Artificial Neural Nets for Retail Forex Trading

Andrew McMahon
Advisor: Dana Angluin
April 2015

Abstract

The high frequency trading (HFT) industry has exploded in recent years. HFT firms now account for over half of all equity trading volume in the United States and make over $1 billion in profit per year.\(^1\)\(^2\) As these firms become more advanced, the strategies they employ move further away from those available to retail investors. While some, such as low-latency arbitrage, will never be viable for a retail investor, other strategies, such as machine learning techniques, can be adapted and developed to work in a low frequency retail environment. Researching these methods might open up new trading strategies to individuals that were previously only available to institutional firms.

In this project, I explored the use of artificial neural nets (ANNs) in trading algorithms intended for retail investing. First, I developed non-neural net algorithms to use as a baseline for later comparison. I did this by automating traditional trading strategies based on several different technical indicators. I then developed two different algorithms based on ANNs. The first is a two layer neural net that uses one indicator, but takes as input its value from several different time periods. The second is a three layer neural net that takes as input several different indicators. I then used a genetic algorithm to optimize the networks and find the best weightings of the inter-neuron connections. Finally, I tested all of these algorithms on multiple currency pairs to determine their viability for real-world trading. The results showed that these neural nets, within limit, show promise for use by retail investors with further research and development.

---

1 Jeremy Grant (Sep 2, 2010). "High-frequency trading: Up against a bandsaw". Financial
Introduction

Machine learning methods are beginning to see widespread adoption across many industries, from cyber-security to insurance. In recent years, the use of these algorithms in financial markets has become one of their most visible applications but also one of their most controversial. While the long-term economic effects of high frequency trading lie in the domain of professional economists and market theorists, computer scientists can address a more immediate concern. Never before in history have ordinary investors felt so alienated from the strategies used by institutional firms, such as banks and hedge funds. Whereas previously, retail investors could analyze and follow the fundamental and technical based strategies used by these firms, machine learning has made only the most computationally and mathematically minded investors able to embrace the newest trading strategies.

In this project, I have explored the viability of using machine learning algorithms in an easily accessible retail-trading environment. Retail investors have three important limits that do not apply to large financial firms: the frequency at which their trades execute, the cost of commission on each order placed, and the availability of real-time market data. By taking these variables into account when designing the algorithms, testing them for profitability, and then making them configurable so that any investor can alter them to fit into his or her portfolio, I have attempted to make advanced trading strategies, using cutting-edge computational methods, available to the common investor.

Project Details

In this project, my goal was to develop trading algorithms that utilized machine learning methods. Both automated trading and machine learning are sizeable areas of study and the result of many years of research and development. Algorithms now trade automatically on nearly every financial market in the world and utilize many different design paradigms. Machine learning is a broad term that can encompass anything from classification
algorithms to clustering algorithms, supervised and unsupervised learning, and much more.

To keep the scope of this project manageable, I narrowed my efforts to applying artificial neural nets to foreign exchange markets. I decided to use a specific platform, MetaTrader5, and its accompanying language, MQL5, to write the algorithms, both for its streamlined development environment for trading algorithms, as well as because it has already seen wide user adoption by thousands of traders across the world. A detailed discussion of these choices is below.

**Foreign Exchange**

I decided to focus on developing algorithms for foreign exchange (FX) markets, where the currencies of the world are traded. Every day, thousands of traders buy and sell currencies, exchanging one currency for another at a rate determined by the market. FX is highly liquid and traded twenty-four hours a day, making it extremely easy for traders to enter and exit positions very rapidly. Furthermore, fundamental analysis, using macroeconomic information and data, is usually used only to predict long-term trends. Shorter-term day-to-day trading decisions are more often based on technical analysis. This sort of analysis is done using numerical data generated by market activity, such as price and volume, to predict short-term movements of a particular currency. These conditions make FX markets ripe for the algorithmic application of advanced computational and statistical methods.

Additionally, algorithmic trading has already seen significant use in FX markets. While the aim of this project was not to develop cutting-edge or highly profitable trading algorithms, but rather to explore whether machine learning can be used in retail trading strategies, this fact provided me a wealth of background knowledge and a solid foundation for my analysis. Without the existing literature and available platforms, the scope of this project would have exceeded what is possible in a single semester.
Artificial Neural Nets

To research the viability of using machine learning based trading algorithms, I focused on artificial neural nets. Neural nets have a long history of use in artificial intelligence research and, more recently, have been used in a multitude of predictive applications. Artificial neural nets use an interconnected network of neurons to compute an output value based on the inputs to the network. Numerical values are given to the network and are then weighted and transformed. If the value of a node then reaches a certain threshold (chosen during the design of the network), the activation of the neuron is then passed on to the next layer of the network. This propagates through until the output neuron is activated.

