Self-Evolving Machine Learning Algorithm for Stock Market Trading Implemented on an FPGA

Front End Dashboard: https://quiet-wave-30496.herokuapp.com/admin/dashboard

D. Bidlack , M. Chaudhary, J. Porter, Z. Yarbrough

***Abstract*—The current wealth gap in the United States is the some of the worst it has been in the last 50 years and it is only widening. Our group was formed with the desire to create opportunities for the average citizen to gain abundance and close the wealth gap by using machine learning to monitor the stock market and learn which trading strategies are best to create wealth from market inefficiencies. This information will then be given to the average citizen for them to optimize their wealth creation. A perceptron classifier on an ensemble boosted decision tree machine learning algorithm was used to test different trading strategies in a live market environment. Traditionally, machine learning models use a back testing approach to train the algorithm to predict what will happen given new stimuli. The backtesting method has been found not to work when applied to the stock market prediction problem. Our algorithm is different because it uses a real time feedback loop to train the model to predict what trades to make instead of traditional backtesting. When the average citizen has access to information that allows them to build net worth at the same rate as the rich, the wealth gap in the United States will be reduced.**

I. INTRODUCTION

Machine learning is a subset of Artificial Intelligence where a computer can dynamically re-evaluate the performance of a task and make the necessary changes to increase this performance. For example, Amazon implements machine learning in its online store. When a customer buys something, the program stores the selection and finds products similar to advertise back to the customer. When the customer makes an additional purchase, the algorithm receives an additional data point about the consumer. As this algorithm continues to collect data about the consumer, it begins to be able to predict future shopping actions. Examples being, when the customer needs laundry detergent or if they would be interested in a certain book. These types of algorithms are implanted all over the world today and are a huge part of our daily lives. Machine learning is still in its development stage and it has difficulty solving the most complex of problems. Huge technology companies are now taking educating their employees with what artificial intelligence is capable of [1] so they can tackle the complex problems. One of these hyper-complex problems is public equity evaluation and trade prediction. To discover the best methods for financial trading, we tested multiple strategies with a machine learning algorithm that utilized a perceptron classifier on an ensemble boosted decision tree. This algorithm was implemented using a field programmable gate array (FPGA) to create a system that is fast enough to make high frequency trades and to learn in real time to discover what strategies are the best at any given time. The machine learning algorithm discovered the best strategies for financial trading, and reevaluated past knowledge to incorporate this new information. With this machine learning implementation built upon well established financial strategies, our hope was to build a machine learning algorithm to generate profits. After much experimenting, and weeks running our algorithm, we were unable to record any profits.

*University of Utah Department of Electrical and Computer Engineering

II. RELATED WORK

Not surprisingly, the idea of applying machine learning to try and guess stock prices isn't novel. From financial institutions to academics to hackers, there are thousands of people trying to solve this problem. More specifically, the idea of using Neural Networks to try and beat Wall Street has been around since the 1980's [4]. This practice went out of popularity in the 1990's because the performance wasn't as good as promised. With recent advancements in technology and machine learning abilities, Neural Networks are again gaining popularity as a method for stock predictions. Now, many investment strategies are using Machine Learning and big data to analyze financial markets and make investment decisions. Whether their algorithms actually work or not is hard to say since most groups are not releasing their proprietary design or trading information. What remains clear is that many organizations are attempting to apply machine learning to trades in hopes of some financial reward [4]. A research group from India tried

Fig. 1. Four Candle Hammer image explanation. Image from https://tradingstrategyguides.com/technical-analysis-strategy/

applying a machine learning algorithm with only one indicator for analysis. They used sentiment analysis from twitter and desired to predict the values of stocks. [2] They found that their model was successful 60% of the time. A team from Turkey built a model that used 25 different indicators for analysis all applied to a neural network which was only successful 50%-60% of the time. [3] What these two have in common is the fact that they store and save their data. Our theory is that the longer you hold data the less useful it becomes. In some cases, this data could even be harmful. For example, If a company is the most popular on social media for years but are suddenly hated. These stored models will still see the company as positive and might buy stocks as the price tanks. Although machine learning is far from perfect, the fact that private groups won't release their findings tells us that there is positive progress in this field that is worth pursuing.

