# Machine Learning Engineer Nanodegree

## Capstone Project

Andy Wilson
May 2, 2017

## I. Definition

### Project Overview

This project aims to construct a predictive financial model, capable of estimating the future price movement of traded financial instruments based primarily on historical price data.

This is a challenging project, since there are a range of views held as to whether this is feasible. On one hand, the Efficient Market Hypothesis (EMH) [1] argues that for efficient markets, it should not be possible to predict future prices, based on historic price information alone. On the other hand, investing strategies which rely on exactly this approach, such as momentum and trend-following approaches [2] do so with empirical success under certain conditions, suggesting that there may indeed be structure in historic price information which can be used to predict future price movements.

There are many factors that influence the price of a financial product traded on any particular market at a given time. These range from the sentiment and demand for the product by the market participants/investors to the dynamics of the systems used to operate the marketplace itself.

The hypothesis proposed here is that some of these factors are not completely independent of recent changes in the price of the product, and that by analysing enough data, some structure may be found in historic price movements that would be a predictor of future price movements.

It is appreciated that any such structure discoverable is likely to be a very weak underlying signal amid a lot of noise. Therefore it's entirely feasible, indeed likely, that any predictive signal obtained, would not be strong enough to trade profitably on its own. Trading any systematic trading strategy incurs execution costs, typically due to price spreads (the difference between the cost to buy and that to sell a given product) and fees or commissions charged by the various brokers or marketplaces involved.

Nevertheless, it is proposed that even the identification of a weak but measurable signal would be a valuable contribution, since it would confirm the existence of predictive structure in historic price series, and could potentially be incorporated as an additional input into existing systematic investment strategies to increase their profitability or diversity.

#### Dataset and inputs

To have confidence in any possible discovery of a weak signal a lot of data is required. Hence the direction of the project was driven to some extent by the availability of such data. Whilst there is a wealth of financial data potentially available from market data vendors such as Bloomberg or Reuters, most of this comes at significant cost.

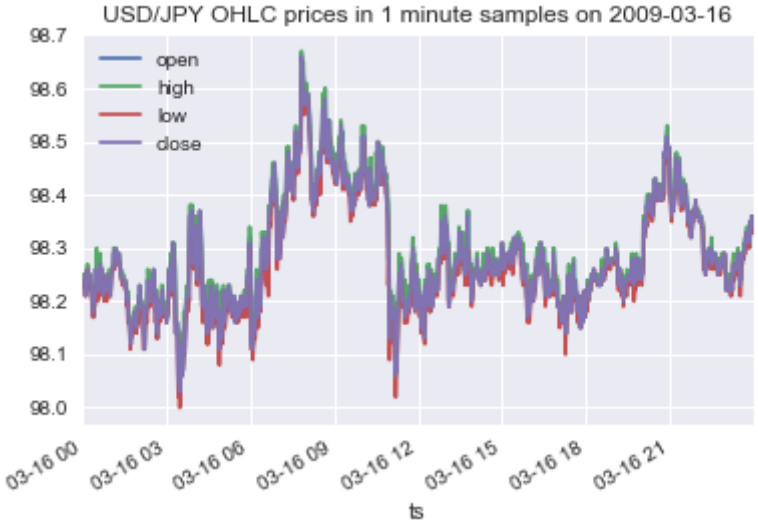
However, a source of freely available Foreign Exchange (FX) and Precious Metals (PM) price data was found at [4], which provides historical open/high/low/close prices sampled at 1 minute intervals across many currency pairs. In most cases, this data extends back to 2009, giving around 2.5 million samples per currency pair.

This led to the proposal to attempt the prediction of short term price movements of FX currency pairs, based on recent price history.

An example extract illustrating the nature of the data available is shown below, in this case for the USDJPY FX currency pair. This shows examples of 1 minute samples of "Open/High/Low/Close" ("OHLC") price samples from one day in 2009.

USDJPY

| timestamp | open | high | low | close |
|---|---|---|---|---|
| 16/03/2009 00:00 | 98.25 | 98.25 | 98.23 | 98.24 |
| 16/03/2009 00:01 | 98.23 | 98.24 | 98.23 | 98.24 |
| 16/03/2009 00:03 | 98.25 | 98.25 | 98.24 | 98.24 |
| 16/03/2009 00:04 | 98.25 | 98.25 | 98.23 | 98.24 |
| 16/03/2009 00:05 | 98.23 | 98.23 | 98.22 | 98.23 |




## Problem Statement

We propose: "*to construct a predictive financial model, capable of predicting the direction of short-term future price movements, based on historical price data*".

The goal is not necessarily to produce a model which could be traded profitably, (which is notoriously difficult in practice). Rather, to consider the project as a proof-of-concept assessing the viability of using machine learning to detect structure in historical price data.

The model will take as its main input a dataset consisting of recent historical prices of a financial product and output a signal indicating the direction of any expected future price movement within a short time period.

The input to the model at any given time will be a snapshot of recent price information up to that time, looking back over a given time window of W samples.

The model is to be implemented as a classifier, returning a categorical output, predicting whether the price will rise or fall.

Finally, the output of the project will be a trained model along with test results, analysis and measurements of its performance on a held-out test data set unseen during its training.

Care is taken to avoid "data leakage" [3] (whereby implicit predictive information from a test or evaluation scenario is inadvertently used in the training process, leading to unrealistically good predictions), by ensuring that the test data sets used cover time ranges that are later in time and distinct from those used for training.

### Proposed Solution

The approach taken is to use a deep neural network trained on historical price information to produce a model providing predictions on future pricing movements.

Deep neural networks were chosen as they are known to be capable of capturing arbitrarily complex patterns in high dimensional data, and they may be trained using Stochastic Gradient Descent (and variants thereof) which is a scalable approach to dealing with large volumes of data.

Note: although the latest price sample at any point in time has only a few dimensions (e.g. open/high/low/close), the recent *history* up until that point is a sequence of, say, *W* elements, where *W* is a parameter describing the length of the price history to be considered.

There are several approaches to processing sequential information for machine learning models. One approach is to use a Recurrent Neural Network (RNN), which explicitly models the input as a sequence of low-dimensional samples. Alternatively, the "sliding window" approach [6] creates one or more input features for each element of the recent history, i.e. having at least *W* features or dimensions. This approach allows the use of regular fully-connected feed forward networks (Multi-Layer Perceptron networks).

Although both approaches are potentially interesting and worthy of comparison, in order to limit the scope, this project focuses on the use of Multi-Layer-Perceptron networks trained using the sliding window method. Some focus is given to assessing the impact of different aspects of this model architecture, such as numbers of network layers and layer sizes.