Neural networks are useful because they can be designed to perform a variety of functions and can be easily modified to include other features. In this project, I created one neural net that relies on genetic algorithm optimization to find the best weightings for the connections between nodes. The goal of this optimization was to find the best weighting of a single indicator over a spectrum of time frames. My second neural network used multiple indicators and the optimization was used to find the best combination of these, rather than just trying to optimize a single one.

MetaTrader5

To streamline the process of testing on historical data, both to check for profitability and to use as training data for the learning component, I used a robust electronic trading platform that already had this feature built in. This choice was so that I would not have to go through the difficulty of finding and scrubbing my own historical data sets and so that I would not have to build (or incorporate) my own back testing engine. By using an environment that already has this functionality built in, I could focus more on the algorithms themselves. I chose MetaTrader5, the newest version of the popular MetaTrader software. This software, especially the previous version, has seen widespread adoption from retail FX traders around the world. It provides a wealth of charts and technical
indicators and a full set of tools for automated trading. In addition to the testing features, it provides an IDE for automated trading using its own programming language MQL5.

**MQL5**

MQL5 is the built-in language used by MetaTrader5 to make custom automated trading algorithms. It is a high-level, object-oriented language with syntax very similar to that of C++. Although it is proprietary to the MetaTrader platform, it is a fully featured programming language and can accomplish nearly anything that any modern language can. Additionally, its added functionality alleviates the pains of developing trading algorithms, primarily the built-in event listener. This is a necessary part of any trading algorithm, as the program needs to be able to listen to price tics from the market and act on them. Finally, the language has built in functions to determine technical statistics, e.g. the 10-day moving average, directly. This is the foundation for technical analysis and is at the core of all trading algorithms. These factors make MQL5 quite a powerful language not only for vanilla algorithmic trading, but also for the application of more advanced methods.

**Trading Algorithms**

In creating the algorithms, I divided the project into two parts. First, I developed two basic trading algorithms. This allowed me to gain a basic familiarity with the language and the platform before I jumped into developing the neural nets. These algorithms use relatively straightforward trading strategies based on simple technical indicators. Indicators, at their most basic, are metrics derived from market data. They are usually based on some underlying statistic that assists traders in predicting the short-term movement of an asset.

For example, two common indicators (and ones that I used in this project) are the moving average and the MACD (Moving Average Convergence-Divergence). The moving average is a simple indicator that is used to smooth out price movement over a given period. It has two commonly used variants: the
simple moving average (SMA), which is just a simple average of the price of the asset, and the exponential moving average (EMA), which weights recent prices heavier than older prices. Thus a 10-day SMA looks at the prices (most often taking the closing prices of the day) of the previous 10 days and calculates a simple average. When this is plotted over time, it provides a smoother graph of the asset’s price.

The MACD indicator is a bit more complex. It is intended to follow trends by showing the relationship between the EMA of two different periods. It is calculated by subtracting the 26-day EMA from the 12-day EMA. This indicator will thus not only smooth out prices by following the trend, but also signals momentum to the trader. A very simple algorithm, for example, would keep track of the nine-day EMA and the MACD, buying when the MACD rises above the EMA and selling when it falls below it.\(^3\)

The intention of the first part of the project was not to develop new trading strategies. My priority was to use a variety of indicators and different types of algorithms so that I could rapidly gain experience in the language and in testing the algorithms. I also explored which indicators worked well together and which ones did not, so that later, I could incorporate this knowledge into building the neural net algorithms. Finally, these algorithms provided me with a baseline for what to expect in terms of profitability for retail algorithmic trading. I adapted strategies that already existed and are known to provide positive returns, so that later I would have something with which to compare my neural net algorithms.

The second part of my project was developing trading algorithms using neural nets. The MetaTrader platform, while powerful in some respects, is limited in its capability for training advanced algorithms. Because of this, I structured my neural nets so that I could use the built-in optimization functionality to train the network. I did this by making the weights of the connections between neurons into input variables. This allows the platform to optimize all of the weights during testing to find the best possible connections between neurons.