III. Trading Strategy

We have focused our research on five well known investment strategies: Relative Strength Index, Twitter Feed Sentiment, Bollinger Bands, Four Candle Hammer, and company news sentiment. Each of these five strategies have all been tested and proven in the market, but using these strategies together has not been proven before.

Relative Strength Index The Relative Strength Index or RSI is a financial instrument used to measure the strength of price changes over a time period. It measures if a stock or financial asset is overbought or oversold. Supply and demand rule the price of a stock. Overbought means that the financial asset is due for a price decrease due to the demand of purchasing decreasing at the current price range. Oversold means that the relative supply is being depleted and the demand at the price range should start to rise. The RSI of a stock generally oscillates up and down reaching a high of 100% (completely overbought) to a low of 0%

Fig. 2. RSI Example. Image from https://www.tradingview.com/chart/?symbol=NASDAQ%3ATSLA

(completely oversold). For our strategy when the RSI broke the 65% barrier with an upwards momentum we deemed the stock bullish. When it broke in a downwards direction under 30% we deemed it bearish. The reverse happens when the RSI breaks in the opposite direction ie; when the price breaks under the 65% barrier the RSI becomes bearish, and when it breaks up from the 35% barrier we deemed it bullish. When the RSI is between those 2 values we follow the trend, if it's continually going down then we are bearish, and if the rsi rises we deem it bullish. The RSI is calculated with the following equation;

$$RSI = 100 - \frac{100}{1 + \frac{AverageGain}{AverageLoss}}$$

We chose this Indicator due to it being a very popular indicator within the industry. Fig 2 below is an example of a graph with RSI indicator at the bottom. The purple region shows the breakout points, above the purple means its currently bullish, below the purple means its bearish. Within the purple region we follow the trend. **Twitter Feed Evaluation:** The public's opinion of a company is very important in a company's stock price. Twitter is an easy way to get a glimpse at the opinion around a company. The general trend is that if the general opinion of a company is positive, the stock price will trend upwards. If the opinion on a company is negative, the stock price will trend downwards. By pulling random "tweets" that contain the company's name, we are able to analyze the wordage and determine the sentiment of the tweet. When the algorithm wants to know when to buy or sell based off the tweets, it will send it the Company name. With the name, it will hit Twitter api

with a with the name and ask for 100 tweets. Once we receive the tweets, we need to make sure there are no images or hyperlinks in the text since that can through off the sentiment of the tweet. After sanitizing the tweets, we use a library called Textblob. Textblob is a python api library that is used for textual analysis and helps developers with natural language processing. We choose to utilize it for their sentiment analyzer. A sentiment analyzer looks for key words and phase in the text to understand what emotions are used in the text. For example: If a tweet contains the word "love" it is likely the sentiment is very positive. If the Tweet contains the world "hate" it tends to mean the text is negative. If the text holds both "love" and "hate" it could be either negative or positive which usually gets decided as neutral. In our case for this project we used it to understand if the tweet is just positive or not. If the tweet is positive, we set the value to 1, if negative or neutral, the value is set to 0. Since the api we used is limited on how many times we can use it a day, we decided to save the tweets in our database. Once all 100 tweets have been analyzed. We sum up all the values and if the sum is greater than the number sent to the program, it will suggest buying the stock by returning 1, if lower, it will return a 0.

Table I shows an example of this algorithm in production. We have various tweets, with the company they are associated with, and the rating.

TABLE I
THIS TABLE HOLDS REAL TWEETS SAVED TO THE DATABASE AND THEIR SENTIMENT READING.

