Kris Walsh
kmw2221
Columbia University Department of Economics
Master's Thesis
"Time Series Models in Foreign Exchange"
Advisor: Professor Steven Ho
December 4, 2020

Abstract

This paper compares the predictive power of three different foreign exchange rate prediction models: a random walk, a linear ARIMA model, and a non-linear neural network model. The neural network model used is a long short-term memory cell. The academic literature has shown that a wide range of linear models fail to outperform the random walk. Neural networks have advantages that make them ideal for time series forecasting, though in the literature they have not been used extensively in exchange rate analysis. The goal of this paper will be to directly compare to performance of the neural network and the linear models in predicting the path of six major currency pairs, to measure if the non-linear structure of the neural network provides a better forecast. The data set covers the range of 2015-2020, in order to put the comparison in context with recent market volatility. The linear model fails to perform any better than the random walk. The neural network performs better than the random walk on a mean squared error comparison, though on a profit and loss analysis the model will gradually lose money over time. A neural network model with a similar structure but with additional variables may mitigate or erase these losses.

Keywords

Foreign Exchange Rates · Exchange Rate Prediction · Time Series Forecasting · Neural Networks · Long Short-Term Memory · ARIMA · Machine Learning

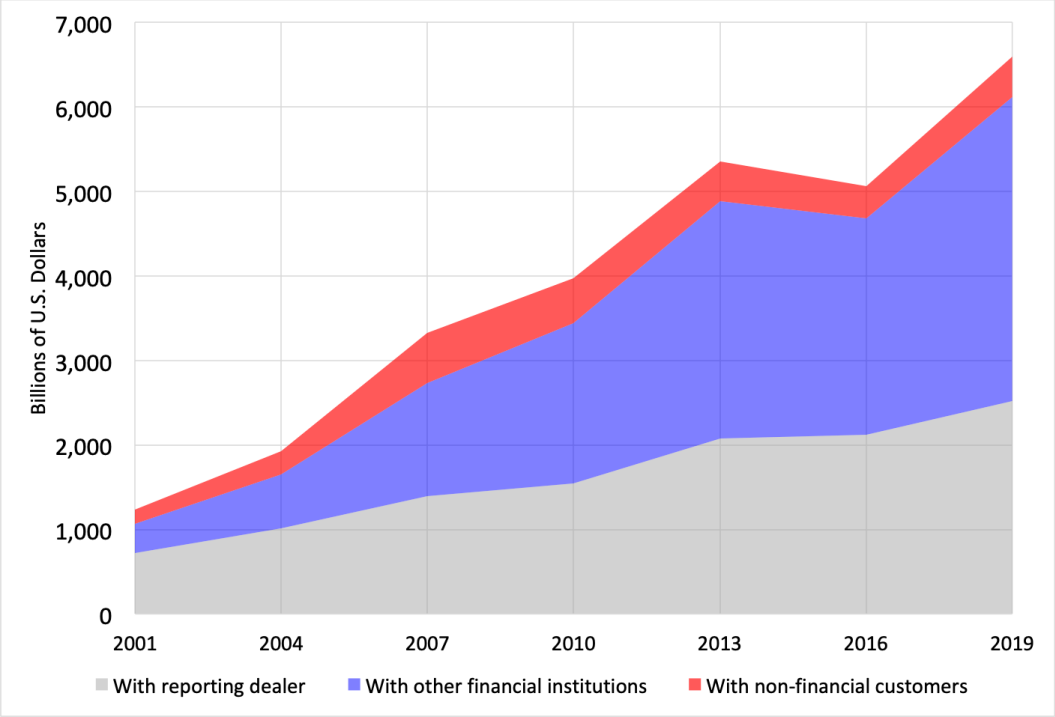
Outline

I. Introduction

Prediction of foreign exchange (FX) rates has a history that goes back centuries, though most of what we would consider modern currency valuation has been around for only 50 years. In 1971, U.S. President Nixon severed the dollar's direct convertibility into gold, beginning a new age of fiat money. This period would coincide with the development of econometric methods that allowed researchers to model FX markets deterministically. Rapid advancement in computing power quickly followed, opening the door for sophisticated machine learning techniques to explore seemingly limitless quantities of exchange rate time series data. Furthermore, the volume of exchange rate market turnover expanded exponentially, from barely anything in 1986 to more than $6 trillion in 2019 (Bank of International Settlements, 2019, Figure 1).

Figure 1: Global Foreign Exchange Market Turnover, 1986-2019

Source: Bank of International Settlements, 2019



Despite the rapid expansion of both the FX market and capabilities in analyzing it, a model that can successfully predict outcomes and create abnormal profit still eludes us. Debate about whether such prediction is theoretically possible remains vigorous in the literature. Central

to this question is the efficient market hypothesis, which states that the market price reflects all current information (Fama, 1970). It follows that the history of the exchange rate cannot be used to increase investors' expected profits, and that the price of the currency is following a random walk. Even Fama has stated that the extreme version of this hypothesis is surely false (Fama, 1991), and that market randomness likely exists on a continuum. It therefore remains of academic interest to apply various time series models to FX markets, and to explore their power to detect patterns in the data, even if those patters are nonlinear or even unknown entirely.

The paper has two components. The first is a review of literature in the field of foreign exchange rate prediction, beginning with Meese and Rogoff's watershed 1983 article which sparked the debate on whether or not FX follows a random walk. This is followed by a review of various models that have been developed since, and a review of how they have performed. Of particular interest will be the application of machine learning, specifically neural networks which have shown promise in outperforming random walks and linear models in predicting FX.

The second component of this paper is an empirical study which compares the predictive performance of a small group of models in predicting six major currency pairs. The data set is daily FX rates of GBP, EUR, JPY, CAD, MXN and ZAR from 1-Jan-2015 to 30-Jun-2020, for a total of 1434 observations. The value of each currency is relative to the U.S. dollar. This study will restrict itself to the univariate case, and the performance measure will be MSE (mean squared error) relative to the actual path of the currency. The models selected are a baseline random walk, the autoregressive integrated moving average (ARIMA) model that best fits each currency pair, and a long short-term memory (LSTM) neural network model that will allow for estimation without the assumption of linearity.

II. Literature Review

A. Linear Models

In their paper, Meese and Rogoff developed the puzzle which states that there is no better economic model for floating exchange rate forecasting than the simple random walk (Meese and Rogoff, 1983). This is a puzzle because to bring a wide range of predictive variables into a

structural model, including macroeconomic fundamentals and market technicals, seemingly should provide some analytic advantage that leads to profitable investments.

The authors considered a wide range of linear models. The first is an autoregressive integrated moving average (ARIMA) econometric model, which was developed in the 1970s. ARIMA models have three components: lags of the *y* variable itself, lags of the error terms, and a differencing parameter. These parameters are identified as *p*, *q* and *d*, respectively. That is, an ARIMA model will include *p* lags of the explanatory variable, *q* lags of the error terms, and the model will be differenced *d* times. What is notable about the ARIMA model is that no other variables are added. Therefore the model tries to find predictive power in the path of the time series itself.

A more complicated version of the ARIMA model, also considered by Meese and Rogoff, is the vector autoregressive (VAR) model. VAR is the same as ARIMA, though instead of being univariate, many more variables are considered. Specifically, money supply, growth, interest rates, inflation, and trade balances are all incorporated into the model. In other words, *y* is a vector rather than a single variable. As the authors point out, "the VAR is important to include in our forecasting experiments since it does not restrict any variables to be exogenous a priori."

The authors found that adding these variables to an ARIMA model, enriching it to a VAR model, neither affects nor improves predictive power. Neither ARIMA nor VAR could outperform the random walk. Therefore in this paper, only the univariate case will be considered, as it has been shown that complicating the econometric model with additional variables is not helpful.

The next group of models considered in this important 1983 work is a broad group of monetary models. Monetary models have their foundation in purchasing power parity, a fundamental economic concept that goes back centuries, to whenever civilizations engaging with trade with one another were using bits of metal currency to store relative value. If purchasing power parity does not hold, then by definition one currency is rich and the other is cheap, relative to each other, in terms of their power to buy an identical basket of goods in each location.

It is entirely logical that if a currency is cheap in terms of purchasing power parity, it should eventually appreciate relative to the other currency. The problem is that deviations in

purchasing power parity can last decades, making it a near-useless tool in making currency investment decisions on any time horizon (Rosenberg, 1996).

Monetary models attempt to address the shortcomings of purchasing power parity by enriching this measure with other, measurable data that are relevant to a currency's value, namely money supply and demand, GDP, interest rates and inflation. It should be noted that this is essentially the same list of variables that was used to construct the VAR model. However unlike in VAR where lags of the variables and the error terms are measured, the monetary model is a straightforward linear regression. Monetary models became increasingly popular in exchange rate prediction in the late 1970s as better data was collected on these variables, and as more participants began to enter the foreign exchange market.

Meese and Rogoff considered the Frenkel-Bilson flexible-price and Dornbush-Frankel sticky-price monetary models, and the Hooper-Morton model which extends the Dornbush-Frankel model with the inclusion of the current account balance as a variable. The authors considered these to be the most promising of linear models, compared to more simple valuations constructed by purchasing power parity (PPP) or interest rate differentials. Still all of them failed against the random walk.

Hope was not entirely lost for linear models. In the following decade John Williamson developed his fundamental equilibrium exchange rate (FEER) model which is designed to value a currency based on deviations from optimal conditions, both internal as measured by output and unemployment, and external as measured by the balance of payments (Williamson, 1985). A major limitation of this model, even according to Williamson, is that it requires a large amount of subjective assumptions, from how to define the external balance to estimating a country's noninflationary and full employment level of output. This means that there are a large number of FEER estimates that are arguably plausible, and to this day this measure is popular among investment professions as a baseline for a currency's fair value.

Several years later Hamid Faruqee introduced the behavioral equilibrium exchange rate (BEER) model which similar to the FEER model uses internal and external macroeconomic conditions to value the exchange rate (Faruqee, 1994). However, the BEER approach uses a stock equilibrium condition to measure the external balance directly, rather than estimating

savings and investment as is done with the FEER approach. The BEER model has the advantage of breaking down the total exchange rate misalignment into the effect of transitory factors, random disturbances and macro imbalances (Macdonald and Clark, 1998). BEER models also remain popular, and a recent study showed that among a wide class of fundamental and theoretical linear models, a first difference specification BEER model was the only one considered which outperformed the random walk, and it did so by a wide margin, using a mean squared error and direction of change criteria (Cheung, 2018).

Other models that did not perform well in the Cheung study include a Taylor rule model, valuations with real rates and yield curve sloped, a sticky price model with risk and liquidity factors, and a PPP model. This therefore provides support to the original conclusion of Meese and Rogoff from almost forty years prior, i.e. that traditional exchange rate forecasting models do not have predictive power. This is despite an explosion in foreign exchange rate trading volumes in the interim years. As a result, it must be considered that there is a fundamental problem with the linear structure of all the models so far mentioned.

B. The Efficient Market Hypothesis

Meese and Rogoff were attempting to answer a simple question: Can foreign exchange (FX) rates be predicted? They considered a vast range of models, applying them from one to twelve month horizons on the three most important currency pairs of the early 1980s; the dollar vs. the British Pound, German Mark (which no longer exists thanks to the advent of the euro "single currency" on January 1, 1999) and the Japanese Yen.

Their resounding conclusion of "No" across models, time horizons and currency pairs on the prediction question would seem to make building a currency forecasting model a futile exercise. It would seem that markets and entirely random and we cannot try to predict them.

This is the fundamental basis of the efficient market hypothesis, which states that the asset price already reflects all available information. While some previous authors have addressed the idea that financial markets are difficult to predict, the efficient market hypothesis is closely associated with Eugene Fama and his 1970 paper "*A Review of Theory and Empirical Work*" (Fama, 1970). In that paper, Fama says that markets are "informationally efficient," in the sense that prices immediately reflect all available information that is in the marketplace.

However, even Fama updated his view in 1991 when he noted, “markets are not simply either efficient or inefficient. Market efficiency can be viewed as a continuum running from the perfect market (i.e., precisely strong form efficient) to the grossly inefficient market where excess earning opportunities abound” (Fama, 1991).

In the context of foreign exchange rate prediction, this opens the door to the idea that some currency markets may be more or less efficient than others. For example, the Mexican peso or South African rand markets may be less efficient than the Japanese yen or British pound markets, meaning that opportunities for abnormal profits may exist in emerging market currency pairs. This is a parallel to Fama’s similar observation that perhaps the New York Stock Exchange is more efficient than the New Zealand stock exchange.

Shortly after Fama provided this clarification, a pair of Australian authors published an article further exploring the reasons why a market may be more or less efficient (Bowman, 1995). Their hypothesis is that people will systematically underestimate the level of efficiency in a market or a security. Essentially, in their view, the market is a reflection of “mass psychology” in which human behavior is amplified and reflected in prices. Since human behavior is obviously imperfect, a market’s structure and therefore its ability to accurately reflect all available information in a price must also be imperfect.

This is great news for currency forecasters. But what are we to make of the empirical results reached by Meese and Rogoff? If markets were even slightly inefficient, shouldn’t even one of the models they considered have demonstrated some predictive power? Particularly since the authors considered a wide range of models and time horizons, it would seem difficult to improve upon the scope of their analysis simply by adding a new time horizon or an extra explanatory variable.

The premise of this paper is that the problem with Meese and Rogoff’s conclusion is that all of the models they considered were linear. The very title of their publication, “Empirical Exchange Rate Models of the Seventies” implies that the models considered may be outdated in some way. Not only did the authors not have the computing power to consider non-linear models, but there were not any non-linear models in existence, particularly when it came to financial market forecasting. Furthermore, the vast expansion of both market volumes and

participation in the ensuing decades is not reflected in their paper. Perhaps data availability has improved since then, and even linear models may perform better in forecasting a currency's value than they did forty years ago.

In order to re-evaluate the original "puzzle" of Meese and Rogoff, this paper will consider the hypothesis that a recurrent neural network, specifically a long short-term memory (LSTM) model, may do better than a random walk or a linear model in predicting the path of a foreign exchange rate. Futhermore, the empirical study will be conducted on exchange rate closing rates from 2015 to 2020, when trading volumes and transaction costs were significantly reduced compared to the 1970s. A comparison will be made between the mean squared error (MSE) of the forecasts for the three different univariate models vs the actual exchange rate. If the mean squared error is the same for all three models, the original puzzle will still hold. However it is possible that either the linear ARIMA models will perform better with more recent data than it did several decades ago, or that a non-linear neural network will provide explanatory power than was not available to Meese and Rogoff. Furthermore, the availability today of computer programming packages will enhance the ability to make a constructive comparison among the various models.

C. Neural Networks

The concept of building a mathematical model that resembles a biological neuron was developed by psychiatrists at the University of Illinois in 1943 (McCulloch and Pitts, 1943). Their basic neuron model takes in multiple inputs, performs an operation on them, and delivers an output. Structures can be formed with multiple layers of neurons that are either parallel or hierarchical, allowing output variables to be determined by a vast range of combinations of input variables. Most importantly, this structure allows for backpropagation, in which the model corrects itself based on its own performance. This mimics how biological neurons become better and faster at performing tasks with repetition. All of these traits characterize a neural network.

The application of McCulloch and Pitts' neural model to financial markets began around 1990, coinciding with an expanding financial sector and rapidly growing computational power. Early studies noted that neural networks were useful in time series forecasting, since they do not

require any assumption on functional form (Wong, 1990). Wong ran his neural network on a primitive Apple Macintosh II with a 20 megabyte drive, and found his model better predicted the stock market than a linear regression.

