A Bayesian regularized artificial neural network

Zimo Zhu& Jason Teng

2023-12-09

Introduction

Neural networks mimics the human brain's process of adapting information and recognize relationships among data. Much like the human brain, neural nets utilize these relationships for prediction and forecasting. Neural networks are powerful. They are nonparametric and non-linear models that are capable of solving a wide variety of numerical problems.This makes them one of the most flexible models in statistical learning. Neural network models do not have restricted parameters, and some parameters may not need to be known for forecasting. In the realm of financial investment, accurately forecasting financial product prices or predicting market trends is crucial for decision-making. Consequently, there is a growing interest in applying neural networks in the stock market. However, feed-forward neural networks face the challenge of overfitting. To address this, several procedures have been applied, such as incorporating a dropout layer, where several neurons are randomly dropped during each fitting of the data.

In this study, we attempt to emulate and understand the study of Bayesian Regularized Neural Networks inspired by the paper 'A Bayesian Regularized Artificial Neural Network for Stock Market Forecasting.' The outline of this report will be as follows: In Section Background, we introduce the feed-foward neural network and also introduce some common procedures to reduce overfitting in neural network models. In Section Paper Review, we summarize the paper and provide some elementary proofs. In Section Replication of the Paper Results, we replicate findings from the paper, and in Section Application in Comparison to Forward Neural Networks, we apply this algorithm to another financial problem. Finally, in Section Summary, we summarize our results and discuss future work.

Background

In this section, we review the theory and algorithm part of the paper. Firstly, we introduce the feed-foward neural network. Firstly, we reduce the neural network model to the model with only one layer and one neuron, which is also called perceptron. Mathematically, the perceptron can be shown as:

$$y = \mathbf{f}(w_0 + \sum w_i x_i) + \epsilon,$$

where f is the activation function. And the activation function can be linear or non-linear.

Then, we expand the perceptron to the neural network. And we only consider the neural network with one hidden layer here. Then, mathematically, the neural network can be expressed as:

$$y = \sum \mathbf{w}_j f_j(\sum_{i=1}^{p} w_{i,j} x_i) + \epsilon,$$

where f_j is the activation function of $neuron_j$, $w_{i,j}$ are the weights from input x_i to $neuron_j$, and w_j are the weights from $neuron_j$ to the output.

In neural network, we treat the w as the constant. The optimal weights are estimated by minimizing the loss of the given sample; the usual loss function is mean square error or mean absolute error, where $error = y - \widehat{y}$. In the real data, we obtain the optimal weights by gradient descent.

Given that neural networks primarily aim to minimize the loss on training data, the issue of overfitting is a common challenge. To mitigate this, techniques such as dropout layers and repeated fitting are often employed. In this section, I will briefly introduce these methods, provide my understanding of their similarities with tree-based methods, and highlight the differences.

In the approach of repeated fitting, a neural network is trained multiple times using the same dataset, and the average of the predictions from each training cycle is calculated. This method is somewhat analogous to Bagging, which aggregates predictions from multiple models, typically trees, trained on varied subsets of the data. Bagging reduces the overall flexibility of the model by averaging the results of multiple trees, as opposed to relying on a single tree.

However, there is a notable difference in the data sampling technique between Bagging and repeated fitting in neural networks. Bagging uses bootstrapped samples, whereas repeated fitting for neural networks consistently employs the same dataset for each training phase. The necessity for bootstrapping is circumvented in the case of neural networks because each training phase begins with a different set of initial weights. This inherent variation in the starting weights for each training phase introduces a level of diversity to the learning process, similar to the effect achieved by bootstrapping in Bagging, although achieved through a different mechanism.

In a dropout layer, during each iteration of the neural network's training process, a random subset of neurons, typically around 80% of the total, is selected for activation. This method bears a resemblance to the technique used in random forests, where each tree is trained on a randomly selected subset of predictors, especially when these predictors are correlated. In a neural network, neurons within the same layer receive information from the same predictors, which can lead to correlations in their outputs.

A significant distinction, however, lies in the sampling approach. In random forests, each tree is trained on a different subset of the data, often created through bootstrapping. In contrast, dropout layers utilize the same dataset for training across all phases, without employing bootstrapping. Additionally, the dropout layer is applied consistently, and the initial weights of the network remain unchanged throughout this process.

This consistency in the use of the same dataset and initial weights might be viewed as a drawback of the dropout approach. Furthermore, there is a difference in the proportion of neurons activated in each dropout phase compared to the number of predictors used for each tree in a random forest, highlighting another distinct aspect between these two methods.

Additionally, in regression analysis, regularization techniques such as Lasso and Ridge are employed to reduce model flexibility compared to the original linear regression, thereby mitigating the risk of overfitting. In this study, we explore the concept of regularization within the weights of a neural network through a Bayesian approach to address overfitting issues.

A Bayesian Regularized Artificial Neural Network (BRNN) extends the concept of regularization beyond merely adding constraints on the weight parameters w. It treats them as random variables, subject to prior distributions. This Bayesian treatment reflects our beliefs about the values of these parameters before observing any data. Incorporating uncertainty in this manner allows for improved model generalization and robustness against overfitting.

Paper Review

Since it is under the context of bayesian, then it is not usual as the original neural network with regularization only. And in this section, I provide the details about fitting the model.

In BRNN, firstly, the objective function has been changed:

$$
\begin{aligned}
L &= \beta E_D + \alpha E_W, \\
E_W &= \sum w^2, \\
E_D &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,
\end{aligned}
$$

where α and β are parameters.

Because of the Gaussian assumption in stock price, then we also assume that w follow normal distribution with the pdf:

$$\pi(w|\alpha) \propto exp(-\alpha E_W)$$
$$\pi(w|\alpha) = \frac{1}{C(\alpha)} exp(-\alpha E_W),$$

where $C(\alpha)$ is the normalized function of α.

