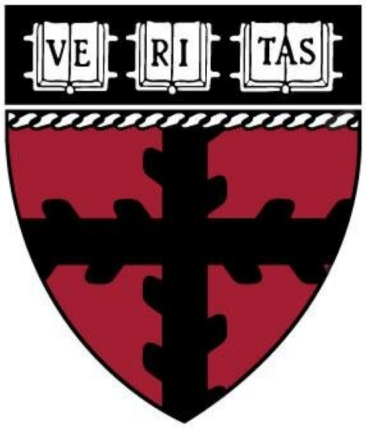
One Step Back, Two Steps Forward - a Machine Learning-Powered Options Trading Strategy

One Step Back, Two Steps Forward

A Machine Learning-Powered Options Trading Strategy



Tomislav Žabčić-Matić

Department of Applied Mathematics

Harvard University

A Senior Thesis presented to the Department of Applied Mathematics in partial fulfillment of Honors in a Concentration in Applied Mathematics with Area of Application In Computer Science

March 29, 2019

Acknowledgements

Thanks to my advisor, Professor Leslie Valiant, whose support of this project was incredible.

Special thanks to my parents and friends, who pushed me to be the best I could be, and without whose love and support this would not have been possible.

Contents

Introduction

The ability to accurately predict volatility in stock markets has become a powerful asset for many quantitative trading firms, as it has opened the possibility of quantitative investing in a range of financial derivatives which offer certain benefits over standard stocks. One common example of such a derivative is what is known as a stock option. Stock options are derivative assets which provide a form of insurance against making a bad trade, at the low cost of an option premium. This has allowed traders to hedge their investments, particularly when it comes to very volatile stocks, and eliminate some of the risk associated with investing in such assets.

One form of financial derivative, which is composed of two stock options, is known as a *straddle option spread*. This is an asset whose profitability is agnostic to the direction of motion of the underlying stock. A straddle will become profitable as long as the price of the underlying asset moves, in either direction, by more than the premium paid for the straddle. Thus, it becomes useful to be able to predict the volatility of stocks, and accurately classify whether or not a given stock's value will move by more than the cost of a straddle option spread for that underlying stock.

In this work, we will explore the use of simple machine learning models, particularly neural networks, in predicting, on a daily basis, whether the price of a stock will move, within the next trading day, by more or less than the cost of its associated straddle option spread, with expiry date set to be the next trading day. We will employ the use of multilayer perceptrons for classification into two classes in order to produce our predictions, and will develop a trading strategy designed to, in expectation, clear any deficits incurred since its last good trade, and increase earnings by a constant amount upon making a good trade.

We will test the performance of the trading strategy by using the predictions of the neural network systems we develop as instructions for when to buy or sell straddle option spreads. Additionally, we will derive a theoretical bound on the expected deficit that the trading strategy may incur in any series of consecutive bad trades, and extract a condition under which this quantity is expected to converge.

Chapters 1 through 3 will develop some of the basic machine learning mechanisms which we will use throughout this work, focusing on variations of small neural networks, trained with different structural and optimizational hyperparameters. From here, we will select the model that will be used in later sections of the work. In Chapter 4, we establish the trading strategy which we use, and proves the important theoretical bound required for the deficit to converge in expectation. Finally, in Chapter 5, we test several different frameworks under which we can apply the trading strategy with our algorithm's predictions, and we generate results for a variety of stocks across several different market sectors.

Chapter 1: The Prediction Problem

Stock options are a common tool that serve as insurance against potentially bad trades. In exchange for a premium, the negative portion of the profit curve becomes zero, so if a stock were to move in the wrong direction, the option need not be exercised, and the trader would only lose the premium on the option. Standard options are divided into two types - *calls* and *puts* - which can then be combined in various quantities to produce what are known as option *spreads*, financial instruments that have payoff and profit curves which are useful under the assumption that one can predict, with a high enough level of accuracy, the volatility of the underlying asset.

A *call option* C with strike price K is defined by the payoff curve at its maturation date T which has the function

$$\mathbf{P}(C) = \begin{cases} 0, \text{ if } S_T < K \\ \\ S_T - K, \text{ otherwise} \end{cases},$$

where S_T is the price of the underlying asset at time T.

A *put option* P with strike price K is defined by the payoff curve at maturity T which has the function

$$\mathbf{P}(P) = \begin{cases} 0, \text{ if } S_T > K \\ \\ K - S_T, \text{ otherwise} \end{cases}.$$

We can now use the two basic kinds of options to define a *straddle option spread*, St, which for strike price K, and maturity T, has a payoff curve of

$$\mathbf{P}(St) = |S_T - K|,$$

or in other words, a payoff curve which is the absolute value of the difference in price between the underlying asset's value at maturity, and the strike price of the option spread.

Thus, since the payoff curve of a straddle is at least zero, when we factor the cost of the spread into the profit function, we see that buying a straddle is an investment which has limited potential loss. Now, the goal is to predict with reasonable accuracy whether or not the price of an asset will experience motion in either direction between current time t and a future time T that is greater than the cost of a straddle with strike price S_t and maturity T.

One basic idea for a prediction setup might be to train a system to predict exactly what was described above, i.e. to classify data points for each given time period into two classes:

1. The price of the underlying asset will move (in either direction) *more* than the cost of the straddle;
2. The price of the underlying asset will move (in either direction) *less* than the cost of the straddle.

Chapter 2: Initial Experimental Results

Due to the difficulty of obtaining historical option prices which arises from prohibitive data costs, we will first tackle the four-class classification problem where the class cutoff occurs at $\left|(\Delta S_{t,t+1})_{\text{pred}}\right| = 1\%$, rather than at $\left|(\Delta S_{t,t+1})_{\text{pred}}\right| = [\text{straddle cost}]$.

In order to be able to assess the success of more advanced methods, we must first establish some baselines using common/basic architectures for predictive systems. For the most basic method, we will select a single-layer neural network that maps 28 features obtained from basic daily stock market quotes (open, high, low, and close prices, along with trading volume) into two outputs, which are then transformed via the *softmax activation function* into class probabilities, from which test predictions are made. The softmax activation function is a multiclass equivalent of the logistic (sigmoid) activation function, which is expressed as $\sigma(x) = \frac{1}{1+e^{-x}}$. We express the softmax activation function for k output classes as

$$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{i=1}^{k} e^{x_i}}, \text{ for } j \in \{1, \ldots, k\}, \text{ and } \mathbf{x} \in \mathbb{R}^k.$$

The loss is calculated by using Cross-Entropy loss [1] for four classes, where the multiclass Cross-Entropy formula with set of classes $\mathcal{C}$ is defined as

$$\mathcal{L}_{CE}(p, q) = -\sum_{c \in \mathcal{C}} p(c) \ln\left(q(c)\right).$$

Here, $p(c)$ denotes the true class probability of class c for a given training example, which equals 1 for the target class an 0 for all incorrect classes. The values $q(c)$ denotes the predicted class probability of class c for the given training example. The predicted class probabilities for the various classes arise from the softmax function, which normalizes its

outputs so that they sum to 1, and in testing, the selected output class is chosen to be the one whose output probability was the greatest.

The results of training these basic systems for 100000 iterations, for several hyperparameter values, are presented in the Table 1. The minibatch size of each iteration is 50, and data points for each minibatch are selected uniformly at random from the set of training data points. Note that these results are based on training the systems on daily quotes for the past 10 years of Apple (AAPL) stock data obtained from Yahoo Finance, with the training set comprising the first $\frac{4}{5}$ of the data, and the test set comprising the remaining $\frac{1}{5}$ of the data. We will explore the results of various networks on multiple assets later in this work.

The network architecture tested here is a simple multiclass perceptron architecture, with 28 input data points, no hidden neurons, and 2 output classes. **End Accuracy** refers to the accuracy of the system after 100000 rounds of training, and **Top Accuracy** refers to the highest accuracy achieved by the network type at any point during training. These two figures are reported as an average across 20 trials of the end and top accuracies of individual neural network models.

From these results, we see that the best results for training of a vanilla single-layer neural network with various hyperparameters is obtained using Adam, with a learning rate of 0.0001, and a weight decay of 0.001. Since the accuracy in the last training iteration is somewhat lower than the best accuracy, it seems reasonable to assume that the network begins overfitting the training data at some point, so in order to reduce overfitting, it is necessary to run several rounds of such training, to determine the average best test accuracy, and average range of iterations within which the maximum occurs, so that we can determine an appropriate early stopping point for the network trained with Adam, under the given learning rate and weight decay.