---

\(^3\) [http://www.investopedia.com/terms/m/macd.asp](http://www.investopedia.com/terms/m/macd.asp)
For the inputs into my networks, I used technical indicators. I did this in two different ways. For the first network, I used one indicator, MACD, over a variety of time frames. During optimization, the network will train itself to find the best weightings of the different time frames to use as a signal for buying or selling. In my second network, I used three different indicators. Rather than optimizing to find the best time frames of these indicators, I converted them into trading signals, as would be done in a traditional trading algorithm. The weightings in the network are then trained to find the best combination of signals to turn into an output signal to buy or sell. I will discuss the structure of the networks and the strategies behind them in more detail below.

The testing for my algorithms consisted of measuring the total profit over the past year using the three main currency pairs: USD/EUR, USD/GBP, and USD/JPY. I consistently used USD as my base currency so that entering into a long position means buying US dollars and selling, or shorting, the traded currency. Conversely, a short position means selling the USD and buying, or being long, the traded currency. Following this convention, a long USD position would be profitable if the USD appreciated versus the traded currency, while a short USD position would benefit if the USD depreciated versus the traded currency. My profits were always measured in dollars, so that if I had any open positions at the end of the testing period, they would be marked at the current value of the traded currency relative to the dollar.

Finally, I was able to optimize my algorithms using the built in optimization functions in MetaTrader5. For the basic algorithms, this meant finding the best parameters for my chosen indicators. For example, if I was using a long period SMA and a short period SMA, they would start using 30 days and 10 days, respectively, and then move around to find the optimal values. For the neural nets, this meant changing the weights between neurons to build the optimal network. I used out of sample testing to ensure that I was not over-fitting my algorithms, so I would optimize on one historical period and then do profitability testing on a different period. In addition, for simplicity’s sake, these tests do not
take transaction costs into account, which would have had a negative impact on the profits of these algorithms.

**Basic Algorithm 1: Moving Average in Trending Markets**

*Strategy*

The first algorithm I developed used the simple moving average in conjunction with another indicator, the average directional movement (ADX), to determine when the market is trending sharply. Specifically, this algorithm enters into a short USD position when the SMA moves above previous close price and enters into a long USD position when the SMA falls below the previous close price. This is only done when the ADX value is above a certain threshold value. The ADX indicator holds two values. The first is a measure of how much the market is moving for a given period, which is the value that we are checking as our threshold. The second gives the current direction of the trend. Thus if the first value is above the threshold and is trending upwards, the algorithm will open a short USD position. If it is above the threshold and trending down, it will open a long USD position. My initial parameters were a period of 10 days for both the SMA and the ADX, and a threshold of 22 for the ADX value.

*Results*

My initial testing for this algorithm was done using the default parameters, a testing period of all of 2014, and a starting account balance of $10,000. Unfortunately, across all three currency pairs, this algorithm lost money. This algorithm lost $6,265 on EUR/USD, $7,680 on GBP/USD, and $2,803 on JPY/USD. For the three currencies, the algorithm traded between 3100 and 3400 times over this time period. The following graphs show the balance of the account throughout 2014 for each currency cross. The balance in USD is on the y-axis and the number of trades is on the x-axis.
I then optimized the algorithm by using a genetic based algorithm on the input parameters. To avoid over-fitting, I optimized using the year 2013, but tested again on the year 2014. The optimization function returned the following parameters: 54 days for the moving average period, 24 days for the ADX period, and 28.6 for the threshold to use for the ADX value. This time around, my algorithm made a slight amount of money for EUR/USD ($534), a little more for JPY/USD ($1,126), but still lost money on GBP/USD ($1,281). For these three tests, the algorithm traded far fewer times, with all three tests trading between 30 and 60 times in 2014. The graphs that show the balance of the account for each currency cross are below.
Basic Algorithm 2: Moving Average in High Volatility Markets

Strategy

The second algorithm I developed used a basic strategy with an added feature to increase its performance. The core of the strategy takes a short period SMA and a longer period SMA and enters into a short USD position (i.e. it buys the traded currency) when the short period SMA moves above the long period SMA and enters into a long USD position (selling the traded currency) when the short period SMA moves below the long period SMA.