### Benchmark Models

The following baseline models are proposed for comparison and benchmarking as predictors of future price movements.

1. last price change predictor - a model which simply predicts the future price change to be the same as the most recent price change. Such a model may be expected to have some success at modelling short term trends.
2. moving average price change predictor - a model which predicts the future price change based on the gradient of the moving average price over a recent window. This model can itself be fine-tuned on the training dataset by selecting the optimal window size, and moving average type (e.g. simple moving average versus exponential moving average).

## Metrics

### F1-Score

The primary metric used to evaluate the success of the model, is the F1-score.

Since the model is a binary categorical type, attempting to predict price rise versus price fall, it may be considered that "accuracy" would a more intuitive metric. However, accuracy is best suited for assessing performance on perfectly balanced classes. In this case, our test data are real price series which have slightly imbalanced number of positive and negative return examples. Although only a slight imbalance, this can be significant given that we are measuring performance of a relatively weak signal.

Hence, F1-score which measures the balance between precision and recall and is designed for measuring performance on examples with imbalanced classes, is proposed as a more robust metric.

### Mean Future Return

The "Mean Future Return" is also offered used as an additional metric, which we devised to provide an indicator of the model's performance from a financial perspective. It is conceived as the mean return which would be achieved over each future time period where we make a prediction, if a "cost-free" investment could be made to take advantage of the price change predicted by the model. By this we mean an investment which provides exposure to the price changes of the underlying asset, but which avoids the costs of entering and exiting positions and all trading costs. It is therefore an

unrealistic measure; however, it is one which introduces a financial perspective to the evaluation of the model and one which places a theoretical upper-bound on the model's financial performance.

This is computed as follows:

- for each prediction; take the actual future price return, and multiply by the sign of the model's predicted direction signal (+1 for predicted price rise, -1 for predicted price fall).
- take the mean of the above across all predictions.

First we define the return at time t as the relative price change since the last period:

$$return_t = \frac{price_t - price_{t-1}}{price_{t-1}}$$
$$= price_t / price_{t-1} - 1$$

Then the mean future return over *n* future time periods:

$$meanFutureReturn = 1/n * \Sigma_{allt}(return_{t+1} * sign(predictedReturn))$$

We also report this in annualized form, to allow comparison over different duration future time periods. The annualized form of this metric represents the mean future return compounded over all future periods of all trading days for 1 year, and is computed as follows, assuming 260 trading days per year for FX products:

$$annualizedFutureReturn = (1 + meanFutureReturn)^{((260*24*60/1)-1)}$$

# II. Analysis

## Data Exploration

### Dataset Characteristics

The FX price data available are provided as a set of "OHLC" samples taken at 1 minute intervals. The datasets were downloaded from the histdata.com website [4] in their "Generic ASCII M1 Bars" format, described at [7]. Comparing the data versus their more granular "Generic ASCII Tick" format, suggests that the bar data are constructed from bid quote samples from the tick format data. Whilst more information is available in the tick data format (such as spread between bid and ask quotes), it is also much more voluminous and would require more computationally intensive pre-processing. Hence it was judged that the use of the bar data would suffice for the proof of concept that this project represents.

The "OHLC" data contains columns representing, respectively:

- Open - the price of the first quote in the time window
- High - the highest priced quote in the time window
- Low - the lowest priced quote in the time window
- Close - the price of the last quote in the time window
  where the window in question is the one minute interval. Each row also contains a timestamp, which in this case is the timestamp at the start of the one minute window.

Example extracts of the data for several currency and precious metal pairs are shown below, along with charts illustrating important characteristics of the data, discussed below.

EURUSD

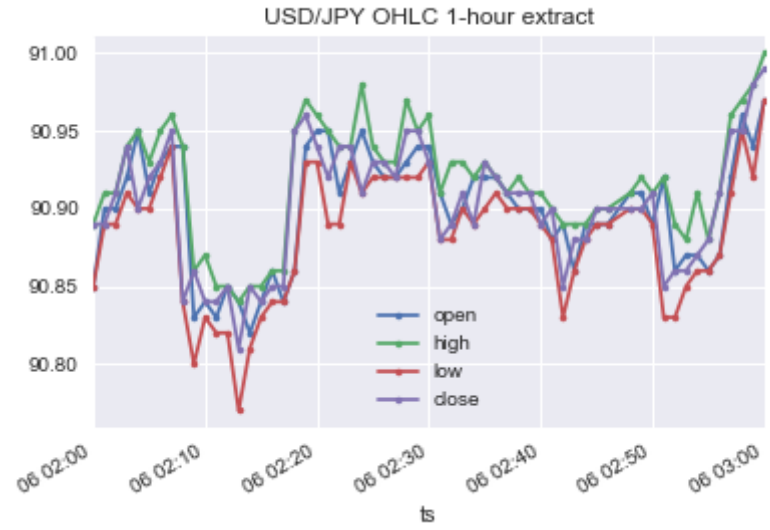
| timestamp | open | high | low | close |
|---|---|---|---|---|
| 16/03/2009 00:00 | 1.2895 | 1.2895 | 1.2895 | 1.2895 |
| 16/03/2009 00:01 | 1.2896 | 1.2897 | 1.2895 | 1.2895 |
| 16/03/2009 00:02 | 1.2896 | 1.2896 | 1.2895 | 1.2896 |
| 16/03/2009 00:03 | 1.2895 | 1.2896 | 1.2894 | 1.2895 |
| 16/03/2009 00:04 | 1.2895 | 1.2895 | 1.2894 | 1.2895 |

USDJPY

| timestamp | open | high | low | close |
|---|---|---|---|---|
| 16/03/2009 00:00 | 98.25 | 98.25 | 98.23 | 98.24 |
| 16/03/2009 00:01 | 98.23 | 98.24 | 98.23 | 98.24 |
| 16/03/2009 00:03 | 98.25 | 98.25 | 98.24 | 98.24 |
| 16/03/2009 00:04 | 98.25 | 98.25 | 98.23 | 98.24 |
| 16/03/2009 00:05 | 98.23 | 98.23 | 98.22 | 98.23 |

XAUUSD

| timestamp | open | high | low | close |
|---|---|---|---|---|
| 16/03/2009 00:00 | 926.7 | 926.7 | 926.38 | 926.55 |
| 16/03/2009 00:01 | 926.6 | 926.6 | 926.33 | 926.55 |
| 16/03/2009 00:02 | 926.6 | 926.7 | 926.53 | 926.7 |
| 16/03/2009 00:03 | 926.65 | 926.7 | 926.65 | 926.68 |
| 16/03/2009 00:04 | 926.63 | 926.63 | 926.43 | 926.43 |


This illustrates important characteristics of the data, as follows:

- some individual samples are missing. e.g. the 2009-03-16 00:02:00 sample for USDJPY. This may be because possibly no price quotes were available during that window, or simply due to issues in the data collection process.
- there are sometimes regular gaps in the price series. In the USDJPY February 2009 closing prices plot, these are clearly seen. In most cases these correspond to weekends or other dates when markets in these instruments are typically closed.
- there is an order of magnitude difference in absolute price value between currency pairs.
- there is a significant long term upward trend in the price series shown in the February 2009 plot.
- the 1-hour extract illustrates the characteristics of OHLC data. Of note is that the 4 traces are of course highly correlated.