Similarly, the conditional distribution of ϵ, which is also the conditional distribution of y can be written as:

$$p(\epsilon|\beta) \propto exp(-\beta E_W)$$
$$p(\epsilon|\beta) = \frac{1}{C'(\beta)} exp(-\beta E_D),$$

Then, based on bayes rule, we have:

$$\begin{aligned}\pi(w|\alpha, \beta, y) &\propto \pi(w|\alpha) p(\epsilon|\beta) \\ &= exp(-\alpha E_W) exp(-\beta E_D) \\ &= exp(-L)\end{aligned}$$

Based on poesterior distribution of w, we obtain the most probable point w_{MP} and Hessian matrix H at w_{MP} of poesterior ditribution by Levenburg–Marquardt training algorithm, which will be used in obtain the evidence. And $H = \beta D + \alpha I$, where D is the hessian of E_D at w_{MP}.

And the optimal value of α and β are obtained by maximizing $P(\alpha, \beta|y)$. And, through bayes rule, we have:

$$\begin{aligned}P(\alpha, \beta|y) &= \frac{P(y|\alpha, \beta) P(\alpha, \beta)}{P(y)} \\ &\propto P(y|\alpha, \beta) P(\alpha, \beta)\end{aligned}$$

Here we can ignore the prior of α and β. Then, the optimal value of α and β are obtained by maximizing $P(y|\alpha, \beta)$. Also,

$$P(y|\alpha, \beta) = \int \pi(w|\alpha) p(y|\beta, w) dw$$

It is hard to get the closed form of the P(y| ,). Then, we approximate the $log(P(y|\alpha, \beta))$ by following:

$$\begin{aligned}log(P(y|\alpha, \beta)) &\approx log(\pi(w_{MP}|\alpha)) + log(p(y|w_{MP}, \beta)) - \frac{1}{2} log(|H|) \\ &= -\alpha E_w^{MP} + \frac{N}{2} log(\alpha) - -\beta E_D^{MP} + \frac{M}{2} log(\beta) - \frac{k}{2} log(\pi) - \frac{1}{2} log(|H|),\end{aligned}$$

where N is number of weights, and M is the number of observations.

Then, we take derivative of $log(P(y|\alpha, \beta))$ with respect to α. And let λ_i be the eigenvalue of D, then the eigenvalue of H should be $\lambda_i + \alpha$ Then,

$$\begin{aligned}\frac{dlog(|H|)}{d\alpha} &= tr(H^{-1}) * \prod (\frac{d(\lambda_i + \alpha)}{d\alpha}) \\ &= tr(H^{-1}) \\ &= \sum \frac{1}{\lambda_i + \alpha}\end{aligned}$$

Then, we get:

$$\begin{aligned}-E_w^{MP} + \frac{N}{2\alpha} - \frac{1}{2} \sum \frac{1}{\lambda_i + \alpha} &= 0 \\ 2\alpha E_w^{MP} &= N - \sum \frac{\alpha}{\lambda_i + \alpha}\end{aligned}$$

Then, we take the derivative with respect to β. Then, we can get:

$$2\beta E_D^{MP} = M - \sum \frac{\alpha}{\lambda_i + \alpha}$$

Here, we let $\eta = \frac{\alpha}{\lambda_i + \alpha}$. Then, we can express α and β as:

$$\alpha = \frac{N - \eta}{2E_w^{MP}}$$
$$\beta = \frac{M - \eta}{2E_D^{MP}}$$

Then, we use this equation to get the optimal value of α and β.

Replication of the Paper Results

In the section below, we try to emulate the results found from the paper. We try to recreate the graphs as well as trying to achieve the same criterion that the authors used to validate their models. Overall, our replication process will employ the BRNN library in R to create a Bayesian Recurrent Neural Network. This package utilizes the same procedure as the research in attempting to follow process set by Forsee and Hagan (1977).

Setup

Below is code to set up the environment and packages that we need to replicate the code. As seen from below, the seed is set to 123 and we utilize both libraries that allow for modeling and functionality in R.

```
set.seed(123)
#importing necessary libraries
library(brnn)
library(TTR)
library(neuralnet)
library(kableExtra)
library(dplyr)

#set working directory
setwd("C:/Users/Diecy/Downloads")

#importing data sets
#--------------------------
#Microsoft
msft <- read.csv("MSFT.csv")
#Goldman Sachs
gs <- read.csv("GS.csv")
#Apple
aapl <- read.csv("AAPL.csv")
#IBM
ibm <-  read.csv("IBM.csv")
```

Next steps, we manipulate the data set to contain the predictors and response variable that is needed for the model according to the paper above. More specifically, there are nine predictors which is needed, out of which three is provided (Open, High, Low) and six are calculated from the data as financial indicators (Please refer to the paper for more information.)

To obtain these additional predictors,this report utilizes the library TTR. This package is created to find financial indicators needed for historical stock prices. For reference, please visit https://cran.r-project.org/

web/packages/TTR/TTR.pdf for documentation on the package and its usage to find Stochastic Oscillators for financial data.

