Neural Networks & Deep Learning

Workshop Report

EXECUTIVE SUMMARY

This report analyses the potential of deep learning model applications, more specifically of neural networks, in two distinct business applications. First, the performance of *Artificial Neural Networks* is assessed through measures surrounding their predictive accuracy on customer attrition. Prior to building a model, the analysis realises *Predictive Mean Matching* as an effective means of imputation for randomly missing numerical values throughout the present dataset. This method shows particularly good performance of retaining the distribution of the original dataset. An optimised *ANN* model, after hyperparameter tuning, predicts customer attrition correctly in over 80% of cases. In the business case at hand, however, this report places greatest importance on *Recall* scores, rather than on overall accuracy. When predicting for customer attrition, it is arguably more damaging to predict customers not leaving when they are leaving in reality, compared to the other way around. In other words, it is most important to minimise the number of false negative predictions, as it makes more sense for businesses to invest a dollar too much into false positive predictions, rather than to lose customers because of not placing enough emphasis on false negatives. The optimal *ANN* chosen achieves an out-of-sample *Recall* score of almost 64%. However, these scores differ by around 23% when altering activation functions, suggesting that users of *ANN*s should place high importance on tuning for this hyperparameter. Ultimately, even though the ability to identify true attrition at 65% seems improvable, the *ANN* does outperform a simple logistic regression model significantly in this metric. As a next step, performance can potentially be enhanced by training on a more balanced dataset regarding the count of the positive and negative responses.

Second, this analysis assesses the potential of both, *Simple Recurrent Neural Networks* as well as *Long Short-Term Memory* models in closing stock price prediction of *Apple* stock and argues that the closing price as well as the number of past days on which to base model training are significant factors to a model's performance. Moreover, it is suggested that the inclusion of 1) additional explanatory variables on pricing information worsen model performance through overemphasising pricing levels and of 2) non-price information such as volume provide additional noise to training. Users of *LSTM* models should be wary of the potential limitations in time-series forecasting, specifically for the stock market. Most notably, large differences in pricing magnitudes across training, validation and testing should be accounted for. As such, an *LSTM*'s architecture, allowing for inference from long sequences of data, might cause the model to place too much importance on the comparatively low values from the past. It can be suggested that model training on asset returns or on the spread between two asset prices as opposed to training on absolute prices mitigates the negative effects of these large discrepancies. In addition, model-based stock price prediction is prone not being able to account well for market-inefficiencies, macroeconomic trends and large anomalies. Training a *Convolutional Neural Network*, with less emphasis on the sequential nature of time-series data, but rather on additional features at one point in time, can thus be suggested as an avenue for future research.

Customer Churn

a) Removing 'customerID' during data pre-processing

The 'CustomerID' column was removed for two primary reasons: First, there is no valuable link between a customer's ID and the outcome attrition variable. A model run including this variable would run the risk of overfitting on this non-useful information. Second, including this variable would provide noise to the training of the *ANN* and thereby manipulate accuracy metrics derived from predictions.

b) Missing values

i) Results of dropping NAs on dataset and the Neural Network training

NA values were dropped in order to ensure that *ANN*s could be trained on the dataset, including the provided explanatory variables that originally contained missing values. Given that only 11 rows were removed following the NA drop (0.156% of the total dataset), this did not affect the distribution of any numerical variables to a considerable degree. Hence, the effects on the subsequent training of the neural network are negligible. However, it is noteworthy that if missing values were present in a systematic manner, that is customers with very distinct characteristics all show NA values in a given variable, dropping all those rows might significantly alter model performance. In these cases, imputation, rather than dropping should be prioritised.

ii) Techniques of handling missing values in machine learning

Replacement with Median or Mean: Simply replacing missing values in continuous variables with the mean or median across the values that are present

Replacement through predictive models: A more sophisticated way of deriving replacement values for NAs by treating the variable with missing values as a dependent variable for which one forecasts, for instance through various regression techniques

Models that work during the presence of NA values: Relying on KNN or Random Forests if alternatives are infeasible

iii) Discuss implementation of two of the above-mentioned techniques

If one compares the results across different techniques of dealing with missing values, it is firstly important to define metrics with which one conducts the comparison. In the case at hand, since the columns for which NA values are being inserted are all numeric, comparison is done based on histograms. This allows for obtaining an idea about the descriptive statistics such as variable distribution, mean, median and skewness.

Comparing the histograms of the original dataset without inserted NA values and the dataset with dropped rows after randomly assigned 35% NA values, it is firstly advantageous to identify that the distributions have stayed stable (compare Appendix 1 and 2). This is important, because it indicates that the NA values have been assigned in a truly random, as opposed to a systematic, fashion. While the general shape of the histograms is very much identical, as one would expect, the count of values is proportionally less after one drops NA values.

Secondly, comparing the two different techniques of replacing NA values, that is mean imputation and predictive mean matching, one can argue that here one faces a trade-off between 1) dropping NA values, loosing large parts of possibly valuable data, and 2) replacing missing values with possibly non-representative data. Here, it again makes sense to compare histograms of either two techniques with histograms derived from the original dataset. One can observe that imputation based on predictive mean matching does a great job at replicating the overall distribution of the original dataset (Appendix 4). At the

same time, as one would expect, simply inserting means has obvious shortcomings compared to predictive mean matching. Given that the former style of imputation uses the mean of the respective variable as a replacement for NA values, the count of values that are equal to the mean gets artificially boosted by around 12% of the entire number of observations (Appendix 3). For all three variables, this poses significant deviations from the statistical metrics of the original dataset. One can conclude that, for the case at hand, predictive mean matching is by far the more adequate option compared to dropping missing values and simply inserting means.

c) Why did we perform log-transformation on the 'TotalCharges' variable?

Log transformation on the TotalCharges variable is performed because the distribution of the variable values is heavily skewed to the right. As such, a log transformation reduces the impact of and puts less importance on heavily outlying values. In essence, logging the values allows for a better linear representation of variable values. At the same time, one enhances the predictability of values in the TotalCharges variable.

d) Explain why we centred and scaled the data.