Tweet	Company Name	Sentiment Value
"TRADING IDEAS EXXON LONG TERM BUY FREE TRADE-OF-WEEK VIA..."	Exxon	1
"MILLENNIALS TRUST BITCOIN BASED EQUITIES OVER MICROSOFT DISNEY STOCK BITCOIN BASED INVE..."	Microsoft	0
"JPMORGAN CHASE BANK REMEMBER YOU ARE NOT THE EMPEROR"	JPMorgan	0
"IBM AND ALIBABA TOPS LIST OF MOST BLOCKCHAIN"	IBM	1
"I AM AS STRONG AS TELSA"	Tesla	1
"...ACCUSES GOOGLE OF ILLEGALLY FIRING WORKERS..."	Google	0

Four Candle hammer: The four candle hammer strategy is a technical analysis of stocks. This strategy also looks for market trends in stock by observing a stock with a positive general trend, but with a recent pullback in the price. The theory is that the general trend will be resumed after a short pullback period. In our implementation of the four candle hammer strategy, we will observe a stock with a 20 day high and a 4-day pullback, if the closing price on the 5th day is higher than the closing price of a stock on the 4th day, we assume the upwards trend will continue. We will then buy that stock on the opening of the 6th day.

Fig 1 from the TechnicalStrategyGuides, shows an explanation for the Four Candle Hammer Technical Trading Strategy. Most days a four candle hammer will not occur, but when it does, there is a large potential for profits to be made. We chose to use the Four Candle Hammer strategy, because it has a history of success, and is a very analytical calculation.

This was implemented by querying from our database, to determine if a 20 day high occurred 5 days ago. If there was a 20 day high, and the past 5 days have been lower than the 20 day high, a positive "buy" signal was returned from this algorithm. It was very important to have an active database we could query from. Without our own database of stock prices, we would have reached beyond the capacity of our allowed data api's and would not have been able to use this metric. Because we had our own local data of each stock and it's daily stock movements, we were easily able to implement and use the Four Candle Hammer strategy.

Bollinger Bands: Bollinger Bands a technical analysis indicator developed by John Bollinger. The Bollinger bands are based on examining the simple moving average of the price over a chosen length of data. For our strategy we used a sample of 14 data points to calculate the simple moving average. The simple moving average is calculated by adding the stock prices for the last n days then dividing by the total number of periods. After calculating the simple moving average we take a look at 2 standard deviation points in the positive and negative direction. These standard deviation points are what create our Bollinger bands. Our Bollinger Band strategy is based on the ideology that if the price is within 3% of the upper standard deviation then we feel the stock will be ready for a drop in price so we deem the stock bearish. On the other hand if the stock is within 3% of the lower standard deviation then we believe the price is due to rise and a buy action should be taken on the stock. If the stock is within the middle region then we should see if its above or below the moving average and follow the trend.[5] Fig 3 below shows an example of how Bollinger Bands look. The line within the middle of the bands is the simple moving average.

Company News Sentiment: The most recent 50 news articles from the previous day are pulled from newsapi.org for each stock and evaluated for writer sentiment.

Fig. 3. Bollinger Bands example. Image from https://www.tradingview.com/chart/?symbol=NASDAQ%3ATSLA

Ticker	Company
AAPL	Apple
AMZN	Amazon
XOM	Exxon
GE	General Electric
GOOG	Google
JNJ	Johnson & Johnson
JPM	JP Morgan
TSLA	Tesla
WMT	Walmart

TABLE II
THIS TABLE CONTAINS THE STOCK UNIVERSE WE CREATED FOR THE PROJECT

This sentiment is processed by using a machine learning driven phrase evaluation algorithm. The algorithm gives the news article a sentiment from zero to one. It then averages the sentiment of the 50 news articles. If the average is above .5 then the stock is recommended as a buy. Otherwise the stock is not recommended as a buy.

Using these trading strategies for the final project demonstrated good results. Other trading strategies could be added to find if those strategies would work better for a given stock than the current strategies. After these trading strategies recommend a buy or not buy this information is sent on to the data processing unit programmed in Python running on the Raspberry Pi.

The stock universe used for executing trades and based on these strategies was a collection of popular stocks from various industries. We chose these stocks, because we knew we would get tweet, news, and strategy information for each of these stocks. We also needed to make sure that the volume of trades were high enough that a sale and buy order would get through within a few seconds of execution.