Please also note value of n in the following equations, they are obtained through common consensus as typical n values for financial periods representing days. In this case, n = 14 which equates to 2 weeks is typically used to observe volatility among stock market changes for the following equations.

```
#adding predictors and six additional financial indicators
#---------------------------
#Microsoft
msft$NextDayClose <- msft$Close[2:755]
msft$FiveDayMovAvg <- EMA(msft$Close,n=5)
msft$TenDayMovAvg <- EMA(msft$Close,n=10)
msft$RSI <- RSI(msft$Close,n=14)
msft$WilliamR <- (max(msft$High) - msft$Close)/(max(msft$High)-min(msft$Low))*100
msftStoch <- stoch(msft[,c("High","Low","Close")],nFastK =14, nFastD=2)
msft$K <- msftStoch[,1]*100
msft$D <- msftStoch[,2]*100
msft$Variance <- log(msft$Close[2:755]) - log(msft$Close[1:754])
msft <- subset(msft[15:753,],
               select=c("Open","High","Low","NextDayClose",
                        "FiveDayMovAvg","TenDayMovAvg",
                        "RSI","WilliamR","K","D"))

#Goldman Sachs
gs$NextDayClose <- gs$Close[2:755]
gs$FiveDayMovAvg <- EMA(gs$Close,n=5)
gs$TenDayMovAvg <- EMA(gs$Close,n=10)
gs$RSI <- RSI(gs$Close,n=14)
gs$WilliamR <- (max(gs$High) - gs$Close)/(max(gs$High)-min(gs$Low))*100
gsStoch <- stoch(gs[,c("High","Low","Close")],nFastK =14, nFastD=2)
gs$K <- gsStoch[,1]*100
gs$D <- gsStoch[,2]*100
gs <- subset(gs[15:753,],
               select=c("Open","High","Low","NextDayClose",
                        "FiveDayMovAvg","TenDayMovAvg",
                        "RSI","WilliamR","K","D"))

#Apple
aapl$NextDayClose <- aapl$Close[2:493]
aapl$FiveDayMovAvg <- EMA(aapl$Close,n=5)
aapl$TenDayMovAvg <- EMA(aapl$Close,n=10)
aapl$RSI <- RSI(aapl$Close,n=14)
aapl$WilliamR <- (max(aapl$High) - aapl$Close)/(max(aapl$High)-min(aapl$Low))*100
aaplStoch <- stoch(aapl[,c("High","Low","Close")],nFastK =14, nFastD=2)
aapl$K <- aaplStoch[,1]*100
aapl$D <- aaplStoch[,2]*100
aapl <- subset(aapl[15:491,],
               select=c("Open","High","Low","NextDayClose",
                        "FiveDayMovAvg","TenDayMovAvg",
                        "RSI","WilliamR","K","D"))


#IBM
ibm$NextDayClose <- ibm$Close[2:493]
```

```
ibm$FiveDayMovAvg <- EMA(ibm$Close,n=5)
ibm$TenDayMovAvg <- EMA(ibm$Close,n=10)
ibm$RSI <- RSI(ibm$Close,n=14)
ibm$WilliamR <- (max(ibm$High) - ibm$Close)/(max(ibm$High)-min(ibm$Low))*100
ibmStoch <- stoch(ibm[,c("High","Low","Close")],nFastK =14, nFastD=2)
ibm$K <- ibmStoch[,1]*100
ibm$D <- ibmStoch[,2]*100
ibm <- subset(ibm[15:491,],
              select=c("Open","High","Low","NextDayClose",
                       "FiveDayMovAvg","TenDayMovAvg",
                       "RSI","WilliamR","K","D"))
```

Partionining Training and Testing Data

In this next portion, we partition the training and testing data for each of our financial data set. For the historical data of Microsoft and Goldman Sachs, it is stated in the research paper that 80% of the first samples served as the training data set while 20% of the latter served as testing. As for Apple and IBM, both are strictly mentioned to have 91 observations as testing while the rest are training.

```
#creating training and testing data sets
#--------------------------
#Microsoft
msft_train <- msft[1:589,]
msft_test <- msft[590:739,]

#Goldman Sachs
gs_train <- gs[1:589,]
gs_test <- gs[590:739,]

#Apple
aapl_train <- aapl[1:386,]
aapl_test <- aapl[387:477,]

#IBM
ibm_train <- ibm[1:386,]
ibm_test <- ibm[387:477,]
```

Creating the Model

Below, we construct the model of interest in the research paper through the BRNN package. In the library BRNN, the function BRNN fits a two layer Bayesian Recurrent Neural Networks with the same methodologies as described in the paper. More specifically, it utilizes Forsee and Hagann(1997) methodology. Below, we reconstruct the models with neurons = 5 to emulate the paper results. However, please keep in mind that this function creates two hidden layers as opposed to one. Thus, our results must be interpreted with care as overfitting for the model is likely due to higher flexibility. The problems with overfitting is greatly discussed in the first part of this report.

Below are results and outputs for each of BRNN() calls for the respective stock data.

```
#Microsoft
msft_model_brnn <- brnn(data = msft_train, NextDayClose~High+
                          Low+Open+FiveDayMovAvg+TenDayMovAvg+
                          RSI+WilliamR+K+D, neurons=5,
                        normalize = TRUE)
```

```
## Number of parameters (weights and biases) to estimate: 55
## Nguyen-Widrow method
## Scaling factor= 0.7019154
## gamma= 15.0657    alpha= 4.6174   beta= 87.5798
```

```
#Goldman Sachs
gs_model_brnn <- brnn(data = gs_train, NextDayClose~High+
                        Low+Open+FiveDayMovAvg+TenDayMovAvg+
                        RSI+WilliamR+K+D, neurons=5,
                        normalize = TRUE)
```

```
## Number of parameters (weights and biases) to estimate: 55
## Nguyen-Widrow method
## Scaling factor= 0.7019154
## gamma= 16.7208    alpha= 5.3152   beta= 155.3326
```

```
#Apple
aapl_model_brnn <- brnn(data = aapl_train, NextDayClose~High+
                        Low+Open+FiveDayMovAvg+TenDayMovAvg+
                        RSI+WilliamR+K+D, neurons=5,
                        normalize = TRUE)
```

```
## Number of parameters (weights and biases) to estimate: 55
## Nguyen-Widrow method
## Scaling factor= 0.7029248
## gamma= 19.2796    alpha= 3.1675   beta= 206.119
```

```
#IBM
ibm_model_brnn <- brnn(data = ibm_train, NextDayClose~High+
                        Low+Open+FiveDayMovAvg+TenDayMovAvg+
                        RSI+WilliamR+K+D, neurons=5,
                        normalize = TRUE)
```

```
## Number of parameters (weights and biases) to estimate: 55
## Nguyen-Widrow method
## Scaling factor= 0.7029248
## gamma= 15.9113    alpha= 3.9752   beta= 71.9087
```

```
#Predicted values for each
msft_model_brnn_predicted <- predict(msft_model_brnn,newdata=msft)
gs_model_brnn_predicted <- predict(gs_model_brnn,newdata=gs)
aapl_model_brnn_predicted <- predict(aapl_model_brnn,newdata=aapl)
ibm_model_brnn_predicted <- predict(ibm_model_brnn,newdata=ibm)
```