Data centring and scaling on the entire dataset is performed in order to account for differences in the nature of the variable values and make them more comparable. During centring, the mean of a variable is subtracted from each respective observation and during scaling, each observation is divided by the standard deviation of the variable values. This ensures that data each variable has a mean of 0 (or as close to 0 as possible) and a standard deviation of 1. Regardless of the units or the magnitude of the original variable, we can now draw more representative inferences about the differences in statistical measures across variables. The practice of centring and scaling the data thus has the potential to speed up the learning process of the later created ANN to a certain extent.

e) Activation and loss function and changes in the case of a multi-class classification

Activation functions in neural networks are used to calculate a weighted sum of the input signals originating in previous neurons in order to determine a signals output signal. In the case at hand, the *sigmoid function* is used to determine a classification result by transforming this weighted sum of input signals into number close to 0 when the input is small vs. close to 1 when the input is large. Following the output given by the activation function at a networks last layer, loss functions start the process of *backpropagation*, in order to update the weights in the neural network. *Binary Cross-Entropy* is useful in classification problems, as it increases the loss incurred the further away the prediction probability is from either 0 or 1 (dependent on ground truth response). The resulting loss is then responsible for the degree of the changed weights.

When classifying for more than two response variables, the loss function will no longer binary. Instead, one can engage in cross-entropy for each of the possible response values. For instance, if one runs multi-class classification for determining a species of flowers, the neural network would run cross-entropy loss functions on each of the possible response values, rather than just on a single binary outcome. This means that the chosen *sigmoid activation function* will output a class probability for every single response, rather than for only two.

f)

i) Architecture and Parameters of best performing Model

The parameters on activation function as well as the number of neurons/nodes in the first two hidden layers in order to identify a model configuration that minimises the *bias-variance trade-off*, in other words that neither under- nor overfits to a certain extent. The optimal model is composed of two hidden layers, with 32 nodes and *softmax* activation function, and an output layer that uses sigmoid as an activation. Learning

is performed through *stochastic gradient descent*, more specifically through the *ADAM* optimiser. The full breakdown of the architecture can be seen in Appendix 5.

ii) Problems of underfitting or overfitting

Given their flexibility, neural networks frequently run the risk of overfitting on trained data. If a neural network is continuously trained while loss metrics stay consistent for multiple simultaneous epochs, overfitting can be assumed. Assessing this risk in Appendix 6, even though loss and accuracy stay constant for a couple of epochs, one can argue that the risk of overfitting is mitigated through some preventive measures employed. First, the hidden layers are trained with 10% of nodes randomly missing, which mitigates a heavy reliance on noise in the data. Second, the activated *callback* parameter ensures that when the model is not learning for five epochs, training will be stopped. Lastly, model evaluation is not based on the training set, but rather on a validation and test set, reducing the risk of overfitting on the training set.

iii) Performance evaluation

A summary of model performance dependent on the tuned number of neurons in the first two hidden layers as well as the activation function can be found in Appendix 7. The performance of the model, and thus the choice for optimal hyperparameter values, is based on the *Recall* metric. Keeping in mind that customer attrition is predicted, for a business it would be worse to predict that a customer is not leaving when he/she is in reality (false negative) as opposed to predicting that a customer is leaving and it is not in reality (false positive). The metric that allows one to evaluate models based on their susceptibility to false negative predictions is *Recall*. *Recall* is highest at 63.57% when training on *softmax activation* [1] and 32 neurons. Interestingly, this combination of hyperparameters also results in the highest predicted out-of-sample *accuracy* and *F1-Score*.

In addition, analysis is conducted on the differences in mean out-of-sample *Recall* metrics across the activation function (Appendix 8) vs. across number of nodes (Appendix 9). Importantly, different activation functions, rather than the number of neurons seem to be the determining factor for discrepancies in model performance. It can thus be suggested that when predicting attrition, hyper tuning should definitely be performed on the chosen different types of activation functions.

g) Classification accuracy ANN vs. Logistic Regression

Comparing the hypertuned *ANN* against a conventional *Logistic Regression*, performance advantages in accuracy are not readily evident. Specifically, one cannot observe a significant increase in out-of-sample prediction accuracy of or *AUC*. However, of particular interest again is the *Recall* metric which is about 15% lower when predicting through *Logistic Regression*. At the same time, it is important to realise that the logistic regression has a significant advantage in the interpretability of model parameters and results, which can be a crucial advantage in business settings. Especially in the case at hand, being able to identify distinct feature importance is crucial to prevent employee attrition. Considering neural networks tendency to being black box algorithms, this becomes more difficult

[1] Similar to *sigmoid*, but with output ranging from -1 to 1 rather than from 0 to 1

Stock Price Prediction

In the case at hand, the predictability of Apple stock prices through the use of Recurrent Neural Networks based on historical data starting in 1980 is assessed. As a base model, a *Long short-term memory* model is set up based on the closing prices of the previous three days. The resulting prediction of upcoming day closing prices consistently underpredicts the dependent variable. Specifically towards the end of the time-series, a significant gap between predicted and actual values emerges.

a) Including multiple predicting variables

Variable selection is performed by assessing the validation *RMSE* for 3 distinct pairs of variables (Appendix 10) to be included in addition to closing price. Optimal performance is achieved when including the following two variables: One, a newly defined variable *mid*, representing the average between each day's high and low price, and two, the trading *volume*. Compared to our initial base model, including data on these independent variables for the past three days alters the predictive accuracy of our model. In statistical terms, the use of additional features causes the out-of-sample RMSE to rise by more than 41%. Applying the significance of this improvement to the case about stock pricing, the reduced predictive accuracy of this magnitude is likely to worsen the quality of decision-making based on this time-series prediction. One can observe this worsening along the plotted time-series values as well (compare Appendix 15 to 16). Here, the model with the additional parameters underpredicts significantly and is thereby less able to account for the underlying daily volatility in the data. Even though this model has not been tuned for yet at all, one major conclusion can be suggested: Given that now information about non-price data (i.e. volatility) is used to make implications about prices, the model seems to model noise in the additionally included variables and is thus less able to draw meaningful inferences about historical patterns compared to a sole reliance on closing prices. In addition, one might suggest that the included *mid* variable, which also adds an additional variable on pricing information next to the closing price variable, causes the model to overfit on the training data and is thus less able to account for the large jumps in prices following the periods of training and validation.