IV. Technologies

A. Software Implementation

This code can be found at https://github.com/xdkxsquirrel/ML-HFT. When the data processing portion of the project initially starts, it creates a unique stock class object for each stock in our universe. While doing this it pulls the current weights that are stored in the remote database. The weights are then sent over universal asynchronous receiver-transmitter (UART) to be stored in memory on the FPGA for use later in the machine learning calculations. Then the program checks if markets are open, if not, it waits until they are. Then it will cycle through each stock in our stock universe. For each stock, the program checks the company news sentiment, four candle hammer, bollinger bands, twitter sentiment, and relative strength index. Then it sends each strategy decision over UART to the machine learning algorithm. The FPGA will then respond with the decision to buy or not to buy the stock. If the decision for the current stock is to buy, the program will buy the stock using the Alpaca API. After cycling through all of the stocks in our universe, the program waits for five minutes. The program then sells all the stocks in currently held in the portfolio. It then cycles through each stock that was previously purchased and sends whether that stock went up in value or went down in value. The data processing program receives the updated weights from the FPGA and sends those new weights to the remote database. Then it starts the buying process again. The buying process could not happen without the most important part of the decision making, the machine learning algorithm.

B. Machine Learning

The machine learning algorithm operates by taking five popular trading strategies and learning which trading strategies work well for each different stock in the stock universe. This kind of machine learning is based upon the multiplicative weight update method. This algorithm works by creating a binary decision that needs to be made based on the different trading strategies proposition to buy or not to buy. In the first round, all strategies' proposition have the same weight. The decision maker will make the first decision based on the weight of the

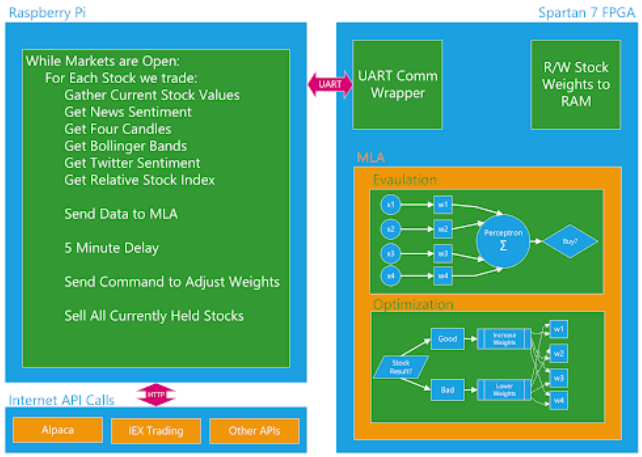

Fig. 4. Diagram of full system on Raspberry Pi.

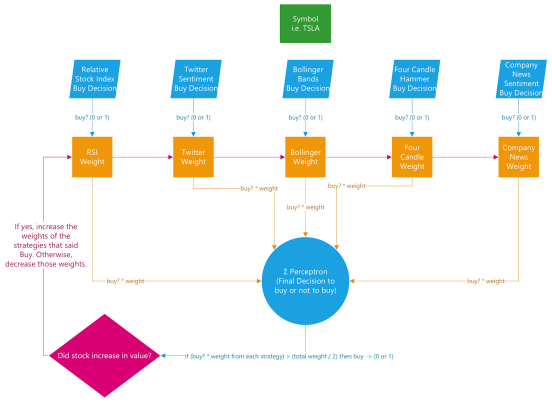

Fig. 5. Multiplicative weight machine learning update method.

strategies' prediction. Then, in each successive round, the decision maker will repeatedly update the weight of each strategy depending on the correctness of its prior predictions.

Fig 5 shows a diagram of the machine learning algorithm that was implemented for our senior project which runs on an FPGA to increase the speed at which the machine learning algorithm calculates.

V. Hardware Implementation

A custom embedded system was designed with an internet communication interpreter and trade execution manager. This allows stock and company information to be transformed from standard hypertext transfer protocol (HTTP) to our custom communication protocol which then transfers data to the FPGA that contains our machine learning algorithm. Data is collected through available online data application programming interfaces (APIs). Trade execution is made using the Alpaca brokerage service. The dashboard front end displays performance in a more easily understandable format for viewers to monitor the machine learning algorithm's performance. The embedded system runs on a Raspberry Pi. The internet communication interpreter and trade execution manager is programmed in Python on an advanced reduced instruction set computing machine (ARM) Linux distribution. All trading algorithms were created and written by our team to provide a unique software component. All hardware was built and implemented on the FPGA to create a unique hardware component.