Plotting the results of Predicted vs. Actual

In this section, we plot predicted results from actual predictions of next day closing prices. As seen from the graph below, the results closely emulate that of the research paper for which this project is trying to emulate. From purely a graphical stand point, it is hard to tell if any difference exist between the results below and that of the research paper. This is both true in both the prediction graph of trajectory of daily closing price of stocks as well as the predicted vs actual points in the scatter plot in our second diagram.

```
#Plotting Results of Predicted vs Actual
#-------------------------
#Microsoft
par(mfrow = c(1, 2))

plot(msft$NextDayClose,type='l',col="blue",
```

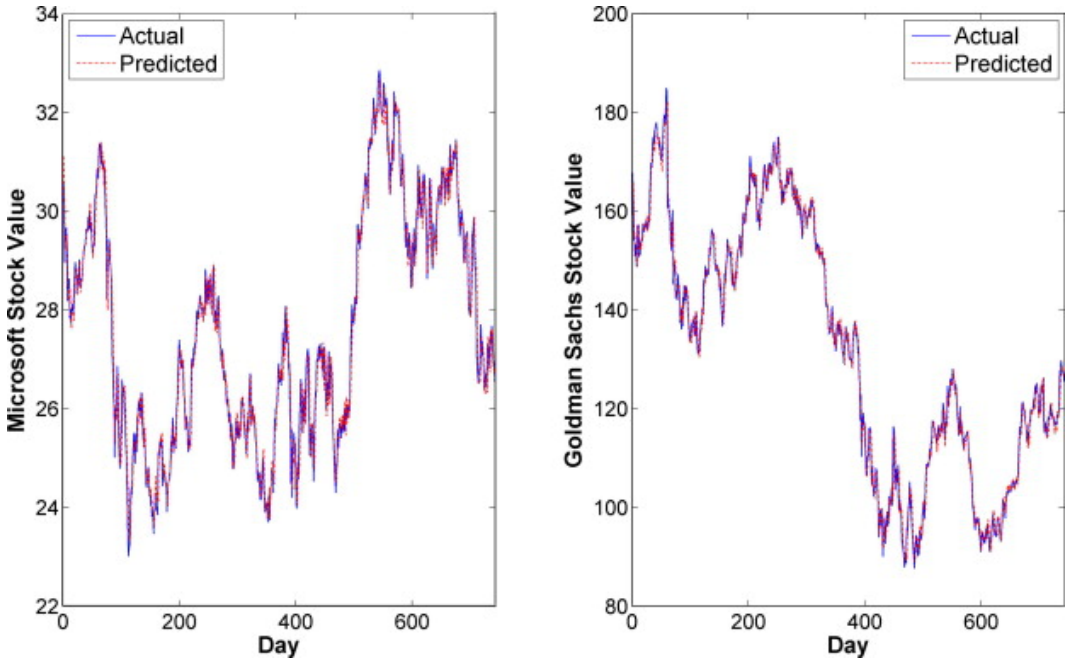
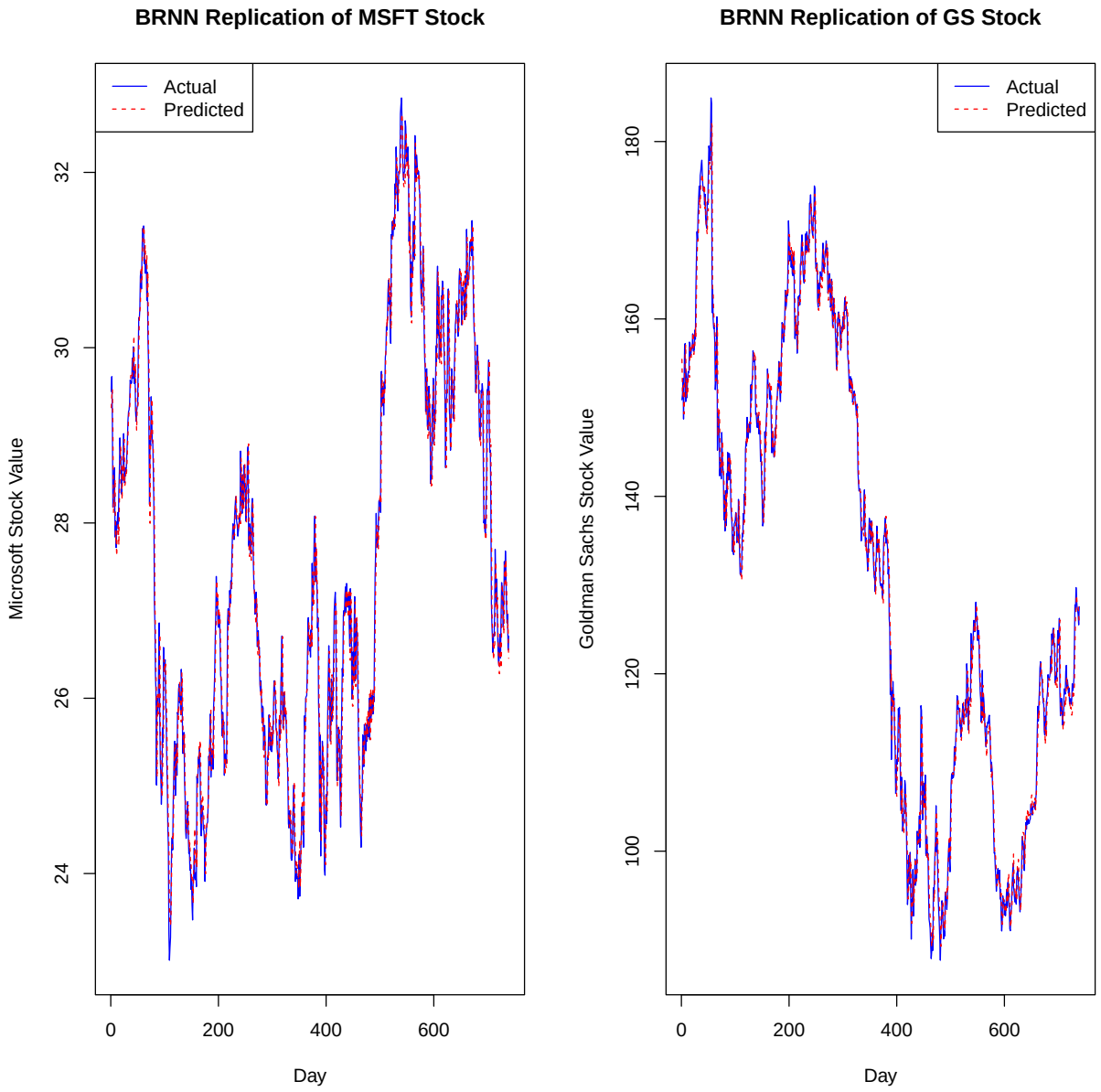