b) Building Simple RNN

Compared to the un-tuned LSTM, the simple un-tuned RNN built seems to be better able to predict the Apple's stock price (Appendix 17). Interestingly, this model also shows by far the best validation *RMSE* when *volume* and *mid* are included as additional variables in addition to closing price (Appendix 11). Even though the resulting implication should be regarded with care before having hyperparameter tuned the model, this variable combination seems to be of some value, possibly attributable to the information that *mid* holds about the spread between daily high and low values.

c) Hyperparametertuning LSTM

Hyperparametertuning is performed on the optimiser as well as on the number of nodes in the hidden layer in order to identify a combination of parameter values that maximises the validation *RMSE*. Appendix 12 shows that these *RMSE*s differ hugely by optimiser chosen and are consistently (aside of one outlying observation) lowest for the *ADAM* optimiser. This suggests that there arises significant value for users of LSTM models when optimising for this parameter. At the same time, the number of nodes chosen seems to be of less importance to model performance. The hypertuned model is able to lower out-of-sample *RMSE* by over 57% and is significantly more accurate in the long-run (Appendix 18).

d) Hyperparametertuning RNN

The analysis does not show a similar picture for the tuned simple RNN, for which performance actually worsens compared to the untrained model. Given that hyperparameter tuning is performed on the validation set, one can suggest that performance is worse because the variable selection only works well in the limited time frame for which the prices seem to be rather different in magnitude to the testing set (Appendix 19).

This introduces one to an underlying problem of the stock-prediction case at hand. Appendix 14 shows how the magnitude of closing prices largely differs throughout the three different training sets. None of the trained models throughout this analysis is able to account for this large difference, suggesting that models should be constantly readjusted to account for systematic changes in price levels.

e) Can we predict for more than one day?

Appendix 20 shows that the hyperparameter tuned model, predicting two consecutive stock prices on the basis of the closing prices of the past 7 days, is the most accurate throughout the entire analysis. This holds for both, one day as well as two-day predictions[2]. As such, this strengthens the above-formed hypothesis on variable selection, stating that the additionally included variables on non-closing price information, cause the model to be trained to too much noise. In addition, it is arguable that the *LSTM*s ability to form meaningful inferences of long-term relationships, as well as to neglect the impact of information that is not useful anymore, is a primary driver of why including a longer explanatory horizon (7 days), is a primary driver for this increased accuracy. The potential at this end should be further explored by increasing the number of included past closing prices.

f) Limitations of Deep Learning on predicting stock prices

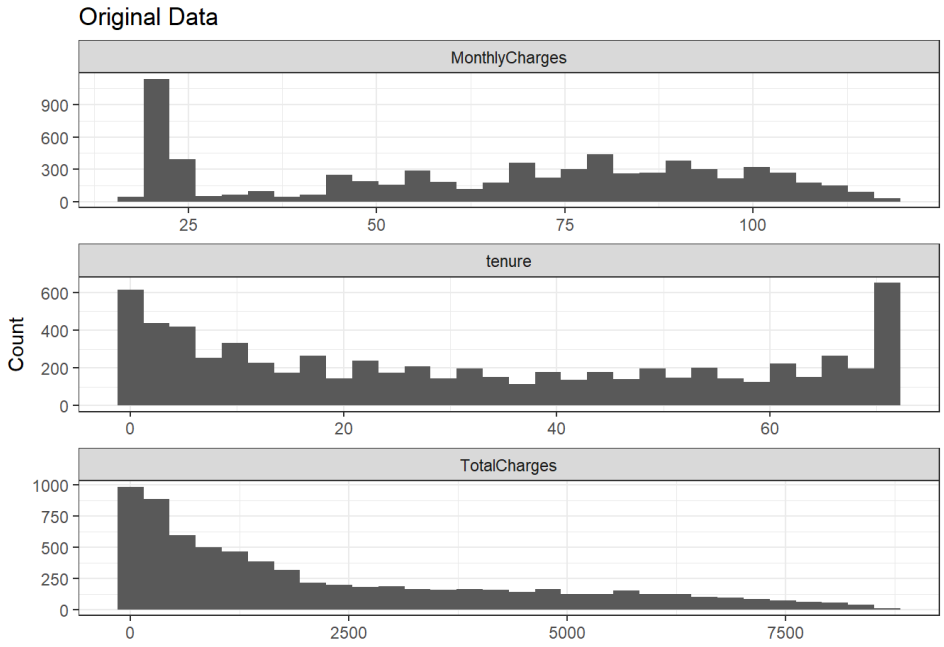
Due to *LSTM* models' abilities to perform well in sequence prediction problems, it appears natural to engage this model in stock price prediction. Through the model's three-gate-architecture, it is able to selectively identify information that is relevant and thereby flush memory from information that is not relevant to the underlying predictive accuracy. That being said, even though *LSTM*s present an upgrade compared to conventional simple *RNN*s, their predictive accuracy is influenced by a number of limiting factors. First, the assumption that is being taken in the model at hand is that future performance of a stock is a function of its past performance. As such, this neglects the *Efficient Market Hypothesis* arguing that all available information is priced in the stock price at each point in time. Even though markets are arguably not entirely efficient in real life, inefficiencies are likely to balance out the longer the investigated time horizon. Thus, the accumulation of deviances between predicted and actual prices enlarges over time and the predictive accuracy of deep learning models diminishes. A primary driver for this diminishing accuracy is the occurrence of anomalies in pricing levels, such as the 2020 drop in prices caused by the outbreak of the *Corona* virus. Ultimately, even though long-term forecasts can provide an idea of the broader direction, it becomes apparent that the magnitude in prediction errors rises.