The approaches chosen for dealing with each of these characteristics are discussed below:

### Missing samples and regular gaps due to market trading hours

Our approach to handling missing price data, for short gaps (up to 10 minutes) is to fill them with the last observed prices. Note that since we have open/high/low/close components to the price data, we fill all 4 of these features with last observed closing price from the previous period. This preserves the meaning of the OHLC features for the filled rows.

For longer gaps, such as over weekends, we don't fill the gaps, but then discard input records which would rely on such missing samples.

For example, if we were to use a lookback window considering the previous 100 prices, then in the event of a missing sample, we would be unable to create a prediction for the next 100 time periods after a long gap. i.e. we wait until we have a full window of prices before attempting to make a prediction.

Given the relatively high frequency and quantity of price samples, this works reasonably well in practice.

### Long-term price movements

A model such as this, which is trying to predict short term direction of future returns is likely to capture significant bias if trained on data where there was a long-term price trend, resulting it seeing different numbers of positive versus negative examples. To counter this, we take care to normalise the input data to remove long term trends in price changes. The details of the mechanisms used are discussed further in the section on data pre-processing.

### Different magnitude price scales

As we aim to develop a model which can be applied to any FX currency pair, the model should ideally not be unduly affected by the magnitude of prices involved. Additionally, many types of machine learning algorithm are sensitive to the scale of their input data.

So, rather than use the raw prices themselves, which may be arbitrarily large, and differ greatly between different FX currency pairs, the data is transformed and normalised, aiming to achieve a mean of approximately zero and a standard deviation of one for each feature.

One important transformation is to base the features on the "returns" of the price series, meaning the relative price change from the previous time interval, which as a relative measure are more comparable between different FX pairs. The "Pre-processing" section below gives full details of the transformations applied to the data, to make it more suitable for input to the models developed.

### Correlation of OHLC features

As seen in the 1-hour chart the Open/High/Low/Close features of the data are seen to move largely in concert. Hence just one of these features should be sufficient to capture the short term historic trends in price movement across the samples. In our data pre-processing we use the returns of the closing price feature to capture this effect.

The additional features do however provide potentially useful additional information such as the range of prices within each time period and whether prices were ultimately rising or falling during each period.

For the other features, we take their values relative to the closing price to capture this information. This de-correlates them from the closing price changes (reducing redundant information), whilst keeping intact the key information that they provide, and also normalising them to similar scales.

### Return series based on closing price

EURUSD

| timestamp | close | return |
|---|---|---|
| 16/03/2009 00:00 | 1.2895 | 0.0000000 |
| 16/03/2009 00:01 | 1.2895 | 0.0000000 |
| 16/03/2009 00:02 | 1.2896 | 0.0000775 |
| 16/03/2009 00:03 | 1.2895 | -0.0000775 |
| 16/03/2009 00:04 | 1.2895 | 0.0000000 |
| 16/03/2009 00:06 | 1.2894 | -0.0000775 |
| 16/03/2009 00:07 | 1.2893 | -0.0000776 |
| 16/03/2009 00:08 | 1.2891 | -0.0001551 |

Summary Statistics of Return Series

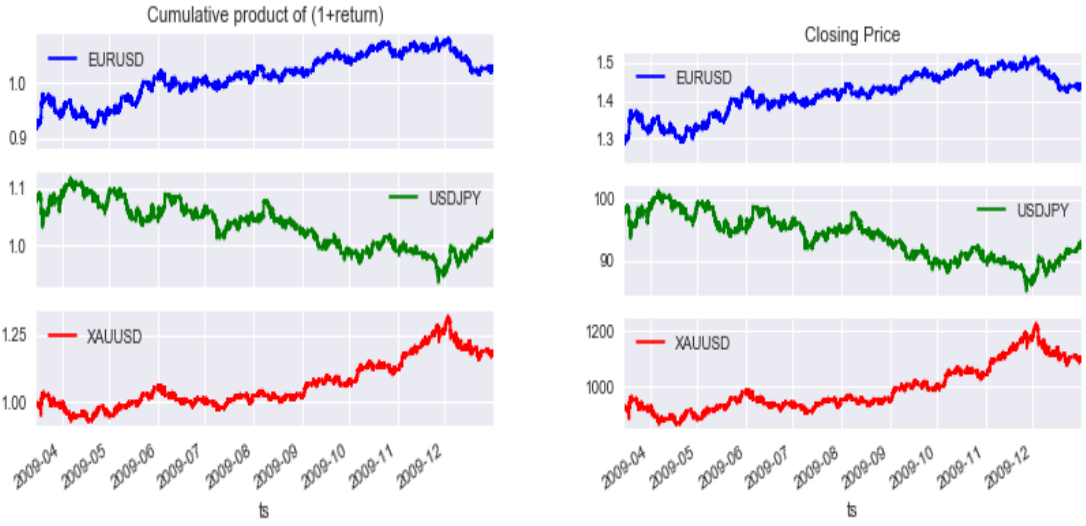
| Column1 | EURUSD | USDJPY | XAUUSD |
|---|---|---|---|
| count | 355,879 | 343,455 | 276,108 |
| mean | 0.0000001 | 0.0000001 | 0.0000007 |
| std dev | 0.0002299 | 0.0002547 | 0.0003415 |
| min | -0.0065299 | -0.0139469 | -0.0078104 |
| 25% | -0.0000780 | -0.0001096 | -0.0001269 |
| 50% | 0.0000000 | 0.0000000 | 0.0000000 |
| 75% | 0.0000784 | 0.0001097 | 0.0001293 |
| max | 0.0175358 | 0.0066100 | 0.0094247 |

The tables above illustrate an extract from a return series produced from closing prices. The summary statistics and the histogram below compare such series for 3 different currency pairs and highlight some important features:

- all have mean of approximately zero
- standard deviation of returns differ, but are of the same order of magnitude
- they have an approximately symmetrical and approximately Gaussian Normal shaped distribution.