Optimizer	Learn Rate	Weight Decay	End (Test) Accuracy	Top Accuracy
SGD (Momentum 0.9)	0.0001	0.001	52.907%	56.85%
SGD (Momentum 0.9)	0.00001	0.0001	43.791%	55.955%
Adam	0.0001	0.001	54.787%	59.563%
Adam	0.00001	0.0001	54.268%	57.835%
Adadelta	0.0001	0.001	46.362%	48.364%
Adadelta	0.00001	0.0001	49.888%	50.203%
RMSprop	0.0001	0.001	54.644%	56.514%
RMSprop	0.00001	0.0001	54.055%	55.234%

Table 1: Results of simple multiclass perceptron with 28 inputs and 2 outputs

A graph of the test loss and test accuracy for Adam(0.0001, 0.001) is shown in Figure 1 below. The x-axis in both graphs represents iterations in hundreds, and the right-hand-side graph shows percentage accuracy. The graph data in both graphs is an an iteration-averaged aggregate of the data from 20 different trials of the same network model. Test loss and accuracy are measured against the last $1/5$ of the full dataset - slightly less than 500 days.

Based on these results, it appears reasonable to conclude that the Adam [2] and RMSprop [3] optimizers will be the best ones to move forward with when testing more complex architectures.

We now present results of testing several two-layer neural networks, with one hidden layer that has a larger number of neurons than there are inputs in each data point (see Table 2). We restrict the options for the learning rate and weight decay values for simplicity, and due to the fact that the values we are testing on are commonly accepted as reasonable values to start with, with the expectation that some fine-tuning will be performed later in the process.

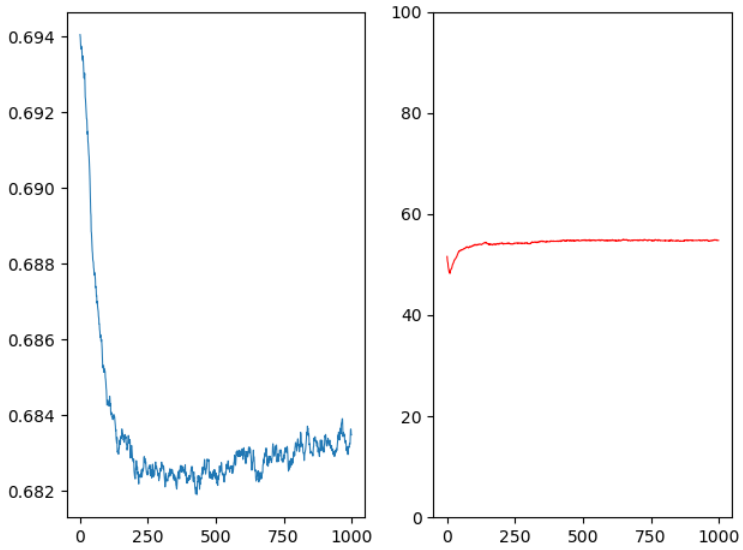


Figure 1: Plots of test loss (left) and accuracy (right) over 10^6 iterations (simple network)

Hidden Size	**Optimizer**	**Learn Rate**	**Weight Decay**	**End Accuracy**	**Top Accuracy**
50	Adam	0.0001	0.001	60.122	64.573
50	Adam	0.00001	0.0001	58.699	63.272
50	RMSprop	0.0001	0.001	60.315	63.455
50	RMSprop	0.00001	0.0001	59.136	62.205
100	Adam	0.0001	0.001	61.118	64.045
100	Adam	0.00001	0.0001	58.831	62.449
100	RMSprop	0.0001	0.001	60.783	64.329
100	RMSprop	0.00001	0.0001	58.526	63.059
200	Adam	0.0001	0.001	60.904	64.665
200	Adam	0.00001	0.0001	58.872	62.368
200	RMSprop	0.0001	0.001	60.508	64.827
200	RMSprop	0.00001	0.0001	58.618	61.494

Table 2: Results of multilayer perceptrons for different hidden layer sizes and optimizers

The results in Table 2 seem to indicate that the networks with 200 hidden neurons, trained with the faster learning rate, would be the best networks in terms of performance, as both the network trained with the Adam optimizer and the RMSprop optimizer have the non-negligibly higher accuracy than the rest of the networks that were tested.

However, an underlying problem with selecting these networks just based on their top accuracy is that their end accuracy is lower than that of their counterparts with only 100 hidden neurons, so if training is allowed to continue to the end of the 50000 iterations, the 200-hidden-neuron networks will end up performing worse on average than the 100-hidden-neuron networks. Additionally, through a series of tests examining early stopping for reduction of overfitting, we have determined that it is easier to achieve better average end performance by stopping the 100-hidden-neuron networks early at a constant early stopping time than it is to do the same for the 200-hidden-neuron networks. Thus, for the rest of this work, we will proceed with the network structures involving 100 hidden neurons.

Chapter 3: Feature Engineering

Barring the use of news articles for natural language processing, the majority of reliably available data for stock prediction is a set of simple daily quotes consisting of **O**pen, **H**igh, **L**ow, and **C**lose prices, and the trading **V**olume for that day. From this data, we may engineer a set of features that include both well-known technical indicators, and some generic features that are applicable to any problem involving time series analysis.

Since we are solving a problem rooted in stock market forecasting, it makes sense for us to first look for features which are widely recognized in the technical trading community as telling indicators of a stock's motion. In particular, there are four indicators that are widely used in momentum trading which are then extended to comprise 12 out of the 28 features used to train the above models. These four indicators are as follows:

- **On-Balance Volume (OBV)**: This indicator is one of the earliest well-known technical analysis indicators, originating in Joseph Granville's 1976 publication *Granville's New Strategy of Daily Stock Market Timing for Maximum Profit* [4]. Granville's logic regarding the OBV indicator lies in the notion that a large change in trading volume without much of a change in the underlying asset's price is indicative of a sharp upcoming shift (upward or downward) in the price of the underlying asset. Denoting a stock's closing price in timestep t by C_t, the volume in timestep t by V_t, and the OBV in timestep t by OBV_t, we calculate the OBV indicator as follows:

$$OBV_t = OBV_{t-1} + \begin{cases} V_t, \text{ if } C_t > C_{t-1}, \\ 0, \text{ if } C_t = C_{t-1}, \\ -V_t, \text{ if } C_t < C_{t-1} \end{cases}.$$

- **Accumulation/Distribution Line (ADL)**: Developed by Marc Chaikin, the Accumulation/Distribution Line [5] is another popular volume-based technical indicator, which is calculated by factoring in the trading volume during a given time period, along with a weighting multiplier derived from a combination of the high, low, and close prices for the given time period. Denoting High, Low, Close, and Volume in timestep t as H_t, L_t, C_t, and V_t, respectively, we can calculate the ADL in timestep t (denoted ADL_t) as follows:

$$ADL_t = ADL_{t-1} + V_t \left(\frac{(C_t - L_t) - (H_t - C_t)}{H_t - L_t} \right).$$

- **Aroon Indicator:** The Aroon Indicator, developed by Tushar Chande in 1995 [6], is a popular price trend indicator that consists of a pair of oscillating values, known as the Aroon-Up (ArU) and Aroon-Down (ArD) indicators. When used as part of a by-hand trading strategy, the relationship between these two values is examined, so in order for these indicators to provide a machine learning system with useful information, it is reasonable to assume that they ought to both be included as features engineered from the dataset. The two Aroon indicators are calculated as follows:

$$ArU_t = \left(\frac{25 - [\#\text{ periods since 25-period high}]}{25} \right) \cdot 100,$$

$$ArD_t = \left(\frac{25 - [\#\text{ periods since 25-period low}]}{25} \right) \cdot 100,$$

where in this work, we consider each "period" to be a single day.

- **MACD:** The MACD (moving average convergence/divergence) indicator is a technical indicator created in the late 1970s by Gerald Appel [7]. The MACD is calculated as the difference between the 12-day *exponential moving average (EMA)* and the 26-day

exponential moving average of a stock's closing price, i.e. for a stock S, we have

$$MACD(S) = EMA_{12}(S) - EMA_{26}(S).$$

Additionally, we form a second feature from this, which is commonly known as the *signal line* - a 9-day exponential moving average of the MACD indicator itself. If we denote this by $SL(S)$, we calculate it as:

$$SL(S) = EMA_9\left(MACD(S)\right).$$

Here, the *k-period exponential moving average* is defined as a weighted average whose value int each timestep is computed using that timestep's value, and the EMA for the previous day, weighted by a small factor so that the value of the EMA is more dependent on recent values than past values. The formula by which we express the k-period EMA in timestep t for a temporal process x is

$$EMA_k(x_t) = (x_t - EMA_k(x_{t-1})) \cdot \left(\frac{2}{k+1}\right) + EMA_k(x_{t-1}).$$

Despite having access to all of these indicators, we still face one great problem that arises when training on a dataset that spans 10 years of daily quotes for a stock that, from the beginning to the end of this 10-year period, increases in value more than tenfold. We wish to take advantage of the possibility that similarities in stock movement data occurring at two potentially vastly different price points will indicate similar future motions, despite the underlying price at those points being vastly different.