Of various mathematical tools available to address nonlinear forms, the neural network is particularly well suited to financial markets, including foreign exchange (Binner, 2005). The network has input nodes, hidden nodes and an output node. The hidden nodes provide the non-linear map (without them you are left with simply a linear model). An important trade off in designing a neural network is deciding how many layers of hidden nodes to include. If you have too many, the model will likely result in overfitting. With too few nodes, the network cannot learn. There are still more parameters to consider, including weights for each input function, the number of input variables, the combination of input variables, the types of activation functions for the hidden and outer layers, the value of the learning and momentum rates, and the amount of training. Similar to the FEER structural model, this wide array of choices for model selection leads to a high subjectivity factor in neural networks.

It has been shown that a three layer neural network with only one hidden layer can approximate any function to any degree of accuracy (Hornik, 1989). This result suggests that when it comes to financial market prediction, a simpler model is a better starting point than a more complicated one. If the model has too many hidden layers, it may overtrain itself when finding unique and highly profitable situations in the training set. The least complex networks turns out to be the most profitable because it will not overspecialize, and will seek a more general solution (Galeshchuk, 2016).

Neural networks form a large class of machine learning algorithms, and different networks will be required in different applications. They are put to use in handwritten text recognition, facial identification, video game design, and finance. All neural networks have the same basic components. First, the “nodes” of the networks take inputs, perform some operation on them, and return an output. Connections between nodes are given “weights,” which will be able to change as the algorithm learns and improves its ability to forecast. Finally a backpropagation function must be included, to provide a mechanism for the weights to change over time.

This set up is similar to the biological functioning of a neuron in the human brain. Neurons that are more relevant to performing a certain task will grow stronger, or have their "weights" increased, as the brain learns with repetition that a particular neuron is relatively more important than others when learning to perform a task. As the brain remembers from experience that particular neurons are useful, it will make them stronger. Such is the case with artificial neural networks. The neuron will recognize a particular input as having an important impact on the output, and will increase the weight of the connection to that input.

In the context of time series forecasting, the "recurrent" neural network is most appropriate. Recurrence refers to the temporal connection between nodes, allowing for previous time steps of an observation to impact the forecast of a future observation. This is a direct reflection of what takes place in a linear ARIMA model, where previous time steps are the inputs when determining a forecast. With the recurrent neural network, some number of past observations will be fed into the system to determine a forecasted output.

D. Long Short-Term Memory

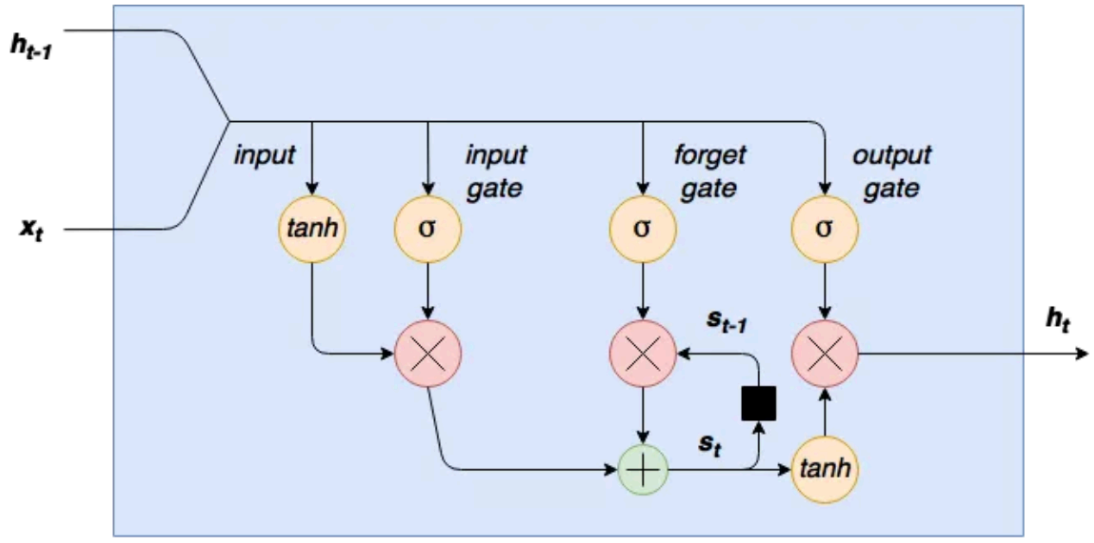
A standard neural network may suffer from the vanishing gradient problem in a time series context (Yang, 2017). As a neural network learns and attempts to update its weights via its backpropagation function, partial derivatives of the error function are typically multiplied in order to determine the new weight. The problem is that since the activation function typically forces inputs and outputs into the range of [-1, 1], for example with a sigmoid or hyperbolic tangent function, the partial derivatives at values close to -1 or 1 will quickly move toward zero. If the system is using multiplication of partial derivatives to update itself, an ever increasing number of inputs will quickly cause multiplication by zero of the error function. In other words, the algorithm will completely stop learning as you add additional inputs to it. A time series algorithm will quickly run into this problem when including more and more previous time steps as inputs to the model.

A particular form of neural network that has shown promise in a financial prediction context is the long short-term memory network (LSTM) (Fischer, 2018). The LSTM network has a memory unit that can store aged information as well as drop it at any time, a structure that

allows for an unspecified gap length between important events and therefore provides a solution to the vanishing gradient. As shown in Figure 2, this is known as a "forget gate," where partial derivatives of the error function are *added* together rather than multiplied. This allows for previous time steps to get "stuck" in the memory network for an indefinite period of time. This is crucial when considering a time series model, because very old observations may still eventually find their way to influencing the output, should the algorithm determine these old time steps to be relevant.

The LSTM neural network cell can be thought of as a stream running slightly downhill through the woods. Most of the information will flow all the way through the stream and off the waterfall at the end. However, as represented by the black box in Figure 2 between state *s(t)* and *s(t-1)*, there is a side stream running in a circle where some water may circle around indefinitely. As time moves on, an individual water molecule has more and more of a chance of finding its way all the way to the waterfall. However, it is entirely possible that a water molecule may trickle through the black box "side stream" indefinitely.

Figure 2: Long Short-Term Memory Cell Diagram



LSTM cell diagram

LSTM is ideal for foreign exchange forecasting. As will be seen in the results of the empirical study, ARIMA models in a financial context rarely find use for more than two or three previous time steps when determining the forecast of the next time step. A better model would be able to include many more time steps, since events in the financial market many time steps prior may be relevant in the price change of an asset. If a traditional recurrent neural network were to try to include even ten previous time steps, the vanishing gradient problem would quickly shut down the learning capability of the algorithm and render it useless. The LSTM gets around this by allowing previous observations of the currency to sit in the black box for many time steps. Therefore "old" information that may be relevant to the forecast but would be missing from an ARIMA or a basic recurrent neural network, can be pocketed away in the LSTM cell until it may be of use in updating the weights of the connections between nodes. In other words, the LSTM has a structure that allows for an unspecified gap length between important events.

One recent study found that LSTM outperforms a standard neural network, a random forest algorithm, and a logistic regression in financial prediction (Fischer, 2018). The authors attribute the success of the LSTM to its ability to extract information from a time series that is particularly noisy. Though they applied their prediction model to the S&P 500, the concept of noisiness is the same in foreign exchange markets. Their conclusion is that "compared to random forests, standard deep networks, and logistic regression, (LSTM) is the method of choice with respect to prediction accuracy and with respect to daily returns after transaction costs."

An even more recent study compared four different types of neural networks (LSTM, Gated Recurrent Unit (GRU), Feedforward Neural Networks (FNN), and Simple Recurrent Neural Network (SRNN)) (Dautel, 2020). The accuracy of all four models was better than that of a random walk, showing promise for the suitability of neural networks for exchange rate prediction in general, but they also noted it is difficult to choose the right parameters for these models, and they did not get very different results among the four structures. They also echo earlier authors when noting that a simple network will likely perform better than a more complicated one, particularly with regard to trading profits.

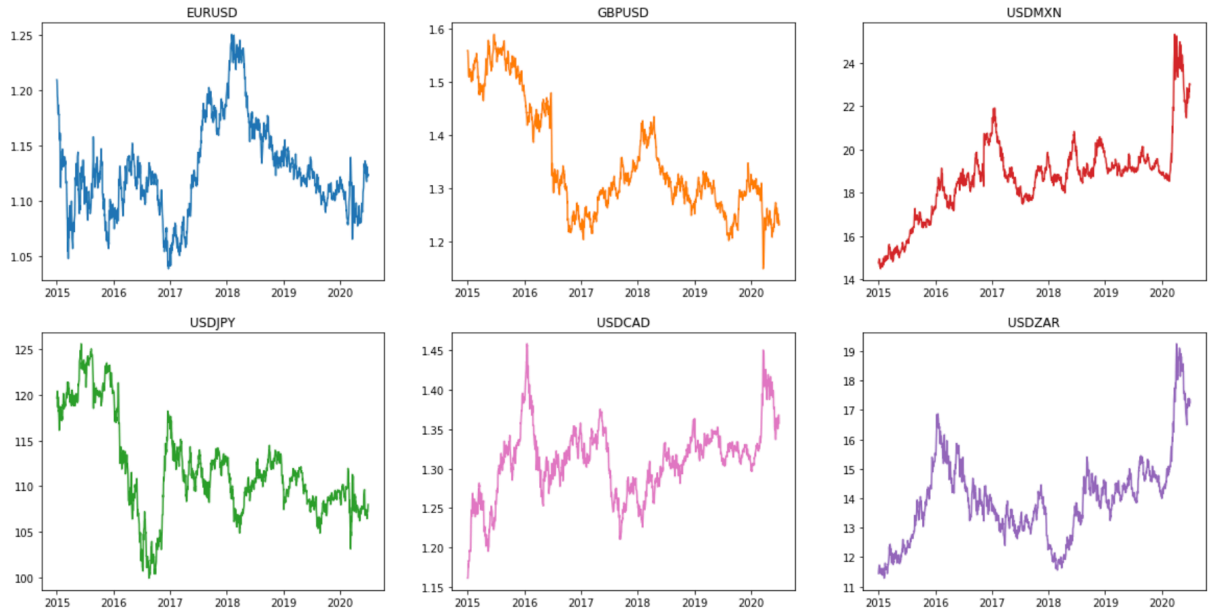
E. Literature Review Summary

Neural networks have shown promise in exchange rate prediction compared to random walks. Both structural and time series linear models have shown lackluster performance, and neural networks offer the advantage of not insisting upon a linear structural form. Long short-term memory (LSTM) models are particularly well suited to chaotic financial time series. An LSTM with a minimalistic structure may be one of the most promising tools in discovering the deterministic nature of exchange rates. To measure the performance of an LSTM model, its mean squared error should be compared against that of a random walk. If it outperforms the random walk model in a meaningful way, the conclusion of Meese and Rogoff may not be correct, and it may indeed be possible to create abnormal profits in FX markets.

III. Empirical Study

A. Data

Figure 3: Data Set (6 Currency Pairs, Daily Observations)



1434 observations daily close observations are collected on six currency pairs, all relative to the U.S. dollar (EUR, GBP, JPY, CAD, MXN and ZAR). The inclusion of two emerging market currencies will allow us to test if higher transaction costs and a larger standard deviation

in returns has any effect on the predictive power of a univariate model. Both the Mexican peso and South African rand are fully deliverable and can be easily traded, though with less ideal liquidity conditions compared to the other four currencies. The date range for the data set is 1-Jan-2015 to 30-Jun-2020. The data is downloaded from the Yahoo Finance API via Python Script (Appendix B).

In the context of machine learning, a data set must be divided into both a training and a test set. The models will "learn" on the training set, using the information therein to determine the ideal *p, d* and *q* parameters (in the case of a random walk and ARIMA), or to update the weights of the neural network backpropagation function. The models are then run on the test set to measure the performance of the models' predictions relative to the actual path of each currency. The performance measure used is MSE (mean squared error), in line with the literature including the paper by Meese and Rogoff.

The training and test set size in this study will be 80% and 20% of the 1434 daily observations respectively. Therefore the models will be trained on 1147 observations from 1-Jan-2015 to 25-May-2019, and the test set will run from 27-May-2019 to the end of the sample, 30-Jun-2020. The test set is thus about a full year of daily observations, and will include the significant market volatility observed during the spring of 2020 as a result of the global pandemic. This period of higher volatility will provide an extra layer of rigor to the test set, and will further help prevent overfitting between the training and test sets.

The inclusion of emerging market currencies will also allow for an examination of the efficient market hypothesis. As discussed in the literature review, market efficiency may exist on continuum with some markets more efficient than others. In the case of less efficient markets such as the Mexican peso or South African rand, relative inefficiency may open the door to a profit opportunity. Including these currencies in the study will allow both the linear and non-linear (neural network) models applied to attempt extraction of relevant forecasting information from these less efficient markets.

The empirical goal of this paper is to compare the performance of a group of univariate exchange rate predictive models: the random walk baseline, the linear autoregressive integrated moving average time series, and the LSTM neural network which will allow for unspecified

functional form and for the inclusion of an unlimited number of time steps that will inform the prediction.

Only the univariate case is considered; the time series history of each currency will be the only input in determining the forecast. Meese and Rogoff showed when comparing their ARIMA and VAR models that adding additional macroeconomic variables such as growth, inflation and interest rates does not improve the performance of exchange rate models. This will allow us to restrict our attention to ARIMA in the linear case. In the case of the neural network, examining a univariate model will be consistent with the idea that a simpler model is generally better than a more complicated one. In addition, having a univariate neural network will be the appropriate direct comparison to the univariate ARIMA model.

B. Random Walk Results

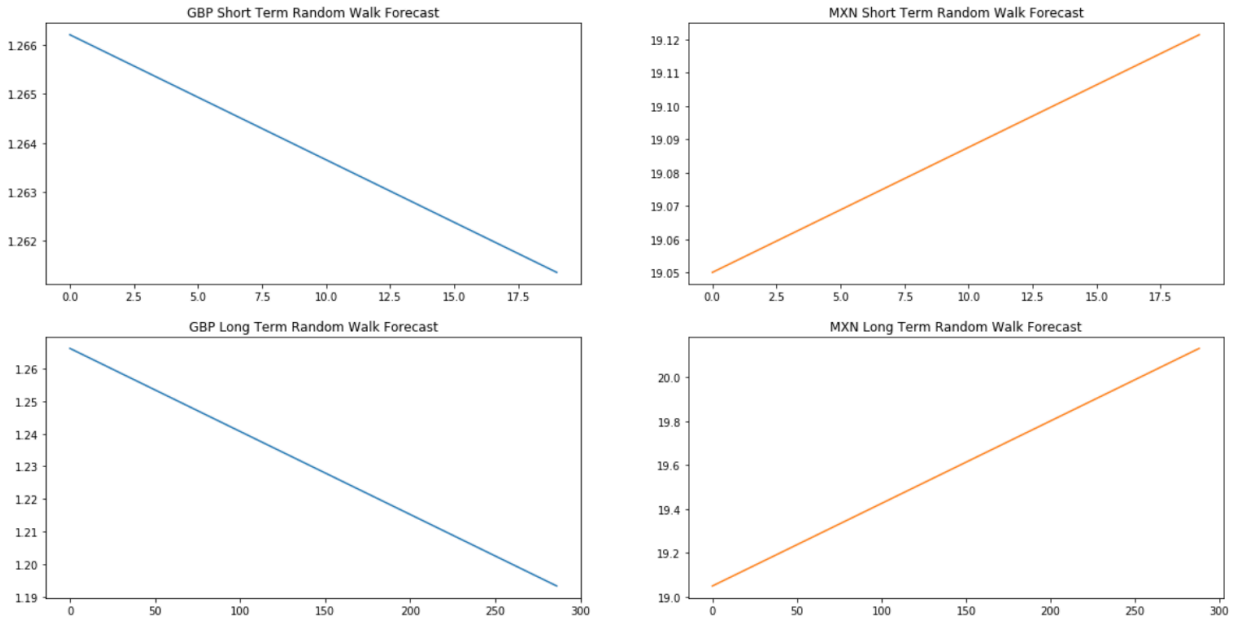
The random walk will serve as the baseline against which the performance of the ARIMA model and the neural network will be compared. A random walk is a pure stochastic process. If a market is completely efficient, then no model should perform any better in prediction than the random walk.