## Exploratory Visualization


The charts below compare the raw closing price of 3 pairs, with the cumulative product of the return series. This clearly illustrates that the historical trends and features of the data are preserved by the conversion to a return series, yet the scale of the values involved are converted to similar ranges.




## Algorithms and Techniques

### Overview

The overall approach taken is to train a multi-layer perceptron neural network on a training dataset constructed from derived features of historical prices, versus labels indicating the direction of future price movement for the next time period.

The model is then evaluated on validation and test datasets calculated in exactly the same way, but on a price series from later in time, for the same FX pair. The validation datasets are used to refine hyper-parameters of algorithms, and compare different approaches, with the test datasets held out to be used only for final evaluation.

### Neural Network Design

In order to make a prediction for a single row of data, the neural network model is required to take a large number of real-valued input features and output a single value indicating the expected direction of the price movement in the next time period.

To achieve this, a multi-layer model architecture was chosen, having an input layer with one node for each input feature, and an output layer with one single node, which outputs a real value.

Between the input and output layers a number of hidden layers are added to allow the model to learn and encode feature representations of the input data.

Both input and hidden layers are formed of "Rectified Linear Units" (Re-LU's) following the recommendations at [8], since they are known to be typically easier to train than other common node types, due to their derivative being piecewise constant, which gives a stable gradient, to drive the gradient descent training algorithm.

The output node uses a sigmoid activation function. This has the property of outputting a real value between zero and one. We can use this output to represent a future price increase if its value is greater than 0.5 and a price decrease for values less 0.5. Following the recommendations at [9] this is used in combination with a binary cross-entropy (log loss) cost function.

The network is trained using the "Adam" [10] optimisation algorithm, which is an enhanced variant of Stochastic Gradient Descent.

Learning rate decay is also used. In the event that the validation loss does not decrease for more than a configured number of epochs, then the learning rate is reduced by a configurable factor, until the validation loss reduces, subject to a configurable minimum learning rate.

### Regularisation

To avoid over-fitting: 2 forms of regularisation are employed:

- Early Stopping [11] is used to stop the training of the model at a point before its performance on the validation set is negatively affected. This is achieved by recording the training and validation set loss of the model at regular intervals during its training, along with the model state. After training, analysis of a chart of the losses, shows the point at which the validation loss is a minimum. The model state at this point is then chosen as the best version of the model.
- Dropout [12]. This mechanism randomly discards the output or "drops-out" hidden nodes during training with a configurable probability. It may be viewed as a form of ensemble learning, where effectively many differently-connected networks are trained and their output averaged at test time. Dropout with probability 50% is employed during training on the output of hidden layer nodes.

### Train / Validation / Test Data split

The datasets used for training, validation and testing were prepared identically but based on different time periods. The training set is based on price data from 2009-2014 inclusive. The validation set from 2015 and the test set from 2016.

### Check for data leakage using random price series.

It is very easy, through careless data processing, to produce a solution which inadvertently leaks information from the future into processing of historic records. To guard against such accidental "data leakage", it was considered important to have some means of verifying that this had not occurred to any significant degree. To this end, a number datasets were constructed using synthetic price series, generated to have a completely random series of returns. Since future price movement of such a series is, by design, impossible to predict, the model's predictions when trained on this series, should be no better than random guessing. If the model were trained on such a series and output a prediction significantly better than random, it would be evidence to suggest that an implementation error leading to data leakage had occurred.

## Benchmark

We use as a benchmark, the F1-score and mean-future-return metrics, of the proposed baseline models.

In addition, we contrast the performance of our model on actual historic price datasets, versus its performance on a dataset constructed from a random return series, to confirm that our model has captured information present in the historic series that's not present in a random return series.

# III. Methodology

## Data Pre-processing

All data manipulation is performed using Python, mostly using the “pandas” and “numpy” libraries.

### Resampling

As discussed, short periods of missing data are forward-filled with the last closing price. The process for performing this is as follows:

- first, we detect missing samples by resampling the data (using the pandas 'resample' function) at 1 minute intervals. This has the effect of inserting empty records in the price series where there are missing samples.
- empty records in the close column are forward filled for a maximum of 10 rows.
- open, high and low columns then have any empty records filled from the content of the close column for the corresponding timestamp.

### Input Features

From the raw OHLC price data, features were computed as follows.

For a number of different time periods, *n*, looking backwards from the latest sample time, *t* the following features were computed:

- "close-*n*" (n-period close-to-close return)
$$close_n = \frac{closingPrice_{t-n}}{closingPrice_t} - 1$$
- "open-n" (open relative to close, n periods ago)
$$open_n = \frac{openPrice_{t-n}}{closingPrice_{t-n}}$$
- "high-n" (high relative to close, n periods ago)
$$high_n = \frac{highPrice_{t-n}}{closingPrice_{t-n}}$$
- "low-n" (low relative to close, n periods ago)
$$low_n = \frac{lowPrice_{t-n}}{closingPrice_{t-n}}$$

Note that the "close-0" feature is always omitted, since it is always zero by the above definitions.

### Selection of lookback periods: n

Different approaches were taken to selecting the number and spacing of the lookback periods used. The first approach was simply to use lookback periods n for $n = [0..60]$ to capture the past 60 minutes of price information.

However, a better approach was to space the intervals farther apart as they increase, for example with the first 5 lookback periods spaced 1 period apart, then expanding the interval between them with the square of the ordinal, using the following the formula:

$$n = [0..4, (4 + i^2)_{i=[1..20]}]$$

With this approach the finer details of recent history is preserved, yet a larger overall historical timespan is covered with fewer samples. In this manner, just over 6 hours of history is covered with 25 periods, greatly reducing the volume of data to be processed relative to the first approach.

## Normalisation:

### Mean Removal

Input feature data was normalised to have approximately mean zero and standard deviation of one. The approach taken was to base this transformation on moving average values of the dataset, rather than on statistics of the batch of training data as a whole, as is often done in other problem domains. This means that each sample is only transformed based on data from earlier in time than itself. This is important when dealing with temporal data, and avoids the subtle data leakage which occurs if say the mean of the whole batch is subtracted from each sample, whereby the resulting data would implicitly incorporate future information in their calculation.

Concretely, from each feature's current value was subtracted, the exponentially-weighted moving average (EWMA) of that feature's historic value up until that sample. The "centre-of-mass" parameter used for the EWMA calculation, (which controls the rate at which the impact of historic samples decays with their age) was chosen as 10 times the value of the largest lookback of any feature in the dataset. For example, the dataset using features based on a window of prices, looking back over 400 time periods, would use a centre-of-mass for the calculation of 400*10 = 4000 periods. This somewhat arbitrary choice was made as a trade-off between acheiving stable mean and minimising the amount of data discarded at the start of the dataset, before enough samples are available for the EWMA calculation.