Second, the performance of employed deep learning models on stock price prediction is heavily dependent on the nature of the underlying dataset, which has become apparent throughout the analysis of the case at hand. Since training of the model is done only on the beginning stage of the time-series data, trends that have emerged over time are not accounted for in the training of the dataset. The impact of this limitation can be observed in the prediction of the testing dataset, since the rather abnormal positive spike in stock prices after around 2014 is constantly underpredicted, even in tuned models. Here, a focus on returns rather than on absolute prices would have the potential to normalise trends over the long run. In essence, it would be optimal if underlying testing data was as similar to a *random walk* as possible, alleviating variances explained through by time. In addition, engaging *deep learning models* on a spread/difference between two asset prices, rather than their absolute values could help to drive out trends. Similarly, changes in volatility of the underlying asset likely influence the predictive accuracy at each time point. In the case at hand, since the model is trained on data that is characterised by a comparatively low volatility, high volatility in the testing data seems to be not accounted for to a sufficient extent.

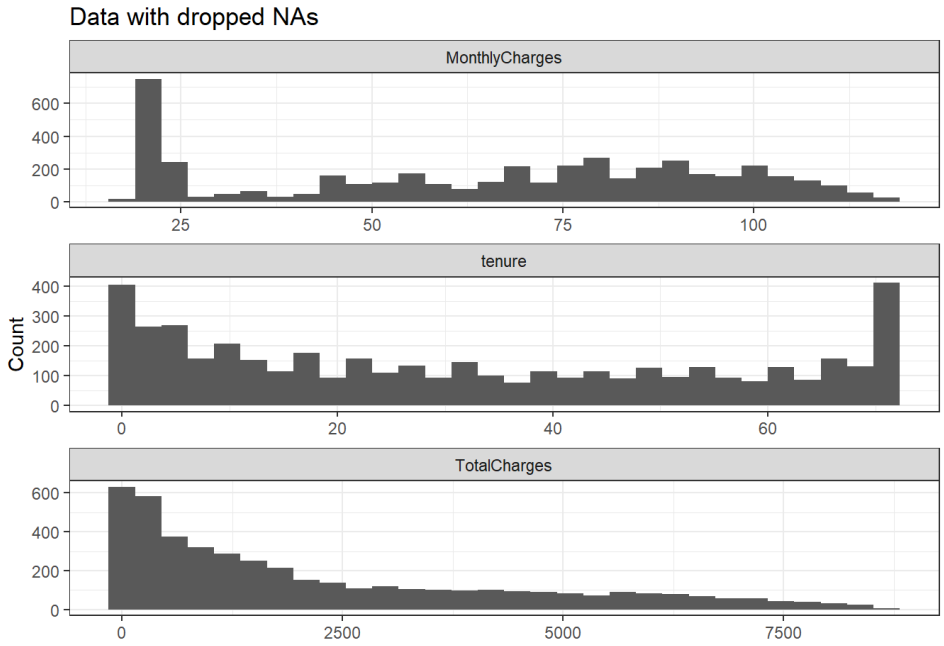
[2] Predictive accuracy for day 2 seems to only worsen marginally

Appendices

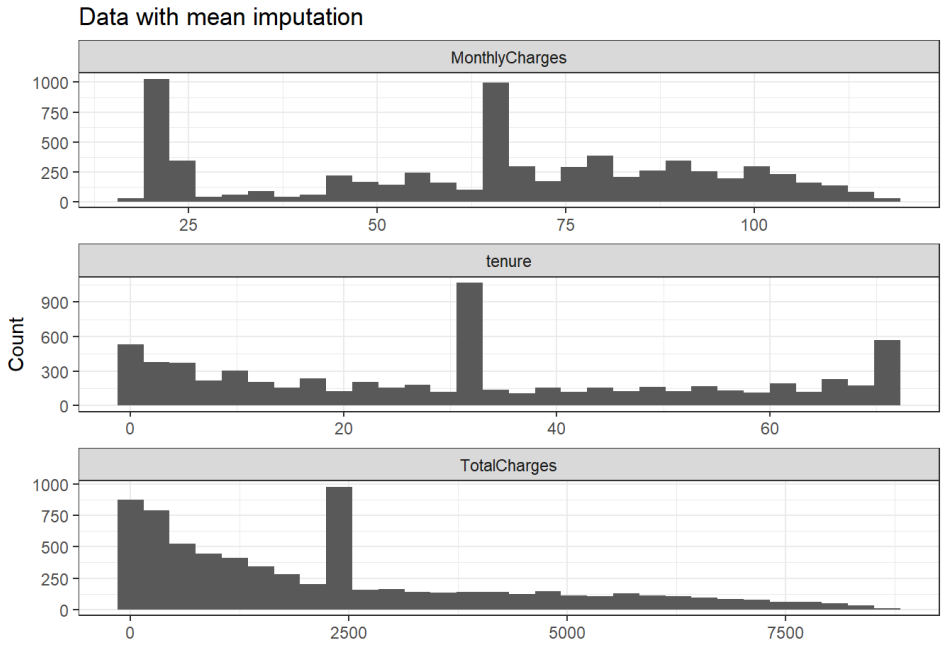
Appendix 1: Histograms of original Attrition Dataset



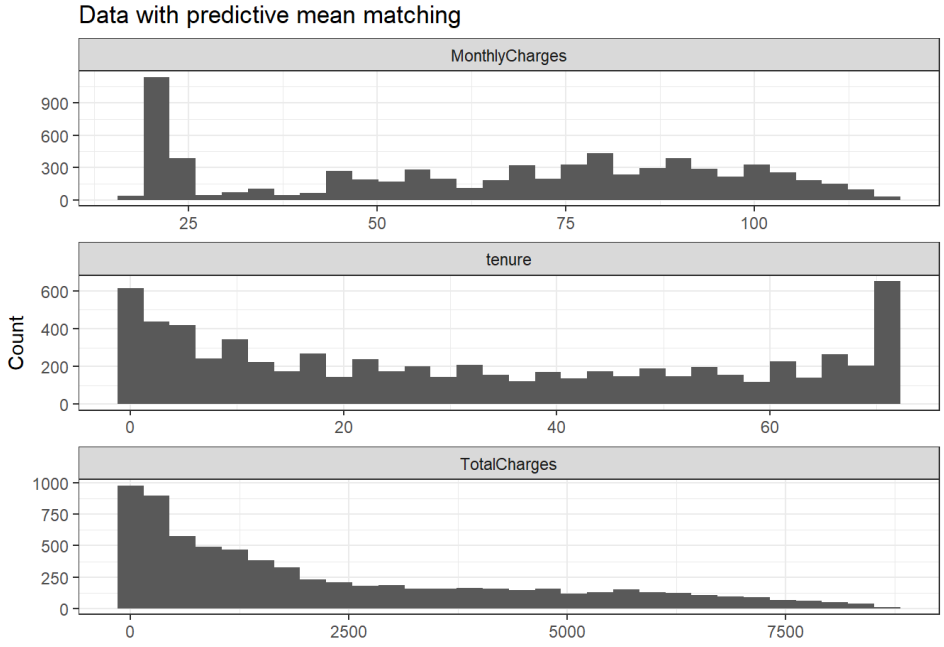
Appendix 2: Histograms after dropped NA Values



