Project 2: Stock Prediction Using Neural Networks

B. Eichmeier[1]
CS 5600 – Intelligent Systems, Logan, UT, 84322, USA

Abstract

This paper explores the ability of neural networks to predict stock change. Both artificial neural networks and convolutional networks perform brute force training to predict the daily stock value change of Apple Inc. The training sequence cycles the hyperparameters of activation functions, learning rate, mini-batch size, layer depth, and input size. Results show this training method produces networks with an average prediction accuracy of 65%. The best performing network correctly predicts stock value at 75%. Trend analysis is done on the hyperparameters to reduce future training time. Future work will include implementing network ensembles and optimizing the amount of training data provided to the networks.

I.Introduction

Predicting stock values from a purely mathematical is a challenge due to seemingly stochastic behavior in the stock data. Analyzing stocks to predict price changes and yield profit involves highly technical analysis methods. The study of finance achieves these predictions by measuring various characteristics of stocks including volatility, public feedback, and market insight. Implementing neural networks to stock prediction eliminates the need for expertise by replacing the expertise with extensive training procedures. Successfully training a neural network would provide novice investors with another tool to make well-informed portfolio decisions. This document will focus on predicting Apple Inc. stock using both traditional artificial neural networks (ANNs) and convolutional neural networks (ConvNets). The networks will be trained to predict whether or not a stock will increase between its opening and closing values.

II.Methods

A. Tools and Third-Party Libraries

This project makes use of the following software and third-party libraries:

Python 2.7.15

Numpy 1.15.4

TFLearn 1.11.0

Fix_Yahoo_Finance 0.0.22: Yahoo Finance is used to collect historical stock data. The Yahoo Finance API lost functionality May 15, 2017. Since then, developers created a new API called Fix_Yahoo_Finance to allow legacy code to function without modification. This API downloads stock data given a stock ticker, a start date, and an end date. The downloaded package holds a record of the opening and closing prices of different stock. The READ.ME file includes instructions for downloading this API.

B. Data Preparation

The training routines for this project used two date ranges for the stock data. The first range included data from May 1, 2017 to Dec 31, 2017. The second data range spanned from May 1, 2016 to Dec 31, 2018. The first data range was chosen from the results during initial testing. The project proposal stated the testing window would include a year of data. This size of input data was altered because the first successful training result occurred when changing the dates. The 7-month data range proved ineffective when testing the networks with input sizes greater than 50. Large input sizes reduced the number of testing samples. Increasing the range of stock data resolved this by collecting more

[1] Senior, Mechanical and Aerospace Engineering, Braden Eichmeier A01662343

data. The This project obtains the stock data by finding the difference between the opening value and the closing value and normalizing the result. The algorithm is the following:

```
FUNCTION: Obtain Stock Data
inputData = []
For each day in the stock data
   Change = openValue[day]-closingValue[day]
   Append Change into inputData
maxValue=MAX(MAX(inputData),-MIN(inputData))
For each Change in inputData
   Change = Change/maxValue
```

Preprocessing the data may be implemented using signal analysis, such as a Fourier transform, in order to extract features in the case of poor initial results.

Next, the data must be split into training and testing segments, along with the corresponding target values. This is done by creating a sliding window along the inputData array. The length of the sliding window represents the number of days used as input to the NN. This project will search for an optimal size for this window. Only sizes corresponding to perfect squares will be considered to allow the ConvNets to use a square window. The window looks like the following for a window size of four:

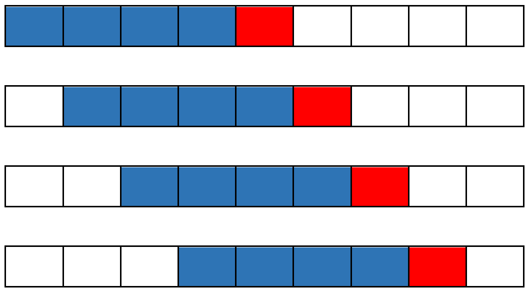

Fig. 1. Graphical representation of NN data collection

The blue cells represent the data and the red cell represents the target value. After the inputData array is segmented, the data is split using a 7/10 ratio between training and testing data.