Note that in the calculation of the EWMA, missing values are explicitly ignored, in order that the calculation handles large regular gaps in the data such as those due to weekends. However, the calculation does require that at least of full window of data is available before outputting a value. This functionality is implemented using the pandas library ewm() function.

The same mean-removal technique was applied both for input features, and also to the future-return data used as the basis of the label generation for training described below.

The EWMA function was chosen to determine a rolling mean value rather than a simple moving average over an equivalent window, since in general the output of an EWMA calculation tends to be smoother than that of a simple-moving average, which can experience jumps in its output over time as outlier values fall out of the lookback window.

### Scaling to unit Standard Deviation

Similarly to the technique used for mean removal, to scale the samples to have a variation of similar order of magnitude, each feature value was divided by the exponentially-weighted rolling standard-deviation of its historic values (with the same centre-of-mass parameter as used for mean removal). This has the effect of each feature's data having a standard deviation being approximately one.

Like for mean removal, missing records are ignored in the calculation of the standard deviation.

In the event that the standard deviation resolves to zero, this is explicitly replaced with a null (N/A) value, since division by a null value is a clearly defined operation in the pandas library, which will propagate the null, leading to an empty value for the feature, which is acceptable and preferable to an infinite value, which would occur when dividing by zero.

## Labels

The labels of the dataset are values of 0 or 1 indicating whether the next period's return was positive or negative (i.e. whether the price goes up or down.).

These are generated from the series of close-to-close returns, but normalised by removing the moving average mean component. If the resulting value is positive then a label of 1 is assigned, otherwise a label of 0.

Note that no standard deviation normalisation is required on the return series for generation of the labels, since we don't consider the magnitude of the return, just its sign.

For final testing, additional test datasets with labels generated from non-normalised returns were also generated, for comparison.

## Implementation

### Details and Approach

All research and analysis was carried out using Python code and Jupyter notebooks. Initial prototype code was written in notebooks, and then once stable, was factored out into reusable python modules which can be called from notebooks or other python modules.

All code and important notebooks are available on github at [13]. Important modules and notebooks are described in Appendix A.

#### Dataset Preparation

Extensive use was made of the "pandas" python library for dataframe manipulation, and loading and saving datasets. The raw CSV files were loading using pandas into DataFrame format (a function in the "utils" module was written for this purpose). The "datasets" module was written and used to perform the calculation of features and labels and data normalisation. The resulting datasets were then saved as files in "HDF5" format for re-use across iterations of model development.

#### Model implementation

The neural network model was implemented using the "Keras" python framework. This is a relatively high-level framework, which allows straightforward definition and training of neural network models, and provides sufficient hooks to allow reasonable customisation. Keras supports plugable backends (Theano, Tensorflow).

The models were trained on an NVidia GTX580 GPU, using the "Theano" backend for Keras, running on an Ubuntu 16.04 workstation. The trained model state was saved to HDF5 files. The model state files are not hardware or backend-specific, so the models could then be loaded for analysis and inference on other hardware (in this case a Macbook Pro using the Tensorflow backend).

The "model01" module provides the implementation of the model, including callback functions which are used to monitor and report on the progress of the training of the model, and facilities to save both the model state at intervals during training, and the history of the model's performance during training against the F1-score metric, in order to construct the learning curve.

To optimise the time taken to train the model, a critical factor turned out to be the mini-batch size used for the SGD algorithm. This controls the number of rows of data passed to the GPU in a single batch. Larger values greatly increase the speed, but use more GPU memory. The best batch size varies with the number of network nodes. It was established by monitoring the GPU memory usage with the "`nvidia-smi`" tool, and increasing the batch size until most of the GPU memory was utilised. (In this case the 3GB of the GTX580.)

#### Model Architecture

The neural network structure has an input layer, with the number of nodes matching the number of features of the dataset, and an output layer with a single node. The layers in between (hidden layers) were parameterised in the implementation, in terms of:

- the number of hidden layers
- the number of nodes per layer
- the amount of dropout applied (if any).

Learning rates were kept at the default for the Keras implementation of the Adam optimiser. Keras has facilities for automatic learning rate decay (`keras.callbacks.ReduceLROnPlateau`), which was used and configured with the following parameters:

Learning Rate Decay Parameters

| Parameter | Value |
|---|---|
| Initial learning rate | 1.0E-04 |
| epochs before reduction | 5.0E+00 |
| reduction factor | 2.0E-01 |
| min learning rate | 5.0E-05 |

Keras' facility for automatic early stopping was also initially used (`keras.callbacks.EarlyStopping`). It proved tricky to tune its parameters for good results, and it often stopped the training too soon. After some experimentation, it was effectively disabled, since learning curves were anyway visually inspected to determine the best model settings (minimum validation loss).

### Model Evaluation

Models were evaluated by loading their saved state from file, loading a test dataset, and producing a performance report using code written in the 'metrics' module.

This performance report contains:

- the F1-score
- mean future return in basis points (bps)
- annualized future return
- cumulative future return chart
- histogram of the real-valued output predictions. (before thresholding at 0.5 to convert to up/down classifications)
- confusion matrix (showing predictions versus actual price movement direction) and heatmap of this confusion matrix.

The confusion matrix was used to qualitatively assess the predictions, e.g. to see if they were balanced across categories.

## Refinement

During the course of the research, various refinements to initial approaches were made. Some of these were driven by quantifiable performance improvements, and others by intuition and practical considerations.

### Window size / lookback period selection

Intuitively, a long lookback window can capture both short and longer term trends. We had no initial preconceptions as to what length was appropriate, but as a starting point, since we were working with data sampled at 1 minute intervals, an initial window of 60 periods was used, to capture the past 1 hour of history.

However, this gave poor results and creating 4 features for each of 60 periods, gave datasets of 240 features, which were very large and slow to work with.

After some thought, it was reasoned that we probably need less granular time intervals as overall timespan increases. e.g. if we use a 50-period lookback, then probably little extra value is added by additionally including the 51- and 52-period lookbacks, rather than say the 70- and 90-period lookbacks. So, using additional features to span a greater time range, should give a greater data efficiency. The approach taken was to keep the first 5 lookbacks spaced every minute, then gradually expand the gap between them, as the square of index of the lookback.

This latter approach resulted in much smaller dataset files which were quicker to train with and which gave better results.