Appendix 3: Histograms after Replacement with Mean Values



Appendix 4: Histograms after Predictive Mean Matching

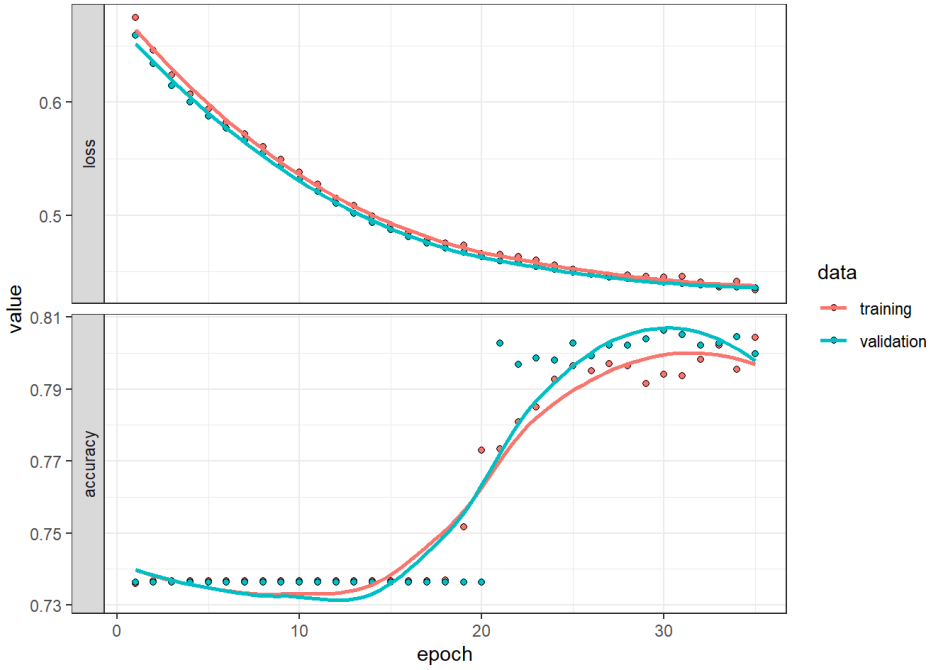


Appendix 5: Architecture of Hyperparameter tuned ANN

Layer (type)	Output Shape	Param #
dense_38 (Dense)	(None, 32)	1152
dropout_25 (Dropout)	(None, 32)	0
dense_37 (Dense)	(None, 32)	1056
dropout_24 (Dropout)	(None, 32)	0
dense_36 (Dense)	(None, 1)	33

Total params: 2,241
Trainable params: 2,241
Non-trainable params: 0

Appendix 6: Epoch Learning of Hyperparameter tuned ANN



Appendix 7: Hyperparameter tuning Performance Assessment

RANK	ACTIVATION	NODES	MAX. VAL. ACCURACY	MEAN VAL. ACCURACY	TOP QUANTILE VAL. ACCURACY	TEST ACCURACY	TEST AUC	TEST PRECISION	TEST RECALL	TEST F1
1	**softmax**	**32**	0.8033	0.7629	0.7980	0.8088	0.8527	0.6578	**0.6357**	0.6465
2	**softmax**	**64**	0.8045	0.7621	0.7959	0.8024	0.8525	0.6430	**0.6331**	0.6380
3	**sigmoid**	**16**	0.8081	0.7569	0.8009	0.8031	0.8526	0.6503	**0.6150**	0.6321
4	**sigmoid**	**64**	0.8086	0.7998	0.8057	0.8081	0.8524	0.6686	**0.5995**	0.6322
5	**relu**	**64**	0.8063	0.7965	0.8045	0.8060	0.8467	0.6638	**0.5969**	0.6286
6	**softmax**	**16**	0.8045	0.7646	0.8024	0.8045	0.8537	0.6600	**0.5969**	0.6269
7	**sigmoid**	**32**	0.8075	0.7945	0.8048	0.8067	0.8527	0.6716	**0.5814**	0.6233
8	**tanh**	**32**	0.8086	0.8036	0.8060	0.8088	0.8535	0.6821	**0.5711**	0.6217
9	**tanh**	**16**	0.8098	0.8030	0.8060	0.8081	0.8533	0.6869	**0.5556**	0.6143
10	**relu**	**16**	0.8027	0.7964	0.8021	0.8031	0.8497	0.6846	**0.5271**	0.5956
11	**tanh**	**64**	0.8081	0.8041	0.8054	0.8017	0.8517	0.6812	**0.5245**	0.5927
12	**relu**	**32**	0.8045	0.7963	0.8021	0.8010	0.8502	0.6826	**0.5168**	0.5882

Appendix 8: Mean Recall by Activation Function

RANK	ACTIVATION	RECALL
1	softmax	0.622
2	sigmoid	0.599
3	tanh	0.55
4	relu	0.547

Appendix 9: Mean Recall by Number of Nodes

RANK	NODES	RECALL
1	64	0.589
2	32	0.576
3	16	0.574

Appendix 10: LSTM Feature Selection in Addition to Closing Price based on Validation RMSE

RANK	FEATURE 1	FEATURE 2	VALIDATION RMSE
1	mid	volume	0.000185
2	volume	mid	0.000251
3	mid	open	0.002581
4	volume	open	0.003135
5	open	volume	0.004722
6	open	mid	0.005977

Appendix 11: Simple RNN Feature Selection in Addition to Closing Price based on Validation RMSE