It should be noted that an ARIMA model with the *p*, *d*, and *q* parameters of (0, 1, 0) is identical to a random walk. Zero values for *p* and *q* indicate the model has no autoregressive terms, neither of itself nor of the errors. The 1 in the *d* term implies that $y(t) = y(t-1)$, or that the best prediction of a currency's value at the end of any given day is its previous close. This "naïve" forecast is our random walk, and serves as the baseline against the performance of the other models considered in this study. The MSE for the random walk model is therefore reported as that of an ARIMA (0, 1, 0) model.

As can be seen in Figure 4, the random walk has zero predictive power. The upper two panels are a zoom in of the first 30 observations of the test set, while the lower two panels show the forecast of the entire test set. The model will observe the last return in the training set in percentage terms, and then will apply that same return to all of the observations in the test set. Figure 4 shows the output for both the British pound and Mexican peso (the other four currency pairs do not have a different result). The last date of the training set was likely a negative risk

day in the market, with the pound and peso both weakening relative the dollar. The random walk for both currencies extrapolates this negative return to both currencies across the test set. (Note that GBPUSD and USDMXN are quoted in opposite terms relative to the dollar, i.e. a lower GBPUSD and higher USDMXN are equivalent in terms of the non-dollar currency weakening).

Figure 4: Random Walk Results (GBP and MXN shown)



C. ARIMA Results

The linear ARIMA model regresses the exchange rate value on its own past values, as well as lags of the error terms in the regression. This is one of the models that did not outperform the random walk in Meese and Rogoff's study. Before an ARIMA model can be estimated, its ideal parameters *p, q* and *d* must be selected. The Akaike Information Criterion (AIC) is a measure that gives the optimal parameters, balancing both the maximum predictive power of the lags and the efficiency of the model. In this context, optimal means the greatest predictive accuracy while minimizing the number of lags (both of the variable and of the error) that are included. In other words, adding more and more terms must be balanced against the computing efficiency gained by having the most parsimonious model possible. Furthermore, the

AIC measure allows for direct comparison of different ARIMA models that have varying parameters.

In this study, a range of ARIMA models are considered for each currency pairs, scanning a range of *y(t-1)* lags ("*p*") from 0 to 5, and any range of *ε(t-1)* lags ("*q*") from 0 to 5. A grid search is performed over a reasonable range of permutations to find the ARIMA model that delivers the lowest AIC value. This model is considered the "best-fit" ARIMA model, and should be used when comparing the performance of the ARIMA model to any other model.

In the case of all six currency pairs, the greatest number of lags in the best-fit model was never larger than 2 for both *p* and *q*. We can therefore rule out the idea that an ARIMA model with a far larger number of lags is worth investigating.

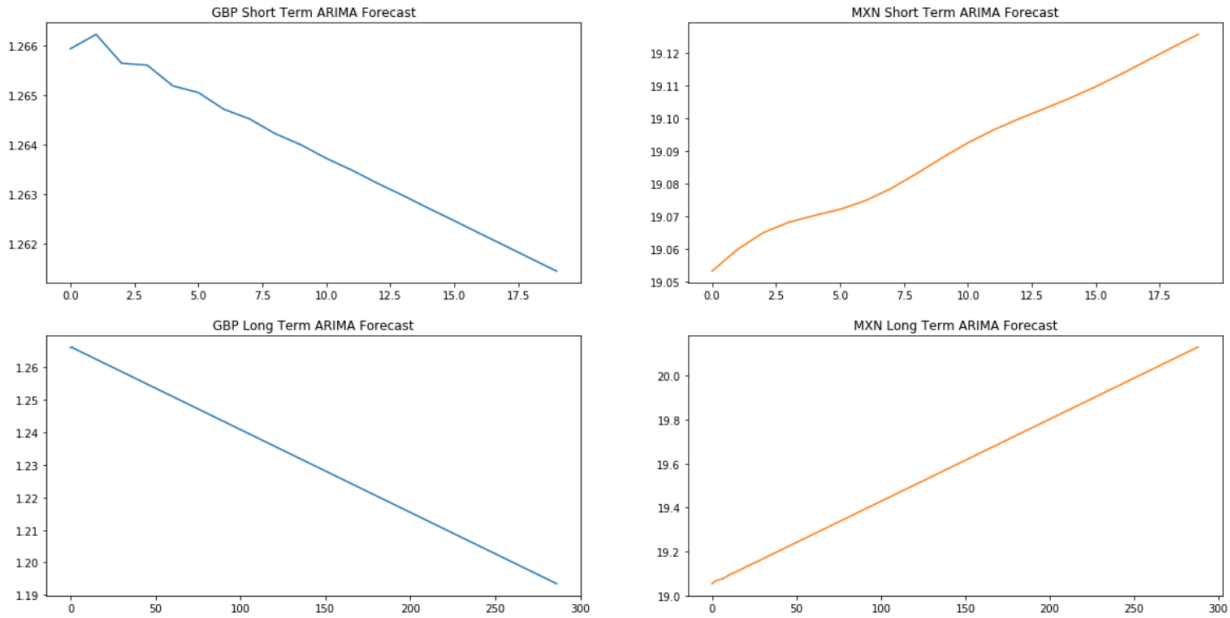
There is a wide precedent in the literature for using a grid search over a subset of candidate numbers of lags, and choosing the model with the lowest AIC. Sk takes 1284 daily observations from 2010 to 2015 for six major currencies (USD, GBP, SGD, NZD, CHF and JPY, all relative to AUD), and grid searches across *p* and *q* values of 0, 1 and 2 to identify the best model. He assumes a *d* value of 1, and finds a different optimal combination of parameters for each currency's ARIMA model (Sk, 2015). Bissoondeeal takes a much larger set of 5661 daily observations of AUD and GBP from 1986 to 2007, and scans across a wider range of *p* and *q* values from 0 to 5 in each case. He also assumes a *d* value of 1, and finds that ARIMA (5, 1, 2) best fits AUD, and (5, 1, 5) best fits GBP (Bissoondeeal, 2008). Both authors compared the performance of ARIMA models to neural networks, and got conflicting results. Sk found that ARIMA models outperform his neural network, while Bissoondeaal reached the opposite conclusion.

The other parameter to be set in an ARIMA model is the number of differences applied to the y variable before the regression, identified as *d*. In the context of time series analysis, the series must be stationary before statistical analysis can be done. The time series of a financial asset, including an exchange rate, is highly unlikely to be stationary. To be sure, we run an Augmented Dickey-Fuller (ADF) test on each of the six series to determine if it is stationary. With a p-value limit of 0.01, we reject the null hypothesis that the series is stationary in five out of the six cases. We cannot reject the null hypothesis in the case of the Canadian dollar.

Therefore, the differencing parameter *d* in the ARIMA model must be at least 1, except in the case of the Canadian dollar where it may be 0. It is possible to difference more than once in an ARIMA model, so a grid search is done across potential *d* value from 1 to 3, and from 0 to 3 in the case of the Canadian dollar.

The best-fit ARIMA model as identified by the grid search varies for each currency pair. In the case of EUR, (1, 1, 1) is the optimal fit, the most parsimonious among the six. JPY, CAD, and ZAR are optimally fit by (2, 1, 2), meaning one differencing of the time series, and the inclusion of two lags of the dependent variable and two lags of the error term in the regression. GBP and MXN are both optimally fit by a model that includes yet another autoregressive term, *y(t-3)*.

Figure 5: Best-Fit ARIMA Results (GBP and MXN shown)



As can be seen in Figure 5, the best-fit ARIMA model does not do any better than the random walk. As was the case with the random walk, a zoom in on the first 30 observations of the test set is shown in the upper panels, while the bottom panels include the entire test set, which is about one full year of daily observations.

The best-fit ARIMA model for British Pound is (3, 1, 1). Therefore 3 lags of the input variable are included when making the forecast. This accounts for the brief wobble in the

forecast trend, after which the model quickly collapses back to the random walk result. The same can be seen for the Mexican peso, where the best-fit ARIMA model was (2, 1, 2). In other words, the ARIMA models are almost entirely informed only by the last observation in the training set, as was the case with the random walk.

As is shown in the full results table in the next section, the mean squared error of the forecasts for the random walk and the best-fit ARIMA models are almost identical for all six currency pairs. This is despite the ARIMA parameters having been selected over a wide range of permutations of possible values. This implies that with the assumption of a linear form, a univariate autoregression does not have any predictive power for the path of a currency. The result is thus the same as that reached by Meese and Rogoff as far as linear models are concerned, i.e. that the best-fit ARIMA model does not have predictive power relative to the random walk.

D. Neural Network Results

The LSTM model requires us to specify the number of previous observations to include in its estimation. With the ARIMA model, a grid search for ideal number of autoregressive terms ("*p*" values) was never any larger than 3, while the LSTM will be set to 10 previous time steps. Therefore the predictive power of an individual observation can be fed into the the model's prediction up to two weeks in advance. This parameter represents the main advantage of the long short-term memory model over the ARIMA model, in that previous time steps are able to contribute to the output prediction with an unlimited distance into the past.

As was the case with the random walk and the ARIMA model, the data set is split into a training set and the test set, and the training set contains the first 80% of daily observations while the test set contains the last 20% of daily observations, or roughly the last calendar year of the data set.

Another key parameter the long short-term memory model requires is the number of training epochs to be applied to the training set. One training epoch means that the entire data set is passed forward and backwards through the neural network only once. An increased number of epochs will prevent overfitting. With all six currency pairs, the loss function

approaches zero quickly, within 2-3 epochs, so the epoch number is set to 5 for all six currency pairs. Attempting larger epoch sizes increases computational time significantly and does not improve the results.

The final parameter to tune in the neural network is the batch size, which is the total number of training examples present in a single batch. Adjusting the batch size does not seem to affect the results much either, so we leave the batch size set to 1. That is, the training set will be treated as a single entity with the model trained on the entire training set with each epoch.

Figure 6: Results Table (Mean Squared Error for Each Model)

Results Table (MSE = Mean Squared Error)

	ADF Test	Best ARIMA	AIC	MSE ARIMA	MSE RW	MSE LSTM NN
Currency	(p-value)					
EUR	0.033	(0,1,1)	-8376	0.0002	0.0002	2*e^-5
GBP	0.389	(3,1,1)	-7679	0.0027	0.0027	0.0001
JPY	0.163	(2,1,2)	2131	2.32	2.32	0.311
CAD	0.005	(1,1,0)	-8275	0.0017	0.0017	7*e^-5
MXN	0.334	(2,1,2)	-1120	3.65	3.65	0.272
ZAR	0.478	(0,1,1)	-1162	2.24	2.25	0.094

Figure 6 displays the results of the mean squared error comparison for the three models. The Augmented Dickey-Fuller test was conducted to determine if the time series of each currency pair is stationary. Interestingly in the case of CAD, we cannot reject the null hypothesis that the time series is stationary with a p-value threshold of 0.01. This reflects that in the period of 2015-2020, the Canadian did not exhibit much of a trend. However, the path of a financial asset is highly unlikely to be stationary over time. Therefore the data set is differenced at least once when running the random walk and ARIMA models. The next column displays the best-fit *p, d,* and *q* parameters as selected by the minimum Akaike Information Criterion (AIC) in the next column. We select the best-fit ARIMA model based on the lowest AIC measure among the options available in the parameter grid search, regardless of absolute value.

The last three columns of Figure 6 are the mean squared error comparison among the three models. The mean squared error of the random walk and the best-fit ARIMA model for each currency pair is almost exactly the same, which is in line with the conclusion of Meese and Rogoff. The last column shows the mean squared error of the long short-term memory neural network for each currency, and shows an improvement over both the random walk and the ARIMA model. This is an encouraging sign that the neural network may be informative.

Whereas the ARIMA and random walk models returned a straight line prediction forecast on the test set, reflecting a constant daily rate of return that is identical to the final training set observation in the case of the random walk, and roughly the same result for the ARIMA model, the neural network produces a forecast that is close to the actual path of the currency.

As the visuals below show, the neural network forecast closely tracks the previous time step throughout the test set. This suggests that the neural network has trained the model to regularly update the forecast throughout the test set, rather than simply mimic the daily return of the last observation of the training set. This is already a substantial improvement in forecast ability relative to the linear model.