C. Network Architectures

The presented results include network architectures with one to three layers. Initial testing produced results equivalent to guessing when using deep (more than three layer) networks. Reducing the network depth yielded training cost improvements after as few as five training epochs. Each layer for a given network will employ a single activation function. Table 1 shows all of the hyperparameters that will be cycled during testing. Some parameters were eliminated or condensed during testing. This was due to eliminate parameters that did not produce accurate results and reduce training time.

Table 1. Hyperparameters used to test the ConvNets and ANNs

Hyperparameters	**Convolutional Networks**	**Artificial Neural Networks**
Activation Functions	Tanh, Sigmoid, Linear, ReLu	
LR	0.1, 0.01, 0.001, 0.0001	
MBS	10, 20, 50	
Layer Depth	5, 10, 15, 20	10, 20, 50, 100, 200, 500

Similarly, initial testing produced the best results when using a window size smaller than 100 days. This difficulty likely occurs due to the volatility of the stock market. Stocks tend to behave differently over short periods of time. As such, using large input windows may incorporate several segments of behavior and fail to teach the network. The networks for this project were trained using windows the sizes of the perfect squares from 4→100.

D. Training Routine

The network architectures will train using a brute-force method to test all combinations of the hyperparameters. Due to the computationally intensive approach, testing for the three-layer networks will be done in segments to reduce strain on system resources. Each network will train for 60 epochs. For each network, the algorithm will print the progress in the training regimen, train the network, calculate the accuracy of the trained network on the testing data, and persist the network if it produces the highest accuracy. The pseudocode of the training is as follows:

```
For all combinations of the hyperparameters
      Print training statistics and remaining time
      Train network
      Calculate networkAccuracy on testData
      If networkAccuracy > bestAccuracy
              Save current network as bestNetwork
```

III.Results and Discussion

This section reviews the results of the network. In total, the results reflect 30,000 trained networks using the given parameters. The results are broken down by the effectiveness of each parameter. The recorded accuracies from the best networks range from 58-75%. Ten networks produced a testing accuracy over 70%. Repeating the training sessions shows the inconsistency of neural networks. When repeating a training session, the next version of testing unanimously produces a network with different hyperparameters. This behavior likely occurs due to the random initialization of the networks' weights and biases. The Appendix contains the results for every training session.

E. Training Window Size

The training window is the number of input cells into the networks. This value represents the number of previous days used to predict the following day's stock behavior. Figure 2 shows the best accuracies for each window size.

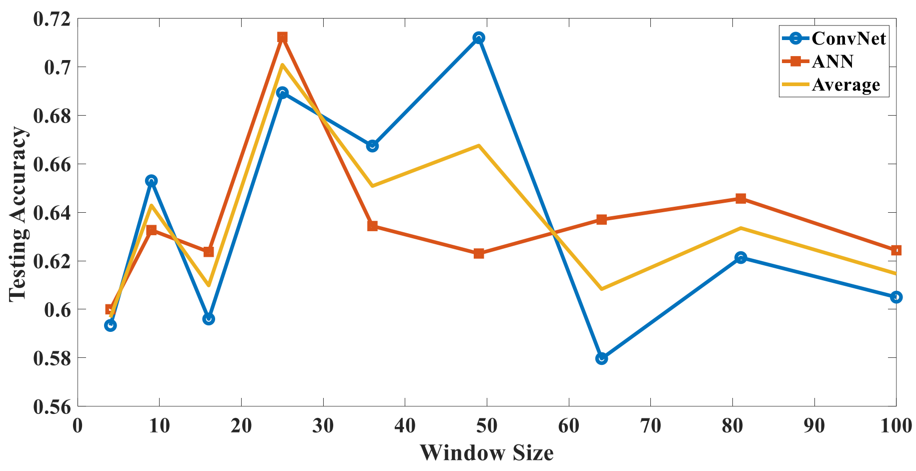

Fig. 2. Testing accuracies as a function of window size.