A. FPGA

CMOD S7: Breadboardable Spartan-7 FPGA Module https://store.digilentinc.com/cmod-s7-breadboardable-spartan-7-fpga-module/

FPGA board that is used to run the machine learning algorithm that communicates with the Raspberry Pi using UART to interface with the financial markets.

B. Raspberry Pi

Raspberry Pi 3 Model B + https://www.adafruit.com/product/3775

Used to run the software that will interface the machine learning algorithm with the internet markets using APIs.

C. Ticker/Dot Matrix

https://www.adafruit.com/product/420

Towards the end of the project, we decided we wanted to include something to make the project more visual when displaying the project to our peers without changing the project specifications. What we came up with a ticker that displays the daily price changes of our universe. Tickers are used in trading to help traders identify potential buy/sells. We used three, 16x32 LED panels placed in a row connected via GPIO pins on a Raspberry Pi to create our small-scale ticker. Utilizing the library provided by Heller Zeller, we were able to query our database and do the necessary calculations to get a working ticker for our price universe.

D. MLA Design on the FPGA

This code can be found at https://github.com/xdkxsquirrel/ML-HFT-FPGA. The machine learning algorithm that was built on the FPGA was built using eight modules. The first module labeled "UART_RX" is a state machine that takes the serial bits coming in from the UART receive line from the Raspberry Pi and outputs an ASCII character for each UART packet it receives. The next module, which

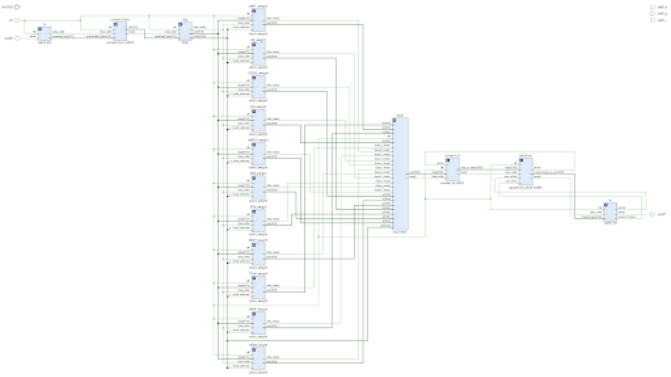

Fig. 6. The Representation of the FPGA's Design

is labeled "Convert_from_ASCII," is a state machine that takes two ASCII characters and outputs the corresponding byte. For example, the ASCII characters "F" and "4" are converted to the hexadecimal byte 0xF4. This Byte is then sent to the next module, which is labeled "MLA," which is a state machine that is acting as a buffer. It waits for six bytes to be sent, then it sets the command state and which stock is being used based on the first byte sent. Then it passes on the next five bytes to the actual machine learning portion of the code. For example, the message string of "A1AB553321C3" would be processed as: A = set weight command, 1 = TSLA, AB = company news sentiment weight of 171, 55 = four candle hammer weight of 85, 33 = bollinger bands weight of 51, 21 = twitter message sentiment weight of 33, and C3 = relative strength index weight of 195.

Once all five weights, the command, and stock data has been received the information is passed on to the next module labeled "stock_weight." This module is a state machine that changes state depending on what command has been sent. The five commands are set weight, get weight, calculate buy, calculate weight gain, and calculate weight loss. This module also holds the weight of the stock in memory. This module is duplicated eleven times, one for each stock in our trading universe. When the command calc buy is sent, the module takes which strategy is currently a buy, adds the weight of those strategies for the current stock being decided on, and sends a buy if the result of that addition is greater than half of the sum of all of the weights. Otherwise, it sends a do not buy.

The next module is an eleven to one mux with a selector based on which stock is currently being processed. It routes the output from the stock's weight module and sends it to the next module labeled "convert_to_ASCII." This module takes the five hexadecimal numbers and converts them to their respective ASCII representations.