Figure 7: LSTM Neural Network Result (GBPUSD)

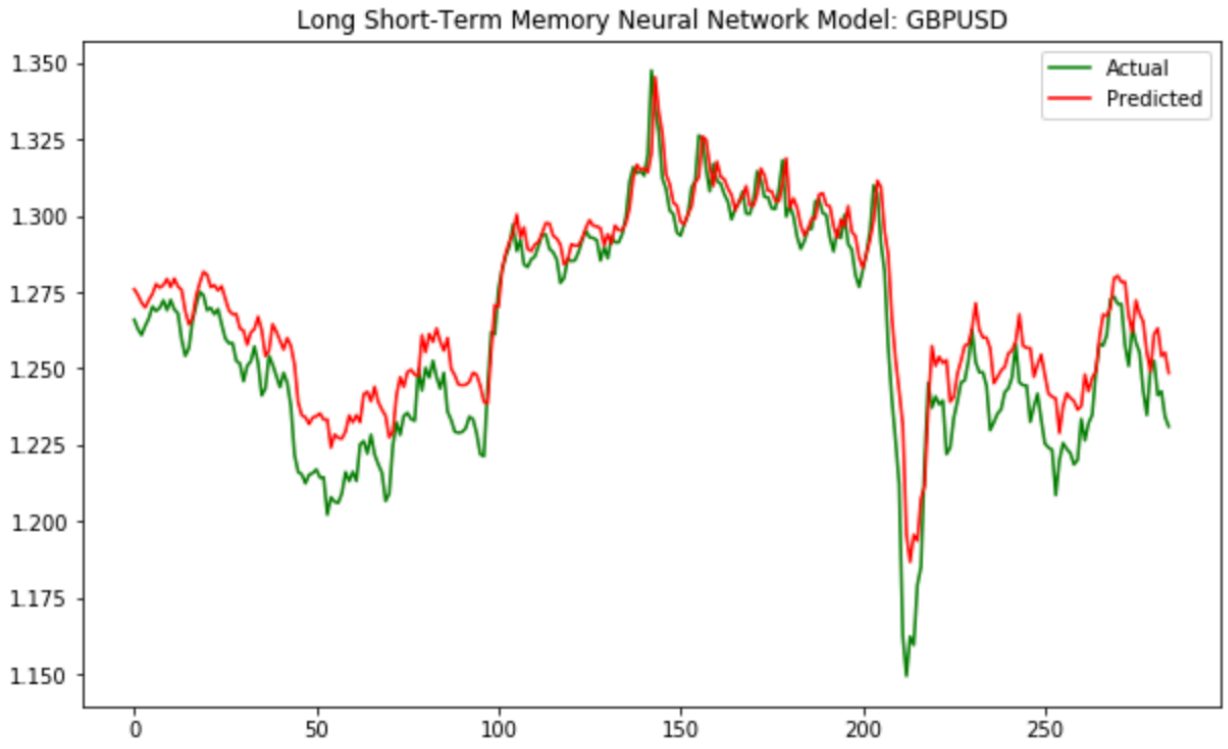


Figure 8: LSTM Neural Network Result (EURUSD)

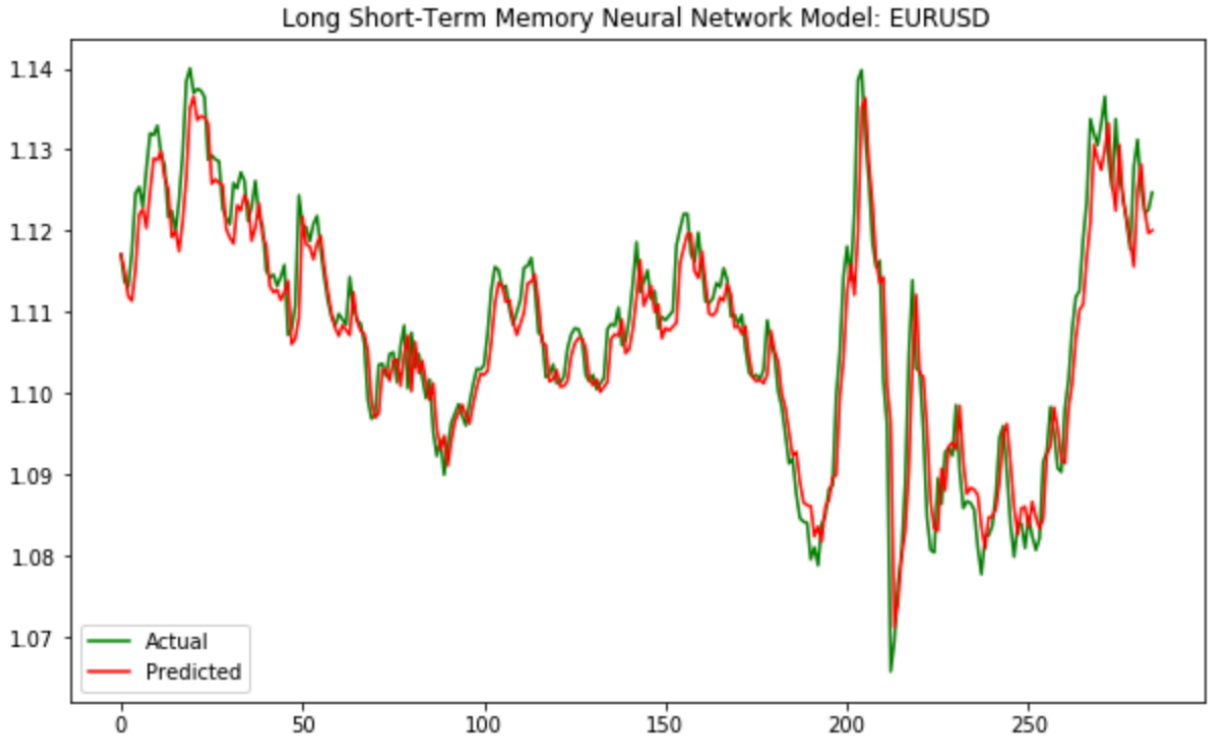


Figure 9: LSTM Neural Network Result (USDJPY)

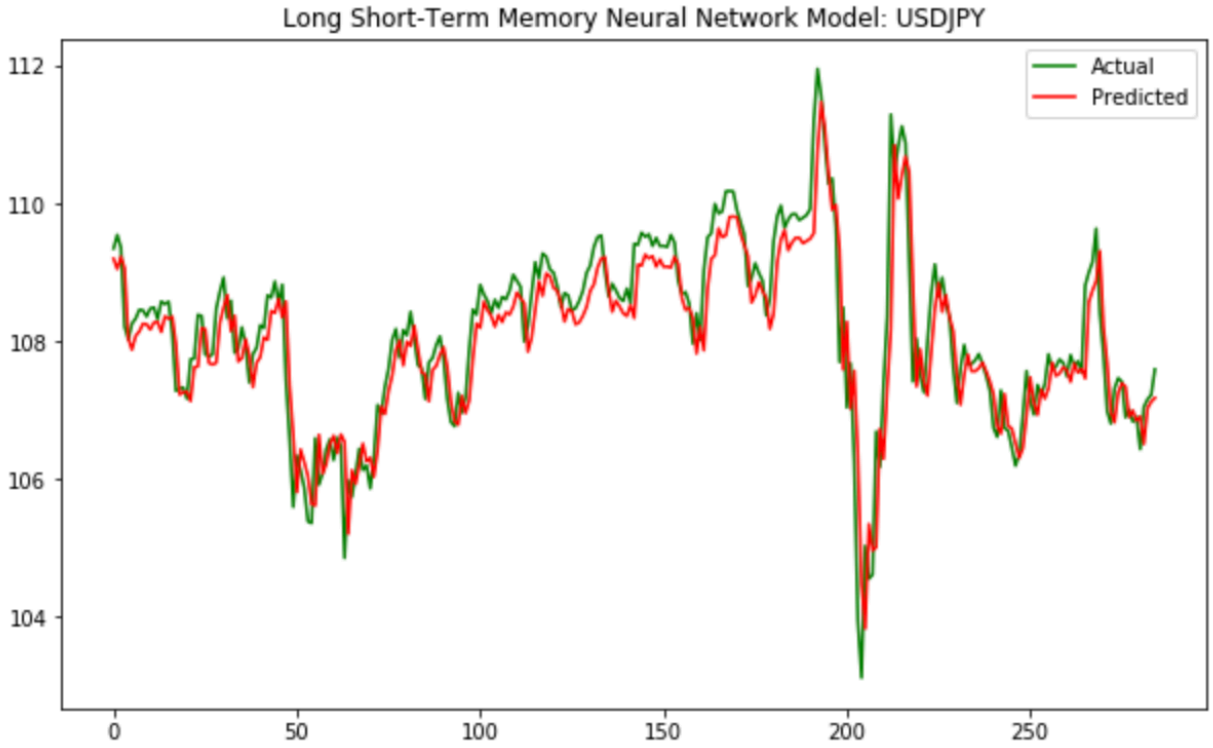


Figure 10: LSTM Neural Network Result (USDCAD)

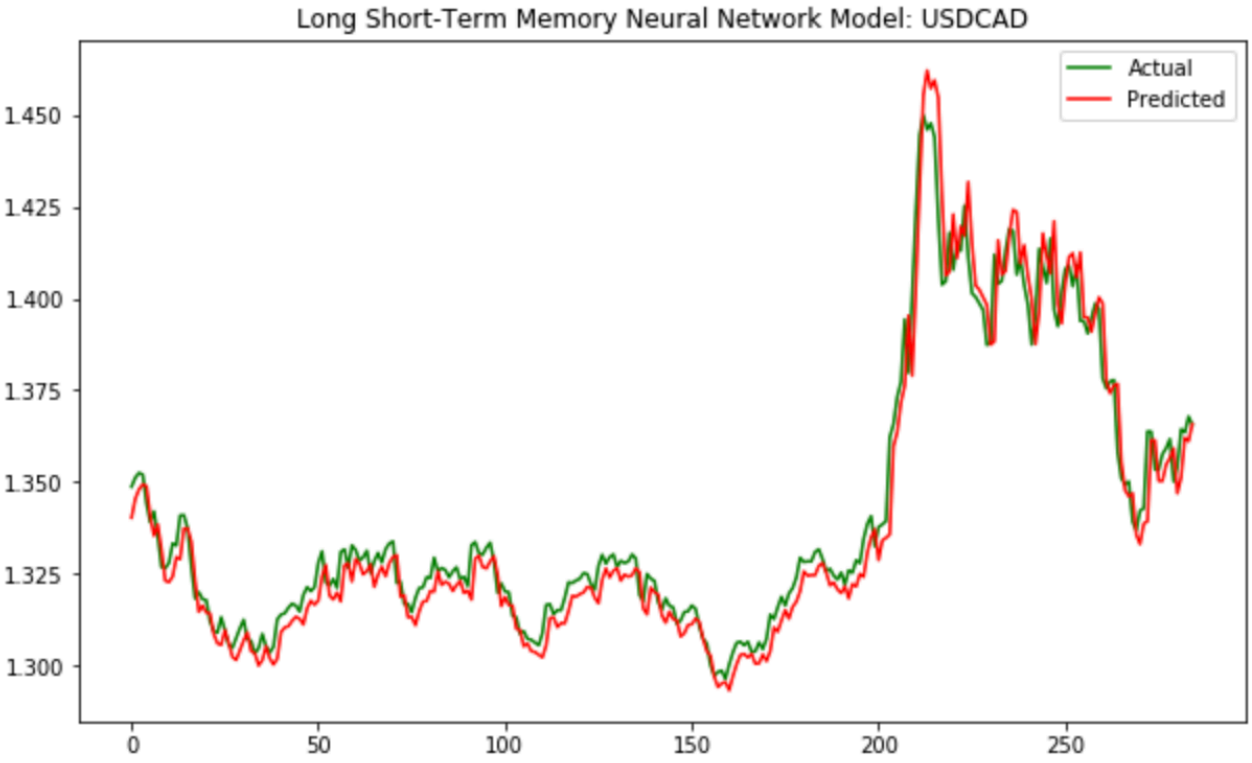


Figure 11: LSTM Neural Network Result (USDMXN)

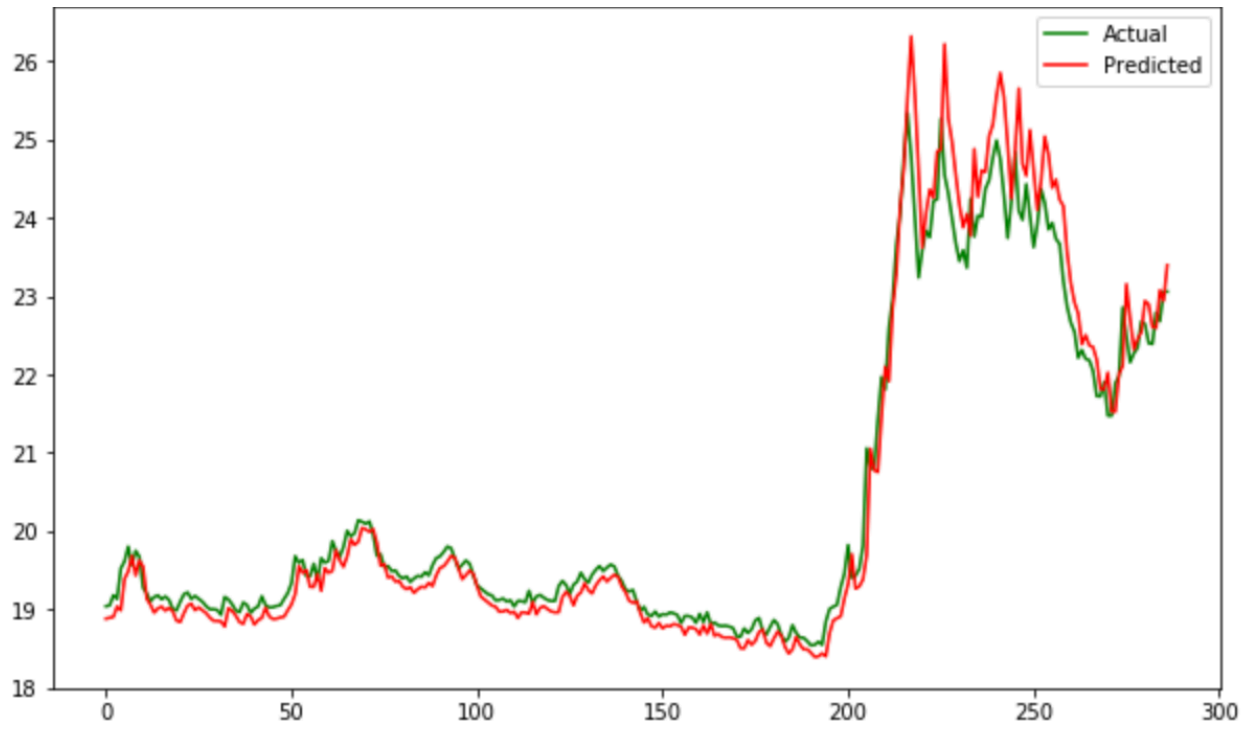
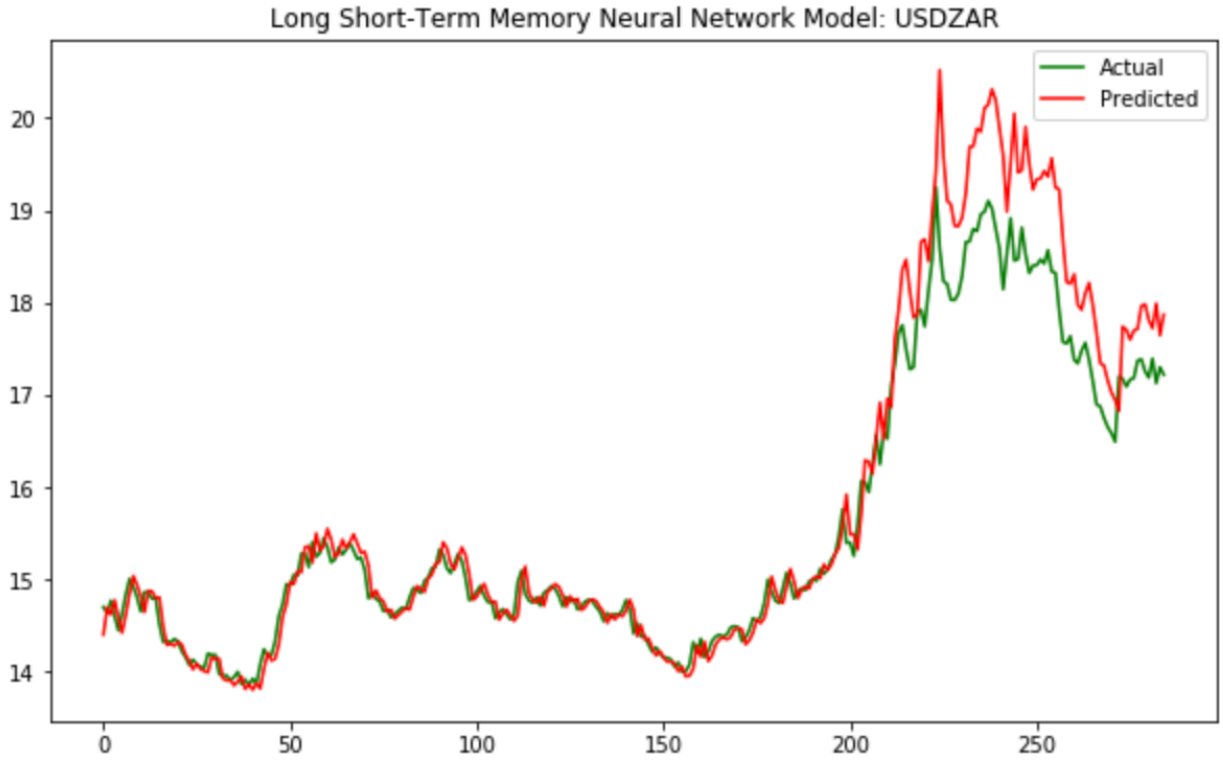