As shown in Fig. 2, the networks performed best when using a window size of 25. This general accuracy stays relatively constant with window sizes of 36 and 49. Figure 3 shows the distribution of the networks that exceeded 70% in testing accuracy. by the window size. Note the average network performance and the number of accurate networks follow a similar trend.

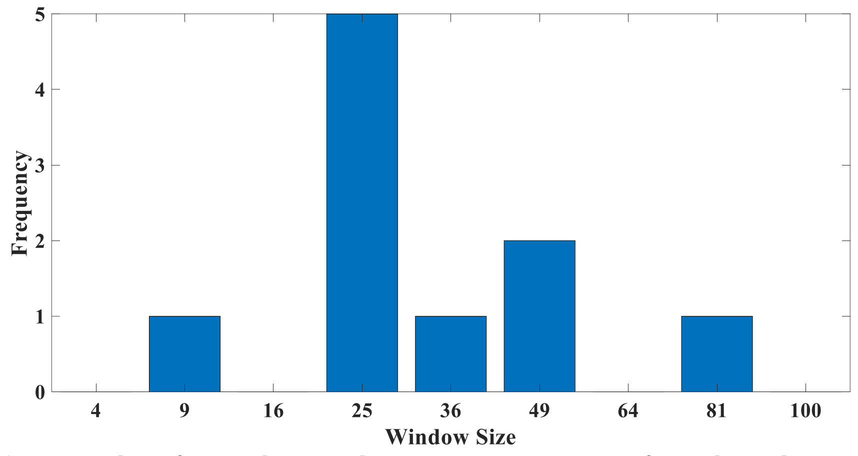

Fig. 3. Number of networks exceeding 70% testing accuracy for each window size.

F. Activation Functions

The functions employed in testing include ReLu, sigmoid, linear, and tanh. Multiple functions were removed when training the three-layer ConvNets. The sigmoid and tanh functions were removed due to poor performance in the one-layer and two-layer networks.

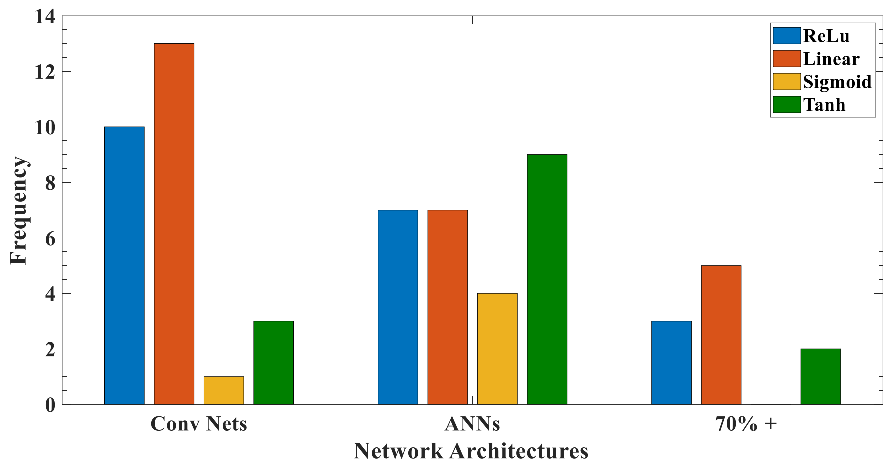

Fig. 4. Distribution of activation functions for the different network architectures.

The data shows the linear and ReLu activation functions are the most consistent in producing the most accurate network. The sigmoid and tanh functions perform poorly in the convolutional networks. The frequency trend stays consistent in the number of activation functions that produced networks with a greater than 70% testing accuracy.