Figure 1: Original Research Results: BRNN

```
     ylab = "Microsoft Stock Value",
     xlab = "Day", main = "BRNN Replication of MSFT Stock")
lines(msft_model_brnn_predicted,col="red",lty=2)
legend(x = "topleft",
       legend=c("Actual", "Predicted"),
       col = c("blue","red"),lty=c(1,2))


#Goldman Sachs
plot(gs$NextDayClose,type='l',col="blue",
     ylab = "Goldman Sachs Stock Value",
     xlab = "Day", main = "BRNN Replication of GS Stock")
lines(gs_model_brnn_predicted,col="red",lty=2)
legend(x = "topright",
       legend=c("Actual", "Predicted"),
       col = c("blue","red"),lty=c(1,2))
```

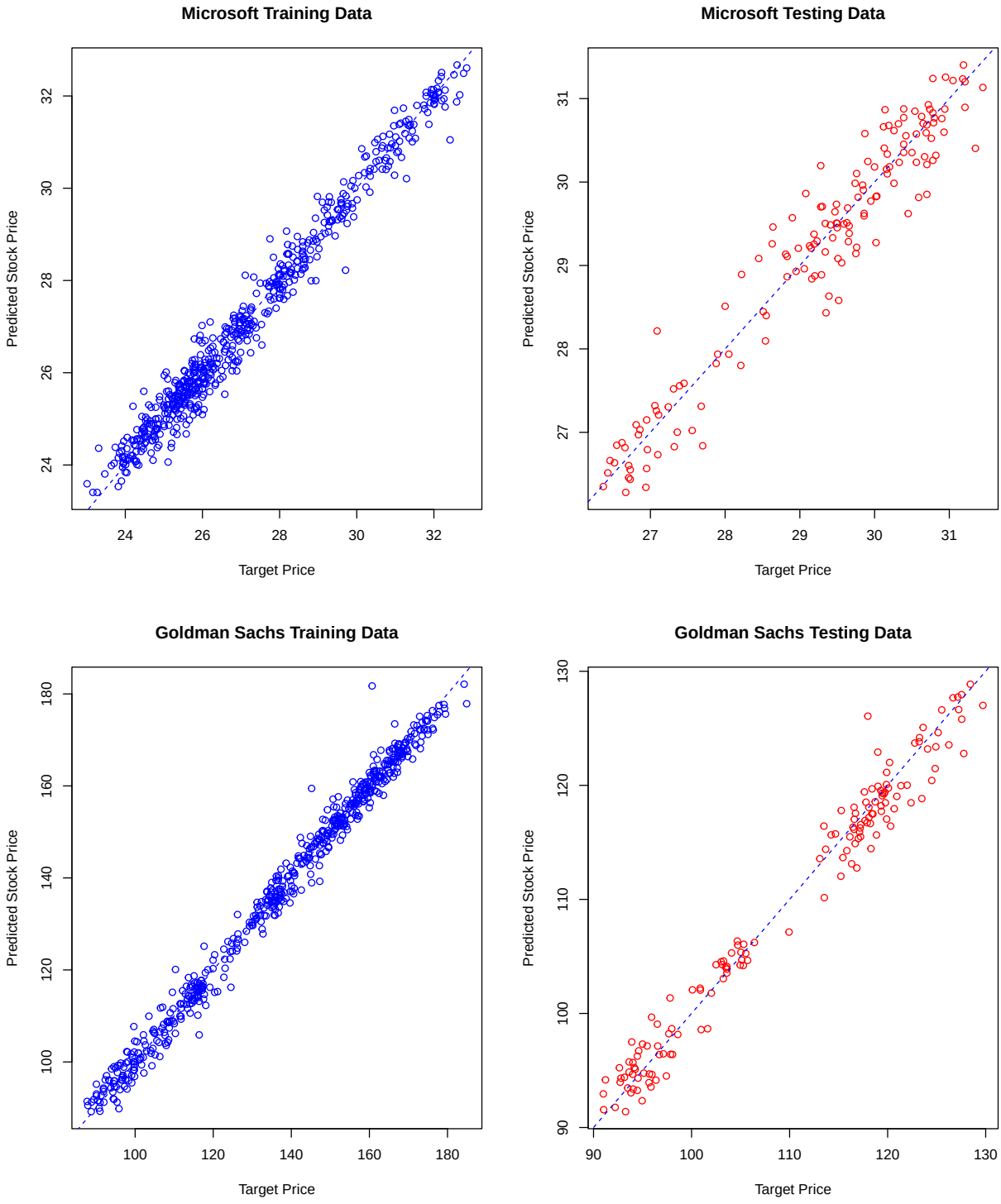


As seen from above, the graphs do not differ greatly from that of the graphs in the research paper. Thus, to further identify whether our results is similar, we utilize the MAPE criterion that is set out to measure accuracy of our predictions.