RANK	FEATURE 1	FEATURE 2	VALIDATION RMSE
1	volume	mid	0.000424802
2	mid	volume	0.000469017
3	volume	open	0.005976722
4	mid	open	0.011033021
5	open	volume	0.011760925
6	open	mid	0.018648507

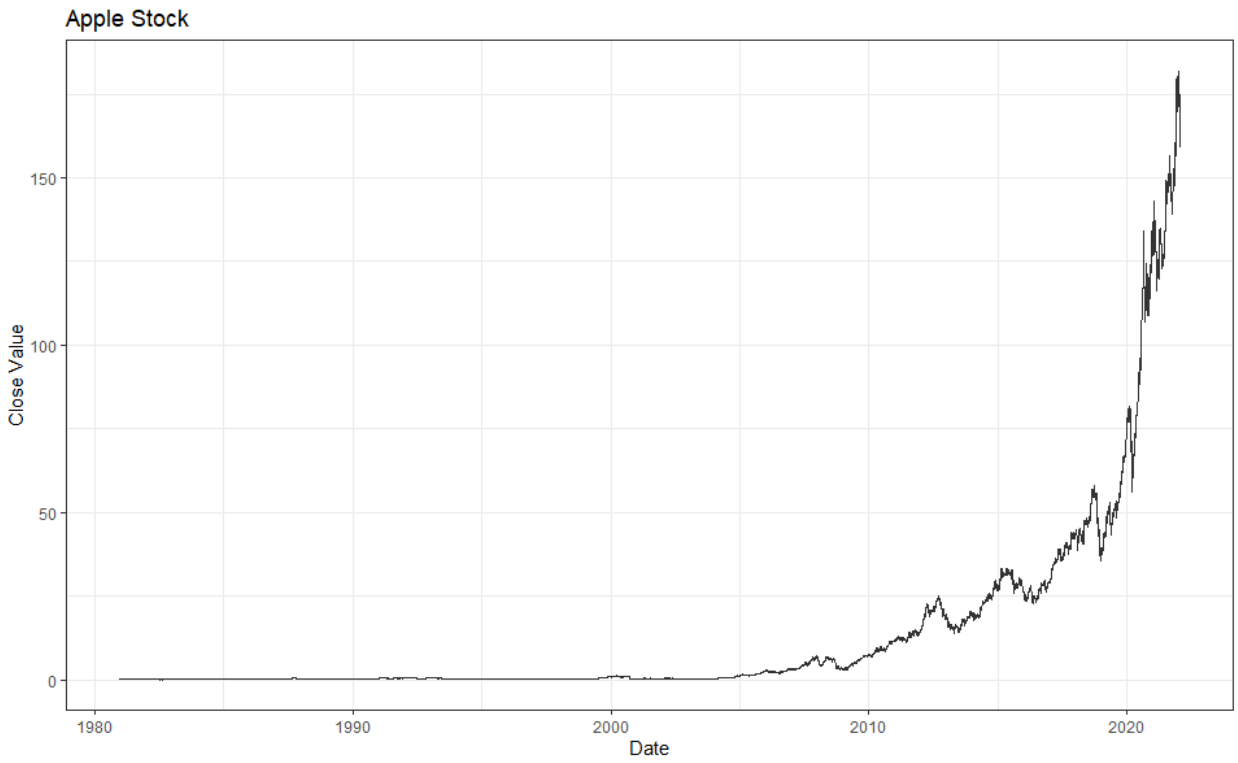
Appendix 12: LMSE Validation RMSE by tuned Hyperparameters

RANK	NODES	OPTIMIZER	VALIDATION RMSE
1	64	adam	0.000179
2	128	adam	0.000179
3	32	adam	0.000209
4	128	RMSprop	0.000236
5	16	RMSprop	0.000236
6	64	RMSprop	0.000259
7	32	RMSprop	0.000339
8	16	adam	0.00036
9	128	sgd	0.001087
10	64	sgd	0.005839
11	32	sgd	0.016376
12	16	sgd	0.04486

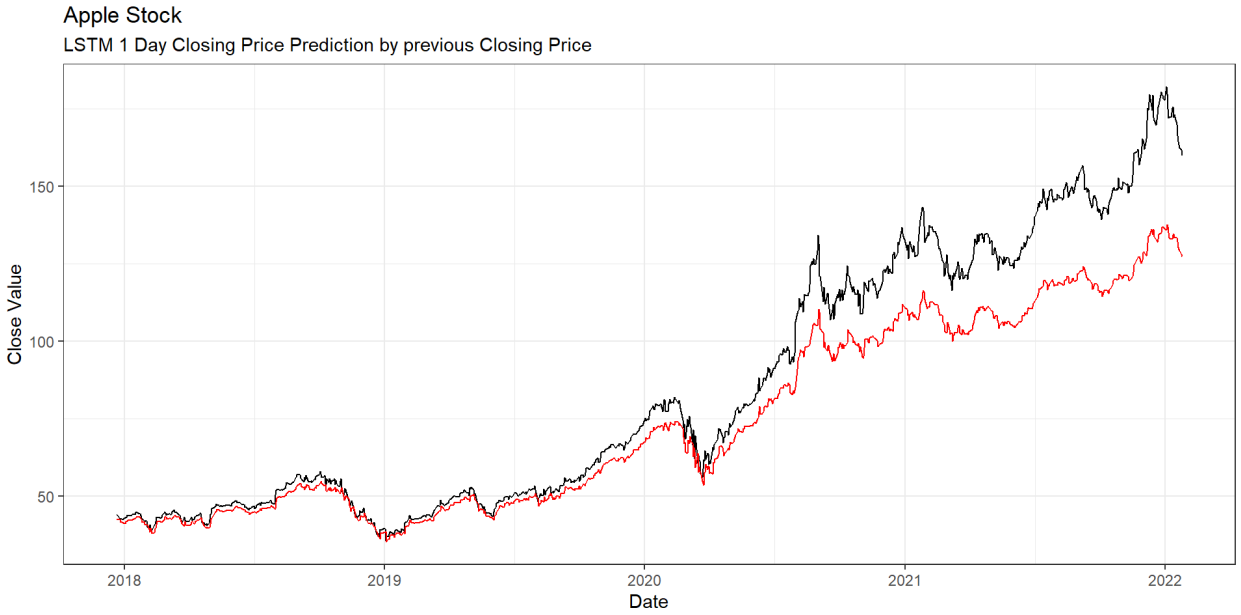
Appendix 13: Simple RNN Validation RMSE by tuned Hyperparameters