Figure 12: LSTM Neural Network Result (USDZAR)



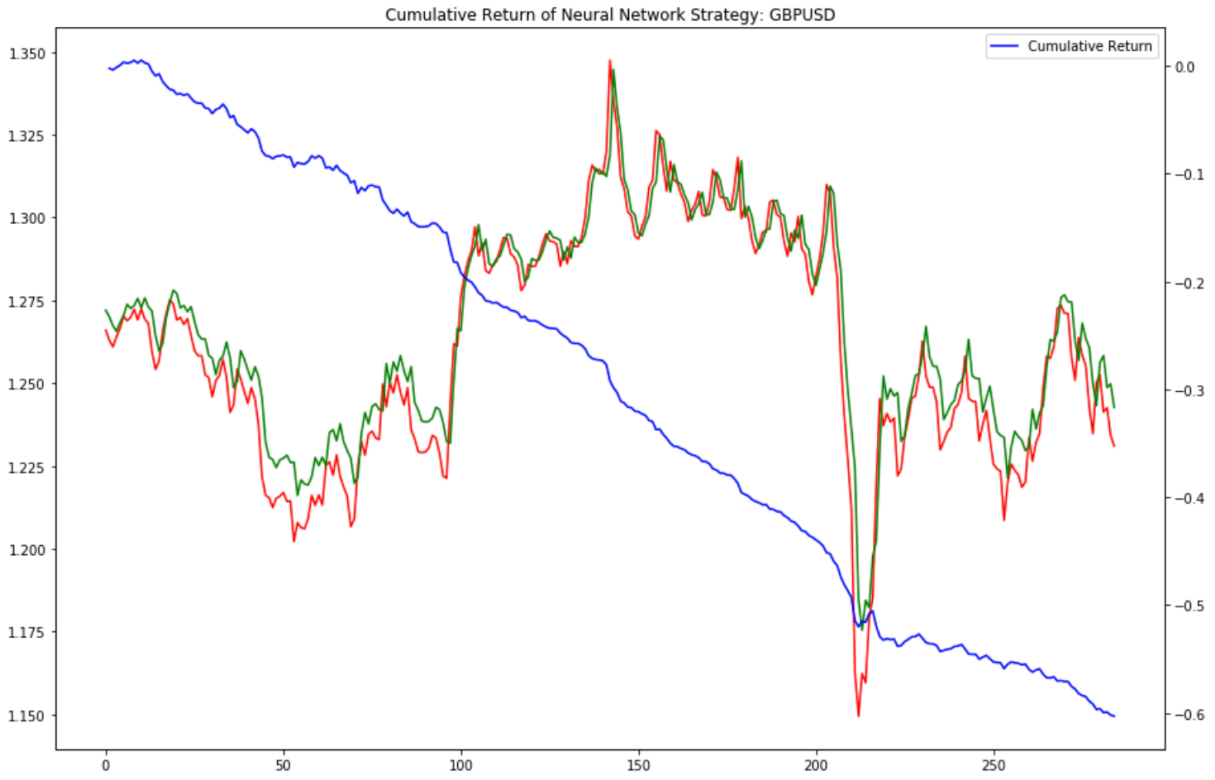
The visuals suggest that the neural network models for EUR, CAD, and JPY very closely track the path of the actual currency. This is not very encouraging and suggests the model may suffer from overfitting. It is noteworthy that this occurs with the three currencies that are relatively less volatile. In the case of CAD, we could not even reject the null hypothesis that the time series is stationary with the Augmented Dickey-Fuller test.

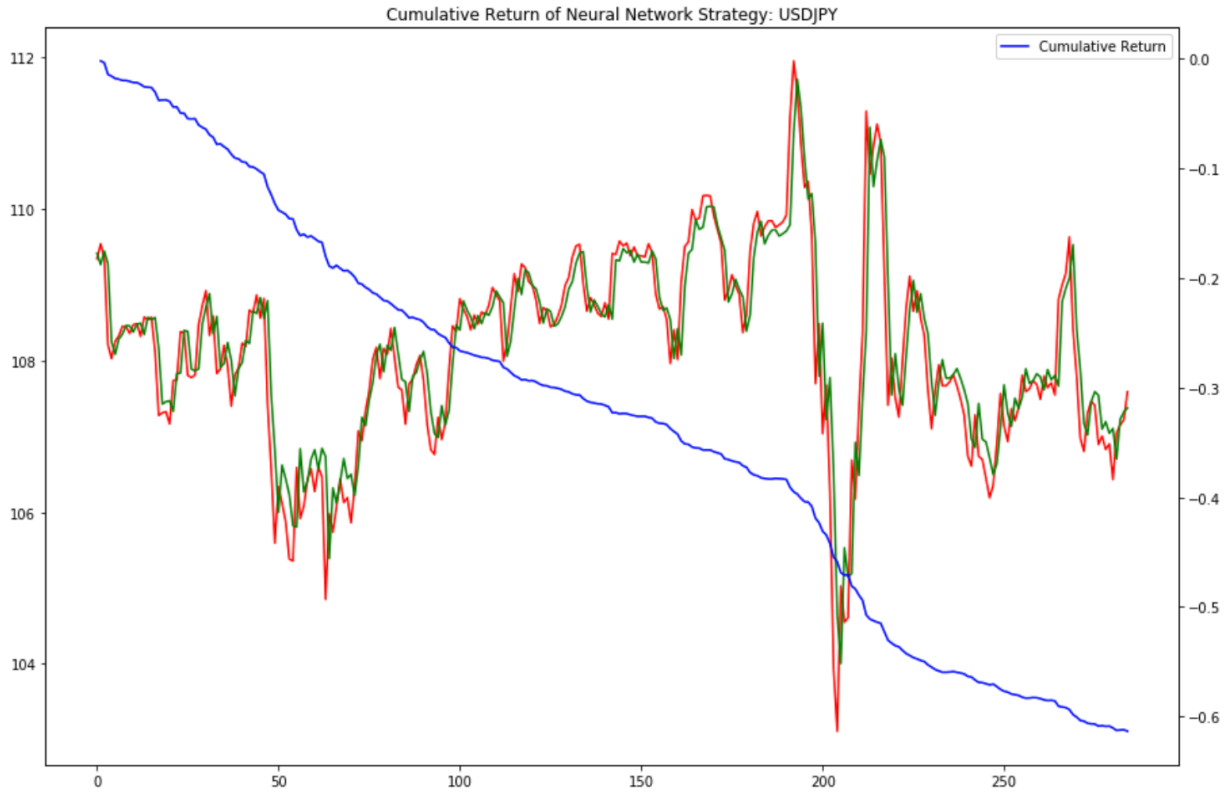
The result is more interesting in the case of the more volatile currencies; GBP, MXN and ZAR. GBP is the only model out of the six attempted in which the forecast deviates from the actual path of the currency from the very beginning of the test set. This suggests that the currency displayed unusual volatility throughout the duration of the training set, giving the model more information when training. ZAR and MXN closely resemble the less volatile currency models for most of the test set, but the prediction begins to deviate with the market hit a pocket of significant volatility in the spring of 2020, as reflected beginning shortly after time step 200 in the figures.

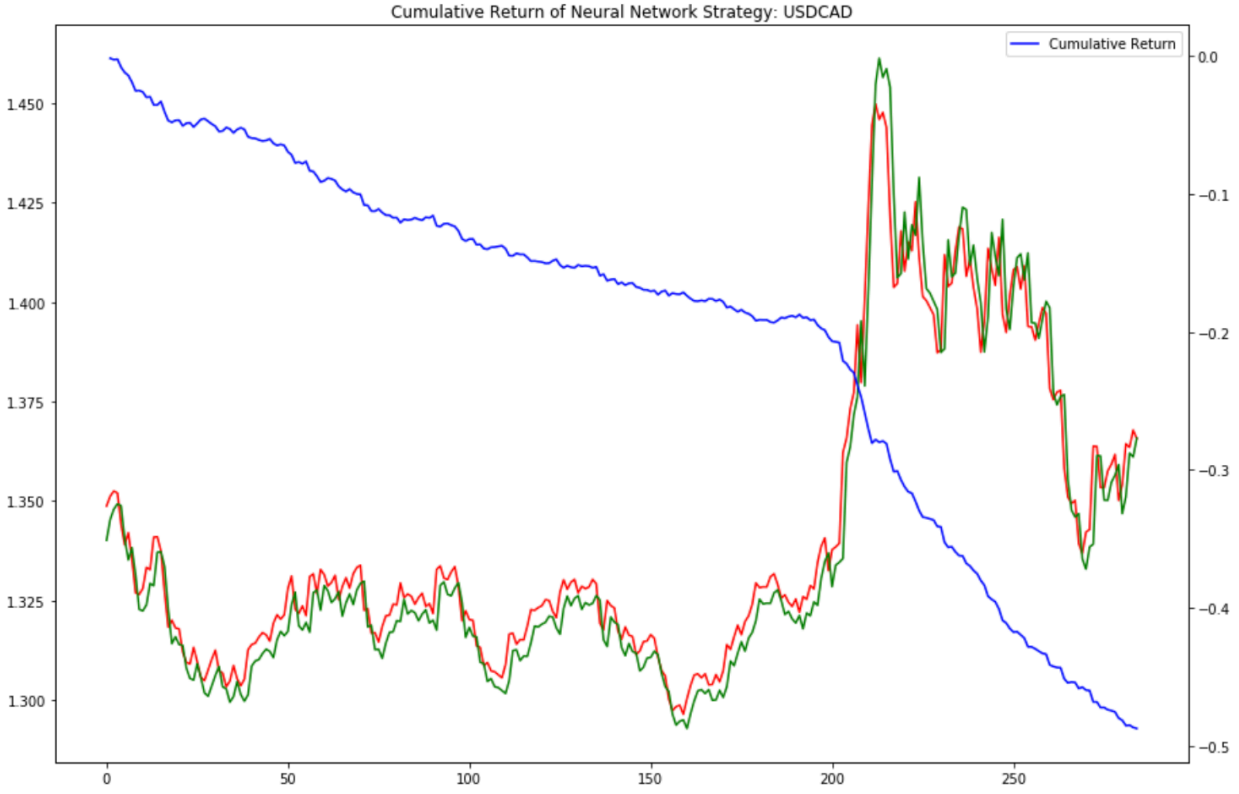
In the case of GBP, during the spring of 2020, the model suggests that the drop in the British pound's value had gone too far. The opposite result is reached in MXN and ZAR, where the model's prediction is that the value of the foreign currency should be even more volatile than the actual path of the currency.