However, first we check to see plotted comparisons of the scatter plot from our prediction results below. Contained directly below is code used to generate the scatter plot figures that is in the next page.

```
#Plotting comparison of target and predicted stock price
#----------------------------
par(mfrow = c(2, 2))
#Microsoft
msft_train_predicts = msft_model_brnn_predicted[1:589]
msft_test_predicts = msft_model_brnn_predicted[590:739]
#plotting training
plot(msft$NextDayClose[1:589],msft_train_predicts,col="blue",
```

```
     main = "Microsoft Training Data",
     xlab = "Target Price",
     ylab = "Predicted Stock Price")
abline(a=0,b=1,col="blue",lty=2)
#plotting testing
plot(msft$NextDayClose[590:739],msft_test_predicts,col="red",
     main = "Microsoft Testing Data",
     xlab = "Target Price",
     ylab = "Predicted Stock Price")
abline(a=0,b=1,col="blue",lty=2)

#Goldman Sachs
gs_train_predicts = gs_model_brnn_predicted[1:589]
gs_test_predicts = gs_model_brnn_predicted[590:739]
#plotting training
plot(gs$NextDayClose[1:589],gs_train_predicts,col="blue",
     main = "Goldman Sachs Training Data",
     xlab = "Target Price",
     ylab = "Predicted Stock Price")
abline(a=0,b=1,col="blue",lty=2)
#plotting testing
plot(gs$NextDayClose[590:739],gs_test_predicts,col="red",
     main = "Goldman Sachs Testing Data",
     xlab = "Target Price",
     ylab = "Predicted Stock Price")
abline(a=0,b=1,col="blue",lty=2)
```



Notice that the plots do not differ much from the actual plots in the research. In fact, all points in both the training and testing data sets of both stocks follow a similar pattern in our graphs compared to that of the research. Thus, to further deduce whether or not our model compares to their findings, we must numerically assess the results by validating MAPE.

Validating MAPE

To validate our results, we check to see if our MAPE values resemble that of the research. Given below is the code of the MAPE function as well as the values of the MAPE for both of our stocks.

```
#MAPE Function
#-------------------------
MAPE <- function(actual,predicted){
  result = (sum(abs(actual-predicted)/actual)/length(actual))*100
  return(result)
}
```

Above is the function used to calculate MAPE score. Recall that it is

$$MAPE = \frac{\sum_{i=1}^{r} abs(y_i - p_i)/y_i}{r} * 100$$

Original Research Results for MAPE Table

Stock	Training MAPE (%)	Test set MAPE (%)	Total MAPE (%)
Microsoft (MSFT)	1.0494	**1.0561**	1.0507
Goldman Sachs (GS)	1.5235	**1.3291**	1.4860

Figure 2: Original MAPE Results

And below is code to supplement how to obtain the values above.

```
#Code for calculating MAPE
#-------------------------
msft_train_MAPE = MAPE(msft$NextDayClose[1:589],
                       msft_train_predicts)
msft_test_MAPE = MAPE(msft$NextDayClose[590:739],
                       msft_test_predicts)
msft_total_MAPE = MAPE(msft$NextDayClose,msft_model_brnn_predicted)
gs_training_MAPE = MAPE(gs$NextDayClose[1:589],
                        gs_train_predicts)
gs_test_MAPE = MAPE(gs$NextDayClose[590:739],
                      gs_test_predicts)
gs_train_MAPE = MAPE(gs$NextDayClose[1:589],
                        gs_train_predicts)
gs_total_MAPE = MAPE(gs$NextDayClose,gs_model_brnn_predicted)

table_summary = data.frame(Stock = c("Microsoft(MSFT)","Goldman Sachs (GS)"),
                           TrainingMAPE = c(msft_train_MAPE,gs_train_MAPE),
                           TestingMAPE = c(msft_test_MAPE,gs_test_MAPE),
                           TotalMAPE = c(msft_total_MAPE,gs_total_MAPE))
kbl(table_summary, booktabs = T) %>%
kable_styling(latex_options = "hold_position")
```

As seen from the above results in our table, our MAPE values closely aligns with the results of the research.

Stock	TrainingMAPE	TestingMAPE	TotalMAPE
Microsoft(MSFT)	1.041981	1.015918	1.036691
Goldman Sachs (GS)	1.511195	1.392217	1.487045

Summary output in comparison to Hassan et Al. (2007)

To further validate our findings, we check for any differences in our findings from that of the research. In the result below, it will be seen that our two layer hidden model fails to capture the upward trend in the Apple stock compared to the results in the paper. The plot below showcases this difference clearly as our model tails off to ensure a less volatile increase for predicting next day prices. This inability to predict the drastic increase may be due to the difference in added hidden layer in our Bayesian Recurrent Neural Network, causing our model to be more flexible and potentionally overfitting to skew the desired results.

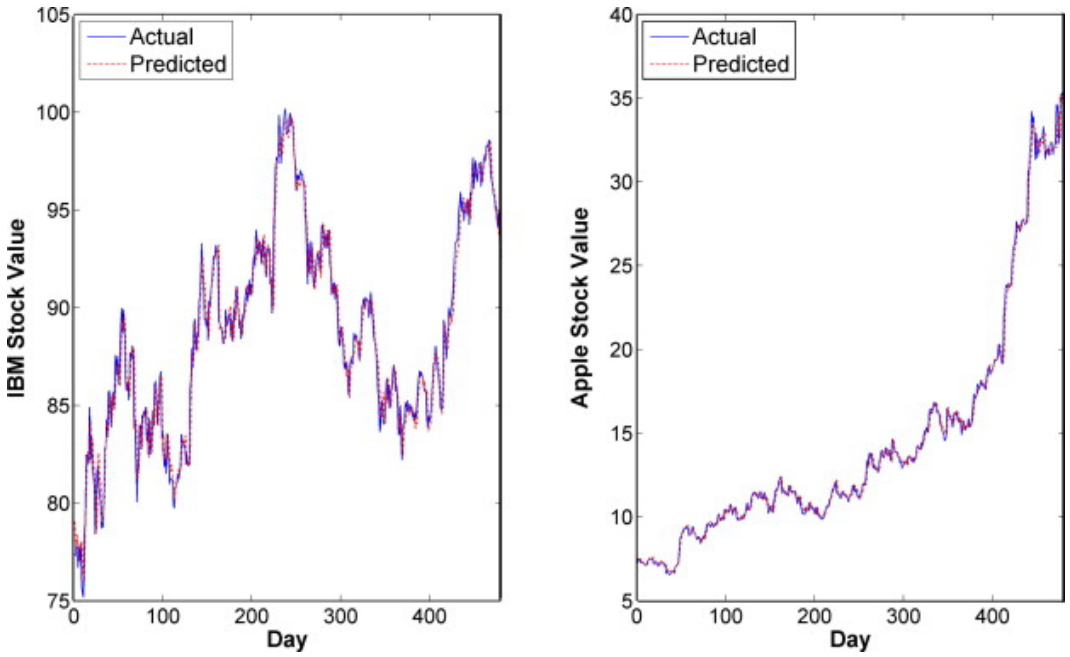
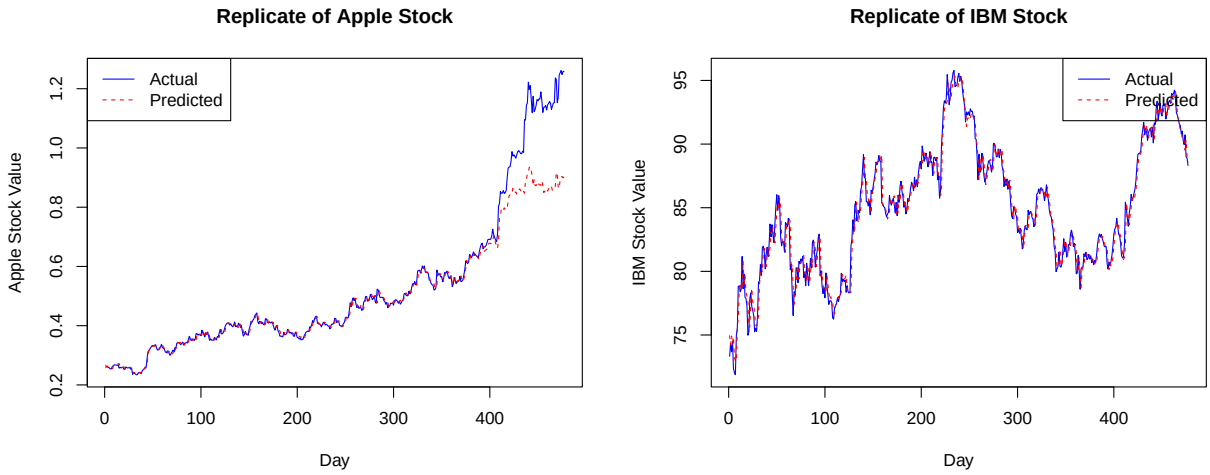