G. Learning Rate

The testing routines in this project cycled through four different learning rates. These included 0.1, 0.01, 0.001, and 0.0001. Figure 5 shows the distribution of the learning rates which resulted in the best network for each training routine. Note, the 3-layer ConvNets did not use 0.1 or 0.0001 due to poor results when testing the 1-layer and 2-layer ConvNets.

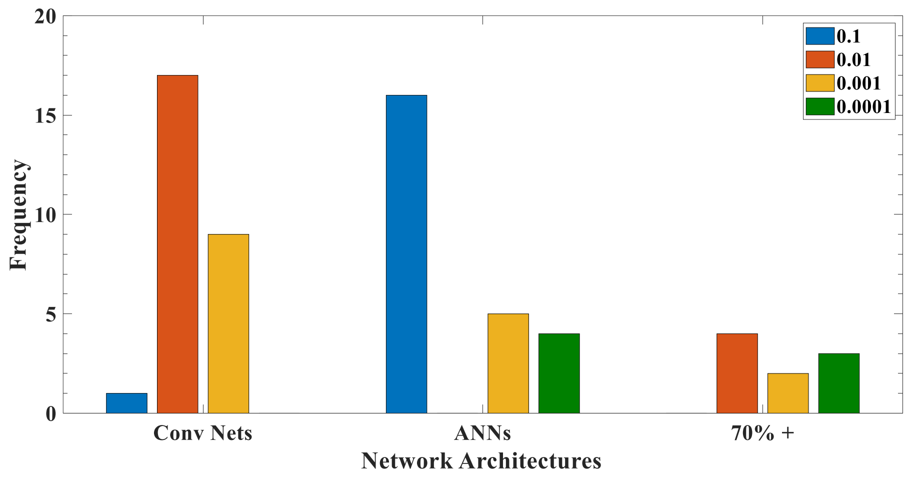

Fig. 5. Distribution of the best learning rate for the trained networks.

The results show starkly different behavior between ConvNets and ANNs. The ConvNets produced better accuracy when using a learning rate between 0.01-0.001. The ANNs most frequently performed best when using 0.1 for the learning rate. The networks recording an accuracy above 70% exhibits yet another unique behavior. The best performing networks used a nearly even spread between learning rates of 0.01, 0.001, and 0.0001. Analyzing the data shows the four ANNs achieving 70% accuracy used a learning rate of 0.0001. The Conv Net learning rates also show a clear divide with regards to memory size. The ConvNets with a memory less than 40 all use a learning rate of 0.01; above a memory of 40, all of the learning rates are 0.001.

H. Mini-Batch Size

The mini-batch sizes (MBS) used in training include 10, 20, and 50. All three options were used when training the three ANN structures. Training for the 3-layer Conv Net used only 10 and 20 due to low accuracy from 50 when training the 1-layer and 2-layer ConvNets. Figure 6 shows the MBS distribution for the trained networks.

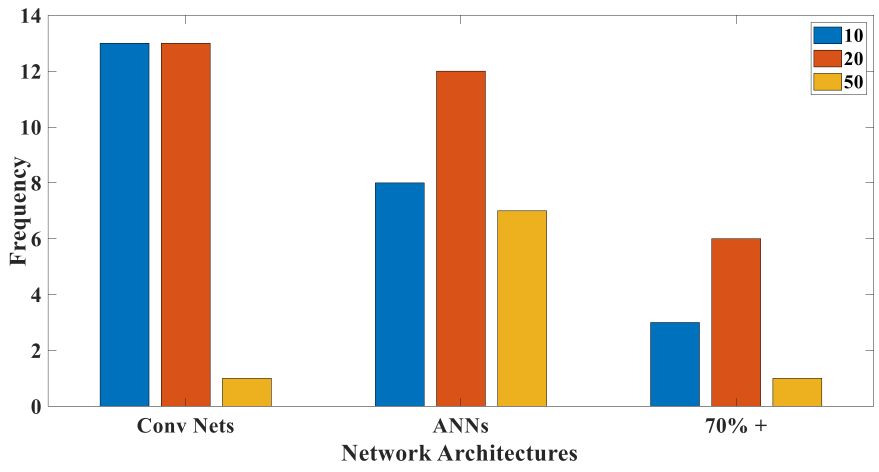

Fig. 6. Mini-batch size distribution for the trained networks.