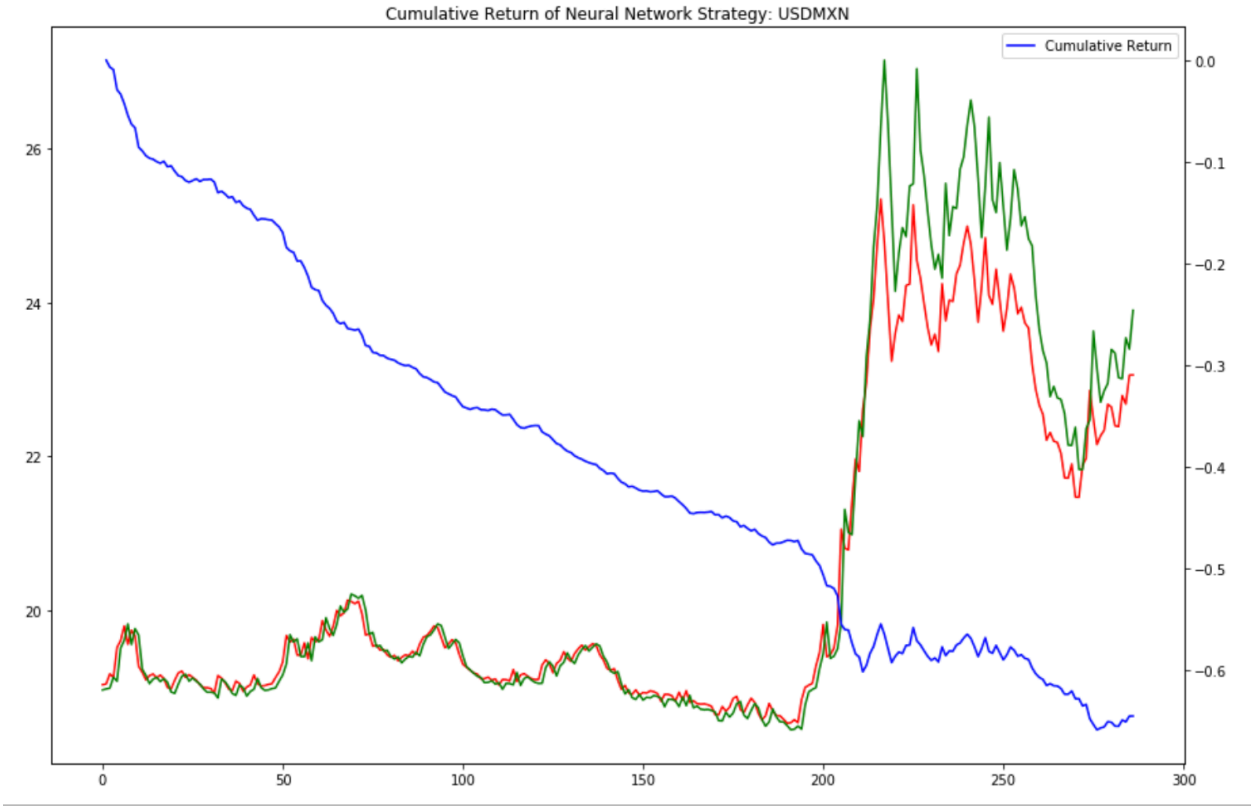
E. Profit and Loss Analysis

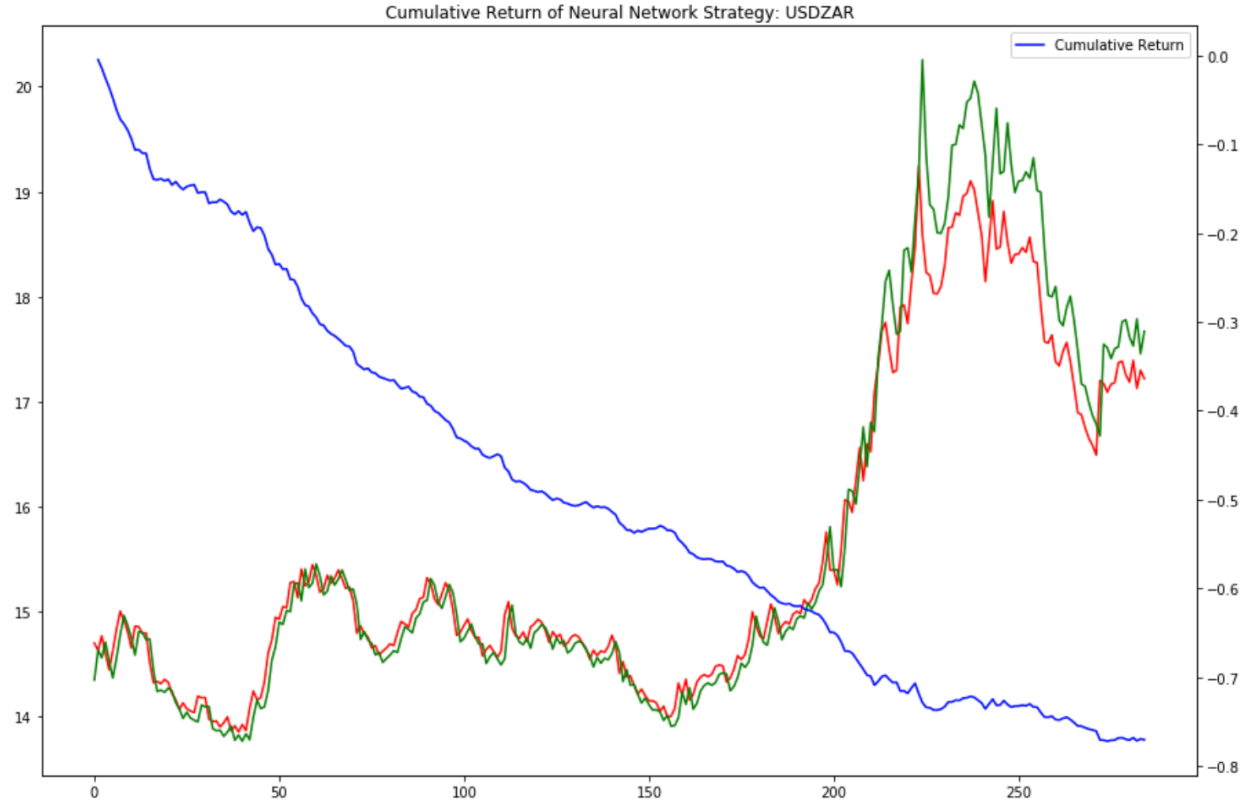
A financial asset prediction model is informative only in the sense that it can contribute to increased profits. In the case of the neural network, we can run a profit and loss analysis as follows. For each daily observation throughout the test set, we set a "long" position of the base currency (long EUR and GBP vs. USD, or long USD vs. JPY, CAD, MXN or ZAR), whenever the neural network's prediction is above the actual value of the currency. We set a "short" position of the base currency whenever the opposite is the case. The hope is that the model will guide us toward the true path of the currency over time, leading to an abnormal profit. The position can be updated at the close of each business day, i.e. with each daily observation. The visuals below show that simply sitting in this long/short position based on the prediction of the neural network leads to a gradual loss over time.

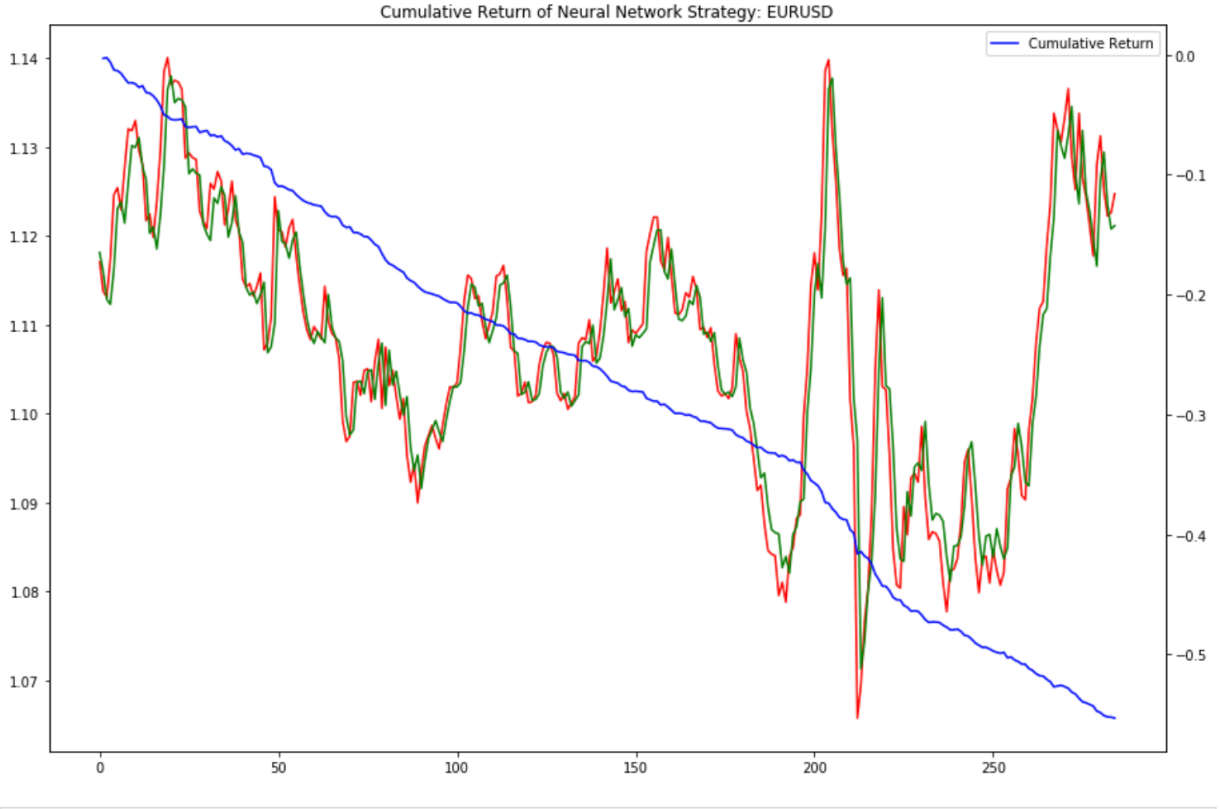



Cumulative Return of Neural Network Strategy: USDJPY
Cumulative Return
112
110
108
106
104
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
0
50
100
150
200
250


Cumulative Return of Neural Network Strategy: USDCAD
Cumulative Return
1.450
1.425
1.400
1.375
1.350
1.325
1.300
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0
50
100
150
200
250


Cumulative Return of Neural Network Strategy: USDMXN
Cumulative Return
26
24
22
20
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
0
50
100
150
200
250
300


Cumulative Return of Neural Network Strategy: USDZAR
Cumulative Return
20
19
18
17
16
15
14
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
0
50
100
150
200
250



In the figures above, the cumulative return is shown by the blue line, with the overall loss in percentage terms display on the right hand side axis. The loss becomes more exacerbated with the more volatile currencies, i.e. the CAD model loses money the slowest over time, and the ZAR model loses money the fastest over time. However, in the case of MXN and ZAR, the losses build more rapidly when the model is running close to the actual path of the currency. During the market volatility of spring 2020, the MXN and ZAR models actually stop losing money, while the losses in the CAD, JPY and EUR models accelerate. The GBP model also shows a slow down in the pace of loss whenever the model deviates from the actual path of the currency.

While a model that gradually leaks out losses cannot be deployed as a trading strategy in and of itself, the result gives some clues about the efficacy of the long short-term memory network in predicting the exchange rate. The models that are likely more overfit are the ones that will lead to more substantial losses. A model that can show more deviation from the actual path of the currency can help mitigate the pace of the loss.

IV. Conclusion

This paper contributes to the academic literature on foreign exchange rate prediction by investigating the forecasting power of neural networks, specifically a long short-term memory model. The vast majority of currency valuation models that have been developed in recent decades insist on a linear functional form, and are unable to include an indefinite number of previous time steps when informing the forecast value. The neural network shown in this paper allows for up to 10 previous time steps, as well as unspecified functional form.

The neural network was compared against a random walk and the linear ARIMA model, and the neural network manages to outperform on a mean squared error basis. However, the profit and loss analysis shows that the univariate neural network model on its own will gradually lose money, meaning it must be expanded upon before it can be utilized to directly inform investment decisions.

Perhaps the most obvious way to improve the neural network would be the inclusion of additional variables. While it has been shown that a simpler model is often better than a more complicated one, surely the univariate consideration is oversimplified. A multivariate neural network that accounts for such macroeconomic inputs such as money supply, growth, inflation and interest rates may vastly improve the precision of the model. This will form the basis for further study and the improvement of the long short-term memory network in forecasting the value of an exchange rate.

Appendix A: Bibliography

Alvarez-Diaz, Marcos. "Exchange rates forecasting: local or global methods?" *Applied Economics* 40, no. 15 (2008). https://doi.org/10.1080/00036840600905308

Bank of International Settlements (BIS) Triennial Central Bank Survey of Foreign Exchange and Over-the-counter (OTC) Derivatives Markets in 2019. (2019). Retrieved from https://www.bis.org/statistics/rpfx19.htm

Binner, Jane, Rakesh Bissoondeeal, Thomas Elger, Alicia Gazely, and Andrew Mullineux. "A Comparison of Linear Forecasting Models and Neural Networks: an Application to Euro Inflation and Euro Divisia." *Applied Economics* 37, no. 6 (2005): 665–80. https://doi.org/10.1080/0003684052000343679

Bissoondeeal, Rakesh K., Jane M. Binner, Muddun Bhuruth, Alicia Gazely, and Veemadevi P. Mootanah. "Forecasting Exchange Rates with Linear and Nonlinear Models." *Global Business and Economics Review* 10, no. 4 (2008): 414. https://doi.org/10.1504/gber.2008.020593

Bowman, Robert G. & John Buchanan. "The Efficient Market Hypothesis - A Discussion of Institutional, Agency and Behavioral Issues." *Australian Journal of Management* (1995). https://doi-org.ezproxy.cul.columbia.edu/10.1177/031289629502000203

Cheung, Yin-Wong, Menzie Chinn, Antonio Garcia Pascual, and Yi Zhang. "Exchange Rate Prediction Redux: New Models, New Data, New Currencies," *Journal of International Money and Finance (*2017). https://doi.org/10.3386/w23267

Colacito, R., Riddiough, S., & Sarno, L. "Business Cycles and Currency Returns." *National Bureau of Economic Research Working Paper* (2019). https://doi.org/10.3386/w26299

Dautel, A. J., Härdle, W. K., Lessmann, S., & Seow, H. "Forex exchange rate forecasting using deep recurrent neural networks." *Digital Finance, 2*(1-2), 69-96 (2020). https://doi.org/10.1007/s42521-020-00019-x

Fama, Eugene. F.. "Efficient Capital Markets: A Review of Theory and Empirical Work." *The Journal of Finance, 25*(2), 383 (1970). https://doi.org/10.2307/2325486

Fama, Eugene. F. "Efficient Capital Markets: II." *The Journal of Finance, 46*(5), 1575-1617 (1991). https://doi.org/10.1111/j.1540-6261.1991.tb04636.x

Faruqee, H. Long-Run Determinants of the Real Exchange Rate: A Stock-Flow Perspective. *IMF Working Papers, 94*(90), 1 (1994). doi:10.5089/9781451851359.001

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research, 270*(2), 654-669. https://doi.org/10.1016/j.ejor.2017.11.054

Galeshchuk, Svitlana. "Neural Networks Performance in Exchange Rate Prediction." *Neurocomputing* 172 (2016): 446–52. https://doi.org/10.1016/j.neucom.2015.03.100

Hopper, Gregory P. "What Determines the Exchange Rate: Economic Factors or Market Sentiment?" *Federal Reserve Bank of Philadelphia Business Review* (1997): 17-29

Hornik, K., Stinchcombe, M., & White, H. "Multilayer feedforward networks are universal approximators." *Neural Networks, 2*(5), 359-366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8

Macdonald, R., & Clark, P. B. "Exchange Rates and Economic Fundamentals: A Methodological Comparison of Beers and Feers." *IMF Working Papers, 98*(67), 1 (1998). https://doi.org/10.5089/9781451961683.001

McCulloch, W. & Pitts, W. "A logical calculus of the ideas immanent in nervous activity" *Bulletin of Mathematical Biophysics* 5, 115-133 (1943). https://doi.org/10.1007/BF02478259

Meese, Richard A., and Kenneth Rogoff. "Empirical Exchange Rate Models of the Seventies." *Journal of International Economics* 14, no. 1-2: 3–24 (1983). https://doi.org/10.1016/0022-1996(83)90017-x

Rosenberg, Michael R. *Currency Forecasting: a Guide to Fundamental and Technical Models of Exchange Rate Determination*. Chicago: Irwin, 1996

Sk, Babu As Reddy. "Exchange Rate Forecasting Using ARIMA, Neural Network and Fuzzy Neuron." *Journal of Stock & Forex Trading* 04, no. 03 (2015). https://doi.org/10.4172/2168-9458.1000155.

Williamson, J. *The exchange rate system*. Washington, D.C.: Institute for International Economics, 1985

Wong, F. (1991). "Time series forecasting using backpropagation neural networks." *Neurocomputing, 2*(4), 147-159 (1991). https://doi.org/10.1016/0925-2312(91)90045-d

Yang, H., Pan, Z., & Tao, Q. Robust and Adaptive Online Time Series Prediction with Long Short-Term Memory. *Computational Intelligence and Neuroscience, 2017*, 1-9 (2017). https://doi.org/10.1155/2017/9478952

Zhang, Y., & Wan, X. "Statistical fuzzy interval neural networks for currency exchange rate time series prediction." *Applied Soft Computing, 7*(4), 1149-1156 (2007). https://doi.org/10.1016/j.asoc.2006.01.002

"Time Series Models in Foreign Exchange"

Appendix B: Python Code

Kris Walsh (kmw2221)

Columbia University Department of Economics

Master's Thesis

Advisor: Professor Steven Ho

December 4, 2020

Import Python Packages

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import yfinance as yf
import pandas_datareader as pdr
import itertools
import warnings
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

Download FX Time Series via Yahoo Finance API

In [2]:

```
fx_list = sorted(['eurusd=x', 'gbpusd=x', 'jpy=x', 'cad=x', 'mxn=x', 'zar=x'])
start = dt.datetime(2015,1,1)
end = dt.datetime(2020,6,30)
```

In [3]:

```
fx = pdr.DataReader(fx_list, 'yahoo', start, end)['Adj Close']
```

In [4]:

```
eur = fx['eurusd=x'].dropna()
gbp = fx['gbpusd=x'].dropna()
jpy = fx['jpy=x'].dropna()
cad = fx['cad=x'].dropna()
mxn = fx['mxn=x'].dropna()
zar = fx['zar=x'].dropna()
```