Figure 3: Original Research Results: BRNN

```
#Apple
par(mfrow = c(1, 2))

plot(aapl$NextDayClose,type='l',col="blue",
     ylab = "Apple Stock Value",
     xlab = "Day")
lines(aapl_model_brnn_predicted,col="red",lty=2)
legend(x = "topleft",
       legend=c("Actual", "Predicted"),
       col = c("blue","red"),lty=c(1,2))
title("Replicate of Apple Stock")


#IBM
plot(ibm$NextDayClose,type='l',col="blue",
     ylab = "IBM Stock Value",
     xlab = "Day")
```

```
lines(ibm_model_brnn_predicted,col="red",lty=2)
legend(x = "topright",
       legend=c("Actual", "Predicted"),
       col = c("blue","red"),lty=c(1,2))
title("Replicate of IBM Stock")
```



From the MSE results below, we see a major increase in our Test MAPE in our attempt to predict next day closing prices for Apple stocks. This major increase is also reflected above in the graph as it is clearly evident that our model fails to predict the drastic upward trend in the Apple stock. Although our IBM stock did perform well to emulate the desired results, more volatile changes in the Apple stock made it so that our model can not successfully emulate that of the research paper. A more keen look at tuning of our neural parameters may be the root cause of this issue.

Original Research Results For MAPE Table

Stock name	Mean absolute % error (MAPE) in forecast for 91 test dataset		
	Proposed Bayesian ANN	Fusion model with weighted average[a]	ARIMA model[a]
Apple Inc. (AAPL)	**1.9688**	1.9247	1.8009
IBM Corporation (IBM)	**0.7441**	0.8487	0.9723

Figure 4: Original MAPE

```
aapl_train_predicts = aapl_model_brnn_predicted[1:386]
aapl_test_predicts = aapl_model_brnn_predicted[387:477]

ibm_train_predicts = ibm_model_brnn_predicted[1:386]
ibm_test_predicts = ibm_model_brnn_predicted[387:477]

aapl_test_MAPE = MAPE(aapl$NextDayClose[387:477],
```

```
                        aapl_test_predicts)
ibm_test_MAPE = MAPE(ibm$NextDayClose[387:477],
                        ibm_test_predicts)
table_summary = data.frame(Stock = c("Apple(Aapl)","IBM (IBM)"),
                           TestingMAPE = c(aapl_test_MAPE,ibm_test_MAPE))
kbl(table_summary, booktabs = T) %>%
kable_styling(latex_options = "hold_position")
```

Stock	TestingMAPE
Apple(Aapl)	15.1088064
IBM (IBM)	0.6320563

The above is the MAPE values of the testing date set in our replication of the research.

Application in Comparison to Forward Neural Networks

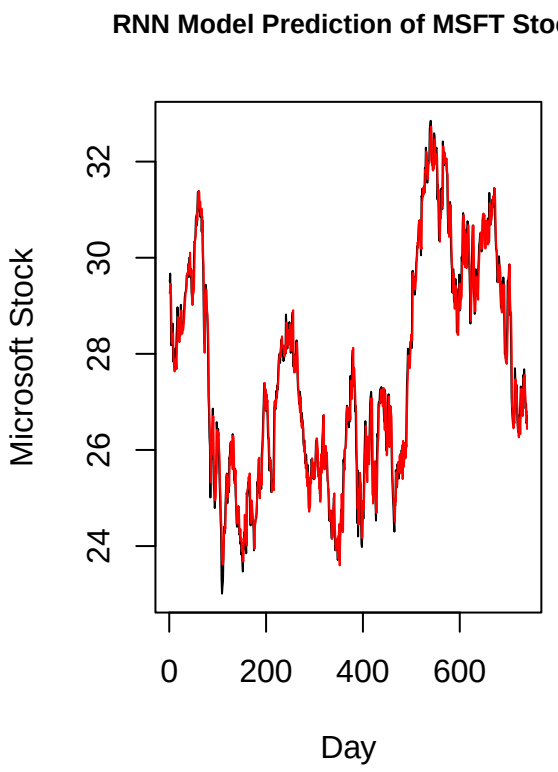
To compare our application of the above with other methods, we use a Forward Neural Network with 10 averaged validation result to evaluate how effective BRNN is to traditional methods.

