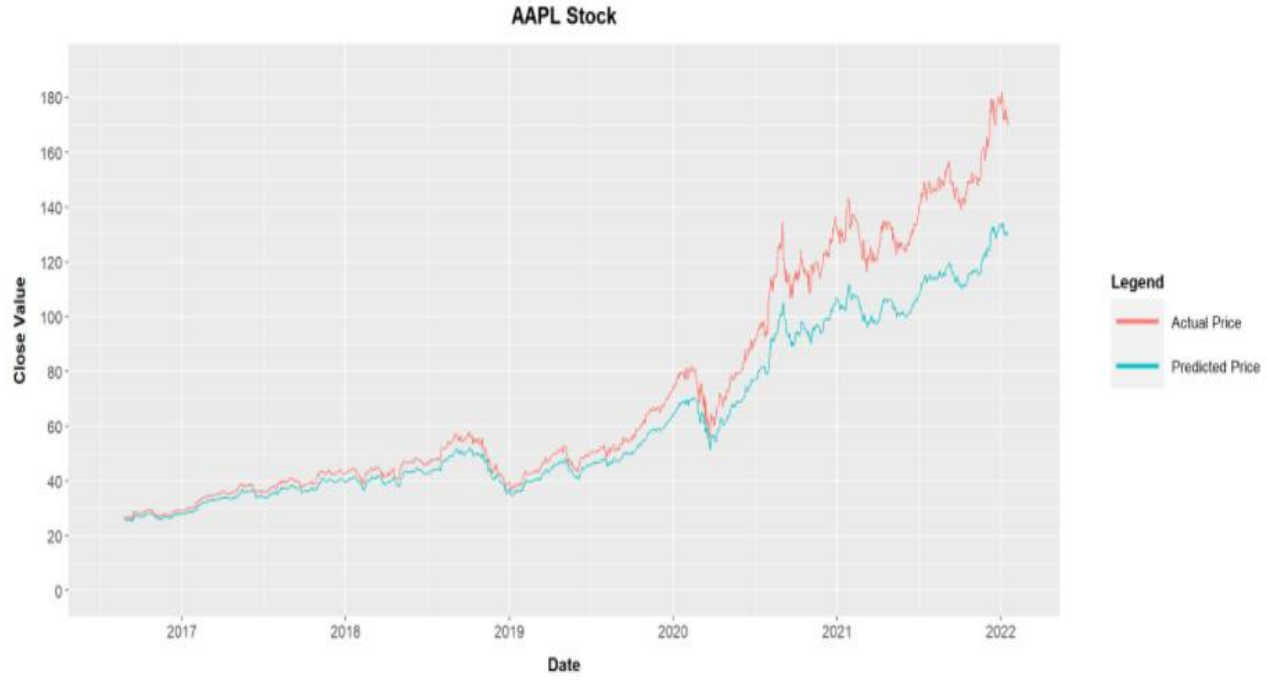
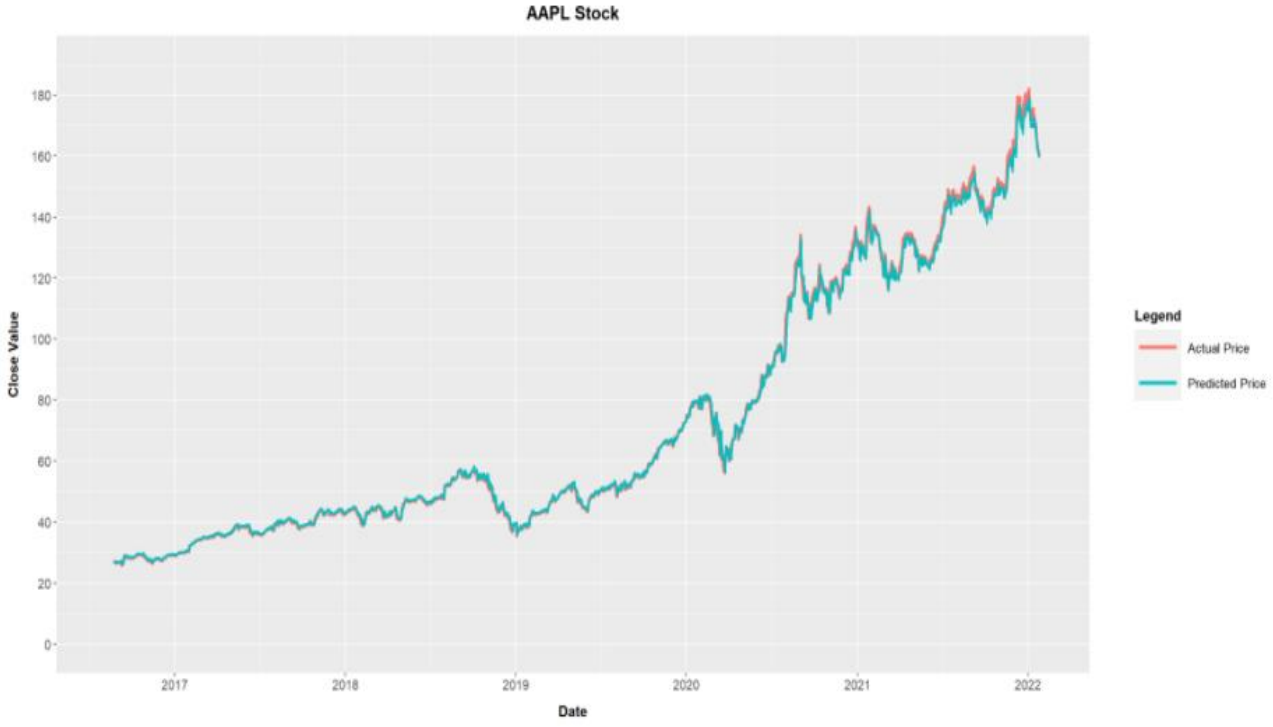


Figure 4: Tuned RNN Predicted VS. Actual Plot



Appendix 6

Figure 1: One Day & Two Day Predicted VS Actual Plot