In [5]:

```
fx_basket = [eur, gbp, jpy, cad, mxn, zar]
```

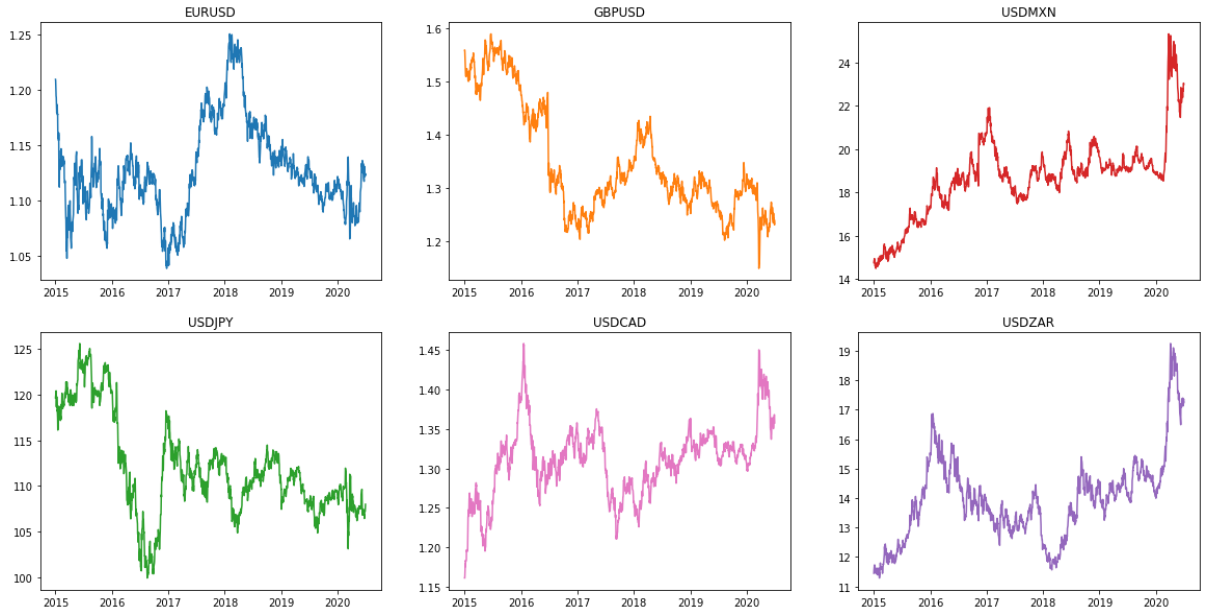
Plot of the 6 time series

In [6]:

```
fig, axs = plt.subplots(2, 3, figsize=(20,10))
axs[0, 0].plot(eur)
axs[0, 0].set_title('EURUSD')
axs[0, 1].plot(gbp, 'tab:orange')
axs[0, 1].set_title('GBPUSD')
axs[1, 0].plot(jpy, 'tab:green')
axs[1, 0].set_title('USDJPY')
axs[1, 1].plot(cad, 'tab:pink')
axs[1, 1].set_title('USDCAD')
axs[0, 2].plot(mxn, 'tab:red')
axs[0, 2].set_title('USDMXN')
axs[1, 2].plot(zar, 'tab:purple')
axs[1, 2].set_title('USDZAR')
```

Out[6]:

```
Text(0.5, 1.0, 'USDZAR')
```



ADF Test for Stationarity

In [7]:

```
def adf_test(currency):
    adf_test = adfuller(currency)
    print('stat=%.3f, p=%.3f' % adf_test[0:2])
    if adf_test[1] > 0.05:
         print('Probably not Stationary')
    else:
         print('Probably Stationary')
```

In [8]:

```
for i in fx_basket:
    adf_test(i)
```

```
stat=-3.023, p=0.033
Probably Stationary
stat=-1.783, p=0.389
Probably not Stationary
stat=-2.328, p=0.163
Probably not Stationary
stat=-3.622, p=0.005
Probably Stationary
stat=-1.896, p=0.334
Probably not Stationary
stat=-1.610, p=0.478
Probably not Stationary
```

Split into Training and Test for ARIMA and Random Walk

In [9]:

```
eur_train = eur[:1144]
eur_test = eur[1145:]
gbp_train = gbp[:1144]
gbp_test = gbp[1145:]
jpy_train = jpy[:1144]
jpy_test = jpy[1145:]
cad_train = cad[:1144]
cad_test = cad[1145:]
mxn_train = mxn[:1144]
mxn_test = mxn[1145:]
zar_train = zar[:1144]
zar_test = zar[1145:]
```

Grid Search for ARIMA Parameters

In [10]:

```
warnings.filterwarnings('ignore')
```

In [11]:

```
def grid_search(currency):
    p = range(0, 4)
    d = range(1, 3)
    q = range(0, 3)
    pdq = list(itertools.product(p, d, q))
    aics = []
    params = []
    for param in pdq:
        model = ARIMA(currency, order=param)
        model_fit = model.fit()
        aic = model_fit.aic
        aics.append(aic)
        params.append(param)
    combo = list(zip(aics, params))
    combo.sort()
    combo_array = np.array(combo)
    print(combo_array)
```

In [12]:

```
fx_train_basket = [eur_train, gbp_train, jpy_train, cad_train, mxn_train, zar_train
]
```

(Note: Grid Search may run for a few minutes)

In [13]:

```
for i in fx_train_basket:
    grid_search(i)
```

```
[[-8377.07926460606 (0, 1, 0)]
 [-8376.643272807625 (0, 1, 1)]
 [-8376.584601701035 (1, 1, 0)]
 [-8375.232795621481 (1, 1, 1)]
 [-8375.035461392245 (2, 1, 0)]
 [-8375.017673909657 (0, 1, 2)]
 [-8374.509415157665 (1, 1, 2)]
 [-8374.078956324534 (3, 1, 1)]
 [-8373.827880211886 (2, 1, 1)]
 [-8373.527394480545 (2, 1, 2)]
 [-8373.159511951706 (3, 1, 0)]
 [-8373.021895368869 (3, 1, 2)]
 [-8359.990406415993 (0, 2, 1)]
 [-8359.060025146677 (0, 2, 2)]
 [-8358.877446371665 (2, 2, 2)]
 [-8357.91392173284 (1, 2, 2)]
 [-8355.29734495656 (2, 2, 1)]
 [-8355.000927790197 (3, 2, 1)]
 [-8350.247861641381 (3, 2, 2)]
 [-8330.085198174134 (1, 2, 1)]
 [-8096.456140547243 (3, 2, 0)]
 [-8030.565245196905 (2, 2, 0)]
 [-7877.023336059554 (1, 2, 0)]
 [-7536.585038700723 (0, 2, 0)]]
[[-7678.279482555179 (3, 1, 1)]
 [-7677.820274978869 (1, 1, 2)]
 [-7677.603630289401 (2, 1, 2)]
 [-7677.314521416314 (2, 1, 1)]
 [-7677.128108194673 (3, 1, 0)]
 [-7676.901749403534 (0, 1, 0)]
 [-7676.210939763765 (3, 1, 2)]
 [-7675.466954546855 (1, 1, 0)]
 [-7675.447980059354 (0, 1, 1)]
 [-7673.849050199618 (0, 1, 2)]
 [-7673.750300338184 (2, 1, 0)]
 [-7673.508354229485 (1, 1, 1)]
 [-7660.399231183901 (0, 2, 1)]
 [-7659.752402985554 (3, 2, 2)]
 [-7658.9981579099585 (1, 2, 1)]
 [-7658.976950782977 (0, 2, 2)]
 [-7658.523502080507 (1, 2, 2)]
 [-7655.26314414807 (2, 2, 1)]
 [-7655.177851581371 (2, 2, 2)]
 [-7643.678483987898 (3, 2, 1)]
 [-7416.971673528125 (3, 2, 0)]
 [-7314.628909782701 (2, 2, 0)]
 [-7226.935138109966 (1, 2, 0)]
 [-6903.298563685793 (0, 2, 0)]]
[[2131.2095387124837 (2, 1, 2)]
 [2134.5898422175396 (1, 1, 1)]
 [2136.1087530079494 (1, 1, 2)]
 [2136.134894711842 (2, 1, 1)]
 [2137.5739697470194 (3, 1, 1)]
 [2137.676177014765 (3, 1, 2)]
 [2139.1258115848577 (0, 1, 0)]
 [2140.2639363031303 (1, 1, 0)]
 [2140.309459819902 (0, 1, 1)]
```

```
   [2141.364070237898 (2, 1, 0)]
   [2141.4615621401786 (0, 1, 2)]
   [2142.72233164838 (1, 2, 2)]
   [2142.873161767079 (3, 1, 0)]
   [2144.231097153117 (2, 2, 2)]
   [2145.7090519764943 (3, 2, 2)]
   [2147.213842070709 (0, 2, 1)]
   [2148.4001579302476 (1, 2, 1)]
   [2148.444456704413 (0, 2, 2)]
   [2149.450789456844 (2, 2, 1)]
   [2150.9968748538513 (3, 2, 1)]
   [2389.4525856937184 (3, 2, 0)]
   [2475.0256081014377 (2, 2, 0)]
   [2590.3096595335064 (1, 2, 0)]
   [2960.15474203291 (0, 2, 0)]]
  [[-8277.611633547587 (0, 1, 0)]
   [-8275.654036800324 (1, 1, 0)]
   [-8275.652609249972 (0, 1, 1)]
   [-8274.017752393835 (0, 1, 2)]
   [-8274.002087664818 (2, 1, 0)]
   [-8273.723309507408 (1, 1, 1)]
   [-8273.490768294232 (2, 1, 1)]
   [-8272.0178746326 (1, 1, 2)]
   [-8272.00556215283 (3, 1, 0)]
   [-8271.658502237226 (3, 1, 2)]
   [-8271.542414170035 (3, 1, 1)]
   [-8270.834595485863 (2, 1, 2)]
   [-8260.90668298264 (0, 2, 1)]
   [-8258.953980071385 (1, 2, 1)]
   [-8258.951817423263 (0, 2, 2)]
   [-8257.319994911993 (2, 2, 1)]
   [-8255.951764718062 (2, 2, 2)]
   [-8249.15459446466 (3, 2, 2)]
   [-8240.262403088163 (3, 2, 1)]
   [-8239.089155885284 (1, 2, 2)]
   [-8004.6050028368445 (3, 2, 0)]
   [-7944.5886144440265 (2, 2, 0)]
   [-7821.184077327072 (1, 2, 0)]
   [-7484.92375005609 (0, 2, 0)]]
  [[-1119.957824467529 (2, 1, 2)]
   [-1117.97599775974 (3, 1, 2)]
   [-1115.1726033993832 (0, 1, 0)]
   [-1113.6712455848751 (0, 1, 1)]
   [-1113.641228756887 (1, 1, 0)]
   [-1113.3651493268585 (2, 1, 1)]
   [-1113.0682488743091 (1, 1, 2)]
   [-1112.8890197910391 (0, 1, 2)]
   [-1112.7288970724571 (2, 1, 0)]
   [-1112.5690627866556 (3, 1, 1)]
   [-1112.2975274585147 (1, 1, 1)]
   [-1111.5043852757722 (3, 1, 0)]
   [-1105.0317385977482 (0, 2, 1)]
   [-1103.5383155250515 (0, 2, 2)]
   [-1103.5079199467705 (1, 2, 1)]
   [-1102.583483494404 (2, 2, 1)]
   [-1101.3521442626547 (3, 2, 1)]
   [-1101.12307263319 (1, 2, 2)]
```

```
 [-1100.102739979519 (3, 2, 2)]
 [-1099.2014239457353 (2, 2, 2)]
 [-800.6378451144383 (3, 2, 0)]
 [-759.3753152663107 (2, 2, 0)]
 [-633.8725145036392 (1, 2, 0)]
 [-345.2578603516397 (0, 2, 0)]]
[[-1161.9612498932242 (0, 1, 1)]
 [-1161.8768550305417 (2, 1, 1)]
 [-1161.800494194406 (1, 1, 2)]
 [-1161.7910364990034 (1, 1, 0)]
 [-1160.5229304255759 (2, 1, 0)]
 [-1160.380635748004 (0, 1, 2)]
 [-1160.1807483859225 (1, 1, 1)]
 [-1160.1060165158497 (0, 1, 0)]
 [-1159.8818992917736 (3, 1, 1)]
 [-1159.8817385094103 (2, 1, 2)]
 [-1159.2914508756526 (3, 1, 0)]
 [-1157.8892182881832 (3, 1, 2)]
 [-1151.1630466918255 (0, 2, 2)]
 [-1150.99468437334 (1, 2, 1)]
 [-1149.6829313385883 (2, 2, 1)]
 [-1149.2137691138237 (0, 2, 1)]
 [-1148.1840764306185 (3, 2, 1)]
 [-1147.3480408352457 (2, 2, 2)]
 [-1146.208790236417 (1, 2, 2)]
 [-1145.6712260357795 (3, 2, 2)]
 [-888.2846088970923 (3, 2, 0)]
 [-817.3508440356268 (2, 2, 0)]
 [-701.0075963441209 (1, 2, 0)]
 [-433.8462835401285 (0, 2, 0)]]
```

Run Best-Fit ARIMA Models

In [13]:

```
model_arima_eur = ARIMA(eur_train, (0,1,1))
model_arima_gbp = ARIMA(gbp_train, (3,1,1))
model_arima_jpy = ARIMA(jpy_train, (2,1,2))
model_arima_cad = ARIMA(cad_train, (1,1,0))
model_arima_mxn = ARIMA(mxn_train, (2,1,2))
model_arima_zar = ARIMA(zar_train, (0,1,1))
```

In [14]:

```
model_arima_fit_eur = model_arima_eur.fit()
model_arima_fit_gbp = model_arima_gbp.fit()
model_arima_fit_jpy = model_arima_jpy.fit()
model_arima_fit_cad = model_arima_cad.fit()
model_arima_fit_mxn = model_arima_mxn.fit()
model_arima_fit_zar = model_arima_zar.fit()
```

In [15]:

```
arima_pred_eur = model_arima_fit_eur.forecast(len(eur_test), alpha=0.05)[0]
arima_pred_gbp = model_arima_fit_gbp.forecast(len(gbp_test), alpha=0.05)[0]
arima_pred_jpy = model_arima_fit_jpy.forecast(len(jpy_test), alpha=0.05)[0]
arima_pred_cad = model_arima_fit_cad.forecast(len(cad_test), alpha=0.05)[0]
arima_pred_mxn = model_arima_fit_mxn.forecast(len(mxn_test), alpha=0.05)[0]
arima_pred_zar = model_arima_fit_zar.forecast(len(zar_test), alpha=0.05)[0]
```

Report MSE of ARIMA vs Actual Currency Path

In [16]:

```
print(mean_squared_error(eur_test,arima_pred_eur))
print(mean_squared_error(gbp_test,arima_pred_gbp))
print(mean_squared_error(jpy_test,arima_pred_jpy))
print(mean_squared_error(cad_test,arima_pred_cad))
print(mean_squared_error(mxn_test,arima_pred_mxn))
print(mean_squared_error(zar_test,arima_pred_zar))
```

```
0.0001941545578032443
0.0026633357811622106
2.321791999598577
0.0017357273828908875
3.6527211112489475
2.2369263000545745
```

Zoom in on ARIMA forecasts, short (20 days) and long (286 days)

In [17]:

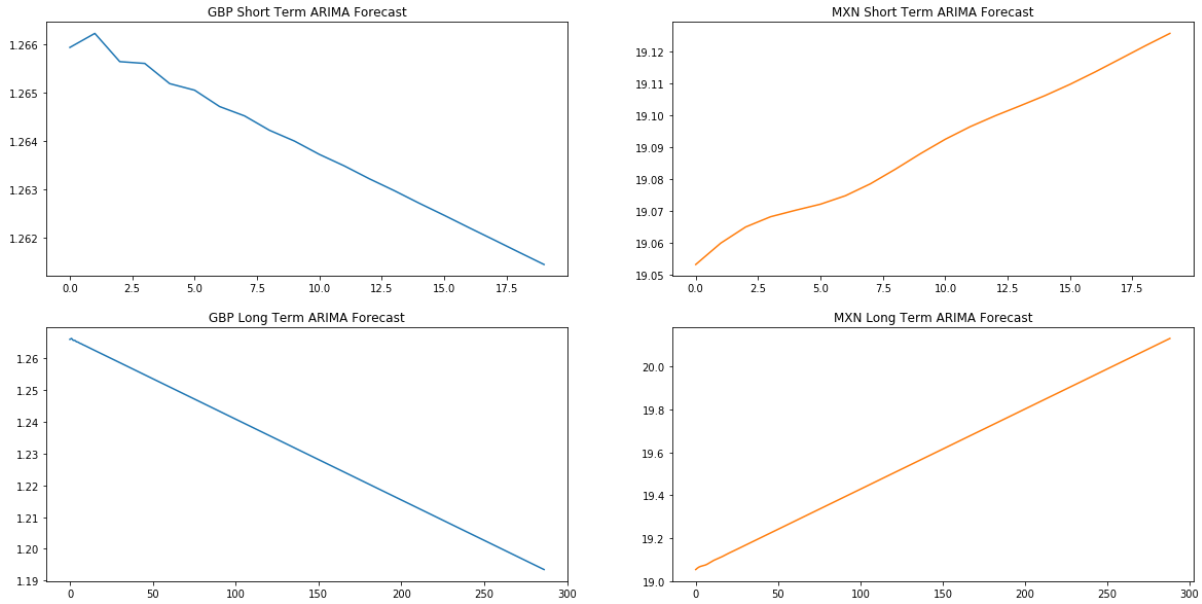
```
arima_pred_mxn_short = model_arima_fit_mxn.forecast(20, alpha=0.05)[0]
arima_pred_gbp_short = model_arima_fit_gbp.forecast(20, alpha=0.05)[0]
```

In [18]:

```
fig, axs = plt.subplots(2, 2, figsize=(20,10))
axs[0, 0].plot(arima_pred_gbp_short)
axs[0, 0].set_title('GBP Short Term ARIMA Forecast')
axs[1, 0].plot(arima_pred_gbp)
axs[1, 0].set_title('GBP Long Term ARIMA Forecast')
axs[0, 1].plot(arima_pred_mxn_short, 'tab:orange')
axs[0, 1].set_title('MXN Short Term ARIMA Forecast')
axs[1, 1].plot(arima_pred_mxn, 'tab:orange')
axs[1, 1].set_title('MXN Long Term ARIMA Forecast')
```

Out[18]:

```
Text(0.5, 1.0, 'MXN Long Term ARIMA Forecast')
```



Random Walk Results

In [19]:

```
model_arima_eur_rw = ARIMA(eur_train, (0,1,0))
model_arima_gbp_rw = ARIMA(gbp_train, (0,1,0))
model_arima_jpy_rw = ARIMA(jpy_train, (0,1,0))
model_arima_cad_rw = ARIMA(cad_train, (0,1,0))
model_arima_mxn_rw = ARIMA(mxn_train, (0,1,0))
model_arima_zar_rw = ARIMA(zar_train, (0,1,0))
```

In [20]:

```
model_arima_fit_eur_rw = model_arima_eur_rw.fit()
model_arima_fit_gbp_rw = model_arima_gbp_rw.fit()
model_arima_fit_jpy_rw = model_arima_jpy_rw.fit()
model_arima_fit_cad_rw = model_arima_cad_rw.fit()
model_arima_fit_mxn_rw = model_arima_mxn_rw.fit()
model_arima_fit_zar_rw = model_arima_zar_rw.fit()
```

In [21]:

```
arima_pred_eur_rw = model_arima_fit_eur_rw.forecast(len(eur_test), alpha=0.05)[0]
arima_pred_gbp_rw = model_arima_fit_gbp_rw.forecast(len(gbp_test), alpha=0.05)[0]
arima_pred_jpy_rw = model_arima_fit_jpy_rw.forecast(len(jpy_test), alpha=0.05)[0]
arima_pred_cad_rw = model_arima_fit_cad_rw.forecast(len(cad_test), alpha=0.05)[0]
arima_pred_mxn_rw = model_arima_fit_mxn_rw.forecast(len(mxn_test), alpha=0.05)[0]
arima_pred_zar_rw = model_arima_fit_zar_rw.forecast(len(zar_test), alpha=0.05)[0]
```

In [22]:

```
print(mean_squared_error(eur_test,arima_pred_eur_rw))
print(mean_squared_error(gbp_test,arima_pred_gbp_rw))
print(mean_squared_error(jpy_test,arima_pred_jpy_rw))
print(mean_squared_error(cad_test,arima_pred_cad_rw))
print(mean_squared_error(mxn_test,arima_pred_mxn_rw))
print(mean_squared_error(zar_test,arima_pred_zar_rw))
```

```
0.00019388120775006782
0.002674872365574409
2.3232498934950008
0.001734354839061851
3.6541375740629225
2.251010706606456
```

Zoom in on Random Walk forecasts, short (20 days) and long (286 days)

In [23]:

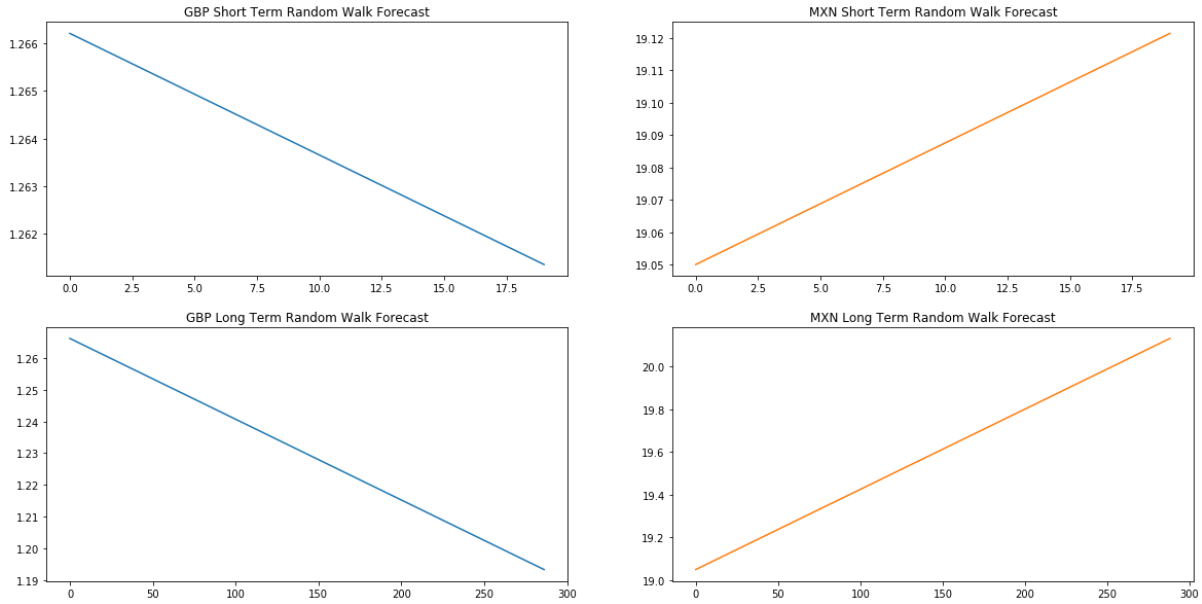
```
arima_pred_mxn_short_rw = model_arima_fit_mxn_rw.forecast(20, alpha=0.05)[0]
arima_pred_gbp_short_rw = model_arima_fit_gbp_rw.forecast(20, alpha=0.05)[0]
```

In [24]:

```
fig, axs = plt.subplots(2, 2, figsize=(20,10))
axs[0, 0].plot(arima_pred_gbp_short_rw)
axs[0, 0].set_title('GBP Short Term Random Walk Forecast')
axs[1, 0].plot(arima_pred_gbp_rw)
axs[1, 0].set_title('GBP Long Term Random Walk Forecast')
axs[0, 1].plot(arima_pred_mxn_short_rw, 'tab:orange')
axs[0, 1].set_title('MXN Short Term Random Walk Forecast')
axs[1, 1].plot(arima_pred_mxn_rw, 'tab:orange')
axs[1, 1].set_title('MXN Long Term Random Walk Forecast')
```

Out[24]:

```
Text(0.5, 1.0, 'MXN Long Term Random Walk Forecast')
```



Neural Network (Long Short-Term Memory)

In [311]:

```
df = gbp
```

In [312]:

```
scaler = MinMaxScaler()
```

In [313]:

```
df = np.array(df).reshape(-1,1)
```

In [314]:

```
df = scaler.fit_transform(df)
```

In [315]:

```
#Training and test sets
train = df[:1145]
test = df[1145:]
```

In [316]:

```
def get_data(data, look_back):
  data_x, data_y = [],[]
  for i in range(len(data)-look_back-1):
    data_x.append(data[i:(i+look_back),0])
    data_y.append(data[i+look_back,0])
  return np.array(data_x) , np.array(data_y)

look_back = 1

x_train , y_train = get_data(train, look_back)
```

In [317]:

```
x_test , y_test = get_data(test,look_back)
```

In [318]:

```
#Processing train and test sets for LSTM model
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)
```

In [319]:

```
print(x_train.shape)
print(x_test.shape)
```

```
(1143, 1, 1)
(285, 1, 1)
```

In [320]:

```
#Defining the LSTM model
n_features=x_train.shape[1]
model=Sequential()
model.add(LSTM(10,activation='relu',input_shape=(1,1)))
model.add(Dense(n_features))
```

In [321]:

```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_9 (LSTM)                (None, 10)                480
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 11
=================================================================
Total params: 491
Trainable params: 491
Non-trainable params: 0
_________________________________________________________________
```

In [322]:

```
#Compiling
model.compile(optimizer='adam', loss = 'mse')
```

In [323]:

```
#Training
model.fit(x_train,y_train, epochs = 5, batch_size=1)
```

```
Epoch 1/5
1143/1143 [==============================] - 1s 829us/step - loss: 0.04
78
Epoch 2/5
1143/1143 [==============================] - 1s 799us/step - loss: 0.00
32
Epoch 3/5
1143/1143 [==============================] - 1s 790us/step - loss: 6.56
29e-04
Epoch 4/5
1143/1143 [==============================] - 1s 797us/step - loss: 6.21
70e-04
Epoch 5/5
1143/1143 [==============================] - 1s 798us/step - loss: 6.05
41e-04 0s - loss: 6.0431e-0
```

Out[323]:

```
<tensorflow.python.keras.callbacks.History at 0x150fed990>
```

In [324]:

```
#Prediction using the trained model
scaler.scale_
y_pred = model.predict(x_test)
y_pred = scaler.inverse_transform(y_pred)
```

In [325]:

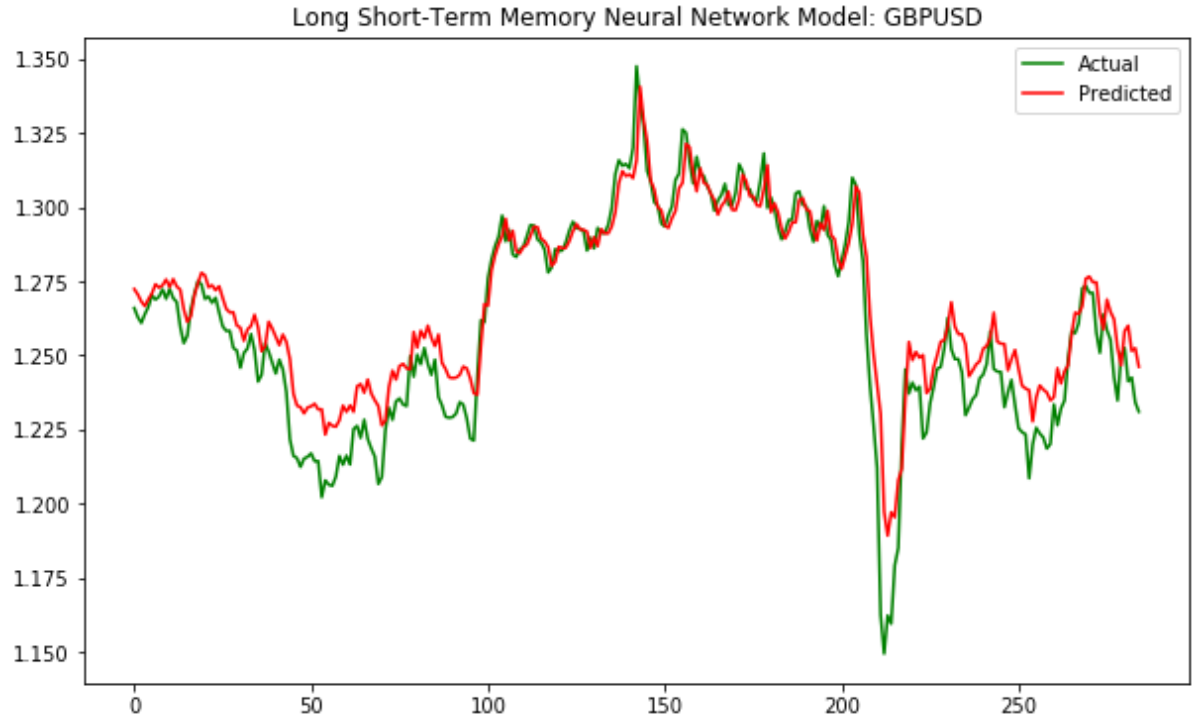
```
#Processing test shape
y_test = np.array(y_test).reshape(-1,1)
y_test = scaler.inverse_transform(y_test)
```

In [326]:

```
#Visualizing the results
plt.figure(figsize=(10,6))
plt.title('Long Short-Term Memory Neural Network Model: GBPUSD')
plt.plot(y_test , label = 'Actual', color = 'g')
plt.plot(y_pred , label = 'Predicted', color = 'r')
plt.legend()
```

Out[326]:

```
<matplotlib.legend.Legend at 0x1512f2210>
```



In [327]:

```
mean_squared_error(y_test, y_pred)
```

Out[327]:

```
0.00015780838623259154
```

In [328]:

```
y_diff = y_pred - y_test
```

In [329]:

```
y_diff_df = pd.DataFrame(data=y_diff)
```

In [330]:

```
y_pred_df = pd.DataFrame(data=y_pred)
```

In [331]:

```
y_actual_df = pd.DataFrame(data=y_test)
```

In [332]:

```
full_table = pd.concat([y_actual_df, y_pred_df, y_diff_df], axis=1)
```

In [333]:

```
table_columns = ['Actual', 'Predicted', 'Gap']
```

In [334]:

```
full_table.columns = table_columns
```

In [335]:

```
full_table['Position'] = np.where(full_table['Gap'] >= 0, 1, -1)
```

In [336]:

```
full_table['Prior Actual'] = full_table['Actual'].shift(1)
```

In [337]:

```
full_table['Return'] = (full_table['Actual'] - full_table['Prior Actual'])*(full_ta
ble['Position'])
```

In [338]:

```
full_table['Return_Percent'] = full_table['Return']/full_table['Actual']
```

In [339]:

```
full_table['Cum Ret'] = ((full_table['Return_Percent'] + 1).cumprod() - 1)
```

In [341]:

```
fig, ax_left = plt.subplots(figsize = (15,10))
ax_right = ax_left.twinx()
ax_left.plot(full_table['Actual'], color='r', label = 'Actual')
ax_left.plot(full_table['Predicted'], color = 'g', label = 'Predicted')
ax_right.plot(full_table['Cum Ret'], color='b', label = 'Cumulative Return')
plt.legend()
plt.title('Cumulative Return of Neural Network Strategy: GBPUSD')
```

Out[341]:

```
Text(0.5, 1.0, 'Cumulative Return of Neural Network Strategy: GBPUSD')
```