I then added in the feature that causes a new position to be opened only when the current market volatility is greater than it was a short time before,
measured using a metric known as the average true range (ATR) indicator. This was done to increase the chance of the market moving after entering into a position because of the higher volatility. ATR is calculated by finding the greatest of the following three values: the difference between the current maximum and minimum price, the difference between the previous closing price and the current maximum price, and the difference between the previous closing price and the current minimum price.\(^4\) However, this metric was only used when opening new positions. If the short period SMA falls below the long period SMA and there is an open long position, this algorithm closes the position whether or not there is high market volatility, and vice versa for closing a short position.

My initial parameters for this algorithm were using a 10-day SMA for the short period, a 30-day SMA for the long period, and 20 hours for the ATR shift. I then optimized these values using a genetic algorithm to find the best values possible. The results for this algorithm are shown in the graphs below.

**Results**

Testing for this algorithm, as before, was done using the default parameters, a testing period of all of 2014, and a starting account balance of $10,000. Also similarly to before, across all three major currency crosses, this algorithm lost money. This algorithm lost $2,614 on EUR/USD, $422 on GBP/USD, and $4,164 on JPY/USD. However this algorithm traded a fewer amount of times, between 1 and 20 times during these three tests. I believe this was due to the construction of the algorithm, and indicates that it likely needs to be adjusted to close stale positions after a certain amount of time. The following graphs show the balance of the account throughout 2014 for each currency pair.

I then optimized the algorithm, much the same as before. I again used 2013 as the training period and 2014 for the testing period. The optimization function returned the following parameters: 34 days for the short SMA period, 56 days for the long SMA period, and 44 hours for the ATR shift. After optimization, this algorithm made $2,884 on EUR/USD, $14,364 on GBP/USD, and $7,655 on JPY/USD. The graphs for the account balances are below.
Neural Net 1: 2-Layer MACD Neural Net

Strategy

The strategy behind my first neural network relies on the MACD indicator. Whereas a traditional algorithm that uses this indicator can be optimized to find the best single time frame for the algorithm, it will still only rely on that one period. In my neural net, I structured it so that the inputs to the network are the MACD indicator for a variety of time frames, which all feed into a single output neuron. This allows the algorithm to combine multiple time frames into an aggregate signal, while weighting each one based on its performance. The output value of the output neuron is calculated using a sigmoidal activation function, which is standard in many neural nets. If the value is positive, it is treated as a buy signal, and if it is negative, it is treated as a sell signal. The structure of the network can be seen in Figure 1 below.
Results
To test this neural net, I first optimized the weights using 2013 as my training period. I used a genetic based algorithm as before, limiting the possible values of the weights to be between -1 and 1. Interestingly, about half of the weights after optimization were negative and half were positive. I then tested the algorithm on the three currency pairs in 2014, and found that the algorithm made a very small amount of money in two out of the three currencies. For EUR/USD, it made a profit of $558 on 459 trades; for GBP/USD it lost $42 on 451 trades; and for JPY/USD, it made $68 on 439 trades. The graphs showing account balances are below.
Neural Net 2: Multi-Indicator 3-Layer Neural Net

Strategy

For my second neural net, I used three different indicators, and one of them twice, for a total of four input nodes. These nodes then feed into a hidden layer of four neurons, where each neuron in the hidden layer is fully connected to the input nodes. Finally, the hidden layer feeds into an output layer with two neurons, one for a buy signal and one for a sell signal. A diagram of the network can be seen in Figure 2 below.

Whereas in the first network, I was able to use the value of the MACD indicator as the values for the input nodes, this would not work in this network
due to the differences between the indicators (i.e. each value means a different thing). So, to normalize the input values, I turned each of my indicators into a signal and then mapped these to values between -1 and 1, where -1 is a strong sell and 1 is a strong buy. These values then get transformed by the weights on the connections into the hidden layer, and then further transformed by the weights going to the output layer, where the final values are turned into a buy or a sell signal.

The first signal was constructed using the ADX indicator, similarly to how I did it in my basic algorithm. The ADX indicator has three components: a value that indicates how sharply the market is trending, a value that corresponds to its upward movement, and a value that corresponds to its downward movement. If it is trending above a certain threshold and its upward movement is greater than its downward movement, a value of 1 is input into this neuron. If it is trending above the chosen threshold and the downward movement is greater than the upward movement, -1 is input into this node. Otherwise, 0 is used as the input, since this indicator does not give us a buy or a sell signal.

The second signal is calculated using the Relative Strength Indicator (RSI). RSI is a price-following oscillating indicator and is calculated based on a ratio between positive price changes and negative price changes over a given time period. I use two different RSI periods, a short and a long period, similarly to how the moving average indicator was used in my basic algorithms. If the short-term RSI is greater than the long-term, an input value of 1 is used. If the opposite is true, an input value of -1 is used.