However, it is not possible to obtain useful insights about stock motion just by looking at fixed price data/engineered features for a given day. In technical analysis, no trader looks

purely at the fixed price/indicator numbers for a given day - instead traders observe changes in both raw price information and indicators by looking at charts which display this data throughout time. Thus, we present a method for simulating this kind of observational ability for a machine, which takes into account changes in the value of a price marker or a derived indicator at the beginning and end of a k-day time period.

Def: We define the *k-day difference of stock S*, denoted $DIFF_k(S)$ to be the difference between the current value of the stock, S_t, and the value of the stock k days prior, (S_{t-k}), all divided by the length of the time window, k. Thus, we can express this as:

$$DIFF_k(S) = \frac{S_t - S_{t-k}}{k}.$$

What we may notice from this is that the k-day difference resembles an approximation of a first derivative, so one may create features resembling further and further derivative approximations of the underlying stock by repeating k-day difference operations (for potentially different values of k). In this work, we produce features from up to two applications of k-day differences (for different values of k).

We create new engineered features from both the raw stock data, and from the special indicators by taking differences corresponding to single trading weeks and approximate trading months (5-day and 25-day differences). Additionally, for the standard price information (Open, High, Low, Close), we can add a level of sophistication to the data arising from these raw data points by taking a 1-day difference of the 5-day and 25-day differences of these various prices. This application of a difference operation to a set of differences is like extracting the equivalent of the second derivative - showing how the k-day difference is changing in two consecutive time periods. This information could prove useful to the machine as a measure of momentum/change in momentum.

Now, we may perform a comparison between the performance of several different data configurations for the same network structures - we re-run the experiments for the network with 200 hidden units, 4 output units, and the Adam optimizer, and the network with 100 hidden units, 4 output units, and the RMSprop optimizer, with the full length-28 feature vector, length-12 feature vector involving only features related to famous indicators, and the length-16 feature vector which contained no information regarding famous indicators, only data obtained from applying k-day and then 1-day differences to the various daily price vectors.

Network Structure	**Input Features**	**End Accuracy**	**Top Accuracy**
100 Hidden, Adam	16	58.414	61.486
100 Hidden, Adam	12	53.161	58.333
100 Hidden, RMSprop	16	58.143	61.576
100 Hidden, RMSprop	12	53.252	57.215
100 Hidden, Adam	28	61.118	64.045
100 Hidden, RMSprop	28	60.783	64.329

Table 3: Results of multilayer perceptrons on subsets of input features with Adam and RMSprop optimizers

We can easily see that the network version that uses the 16 custom inputs has significantly higher accuracy than the network version that uses the 12 inputs arising from famous technical indicators. This is likely due to the fact that there is more information encoded in the 16 variables that make up the custom features than there is in the 12 technical indicator variables, both in terms of price fluctuations, and in terms of local intra-day volatility (obtained from information about each day's high and low prices). Ultimately, however, it is clear that including all 28 features is the best option, as the accuracy is once again considerably higher for all 28 features than it is for only the 16 custom features.

Chapter 4: Trading Strategy

While testing the prediction accuracy of the system is certainly a good way to gain some basic insight into the true performance of the system, nothing can test this accuracy better than applying the prediction results of the system through a trading strategy, and evaluating the strategy's performance directly.

We propose the following two strategies that will be tested and evaluated on real market data via backtesting. For the purposes of using the predictions made by the algorithm, all option spreads are assumed to mature one timestep after they are purchased.

1. When the algorithm predicts that the price of the underlying asset will move (in either direction) by more than the cost of the straddle option spread, buy a fixed number of straddles (assume 1 for simplicity), and collect the profit (or loss) that results at the options' expiry.

2. When the algorithm predicts that the price of the underlying asset will move by more than the cost of the straddle, buy a number of straddles according to the following formula:

$$\#[\text{shares purchased}] = \frac{D_t + 1}{P_{\text{exp}}},$$

where the quantities D_t and P_{exp} denote what we call the "round-deficit at time t" and the expected profit conditional on having made a good trade, respectively. These quantities are discussed in further detail below.

We define the following quantities, which we will use in our analysis of both strategies:

- We denote by D_t the *round deficit* at time t, i.e. the total money lost in a sequence of bad trades that has occurred since the last good trade.

- P_{exp} denotes the *expected profit* per-straddle-share conditional on having made a good trade.

- L_{exp} denotes the *expected loss* per-straddle-share conditional on having made a bad trade (this quantity is positive, so we measure a monetary loss by adding a negative sign when actually calculating profits).

- A_m denotes the *accuracy* of the algorithm when predicting that the price of the underlying will move (in either direction) more than the cost of the straddle option spread for that asset. This quantity can only be obtained experimentally.

Strategy 1 described above is the trivial strategy, and does not merit much discussion, as the success of the algorithm as measured by Strategy 1 will always have the same expected return in every timestep, and thus, the expected value of the strategy as time goes to infinity will be fully determined by the sign of the expected value in a single timestep. This expectation can be expressed (for any timestep t) as:

$$\mathbb{E}_t\left[\text{Strategy 1}\right] = A_m \cdot P_{\text{exp}} - (1 - A_m) \cdot L_{\text{exp}},$$

where A denotes the probability that the algorithm makes a correct prediction, conditional on it having predicted that the underlying asset of the straddle would move more than the straddle price, P_{exp} denotes expected profit (conditional on having made a good trade), and L_{exp} denotes expected loss (conditional on having made a bad trade).

Strategy 2, on the other hand, is designed to provide certain guarantees regarding "eventual" profitability, but comes with some large potential risks. This strategy results from a modification of the *martingale* betting strategy, in which one bets on a game with an outcome in $\{-1, +1\}$, and always doubles their bet after a loss. Due to this doubling of bets, a

string of k losses would result in a deficit of $2^k - 1$, but then winning on the $(k+1)$-th turn would eliminate the entire deficit, and incur a profit of 1. One notable issue with martingale betting is that if the probability of winning is not high, it is more likely than one would expect to incur catastrophic losses by losing many times in a row. However, if it is possible to increase the probability of victory, then the loss that one is liable to incur before the first win becomes, in expectation, exponentially lower.

Strategy 2 emulates the martingale betting strategy in the sense that it is designed to, in expectation, eliminate all current losses, and incur an additional profit of $+1$ upon the trader having made a good trade. The desire for this expected effect is reflected in the trading formula itself, where the numerator, $D_t + 1$, represents the cumulative deficit incurred since the last good trade (which we call the "*round deficit*"), plus one, and the denominator represents the expected profit per-straddle. The numerator is exactly the amount of money that needs to be made in the next trade in order to clear the current round deficit, and incur an additional profit of 1. Assuming an expected P_{exp} amount of profit per-straddle, conditional on having made a good trade in timestep t, the expected payoff in that round will trivially be

$$\frac{D_t + 1}{P_{\text{exp}}} \cdot P_{\text{exp}} = D_t + 1,$$

exactly the amount that we wish to obtain on a good trade. Now, since we are assuming that predictions are made according to our algorithm, which will have a certain level of accuracy, we must be able to calculate how much deficit Strategy 2 is expected to incur before its first good trade, on any given "round" consisting of a series of losses (bad trades) followed by a single good trade.

Now, when calculating the expectation of the deficit incurred in a given round prior to the first good trade, we notice that for timestep t, the deficit, upon which the quantity of shares that will be bought depends, is based on all of the previous deficits incurred, via the trading

formula. We will show the first several steps of how the expected deficit in any given timestep is calculated, and will extrapolate from that a formula for the expected deficit in any given round.

In the first step of any round (which we can denote by $t_1^{(r)}$), the formula for the number of straddles bought gives $\frac{D_{t_1^{(r)}} + 1}{P_{\text{exp}}} = \frac{1}{P_{\text{exp}}}$, since $D_{t_1^{(r)}} = 0$. If this resulted in a bad trade, then in expectation, we would incur a loss of L_{exp} per straddle bought, resulting in a current deficit of $\frac{L_{\text{exp}}}{P_{\text{exp}}}$.