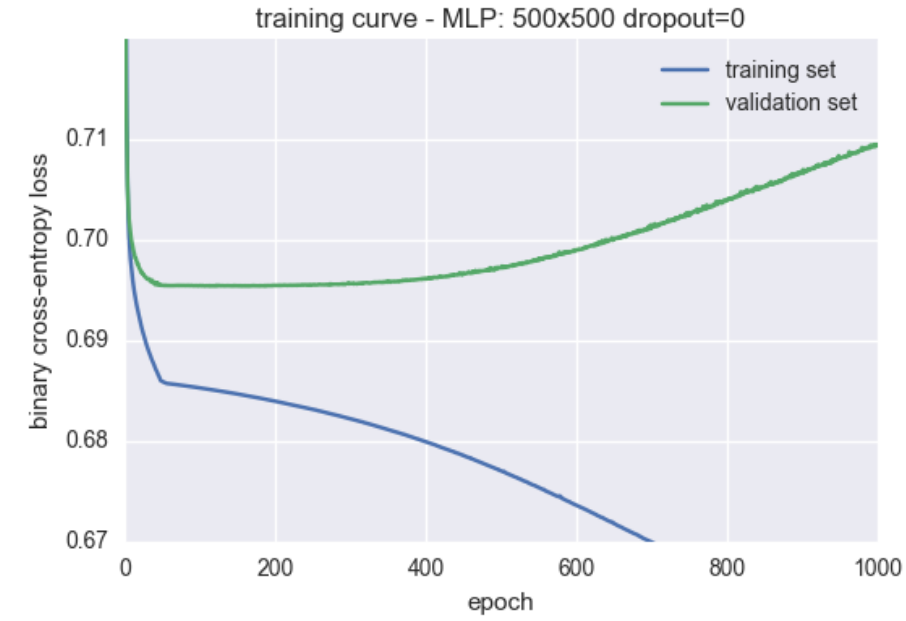
**Use of dropout for regularisation**

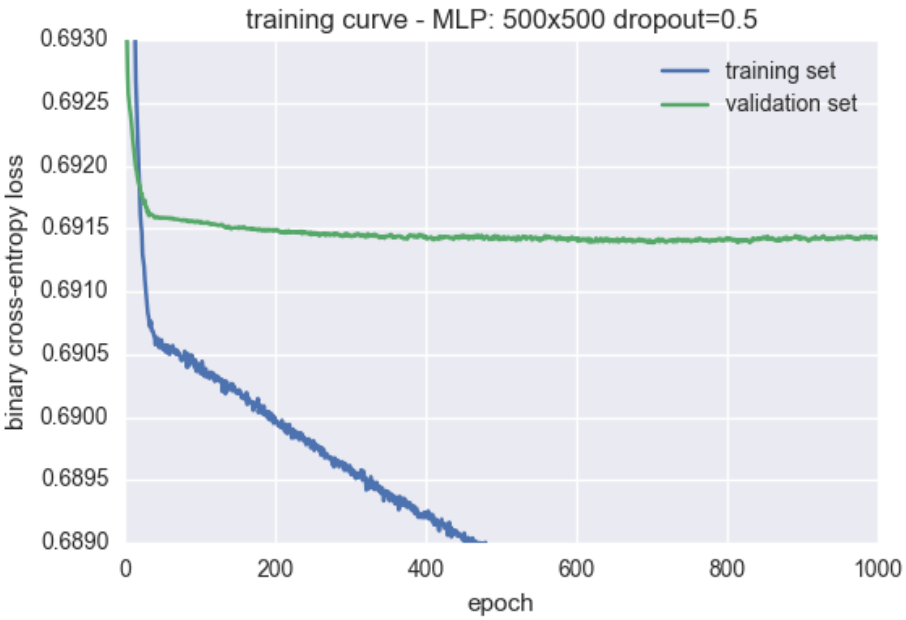
Models were tried both with and without dropout for regularisation. Without dropout, the validation set performance of larger models was very clearly limited, as the model starts to overfit.

This effect is clearly illustrated below, where the learning curves for a 500x500 hidden layer model trained for 1000 epochs both with and without dropout are shown.

In the case where no dropout is applied, the validation set loss reaches a minimum of 0.6954 after 115 epochs, and then starts to rise as the model overfits to the training set.

In the case where dropout is used, the model can be trained for 1000 epochs without signs of overfitting. The validation set loss reduces to around 0.6910 by this time.




The model performance on the EURUSD 2015 validation set is compared below, showing better performance when dropout was used. Note that the performance of the model without dropout is reported for the model in its state after 115 epochs, when the loss was at a minimum.

| Model | f1-score | 1-minute mean future return (bps) | annualized future return |
|---|---|---|---|
| Dropout=0 | 0.518 | 0.023 | 1.38 |
| Dropout=0.5 | 0.526 | 0.03 | 2.123 |

### Model Architecture Refinement

In classical machine learning approaches, hyper-parameters are often automatically optimised by grid search or random search in conjunction with procedures such as K-fold cross validation.

Unfortunately, the high training time of neural network based models, makes exhaustive automatic search of the parameter space impractical. Instead, a manually selected set of architectures was compared and evaluated based on their f1-score metric on the validation data set. Models with between 1 and 3 hidden layers, in layer widths of 100 and 500 nodes were evaluated.

The results are shown in the table below, for training on the EURUSD 2009-2014 dataset and validation performance measured on the EURUSD 2015 dataset.

For comparison, the performance of a Gaussian Naive Bayes Classifier and Random Forest Classifier trained and evaluated on these datasets is shown.