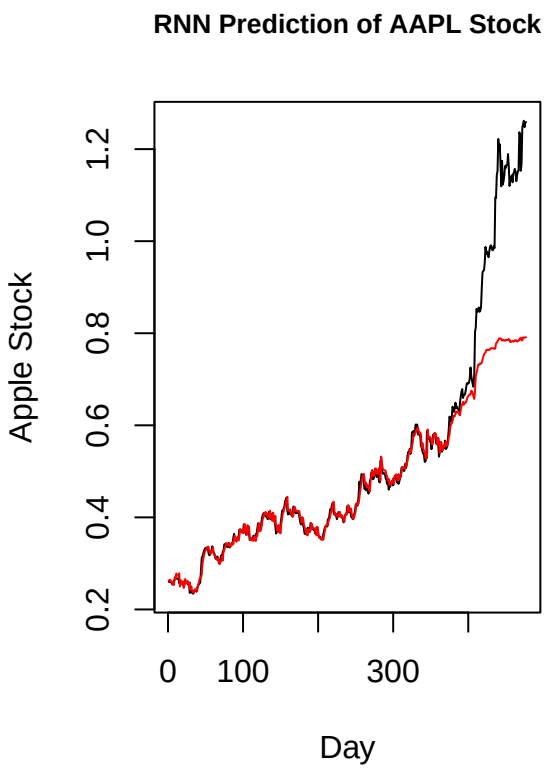
To perform the procedure below, we first normalize the data set so it takes in range from 0 - 1. To do this, we use a minimax function that is written below. The formula is as follows

$$normalized_{value} = \frac{value - max(value)}{max(value - min(value)}$$

```
par(mfrow = c(1, 2))

normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

msft_norm <- as.data.frame(lapply(msft, normalize))

msft_train <- msft_norm[1:589,]
msft_test <- msft_norm[590:739,]

nn<-neuralnet(data = msft_train, NextDayClose~High+
            Low+Open+FiveDayMovAvg+TenDayMovAvg+
            RSI+WilliamR+K+D,hidden=c(5,2),rep=10)

train_result <- predict(nn,msft_train)
test_result <- predict(nn,msft_test)


train_final <- train_result*abs(diff(range(msft$NextDayClose)))+min(msft$NextDayClose)
test_final <- test_result*abs(diff(range(msft$NextDayClose)))+min(msft$NextDayClose)
result_final <- c(train_final,test_final)

#MAPE for Microsoft
msft_total_MAPE <- MAPE(msft$NextDayClose,result_final)

plot(msft$NextDayClose[1:739],type="l",ylab="Microsoft Stock",xlab="Day",
```

```
     main="RNN Model Prediction of MSFT Stock",cex.main=0.8)
lines(result_final,col="red")

aapl_norm <- as.data.frame(lapply(aapl, normalize))

aapl_train <- aapl_norm[1:386,]
aapl_test <- aapl_norm[387:477,]

nn<-neuralnet(data = aapl_train, NextDayClose~High+
            Low+Open+FiveDayMovAvg+TenDayMovAvg+
            RSI+WilliamR+K+D,hidden=c(5,2),rep=20)

train_result <- predict(nn,aapl_train)
test_result <- predict(nn,aapl_test)


train_final <- train_result*abs(diff(range(aapl$NextDayClose)))+min(aapl$NextDayClose)
test_final <- test_result*abs(diff(range(aapl$NextDayClose)))+min(aapl$NextDayClose)
result_final2 <- c(train_final,test_final)

plot(aapl$NextDayClose[1:477],type="l", ylab="Apple Stock",xlab="Day",
     main = "RNN Prediction of AAPL Stock",cex.main=0.8)
lines(result_final2,col="red")
```






```
#MAPE for Apple
aapl_total_MAPE <- MAPE(aapl$NextDayClose,result_final2)
```

```
msft_train_MAPE <- MAPE(result_final[1:589],msft$NextDayClose[1:589])
aapl_train_MAPE <- MAPE(result_final2[1:386],aapl$NextDayClose[1:386])
msft_test_MAPE <- MAPE(result_final[590:739],msft$NextDayClose[590:739])
aapl_test_MAPE <- MAPE(result_final2[387:477],aapl$NextDayClose[387:477])

table_summary = data.frame(Stock = c("Microsoft(MSFT)","Apple (AAPL)"),
                           TrainMAPE = c(msft_train_MAPE,aapl_train_MAPE),
                           TestMAPE = c(msft_test_MAPE,aapl_test_MAPE),
                           TotalMAPE = c(msft_total_MAPE,aapl_total_MAPE))
kbl(table_summary, booktabs = T) %>%
kable_styling(latex_options = "hold_position")
```

Stock	TrainMAPE	TestMAPE	TotalMAPE
Microsoft(MSFT)	1.061927	1.047773	1.058298
Apple (AAPL)	2.034810	30.691492	5.792058

As seem from the MAPE value above, the result of our BRNN model differed greatly from the RNN model above. Although the performance did not vary in more stable stocks such as Microsoft, the difference can be seen clearly in the more volatile stock Apple. The MAPE value is double in the RNN model than the BRNN model for the testing dataset. This signifies that the RNN is much worse at finding patterns compared to the model that we studied in the research paper above.

Conclusion and Future Work

In this study, we explored a novel approach to prevent overfitting in neural networks: Bayesian Regularized Artificial Neural Networks (BRANN). Following the methodology outlined in a seminal paper, we applied this technique to forecast stock prices. In addition to the above, we employed a more conventional method to mitigate overfitting by repeatedly fitting the data using a feed-forward neural network. Our results indicate that BRANN outperforms the repetitive use of a feed-forward neural network.

Looking ahead, we intend to investigate the application of dropout layers, another strategy for preventing overfitting, and compare its efficacy with BRANN. Furthermore, integrating dropout layers into BRANN itself could be a worthwhile experiment. Additionally, considering that our current dataset assumes a normal distribution, we are also interested in exploring data following different distributions, such as gamma distribution, which is prevalent in the financial world. This exploration could provide further insights into the adaptability and effectiveness of neural network models across various types of financial data.