If the second step of a round is reached, that means that a bad trade was made in the first step, so we have an expected deficit of $D_{t_2^{(r)}} = \frac{L_{\text{exp}}}{P_{\text{exp}}}$. Substituting this into the trading formula, we buy

$$\frac{\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1}{P_{\text{exp}}} = \frac{\frac{L_{\text{exp}} + P_{\text{exp}}}{P_{\text{exp}}}}{P_{\text{exp}}} = \frac{L_{\text{exp}} + P_{\text{exp}}}{P_{\text{exp}}^2}$$

shares in round 2. Then, if this results in a bad trade, we incur an additional expected loss of

$$L_{\text{exp}} \cdot \left(\frac{L_{\text{exp}} + P_{\text{exp}}}{P_{\text{exp}}^2} \right) = \frac{L_{\text{exp}}^2 + L_{\text{exp}} P_{\text{exp}}}{P_{\text{exp}}^2}.$$

Since we still need to factor in our original deficit to get the total deficit up to this point, we get that the new total deficit is

$$\frac{L_{\text{exp}}}{P_{\text{exp}}} + \frac{L_{\text{exp}}^2 + L_{\text{exp}} P_{\text{exp}}}{P_{\text{exp}}^2} = \frac{L_{\text{exp}}^2 + 2L_{\text{exp}} P_{\text{exp}}}{P_{\text{exp}}^2}.$$

Repeating the above process for the third step of a given round, the trading formula yields the purchased share quantity $\frac{L_{\text{exp}}^2 + 2L_{\text{exp}} P_{\text{exp}} + P_{\text{exp}}^2}{P_{\text{exp}}^3}$, and thus if this step also contains a bad trade, results in an additional loss of

$$\frac{L_{\text{exp}}^3 + 2L_{\text{exp}}^2 P_{\text{exp}} + L_{\text{exp}} P_{\text{exp}}^2}{P_{\text{exp}}^3}.$$

Adding in the total deficit from the end of step 2, we get that the total expected deficit after 3 consecutive losses is

$$\frac{L_{\text{exp}}^2 + 2L_{\text{exp}}P_{\text{exp}}}{P_{\text{exp}}^2} + \frac{L_{\text{exp}}^3 + 2L_{\text{exp}}^2P_{\text{exp}} + L_{\text{exp}}P_{\text{exp}}^2}{P_{\text{exp}}^3} = \frac{L_{\text{exp}}^3 + 3L_{\text{exp}}^2P_{\text{exp}} + 3L_{\text{exp}}P_{\text{exp}}^2}{P_{\text{exp}}^3}.$$

Repeating the above steps many times, we see that from a sequence of n bad trades, the total expected deficit incurred after the n^{th} bad trade is equal to

$$\frac{(L_{\text{exp}} + P_{\text{exp}})^n - P_{\text{exp}}^n}{P_{\text{exp}}^n} = \frac{(L_{\text{exp}} + P_{\text{exp}})^n}{P_{\text{exp}}^n} - 1 = \left(\frac{L_{\text{exp}} + P_{\text{exp}}}{P_{\text{exp}}}\right)^n - 1 = \left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)^n - 1.$$

From this, we get that the expected deficit incurred in any given round can be expressed as

$$\mathbb{E}\left[D^{(r)}\right] = \left(\frac{1 - A_m}{A_m}\right) \cdot \sum_{i=1}^{\infty} \left((1 - A_m)^i \cdot \left[\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)^i - 1\right]\right)$$

Here, we have normalized the infinite sum by $\frac{1-A}{A}$, since otherwise, the probabilities for the various events would not add up to 1.

Now that we have an expression for the expected deficit of a round, we can derive an elegant convergence condition, under which we have a guarantee that the expected deficit for a round will be finite. We can see that we can split the summation term in our expected deficit formula, so that the formula becomes

$$\left(\frac{1 - A_m}{A_m}\right) \cdot \left[\sum_{i=1}^{\infty} \left[(1 - A_m)\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)\right]^i - \sum_{i=1}^{\infty} (1 - A_m)^i\right] =$$

$$= \left(\frac{1 - A_m}{A_m}\right) \cdot \left[\sum_{i=1}^{\infty} \left[(1 - A_m)\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)\right]^i - \left(\frac{1 - A_m}{A_m}\right)\right] =$$

$$= \left(\frac{1 - A_m}{A_m}\right) \cdot \left[\left(\sum_{i=0}^{\infty} \left[(1 - A_m)\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)\right]^i - 1\right) - \left(\frac{1 - A_m}{A_m}\right)\right] =$$

$$= \left(\frac{1 - A_m}{A_m}\right) \cdot \left[\left(\frac{1}{1 - (1 - A_m)\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)}\right) - 1 - \left(\frac{1 - A_m}{A_m}\right)\right] =$$

$$= \boxed{\left(\frac{1 - A_m}{A_m}\right) \cdot \left[\left(\frac{1}{1 - (1 - A_m)\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)}\right) - \left(\frac{1}{A_m}\right)\right]}.$$

From this final closed-form expression, we can see that the condition for which the expression converges depends on the quantity $\left(1 - (1 - A_m)\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right)\right)$. More specifically, we want

$$(1 - A_m)\left(\frac{L_{\text{exp}}}{P_{\text{exp}}} + 1\right) < 1 \implies$$

$$\implies \frac{L_{\text{exp}}}{P_{\text{exp}}} + 1 < \frac{1}{1 - A_m} \implies \frac{L_{\text{exp}}}{P_{\text{exp}}} < \frac{1}{1 - A_m} - 1 \implies$$

$$\implies \boxed{\frac{L_{\text{exp}}}{P_{\text{exp}}} < \frac{A_m}{1 - A_m}}.$$

We will want to experimentally verify whether our algorithm is capable of attaining the convergence condition, since we cannot give theoretical values for L_{exp}, P_{exp}, and A_m. In order to do this, we will want to extract these values from a trained algorithm, and will evaluate the expected deficit convergence condition on a per-algorithm basis.

In an idealized scenario, the values of L_{exp}, P_{exp}, and A_m would not vary over time, but since the desired algorithm will need to be retrained periodically to account for new information, the desired values will likely vary slightly over time. We will later explore methods to guarantee a more precise estimate of the three desired values that will vary less with time.

Chapter 5: Testing and Results

5.1 - Realistic Option Prices

Now, we will move on to the more difficult, but ultimate goal of this work - to analyze the accuracy of our algorithms with respect to actual option prices. To achieve this, however, we need a way to simulate historical option prices, since real historical options data is prohibitively expensive to obtain. For this data, we will use the Black-Scholes option pricing model [8], with implied volatility estimated from prior data using the GARCH$(1, 1)$ model [9, 10].

Generally, the Black-Scholes model is used to model present option prices, and many online tools contain their own estimators/trackers of the implied volatility of the market/of individual stocks. However, these are only available for the given day on which one visits the tool to obtain an option price estimate, so if one wishes to obtain historical options data from many years ago, one must find a way to realistically model the volatility surfaces for assets of interest, as those are needed to be able to accurately apply the Black-Scholes model to historical data.

In order to more realistically model historical volatility surfaces, we will use the GARCH model [9] with parameters $p = 1, q = 1$ (GARCH$(1, 1)$), as it is recognized as reliable way to improve the modeling of volatility in stocks. The GARCH model with parameters $(1, 1)$ has also been shown in particular to perform better than GARCH with other parameter values on stock market samples [10], so this will be the particular model that we proceed with when computing historical volatility.

The Black-Scholes formula at time t for calculating the price of a call option with strike price K for an underlying asset with price S_t, and maturity T is

$$S_t \cdot \mathcal{N}(d_1) - Ke^{r(t-T)} \cdot \mathcal{N}(d_2),$$

where r represents the *risk-free rate* (usually taken to be the interest rate on treasury bonds), and the quantities d_1 and d_2 are defined as follows:

$$d_1 = \frac{\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{1}{2}v^2\right)(T-t)}{v\sqrt{T-t}},$$

and

$$d_2 = \frac{\ln\left(\frac{S_t}{K}\right) + \left(r - \frac{1}{2}v^2\right)(T-t)}{v\sqrt{T-t}},$$

where $\mathcal{N}$ represents the standard normal cumulative distribution function (CDF).
The quantity of interest in the GARCH$(1,1)$ computations is the value of v, the standard deviation of the underlying asset. In accurate Black-Scholes models, v^2 is generally taken to be the *implied volatility* of the underlying, an estimate of future volatility, and not a simple calculation of the variance of the asset's returns over a pre-specified number of periods prior to the current timestep.

For the value of r, the *risk-free rate*, we will use daily values for 10-year US treasury bond yields, and since we are looking at options whose strike price equals their current price, we have $\frac{S_t}{K} = 1$, which gives $\ln\left(\frac{S_t}{K}\right) = 0$, and thus, our values for d_1 and d_2 become

$$d_1 = \frac{\left(r + \frac{1}{2}v^2\right)(T-t)}{v\sqrt{T-t}},$$

and

$$d_2 = \frac{\left(r - \frac{1}{2}v^2\right)(T-t)}{v\sqrt{T-t}}.$$

5.2 - Basic Testing Framework

We will begin by testing the same networks that were determined to be of sufficient quality in the previous section - the networks with 100 neurons in a single hidden layer - with the slight modification that the networks now accept data points with 33 input features. The 5 extra input features are normalized returns for the Open, High, Low, and Close prices of the S&P 500 market index, plus each day's straddle cost. These features are provided with the aim of giving the network context as to the motion of the given stock with respect to the market, and as to the amount of motion in the given stock that would be necessary in order to predict the class label corresponding to "more" motion than the cost of the associated straddle option spread.