RANK	NODES	OPTIMIZER	VALIDATION RMSE
1	128	adam	0.000308267
2	64	adam	0.000351195
3	16	adam	0.000707225
4	32	RMSprop	0.000715626
5	32	adam	0.000853488
6	64	RMSprop	0.000978781
7	128	RMSprop	0.001052137
8	64	sgd	0.001557083
9	16	RMSprop	0.00166617
10	128	sgd	0.00374183
11	16	sgd	0.005320844
12	32	sgd	0.005638265

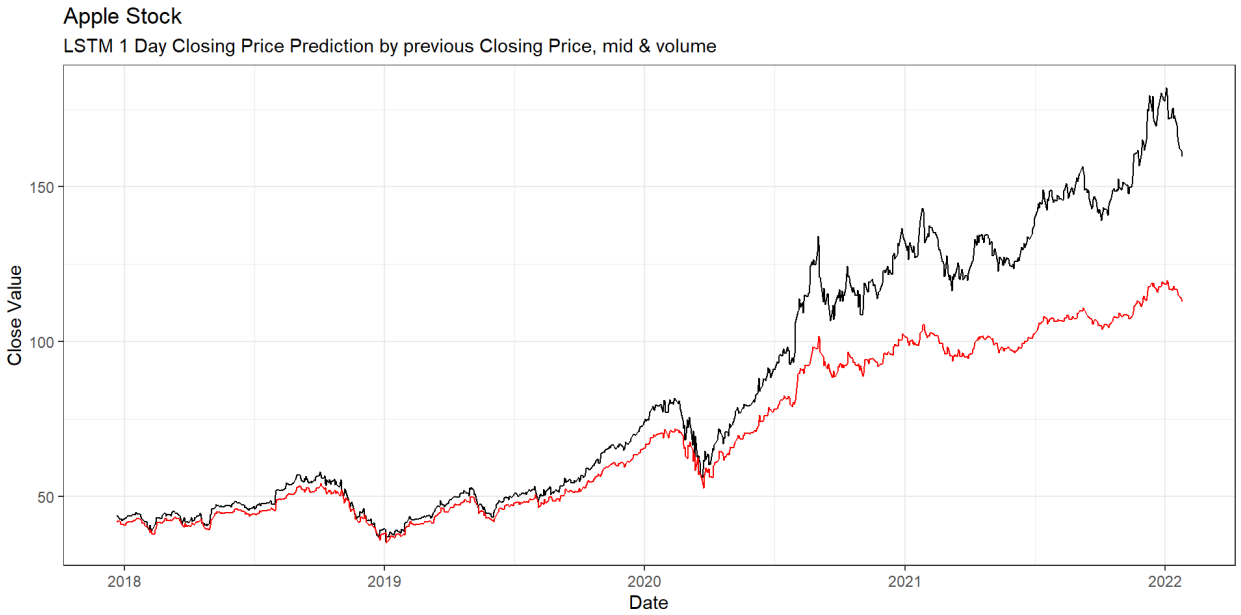
Appendix 14: Evolution of Apple Stock Closing Prices



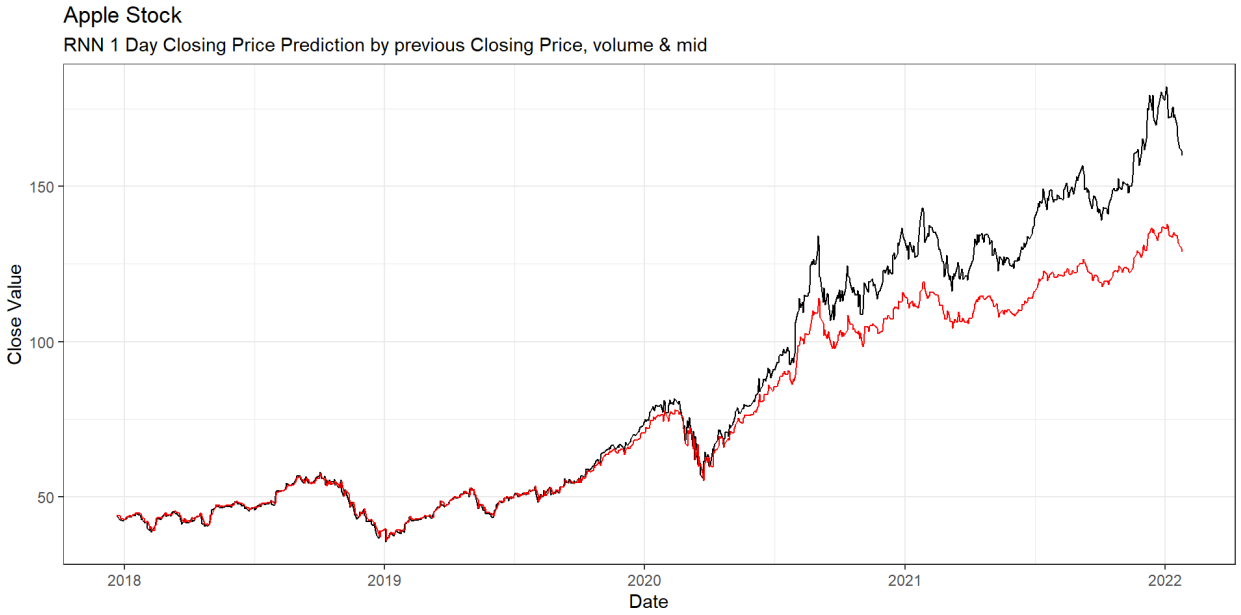
Appendix 15: Un-Tuned LSTM Closing Price Prediction before Variable Selection