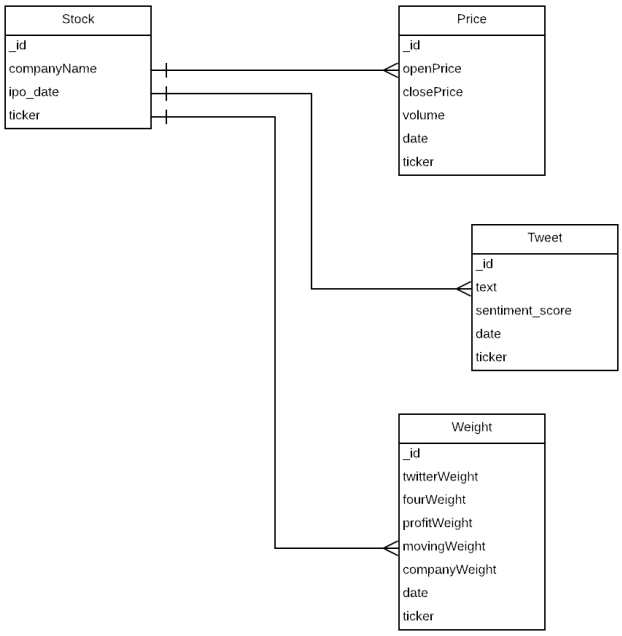

Fig. 7. Example of the database schema

This is then passed to "parallel to serial buffer." This takes the five ASCII characters and sends them one at a time to the final module which is a UART transmitter that will send this data to the Raspberry Pi.

E. Database

A few weeks into our project, we quickly discovered that the limiting factor of this project would be the cost of data. For each data resource, who has an API for us to use, there is a limit to the number of API calls we can do daily. The daily allotment is much smaller than we would need to measure stock prices and perform our finance algorithms individually. To fix this problem, we created a database to store all data, which we could access for free. As a result, we did minimum api calls to APIs containing data, and stored all resulting information into our own internal database.

Below is the database schema for our stock storage. Each stock has a price and weight objects for every day. There was a script which ran daily to scrap the daily stock information, and insert it into the database.

1) APIs: In order to actually mock buying and selling stocks in a real market, we needed a way to automatically ping the market for our buys and sells. There are a lot of companies out there are offering an api to do just that. We decided to choose Alpaca since it was free and we can easily do these pretend trading called paper trading.

Our internal API was a graphQL endpoint. This endpoint was used to access the database, and to communi-

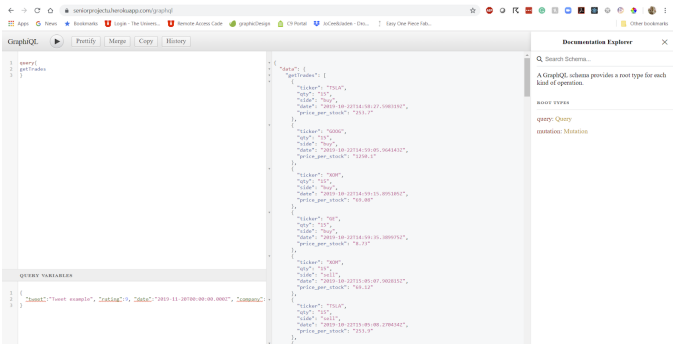

Fig. 8. Example of the API use

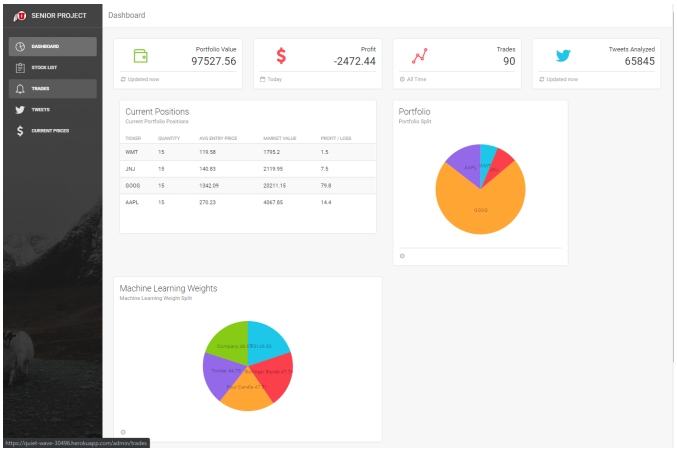

Fig. 9. This is the front page of our dashboard

cate with the other sections of our project. This allowed multiple systems to communicate asynchronously across different devices. The ML algorithm was able to send and retrieve weight information to perform calculations with. Additionally, the Python script saved Twitter information, which was viewed and understood on the website. Fig 8 is an example of a query to get all the most recent trades executed.