Results from the training show the most accurate functions to be 20, 10, and 50. The ConvNets performed best with a MBS of 50 only once. The ANNs' results show a near even success rate between a MBS of 10 and 50; a MBS of 20 produced near double the successful networks than the other MBS options. Analysis of the 70% accurate networks shows a MBS trend similar to the ANNs with the MBS of 20 outperforming the other two MBS options combined.

Analyzing the MBSs in the best networks shows distinct trends between the network styles and the number of input nodes. All but one of the ANNs with a testing accuracy above 70% used an MBS of 20. The ConvNets used a

1:1 ratio of 10 and 20. All of the ConvNets with less than 30 input nodes use an MBS of 10. The ConvNets with more than 30 input nodes use an MBS of 20.

I. Nodes Per Layer

Different values for the nodes per layer were used for the ConvNets and the ANNs. The ConvNets used depths between 5 and 20; the ANNs used depths between 10 and 500. Analyzing the data shows no apparent correlation between the accuracy and the number of nodes for most of the trained networks. Close analysis of the networks with an accuracy higher than 70% show the layer depths are monotonically increasing for each network. Figure 7 shows this behavior.

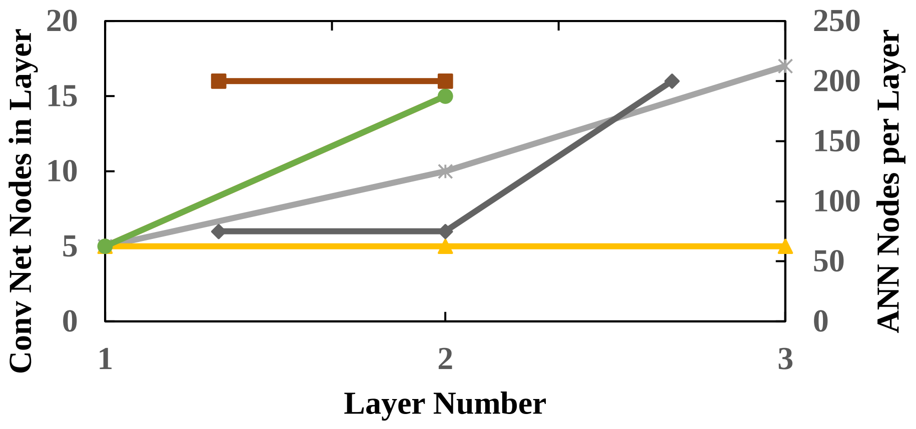

Fig. 7. Monotonically increasing layer depths in the 70%+ networks

IV.Conclusion

Simple neural networks show promising results in prediction stock data. Using brute force training procedures, networks average 64% accuracy on training data. Analysis of the trained network data shows strong trends in the effectiveness and ineffectiveness of the hyperparameters. Table 2 shows the more effective hyperparameters. The data showed not apparent correlation between the layer depths and testing accuracy.

Table 2. Most accurate hyperparameters from trends in the trained networks

Hyperparameters	Convolutional Networks	Artificial Neural Networks
Activation Functions	ReLu, Linear	ReLu, Linear, Tanh
LR	0.01, 0.001	0.1
MBS	10, 20	10, 20, 50

Future work on neural networks in stock prediction may employ testing various techniques. The first parameter to optimize would be the amount of data used to train the networks. The networks with a large number of input nodes performed poorly when trained on 7 months of data. Similarly, the networks with less input nodes performed poorly when trained on 19 months of data. Thus, further work may be done in finding an optimal size of training data with respect to the network's input nodes.