In this testing framework, the values of interest are $\frac{L_{\text{exp}}}{P_{\text{exp}}}$, $\frac{A_m}{1-A_m}$, $\max_t\{D_t\}$, and both the maximum and final amounts of money earned in a given trial. We will average these quantities over 20 different initializations of the same network structure, across 10 different assets in order to both test the ability of the basic system to produce good predictions, and the ability of the trading strategy to generate significant profit.

The quantities L_{exp} and P_{exp} will be experimentally generated using the test data, as will the quantity A_m. Note that A_m will differ from the overall accuracy of the algorithm, as A_m measures the accuracy of the algorithm conditional on it having predicted a motion of the underlying asset that is *greater* than the cost of the straddle option spread. Here, we will compare the difference between the averages of $\frac{L_{\text{exp}}}{P_{\text{exp}}}$ and $\frac{A_m}{1-A_m}$ to the ratio of the averages of $\max_t\{D_t\}$ and the final amount of money earned, and will establish whether there exists a relationship between the two quantities. L_{exp} and P_{exp}, as well as A_m will be computed from the same test data that $\max_t\{D_t\}$ and the final earnings are computed from, though for future testing, when attempting to improve the final earnings and decrease the max deficit,

L_{exp}, P_{exp}, and A_m will be computed from data that is recent to each data point for which predictions and an investment decision are being made, but do not include that new data point, as ultimately, L_{exp}, P_{exp}, and A_m will be used in order to drive decisions regarding whether or not to invest during a given time period.

It should also be noted that in this setup, the value of P_{exp} used in the trading formula $\#[\text{straddles}] = \frac{D_t+1}{P_{\text{exp}}}$ is a value that in practice would not be known exactly, and thus, it is not realistic to base all expectations of future algorithm and trading strategy performance on the results in this section. The primary aim here is to establish a relationship between the difference of $\frac{L_{\text{exp}}}{P_{\text{exp}}}$ and $\frac{A_m}{1-A_m}$, and the ratio of final earnings to $\max_t\{D_t\}$.

Symbol	$\frac{L_{\text{exp}}}{P_{\text{exp}}}$	$\frac{A_m}{1-A_m}$	E_f	$D_{\max}$
AAPL (Apple)	0.439	1.002	59.846	4.5
IBM (IBM)	0.468	0.593	29.291	6.98
MSFT (Microsoft)	0.276	2.547	18.666	0.699
INTC (Intel)	0.289	1.196	56.402	1.061
CSCO (Cisco)	0.238	0.988	46.637	0.57
QCOM (Qualcomm)	0.311	1.273	100.214	1.588
WMT (WalMart)	0.386	0.88	36.458	2.571
GE (General Electric)	0.361	1.49	94.51	0.601
GS (Goldman Sachs)	0.233	1.524	52.351	3.151
AXP (American Express)	0.216	0.773	26.871	2.028

Table 4: Basic investment results for multilayer network with 100 hidden units, Adam

We define the quantities E_f and $D_{\max}$ to be the final earnings and max deficit, respectively, of a given test averaged over 20 trials. For 10 different stock symbols, ranging across a several sectors, we have the following results for the desired quantities for a network with

33 inputs, 100 hidden nodes, and 2 output classes, trained with the Adam optimizer, with learning rate 0.0001 and weight decay 0.001. Results are presented in Table 4.

We also have the following results in Table 5 for the same network setup, but trained with the RMSprop optimizer, with the same learning rate and weight decay factor.

Symbol	$\frac{L_{\text{exp}}}{P_{\text{exp}}}$	$\frac{A_m}{1-A_m}$	E_f	$D_{\max}$
AAPL (Apple)	0.439	1.007	58.98	4.686
IBM (IBM)	0.459	0.592	35.41	6.596
MSFT (Microsoft)	0.279	2.74	17.746	0.602
INTC (Intel)	0.281	1.261	59.379	1.037
CSCO (Cisco)	0.235	1.042	47.232	0.544
QCOM (Qualcomm)	0.320	1.294	96.333	1.605
WMT (WalMart)	0.379	0.883	35.884	2.485
GE (General Electric)	0.353	1.529	94.321	0.534
GS (Goldman Sachs)	0.232	1.563	54.084	2.896
AXP (American Express)	0.214	0.778	26.86	1.823

Table 4: Basic investment results for multilayer network with 100 hidden units, RMSprop

From these results it is not certain whether it would be better to train exclusively with Adam or RMSprop - for some assets, RMSprop outperforms Adam on both E_f and $D_{\max}$, but on others, it does not outperform Adam on both quantities, and on some, it underperforms Adam on both. From this data, we can attempt to model the relationship between $\left(\frac{A_m}{1-A_m}\right) - \left(\frac{L_{\text{exp}}}{P_{\text{exp}}}\right)$ and $E_f - D_{\max}$. The model below is a simple linear regression on the data obtained above, excluding the data points corresponding to MSFT stock, as those are outliers that adversely affect the regression's ability to model the majority of the data.

The R^2 coefficient in the model which is plotted below is 0.44, which indicates a moderate

positive correlation between the variables. As we can see from the plot, however, the variance about the regression line increases as the value of $\left(\frac{A_m}{1-A_m}\right) - \left(\frac{L_{\text{exp}}}{P_{\text{exp}}}\right)$ increases. A potential solution for this may be to record the same figures over a larger amount of time, thus allowing for longer-term accumulation of profits, and longer-term tracking of maximal deficits.

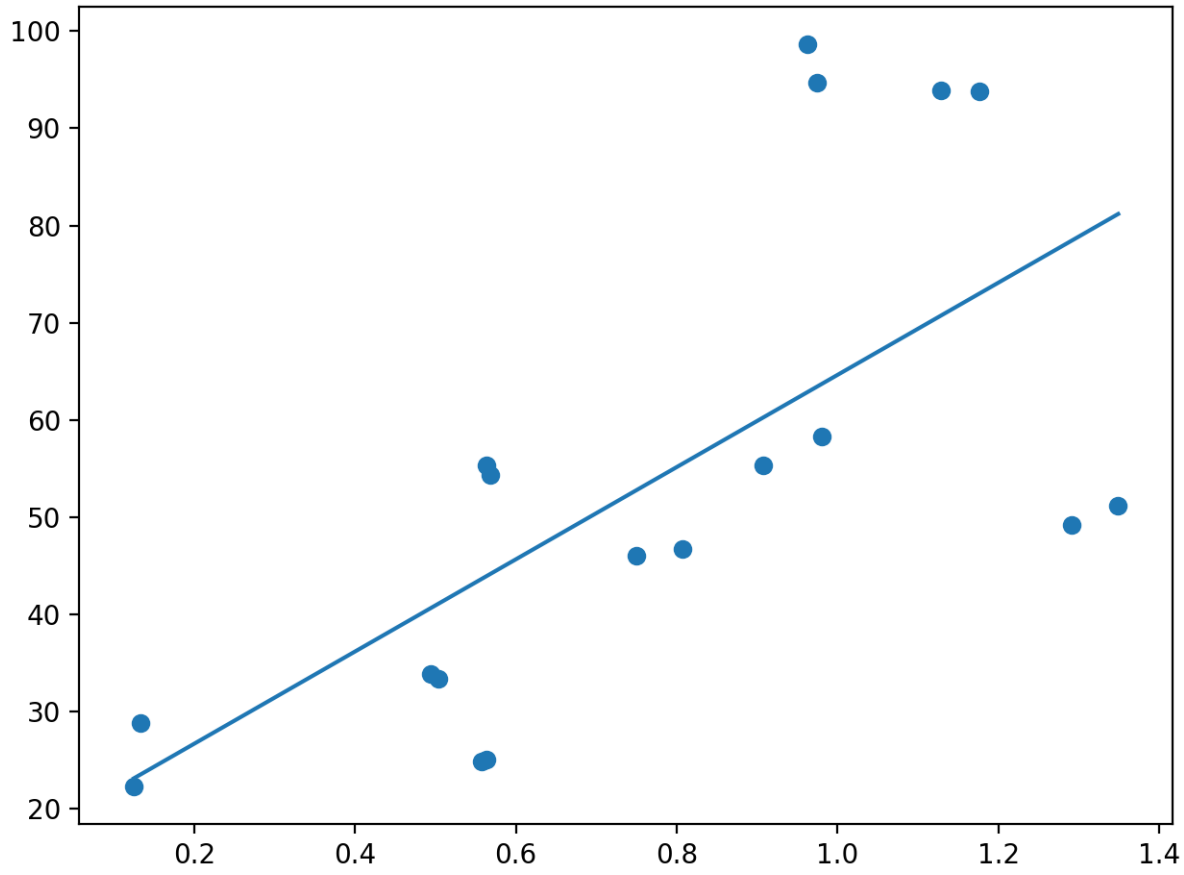


Figure 2: Plot of $E_f - D_{\max}$ against $\left(\frac{A_m}{1-A_m}\right) - \left(\frac{L_{\text{exp}}}{P_{\text{exp}}}\right)$.