Apart from our own internal API and Alpaca, this project used a lot of external API's as well to help us gather and analyze as much data as possible. The Twitter algorithm used a Twitter api for receiving tweets and the Textblob api for textual analysis. The Company Data algorithm used NewsApi to get news articles and then used Indico api for their sentiment analysis. The profit loss algorithm used iexapis api to get the company's quarterly earnings. The Moving Average, Bollinger and RSI algorithms all used the AlphaVantage api to get the daily cost averages.

F. Front End

In order to view the current status of all parts of our system in an easy way we needed to develop a frontend. We chose to develop our front end with React.Js. React is a frontend javascript library which makes websites feel more like desktop and mobile applications. We chose to use react and bootstrap styling for a clean minimalistic look, as well as being able to render mobile and desktop views depending on which device the user opens the website on. React allows an extremely fast way to re-render parts of a website. If you go to our website you can see that when you click on different links there is no refreshing the browser but a re-rendering of only the parts which need it. By clicking on different links our router system removes and adds new elements to our Document Object Model. We have 5 main pages or "components" to our site. The dashboard is the main landing page which displays overall statistics including our current portfolio value, profit/loss, number of tweets we've analyzed, current positions, trades made, portfolio positions and individual profit and loss, as well as overall machine learning statistics. Once the dashboard component is mounted (ready to be displayed) on the browser we make asynchronous calls to our graphql backend to get the data needed for our charts. The asynchronous calls are done by using Apollo and Axios which are both industry standard libraries for graphql queries. Once the data is retrieved we have to process and extract the data needed and update the state of our application. When the page is first rendered the charts are rendered with empty data until our call to the backend retrieves the data. Once the data is retrieved and state is updated, the website re-renders each chart one at a time with the appropriate data. Chartist was our library choice for helping us to create nice looking charts. The next main page of our front end is the Stock List. This page shows us a list of all stocks within our universe and allows us to track the machine learning statistics for each stock over a period of time. The last 3 pages, Trades, Tweets and Current prices are pages listing our most recent trades our system has made, the most recent analyzed tweets and the current prices along with overall bearish or bullish indicators. All of these pages are using asynchronous calls for our data, and can be refreshed by once again clicking on the link to the page or by refreshing the browser. The front end react project is hosted on AWS through Heroku and can be accessed anytime at https://quiet-wave-30496.herokuapp.com/admin/dashboard. Since we are on a free hosting tier it may take up to 30 seconds to load the initial website. Our dashboard theme was supplied by Creative Tim*, and list of installed libraries can be found within the node modules folder.

Fig. 10. Final Profit/Loss Graph

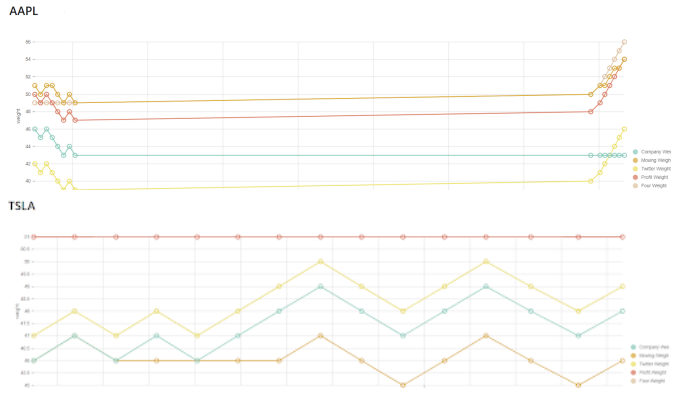

Fig. 11. Apple and Tesla's moving weights

VI. Final Trading Results

Unfortunately, we didn't make any money with our machine learning algorithm. Of the $100,000 of capital we started with in our paper portfolio, we lost $2,435.09 over 4 months. There are many explanations for why our algorithm didn't result in positive cash flow.