Another area of future work will involve network ensembles. A network ensemble uses multiple networks to predict the same value. The program would then return the highest voted prediction. Previous studies in other applications show ensembles typically improve neural network prediction accuracy. To create an ensemble, the prediction algorithm would consider the predictions of all the networks with a testing prediction above 70%.

V.Appendix

Table 3. One-layer ConvNets training results

Memory	Accuracy	Nodes 1	Function	MBS	LR
4	0.6	5	linear	20	0.001
9	0.714	5	relu	10	0.01
16	0.533	5	sigmoid	10	0.1
25	0.636	15	linear	20	0.01
36	0.636	10	tanh	50	0.01
49	0.703	20	relu	20	0.001
64	0.565	5	linear	20	0.01
81	0.602	15	tanh	20	0.01
100	0.619	20	linear	20	0.001

Table 4. Two-layer ConvNets training results

Memory	Accuracy	Nodes 1	Nodes 2	Function	MBS	LR
4	0.6	20	5	linear	10	0.01
9	0.612	10	5	relu	10	0.01
16	0.617	20	10	linear	10	0.001
25	0.705	5	15	linear	10	0.01
36	0.659	10	15	relu	10	0.01
49	0.757	5	15	linear	20	0.001
64	0.563	5	5	linear	10	0.01
81	0.641	20	15	tanh	10	0.001
100	0.598	10	20	relu	10	0.01

Table 5. Three-layer ConvNets training results

Memory	Accuracy	Nodes 1	Nodes 2	Nodes 3	Function	MBS	LR
4	0.58	15	5	10	relu	20	0.01
9	0.633	10	5	10	relu	20	0.001
16	0.638	10	10	10	relu	20	0.001
25	0.727	5	10	17	relu	10	0.01
36	0.707	5	5	5	linear	20	0.01
49	0.676	5	5	10	linear	10	0.001
64	0.611	10	17	5	linear	20	0.01
81	0.621	17	10	10	linear	20	0.01
100	0.598	10	10	17	relu	10	0.01

Table 6. One-layer ANN training results

Memory	Accuracy	Nodes 1	Function	MBS	LR
4	0.6	200	tanh	10	0.1
9	0.653	10	sigmoid	20	0.1
16	0.596	20	sigmoid	10	0.1
25	0.705	500	linear	50	0.0001
36	0.61	500	linear	50	0.1
49	0.622	500	linear	10	0.1
64	0.63	100	relu	20	
81	0.704	500	linear	20	
100	0.608	100	linear	10	0.1

Table 7. Two-layer ANN training results

Memory	Accuracy	Nodes 1	Nodes 2	Function	MBS	LR
4	0.58	15	200	relu	20	0.001
9	0.612	15	15	relu	50	0.001
16	0.615	200	15	tanh	20	0.1
25	0.727	200	200	tanh	20	0.0001
36	0.634	15	15	linear	50	0.0001
49	0.625	200	50	tanh	10	0.1
64	0.625	15	15	relu	20	0.001
81	0.612	15	15	linear	10	0.1
100	0.667	15	15	sigmoid	10	0.001

Table 8. Three-layer ANN training results

Memory	Accuracy	Nodes 1	Nodes 2	Nodes 3	Function	MBS	LR
4	0.62	15	15	15	sigmoid	50	0.1
9	0.633	15	200	15	relu	20	0.1
16	0.66	15	75	200	tanh	10	0.1
25	0.705	75	75	200	tanh	20	0.0001
36	0.659	15	200	200	tanh	50	0.1
49	0.622	15	75	200	relu	20	0.1
64	0.656	15	15	15	relu	20	0.001
81	0.621	200	75	200	tanh	50	0.1
100	0.598	15	75	200	tanh	20	0.1