The y-intercept of this linear regression is 17.211, which indicates an interesting potential conclusion - even for the boundary at which the model's expected round-deficit theoretically does not converge, this linear model predicts that the difference between the end-of-trial profit and the max deficit will still be positive. For additional robustness of this type of modeling, it may be useful to collect data for an even larger number of stocks, and the more

sophisticated testing frameworks that follow will test on a larger range of stock symbols, and across a larger timespan.

5.3 - Advanced Testing

In order to perform more sophisticated testing of this model, we must now establish a method by which to estimate P_{exp} and L_{exp} from data that the model has only seen up to that point. In the basic testing framework, P_{exp} and L_{exp} are taken to be the precomputed average values for those quantities based on the actual profits and losses of the system given predictions on the entire test set. However, in reality, this future data will not be available to the system, and judgements about A_m, L_{exp}, and P_{exp} must be made given only validation data or real historical predictions on data points that have already been observed.

We will introduce several tools with the goal of both improving the accuracy of the model, and making good predictions regarding future values of P_{exp} and L_{exp}. First, we will explore a method by which we can improve overall predictive accuracy, and by which we can also more realistically model a day-by-day investing scenario in which the model is able to take into account the most recent data before making a prediction for the next day.

5.3.1 - Array-of-Networks Model

Though the results so far show promise, it is necessary to also consider the fact that these results are not generated in a way that is realistic in terms of how real-time market predictions would be made. The above systems were trained on 4/5 of a dataset spanning 10 years of a single stock, and test data consisted of the most recent 1/5 of the data, so although test

data was temporally properly aligned with the training data, the fact remains that it is not realistic for one to predict hundreds of days into the future, and measure the accuracy of a system based on how well it predicts for those far-off days without having ever trained on data corresponding to days which more closely precede those far-off days.

To address this problem, we consider a system consisting of an array of neural networks, each with the same structure, and each of which trains on a large, timewise-contiguous portion of the data, with each network's only test data being the single data point corresponding to the date immediately following the last data point in its training set. Thus, for an array of n networks, we effectively generate a test set of size n, and each network trains on $N - n$ data points, where N is the size of the complete dataset.

This new method is aimed at establishing confidence regarding a network structure's expected ability to predict accurately for one unseen data point, which is the data point corresponding to the single next time period, given that it has trained on a large amount data corresponding to a large number of time periods into the past, assuming that the network is completely retrained each day, without any mind paid to optimizing parameter initialization. Additionally, with this method, we are able to easily obtain figures for A_m, L_{exp}, and P_{exp} for the data points leading up to the unseen date that we must make a prediction for. We can select any range of days into the past from which we will compute these quantities, and additionally, for every next day, we can trivially add on to our running list of A_m, L_{exp}, and P_{exp} values.

In the upcoming tests, we will train each network on a range of contiguous data points spanning 3/5 of the length of the full dataset, as it is likely detrimental to the performance of the algorithm to include data points that are too far in the past, and could exhibit significantly different behavior from data points in the recent past, which we assume that new data points are more similar to. An added benefit of this is that we now have an

additional 1/5 of the data set to use as test data when attempting to establish a model for the relationship between $\left(\frac{A_m}{1-A_m}\right) - \left(\frac{L_{\text{exp}}}{P_{\text{exp}}}\right)$ and $E_f - D_{\max}$.

For our initial simulations, we will train the same network as before (33 input features, 100 hidden units, 2 output classes, learning rate 0.0001, weight decay 0.001), but only using the Adam optimizer, as there was no clear distinction between the performance of the Adam-trained network and the RMSprop-trained network. One distinction to note is that these networks will be trained for only 15000 iterations, and with a batch size of 100, as these new hyperparameters produce a performance increase of up to several percent in test accuracy. The greater batch size also reduces the variance of the network parameter updates sufficiently that the test performance also reaches a suitable level of convergence far earlier, and maintains that level for enough iterations so that we are more certain we have not overfit to the training data when we perform early stopping.

To calculate A_m from the past data, we will average the accuracy over the last 200 single-day iterations of network predictions in which the network predicted motion of the underlying asset greater than the cost of its associated straddle option spread. To predict L_{exp} and P_{exp}, however, we will take the sequences of losses given bad trades and profits given good trades, respectively, and compute 10-step exponential moving averages starting from 100 points in the past corresponding to "greater-than-straddle-price" motion predictions. If for either good trades or bad trades there are not enough data points within the 200-day range, we will take the corresponding data points within the 100-day range that are present, and will compute a simple average over those points to obtain the expected value of the corresponding quantity. We will begin gathering performance data 200 iterations into the process, which leaves more than 1/5 of the remaining data as testable data on which we can make predictions.

5.3.2 - Two Testing Regimes

In order to test the outputs of this data, we must only make trades based on the P_{exp} that is computable from already seen data, and cannot make the naïve assumption that P_{exp} is approximately constant through time (which we made in the basic testing framework, when we first computed the P_{exp}, L_{exp}, and A_m from test data, and then held those constant throughout the testing of the trading strategy).

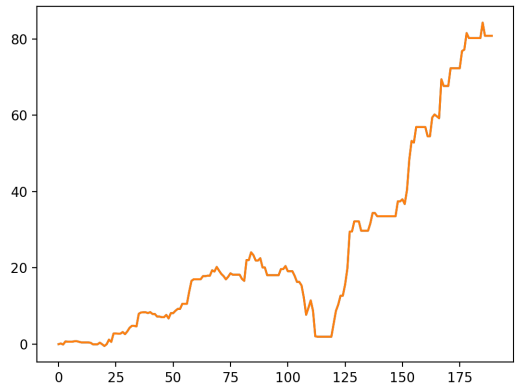
In the basic testing framework, we traded based on this naïve value of P_{exp}, and computed the expected round-deficit using the naïve values of A_m, L_{exp}, and P_{exp}. However, in practice, it may be practical to use the predicted values of A_m, L_{exp}, and P_{exp} in order to establish whether or not a trade would be worth making, in expectation. Recall that the convergence condition for the expected round-deficit is

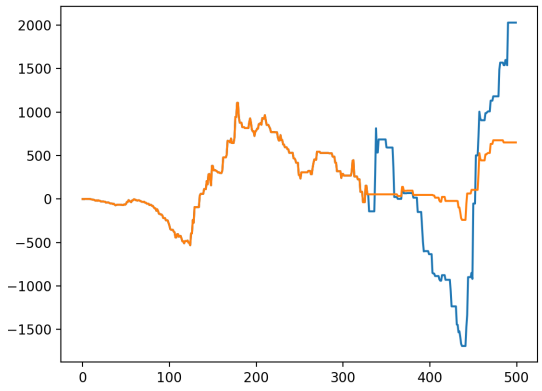
$$\frac{L_{\text{exp}}}{P_{\text{exp}}} < \frac{A_m}{1 - A_m}.$$

Since we have predicted values in each timestep for all three basic components of this condition, we can compute for every new trading day whether or not it would make sense, according to this condition, to place a trade that day. However, since it is statistically highly unlikely that many bad trades in a row would occur, even in the case where the expected round-deficit does not converge, it could still be profitable to trade whenever the algorithm produces a prediction that the underlying asset will move more in price than its associated straddle option. As a matter of fact, if good predictions were to be made during a time period when the expected round-deficit was divergent, the trade-limiting strategy would lose the ability to reap the profits that could have resulted from those good predictions. Conversely, however, the trade-limiting strategy could also prevent more bad trades from occurring during this time period, and thus reduce the maximum round-deficit.