One exciting thing about our results is that our algorithm was completely uncorrelated with the market. Instead of simply following market trends, our algorithm performed independently.

In Fig 11 you can see how the weights for the different algorithms in each stock changed over the course of a day. The weights are correlated with each other, since the cause of price couldn't be completely isolated from the price of a stock. Over time, our machine learning approach was able to calculate the more relevant strategy for that stock. Giving the weight associated with the relevant strategy a higher weight, and lowering the weights of lesser relevant strategies. As the days progressed, we observed the weights fluctuating as new information was being calculated and considered. Over the course of a trading day, up to 100 data points and weight calculations were recorded for each stock.

Again in Fig 11 you can see the weights for Apple and Tesla. As expected, the Twitter weight for Tesla was one of the highest weights, while the twitter weight for Apple was the lowest. If given more time to continue training and adjusting weights from over time, we believe the machine learning algorithm would have been able to more accurately predict future market trends.

VII. Conclusion

As Machine learning is becoming more prevalent in our lives, through asking Alexa to buy items online, to Netflix recommending which show to watch next [1]. It is only natural to let a system that can dynamically adjust to optimize performance make our financial decisions. A system which is able to discover which financial strategies correlate with stock prices. Creating a more informed investment strategy.

Through our implementation of a high frequency machine learning trading algorithm, we analyzed five different stock trading algorithms. Each of these trading algorithms stands alone as a useful and accurate financial strategy. Each strategy can be used alone to evaluate the short term potential profitability of a stock. Utilizing machine learning, we attempted to optimize the performance of these trading strategies for each individual stock in our stock universe. Coding this machine learning algorithm on FPGA hardware, we created a system that is fast enough to handle high frequency trading for a large universe of stocks. Although at the time of this report, our implementation was not at the point of profitability, we believe with adjustments to our machine learning system we can create a reliable system that can accurately discover market trends providing a profit.

References

[1] S. Levy, "How Amazon Rebuilt Itself Around Artificial Intelligence," Wired, 30-Oct-2018. [Online]. Available: https://www.wired.com/story/amazon-artificial-intelligence-flywheel/. [Accessed: 29-Apr-2019].

[2] T. Mankar, T. Hotchandani, M. Madhwani, A. Chidrawar and C. S. Lifna, "Stock Market Prediction based on Social Sentiments using Machine Learning," 2018 International Conference on Smart City and Emerging Technology (ICSCET), Mumbai, 2018, pp. 1-3. doi: 10.1109/ICSCET.2018.8537242

[3] GÜNDÜZ, Hakan, Zehra ÇATALTEPE, and Yusuf YASLAN. 2017. "Stock Daily Return Prediction Using Expanded Features and Feature Selection." Turkish Journal of Electrical Engineering & Computer Sciences 25 (6): 4829–40. doi:10.3906/elk-1704-256.

[4] Ruggiero Jr. "Enhancing Trading with Technology". Futures: News, Analysis & Strategies for Futures, Options & Derivatives Traders, 2000;29(7):56. Available: http://search.ebscohost.com/login.aspx?direct=true &db=f6h&AN=3194784&site=ehost-live. [Accessed: 11-Dec-2019]

[5] A. Hayes, "Bollinger Band®," Investopedia, 23-Apr-2019. [Online]. Available: https://www.investopedia.com/terms/b/bollingerbands.asp. [Accessed: 12-Dec-2019].

[6] "Simplified Text Processing," TextBlob. [Online]. Available: https://textblob.readthedocs.io/en/dev/. [Accessed: 14-Dec-2019].

[7] TradingStrategyGuides, "Technical Analysis Strategy – Four Candle Hammer Strategy," Trading Strategy Guides. [Online]. Available: https://tradingstrategyguides.com/technical-analysis-strategy/. [Accessed: 14-Dec-2019].

[8] Hzeller, “hzeller/rpi-rgb-led-matrix,” GitHub, 07-Oct-2019. [Online]. Available: https://github.com/hzeller/rpi-rgb-led-matrix/. [Accessed: 14-Dec-2019].