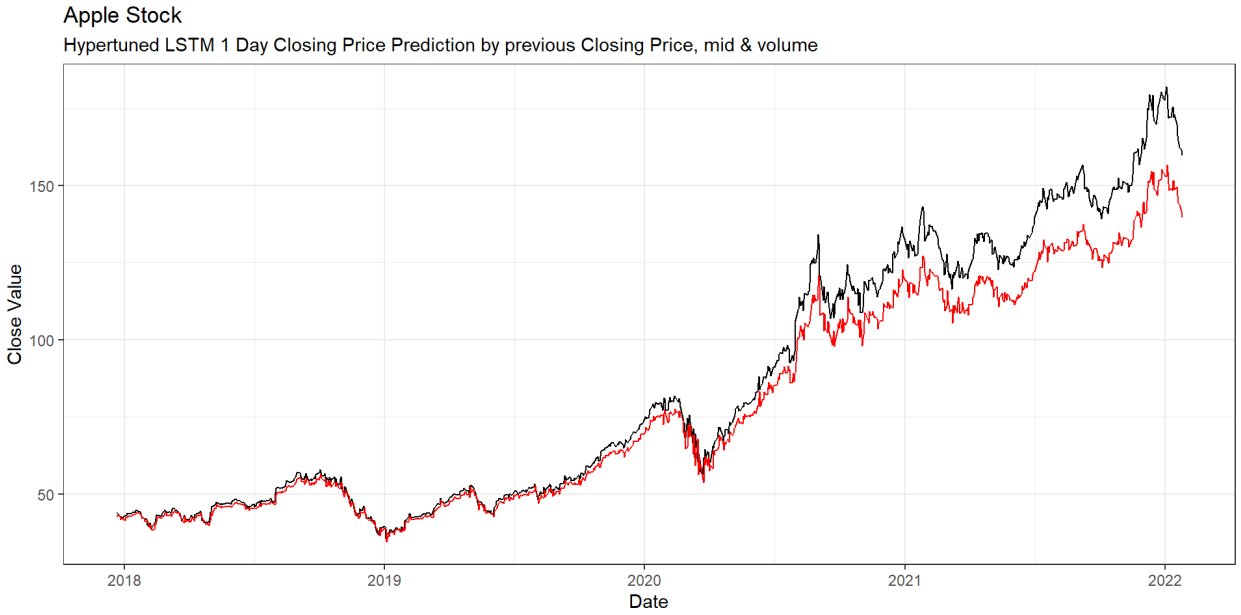
Appendix 16: Un-Tuned LSTM Closing Price Prediction after Variable Selection

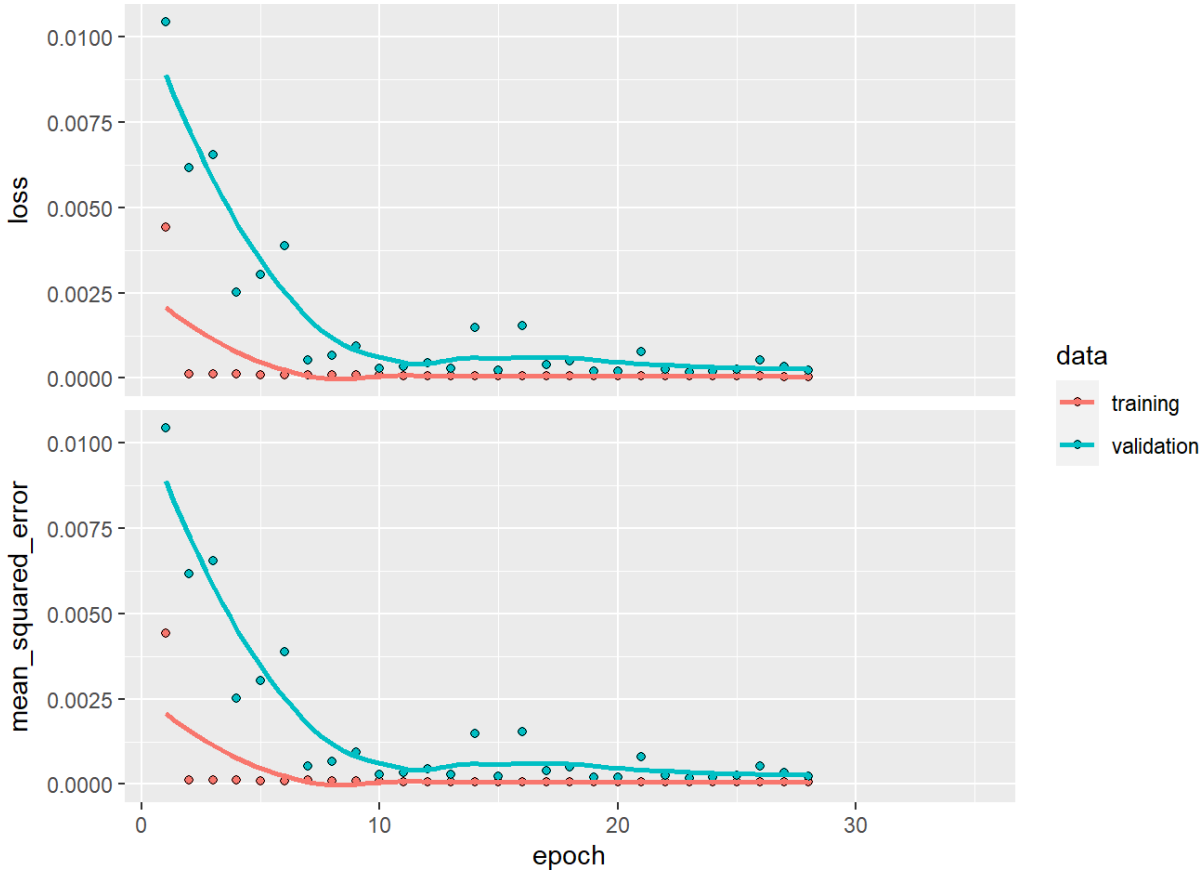


Appendix 17: Un-Tuned RNN Closing Price Prediction after Variable Selection



Appendix 18: Tuned LSTM Closing Price Prediction after Variable Selection





Appendix 19: Tuned RNN Closing Price Prediction after Variable Selection

Appendix 20: Tuned Two-Day LSTM Closing Price Prediction after Variable Selection