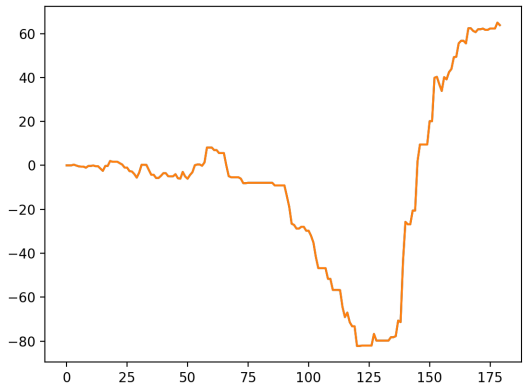
Thus, it becomes of interest to us to study the effects of both trading regimes - regime α, under which trades are always placed when the algorithm predicts a large motion of the underlying asset, and regime β, un which trades are only placed when $\frac{L_{\text{exp}}}{P_{\text{exp}}} < \frac{A_m}{1-A_m}$, based on the predicted values for A_m, L_{exp}, and P_{exp}.

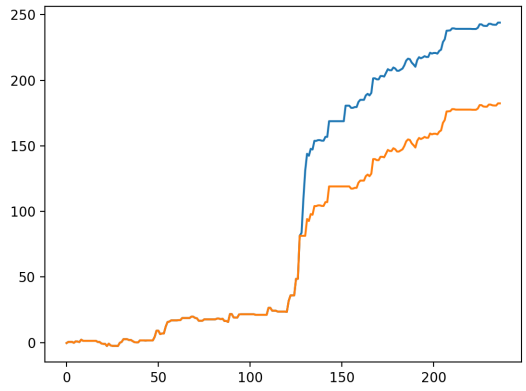
In the following graphs, we plot, for a sequence of trades occurring in the past 4 years, the earnings of regime α in blue, and the earnings of regime β in orange, for the 10 assets considered in the previous chapter. (Note that if only an orange earnings curve is present on a given graph, then regime β produced equivalent trades to regime α). These simulations were performed using a pseudo-array-of-networks model in which each network in the array was responsible for predicting 100 data points past the end of its training set. This was done (as opposed to a pure array-of-networks model) due to prohibitively long runtimes of the pure array-of-networks model (on the order of 100+ hours).



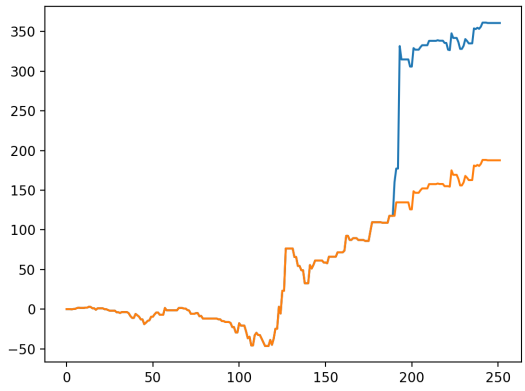


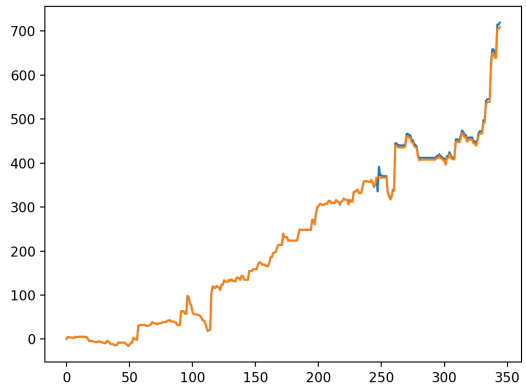
Left: Figure 3: Plot of regimes α and β for AAPL
Right: Figure 4: Plot of regimes α and β for IBM



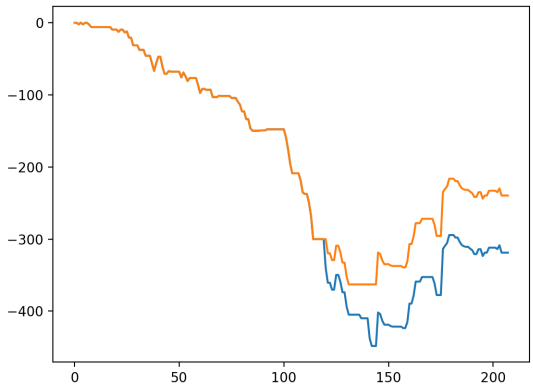


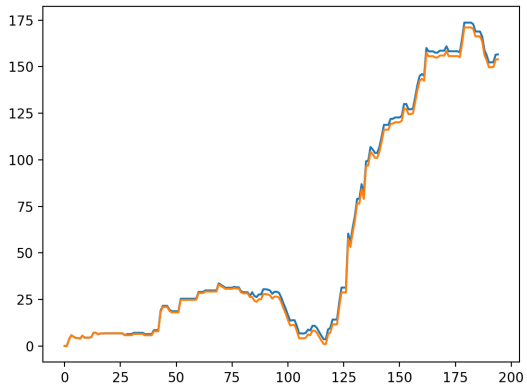
Left: Figure 5: Plot of regimes α and β for MSFT
Right: Figure 6: Plot of regimes α and β for INTC



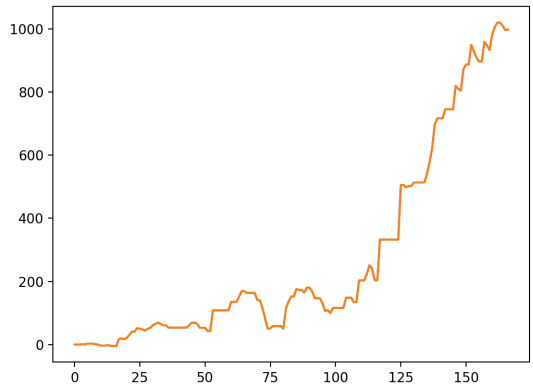


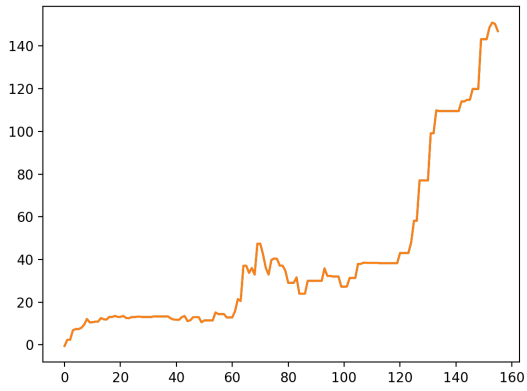
Left: Figure 7: Plot of regimes α and β for CSCO
Right: Figure 8: Plot of regimes α and β for QCOM





Left: Figure 9: Plot of regimes α and β for WMT
Right: Figure 10: Plot of regimes α and β for GE



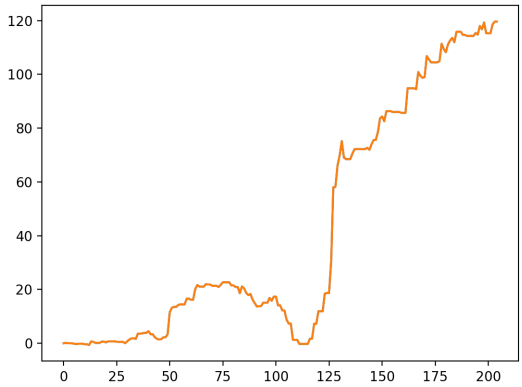


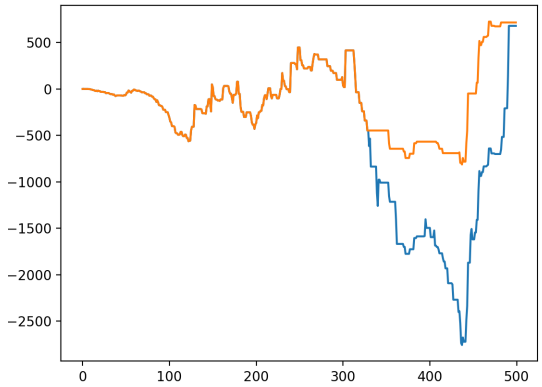
Left: Figure 11: Plot of regimes α and β for GS
Right: Figure 12: Plot of regimes α and β for AXP

One thing that is immediately apparent from observing these graphs is that whatever the overall trend of the earnings curve may be, if there exists a difference between the regime α and regime β curves, the regime α curve will have at least slightly amplified motion of its earnings curve as compared to that of regime β, whether the earnings are moving upward or downward. This corroborates the hypothesis presented above that regime β can both reduce losses but may also inadvertently limit profits. For example, regime β reduces the

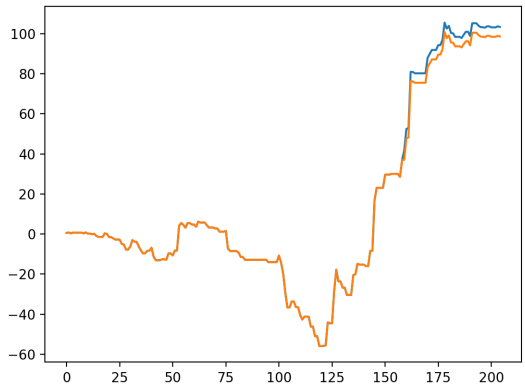
losses incurred in the WMT simulation, but it also limits the profits obtained in the CSCO simulation.