The third signal is calculated using the Money Flow Indicator (MFI). Its value indicates the rate at which money is being invested in an asset (here, the current currency). Usually, a very high MFI is correlated with a peak of the market and a very low MFI is correlated with a trough. However, the intermediate values of the MFI can be used as another insight into which way the market is trending. Because of this, I split this indicator into four possible input values. If the MFI is below 20, the input value used is 1. If the MFI is between 20 and 50, the input
value is 0.5. If the MFI is between 50 and 80, the input value is -0.5, and if the MFI is above 80, the input value is -1. This lets me capture the full spectrum of what this indicator can tell us about the market.

Finally, I used the ADX indicator again, but I added a third condition when calculating the signal. As before, for a buy signal, the first two conditions are having the ADX trend value above the threshold, and the upward movement has to be greater than the downward movement. I then added in a third condition that the ADX trend value also has to be greater than the upward movement value for a buy signal or the ADX trend value has to be lower than the downward movement for a sell signal. This would be indicative of an even more sharply trending market, and so I hoped would strengthen the output signal. Because this signal occurs rarely, I also began the weights for this neuron’s connections at a higher starting value during optimization.

Optimization of this neural network was more difficult than with the first one. This is because this network not only has parameters for the weights between neurons (and there are more of these than in the first network), but also traditional parameters that are used for calculating the signals from the indicators. I found that attempting to optimize both of these sets of parameters together did not yield good results (and took an inordinate amount of time). I then attempted to optimize the signal parameters separately from the weights. However, this seemed to offer no advantage over using the default parameters during the later weight optimization, so these results are using un-optimized signal parameters and a genetic-based optimization for the inter-neuron weights.
Results

Once again, I performed my optimization using 2013 and my actual testing in 2014. This algorithm lost $5,660 on 7 trades for EUR/USD, $6,896 on 79 trades for GBP/USD, $4,310 on 87 trades for JPY/USD. The graphs that show the account balances are below.
From the graphs, we can see that the algorithm actually made quite a bit of money at certain points for the GBP and JPY pairs before losing it by the end of the year. An astute investor, using stop-losses and take-profits, could likely have made more money using this algorithm than the final profit numbers suggest.

**Conclusion**

In this project, I was successfully able to develop and test machine learning-based trading algorithms using neural nets. These neural nets were simply designed, using a small number of layers and incorporating feed-forward functionality. By using technical indicators as the inputs to the networks, they can be easily reconfigured for any trader’s personal taste and trading strategy. Finally, they showed promise for profitability with further testing and refinement.

However, as the results above have shown, neural nets are definitely not a magic “black box” that can be used to make profitable trading algorithms. These algorithms were not as profitable as the very basic algorithms I presented here and still need forward testing (testing them for a certain amount of time on live
markets) before one should begin to consider using them with real money. The potential is definitely there, even if it is more advantageous for a retail investor to use these algorithms to research the best way to use certain indicators in their own strategies, rather than to actually trade.

**Future Work**

A great deal of work remains to be done in this area. There still are a large number of machine learning algorithms that can be researched and developed for use in retail trading, many of which, such as support vector machines, have intuitive underlying ideas that make them viable for non-technical retail investors. With more time, I would have further developed my neural nets and then explored other algorithms. With a lot more time, I would have done this by incorporating open source machine learning libraries. This would streamline the development process, allowing me to focus on producing stronger and faster-learning algorithms (to decrease training time). This would likely require moving away from the MetaTrader platform, trading off accessibility for more advanced algorithms. While this was contrary to the original goals of the project, it might result in developing a new platform, allowing retail traders the greatest access to machine learning yet.
References and Additional Reading

Foreign Exchange

http://www.investopedia.com/walkthrough/forex/

Artificial Neural Nets


Algorithmic Trading with Neural Nets


MetaTrader5 and MQL5

http://www.metatrader5.com

https://www.mql5.com

I used the forums at mql5.com as inspiration for much of the basic strategy in my algorithms and general help learning the language.

https://www.mql5.com/en/articles/100

This article provided a walkthrough into creating trading algorithms and is the source for the trading strategy behind my first basic algorithm

https://www.mql5.com/en/articles/497

This article presented one user’s attempt at using neural nets with MQL5 and helped me create my first network.