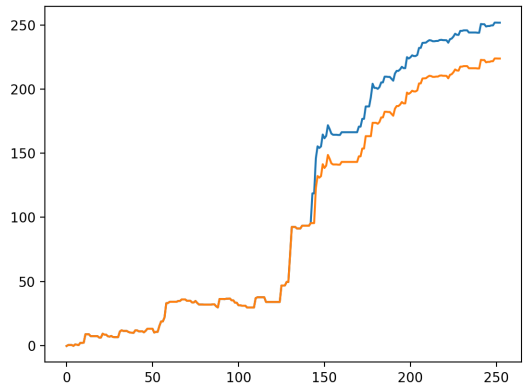
Another clear observation which we draw from these graphs is that there is a period roughly situated between 1/2 to 3/4 of the way into each series of trades during which the algorithm experienced losses for almost every stock, even the ones for which the overall earnings ended up very positive. Next, we will present graphs of simulation results for pseudo-array-of-networks simulations in which each network was responsible for predicting 50 days ahead of the end of its training set.



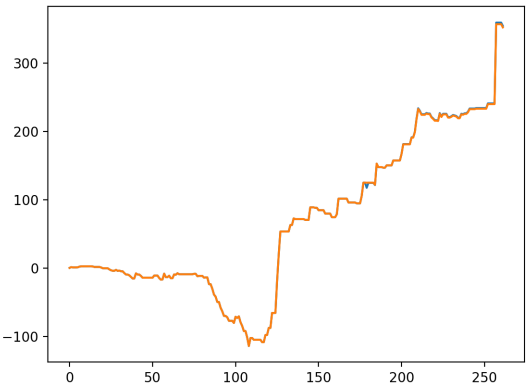


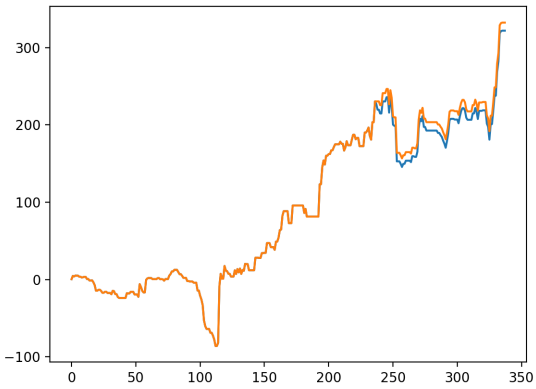
Left: Figure 13: Plot of regimes α and β for AAPL
Right: Figure 14: Plot of regimes α and β for IBM



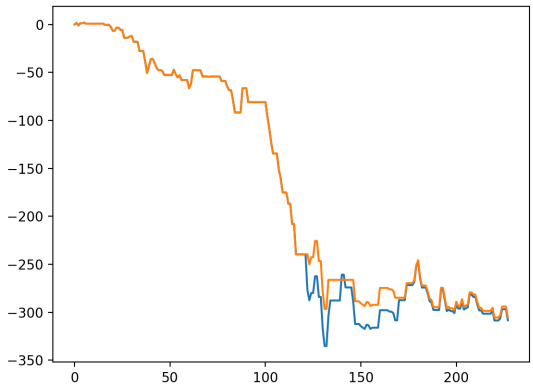


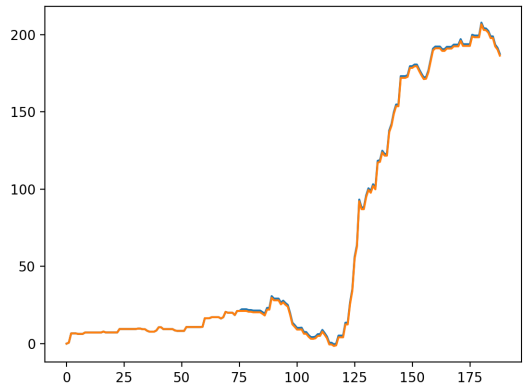
Left: Figure 15: Plot of regimes α and β for MSFT
Right: Figure 16: Plot of regimes α and β for INTC



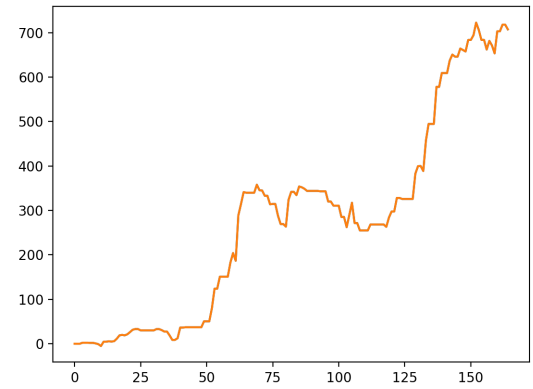


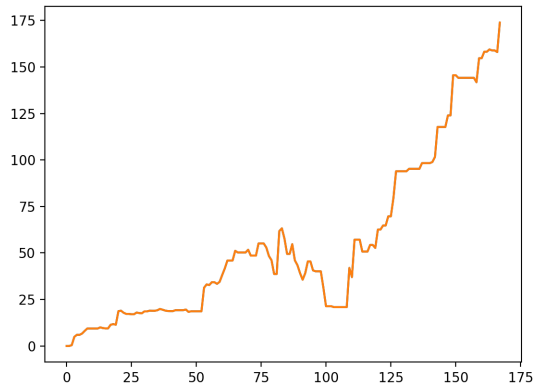
Left: Figure 17: Plot of regimes α and β for CSCO
Right: Figure 18: Plot of regimes α and β for QCOM





Left: Figure 19: Plot of regimes α and β for WMT
Right: Figure 20: Plot of regimes α and β for GE





Left: Figure 21: Plot of regimes α and β for GS
Right: Figure 22: Plot of regimes α and β for AXP

From this set of graphs, it is not apparent that there is any conclusive benefit to using the 50-day pseudo-array-of-networks model rather than the 100-day alternative. It is likely that due to random initializations of network parameters, significant differences may even arise between the performances of the same network structure, under the same array-of-networks configuration, for different trials. Thus, in order to augment the performance of the array-of-networks model for smaller future prediction timeframes, it is likely necessary to fine-tune additional hyperparameters which are outside of the scope of this work.

Chapter 6: Conclusions and Future Work

Although the models explored in this work may not produce favorable investment outcomes on all stocks, the results of the tests we have performed show that for a majority of the stocks tested, which have been selected from a variety of market sectors, the models produce consistent profits, and rarely, if ever, incur negative earnings. It is important to note that the results produced by the models in this work arise from relatively unsophisticated machine learning systems, and that employing more powerful algorithms, along with much more comprehensive datasets, would likely result in a model with higher accuracy, and thus, likely significantly higher earnings.

In future work, one way to potentially combat the adverse effects of a bad initialization of a neural network may be to run, in parallel, a set of k neural networks, and then select from the network predictions the one which the majority (more than $\frac{k}{2}$) of the networks produced. This is a simple consensus algorithm which may provide improvements over running just a single network for each prediction or set of predictions, as bad initializations with incorrect predictions for certain normally easily predictable data points would be outvoted by the majority of networks which produced good predictions. As always, it is necessary to also be cognizant of the possibility that some network initializations may provide good predictions when the majority would provide bad predictions, so this tradeoff must be experimentally explored.

Future attempts to create better predictors will also involve taking into account time series data using recurrent neural networks, and possibly selecting for optimal network structure hyperparameters via a grid-search or evolutionary algorithm.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, p. 23, 2013.

[4] J. E. Granville, *Granville's New Strategy of Daily Stock Market Timing for Maximum Profit*. Prentice-Hall, 1976.

[5] K. S. Kannan, P. S. Sekar, M. M. Sathik, and P. Arumugam, "Financial stock market forecast using data mining techniques," in *Proceedings of the International Multiconference of Engineers and computer scientists*, vol. 1, p. 2, Citeseer, 2010.

[6] T. S. Chande and S. Kroll, *The new technical trader: boost your profit by plugging into the latest indicators*. John Wiley & Sons Inc, 1994.

[7] G. Appel, *Technical analysis: power tools for active investors*. FT Press, 2005.

[8] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of political economy*, vol. 81, no. 3, pp. 637–654, 1973.

[9] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of econometrics*, vol. 31, no. 3, pp. 307–327, 1986.

[10] M. Miah and A. Rahman, "Modelling volatility of daily stock returns: Is garch (1, 1) enough?," *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, vol. 18, no. 1, pp. 29–39, 2016.