#### Training results

| hidden layers | dropout | model parameters | training epochs | training f1-score | validation f1-score | 1-minute mean future return (bps) | annualized future return | notes |
|---|---|---|---|---|---|---|---|---|
| 100 | 0 | 10101 | 1000 | 0.533 | 0.516 | 0.021 | 1.221 | |
| 100x100 | 0 | 20201 | 1000 | 0.542 | 0.520 | 0.023 | 1.329 | |
| 100x100 | 0.5 | 20201 | 1000 | 0.530 | 0.520 | 0.029 | 1.948 | |
| 500 | 0 | 50501 | 1000 | 0.547 | 0.518 | 0.021 | 1.207 | |
| 500x500 | 0 | 301001 | 165 | 0.55 | 0.518 | 0.023 | 1.380 | early stopping at epoch 165 |
| 500x500 | 0 | 301001 | 1000 | 0.599 | 0.516 | 0.025 | 1.574 | |
| **500x500** | **0.5** | **301001** | **1000** | **0.546** | **0.526** | **0.030** | **2.123** | **best performing model** |
| **500x500** | **0.5** | **301001** | **1000** | **0.546** | 0.525 | **0.030** | 2.080 | training repeated for comparison |
| 100x100x100 | 0 | 30301 | 20 | 0.541 | 0.519 | 0.024 | 1.472 | early stopping at epoch 20 |
| 100x100x100 | 0 | 30301 | 115 | 0.545 | 0.521 | 0.026 | 1.653 | f1-score improved with further training, even though cross-entropy loss rose after epoch 20. |
| 100x100x100 | 0.5 | 30301 | 1000 | 0.540 | 0.524 | 0.026 | 1.693 | |
| 100x100x100 | 0.5 | 30301 | 1000 | 0.526 | 0.515 | 0.027 | 1.748 | repeated for comparison |
| 500x500x500 | 0 | 551501 | 15 | 0.539 | 0.518 | 0.023 | 1.328 | very rapidly overfits without dropout, early stopping at epoch 15 |
| 500x500x500 | 0.5 | 551501 | 1470 | 0.547 | 0.523 | 0.029 | 1.919 | early stopping at 1470 |
| 500x500x500 | 0.5 | 551501 | 1970 | 0.551 | 0.521 | 0.028 | 1.862 | continued training |
| 500x500x500 | 0.5 | 551501 | 2000 | 0.551 | 0.520 | 0.028 | 1.840 | continued training |
| 32x32x32x32 | 0.5 | 6401 | 455 | 0.336 | 0.341 | 0.001 | 0.056 | small model: insufficient capacity |
| Gaussian Naive Bayes classifier | | | | 0.522 | 0.514 | 0.021 | 1.216 | alternative model for comparison |
| Random Forest classifier | | | | 0.549 | 0.525 | 0.021 | 1.222 | alternative model for comparison |

### Observations

#### Model performance versus complexity

It was seen that the smaller models, with no dropout, such as the 100-node single hidden layer network, and the 100x100 network had relatively low *training* set f1-scores, even after training for 1000 epochs. This indicates that the model was suffering from high bias and was failing to fit the training set well, suggesting it was limited in expressive capacity.

Increasing the number of nodes in either width or depth improves this, and training set scores are seen to rise, indicating that the model has greater capacity to capture relevant features, and therefore has a lower bias.

Such expressive models are seen to over-fit rapidly without regularisation, leading to limited generalisation performance. However, applying dropout for regularisation slows down the rate of learning during training, and reduces training set performance, but greatly improves validation set performance, and hence the models ability to generalise well, and achieve low variance.

#### Loss function versus evaluation metric

Training curves were plotted based on the cross-entropy loss, which is the quantity being minimised by the gradient descent training procedure.

However, the metric we have chosen to measure the eventual performance of the model is the f1-score. Whilst generally a decrease in cross-entropy loss leads to an increase in f1-score, this is not always the case. For example, in the results above, for the 100x100x100 layer network with no dropout, training was halted by the early stopping algorithm at epoch 115, since the validation loss had not decreased since around epoch 20. However, evaluating this f1-score of the models shows that the validation f1-score rose through further training of the model, even once the minimum validation loss had been passed.

Ideally, the training algorithm would optimise the f1-score directly, however training the neural network using back-propagation requires the use of a differentiable loss function, which f1-score is not.

Furthermore, it can be seen that whilst, in general, an increase in f1-score leads to an increase in mean future return (which we chose as our secondary metric), this is not always the case.

# IV. Results

## Model Evaluation and Validation

The final model selected was a multi-layer perceptron with two hidden layers each of 500 nodes, trained for 1000 epochs using 50% dropout. Of the architectures tested, this gave the best validation set f1-score performance. It also gave the best performance measured in terms of mean-future-return.

Training of this model took several on hours even though a GPU was used. The training procedure was repeated a second time to see how repeatable the procedure was, and similar results were obtained.

This final model was then evaluated on a selection of held-out test sets, both for the same currency pair used in training (EURUSD) and for other pairs for comparison (to assess how transferable the learned features were). The results are shown below:

MLP 500x500 dropout=0.5 trained on 1000 epochs of EURUSD 2009-2014, hyper-parameters tuned on EURUSD 2015

| test dataset | f1-score | 1-minute mean future return (bps) | annualized future return | notes |
|---|---|---|---|---|
| **EURUSD 2016** | **0.520 (0.517)** | **0.025 (0.025)** | **1.557 (1.560)** | |
| USDJPY 2016 | 0.525 (0.521) | 0.019 (0.019) | 1.014 (1.014) | |

| | | | | |
|---|---|---|---|---|
| EURSEK 2016 | 0.528 (0.517) | 0.039 (0.039) | 3.286 (3.283) | |
| XAUUSD 2016 | 0.517 (0.514) | 0.024 (0.024) | 1.461 (1.451) | |
| RND 1 | 0.501 | 0.000 (0.000) | 0.000 (-0.001) | Random dataset #1 |
| RND 2 | 0.501 | 0.000 (0.000) | 0.004 (0.004) | Random dataset #2 |
| RND 3 | 0.5 | 0.000 (0.000) | -0.009 (-0.009) | Random dataset #3 |

Figures in parenthesis are those evaluated against an un-normalized future return series (i.e. with no mean removal).

It can be seen from the above results that the model generalises well beyond the test and validation datasets to produce a good performance on the test sets, both for the EURUSD currency pair, and also on a selection of other currency pairs on whose historic data the model had not been trained. This suggests that the learned features used by the model are not unique to that currency pair.

The performance on the 3 random datasets, is consistent with random guessing. This gives confidence that the model's performance is not simply due to accidental data leakage.

## Justification

### Baseline model performance.

The initially proposed baseline models, based on the last price change, and the rolling average of the last price change, actually turned out to give negative results when tested on the training dataset. That is, they predicted the opposite of the actual price direction. This suggests that the EURUSD return series may be characterised as mean-reverting [14] over short time periods.

However, reversing the direction of the signal that these models provide, by subtracting it from 1.0, led to baseline models with some predictive ability. The window sizes of the moving average models were then tuned on the EURUSD 2015 validation set, and then tested on the EURUSD 2016 test dataset, for comparison with the neural net models, giving results as below:

Comparison of baseline models

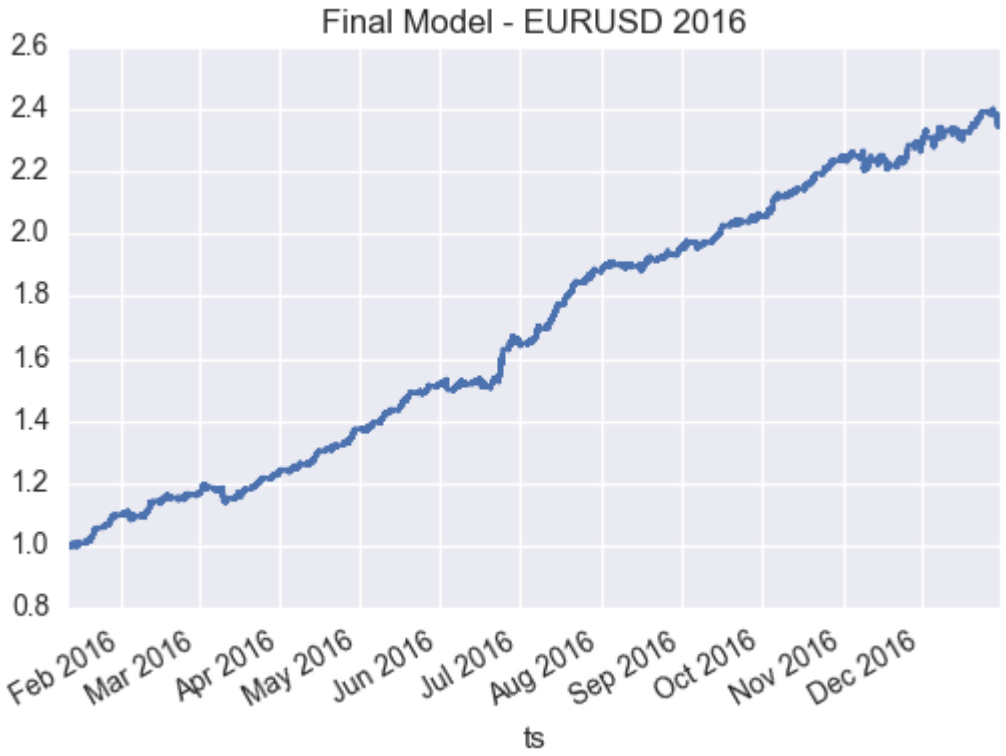
| baseline model | f1-score | 1-minute mean future return (bps) | annualized future return | notes |
|---|---|---|---|---|
| 1. Last price change predictor | 0.511 (0.516) | 0.015 (0.015) | 0.772 (0.775) | |
| 2a. Moving average price change predictor | 0.512 (0.512) | 0.019 (0.019 | 1.044 (1.045) | SMA window=9 |
| **2b. Moving average price change predictor** | **0.516 (0.516)** | **0.025 (0.025)** | **1.535 (1.535)** | **EWMA centre-of-mass=21** |

This demonstrates that these simple baseline models are capable of reasonable performance, however our final neural network model achieves stronger scores, both in terms of f1-score (0.520 versus 0.516) and annualized mean future return (1.560 versus 1.535).

# V. Conclusion

## Visualization

The performance of the final model on the EURUSD 2016 test dataset, may be visualised as a cumulative return series, showing the compounded relative change in value of an investment made in accordance with the direction predicted by the model, in a cost-free environment. The chart for our final model is shown below:




## Reflection

### Hyper-parameter selection

A particular challenge in the research was knowing where best to focus efforts to optimise the large number of hyper-parameters. There were many dimensions which could be tuned, such as:

- lookback window - i.e. how much history to include
- look-ahead period - how far ahead to attempt to predict
- model architecture - number of layers and nodes, loss function and learning rate
- inclusion of additional features

Given the time taken to train a neural network could be anywhere from a few minutes to several hours depending on its size, only a limited number of combinations could be explored.

Hence to limit the scope of the project, some parameters were fixed, such as the look-ahead period, set at 1 minute. This was chosen since intuitively price movement over a shorter term was expected to be more predictable, however, given more time, this is an assumption that should be challenged, particularly since it has the drawback that the potential price movement in a such short timescales is generally too small to be profitably tradeable.

### Model selection

Relatively late into the project, a number of alternative models were tried for comparison, such as Ada Boost, Gaussian Naive Bayes and Random Forest Classifiers. Whilst AdaBoost proved very time consuming to train on the volumes of data available, both Gaussian Naive Bayes and Random Forest Classifiers were much quicker and gave reasonable results. Whilst their performance was below that of the neural network classifier, their quicker training time is appealing, particularly that of the Naive Bayes classifier.

With hindsight, an interesting approach to exploring alternative lookback windows, look forward periods and the effectiveness of different features would be to test them first with the Naive Bayes classifier, which would allow rapid iteration, and then spend time tuning the more complex neural network models only once the most promising combination had been established.

## Improvement

### Additional features

A personal goal from the project was to determine if a successful classifier could be built, learning only from the raw data with minimal pre-processing beyond normalisation, with no preconceptions regarding the applicability of synthetic features manually derived from the data. That is, allowing the model to discover features of interest in the raw data.

However, the relative strength of the exponentially weighted moving average baseline model, suggests that this moving average return would make a good feature. An interesting area for further research would therefore be to augment the datasets with the exponentially weighted moving average close-close return as a synthetic feature, to determine if this leads to faster training and/or improved performance. Intuitively, the inclusion of an additional single feature which provides a large proportion of the final models performance would seem likely to at the very least speed up convergence of the model's training.

### Additional data

The success of the model trained on EURUSD data when evaluated on data from other currency pairs suggests that the distributions of the price data share much in common. Given this, it seems reasonable to consider that the model may benefit from training on a potentially much larger dataset containing data from a range of currency pairs.

# Appendix A - Code Overview

## Python Modules

- `datasets.py`
  contains the `prepare_dataset` functions to generate the datasets. There are several variants of these, reflecting refinements as the project progressed.
- `metrics.py`
  contains code to evaluate and report the performance of a model, given it's predictions and the true results.
- `model01.py`
  contains the implementation of the neural network model.
- `utils.py`
  contains miscellaneous utility functions, including `load_1minute_fx_bars` for loading the raw CSV price data.
- `baseline01.py, baseline02.py`
  contain the implemenation of the baseline models.

## Jupyter Notebooks

- `CapstoneProjectReport.ipynb`
  The document used to prepare this report.
- `PrepareExtractsForReport.ipynb`
  contains the code used to prepare all figures and tables used in this report
- `PrepareDatasets.ipynb`
  Used to prepare the datasets and save them to file.
- `TrainModels.ipynb`
  Used to load datasets from file, create the models and running training sessions upon them
- `LoadAndEvaluateModel.ipynb`
  Used to load a trained model from file, load a test dataset and evaluate its performance.

# References:

[1] https://en.wikipedia.org/wiki/Efficient-market_hypothesis
[2] https://en.wikipedia.org/wiki/Momentum_(finance)

[3] https://www.kaggle.com/wiki/Leakage
[4] http://www.histdata.com/
[5] https://en.wikipedia.org/wiki/Trend_following
[6] Machine Learning for Sequential Data: A Review - Thomas G. Dietterich - http://web.engr.oregonstate.edu/~tgd/publications/mlsd-ssspr.pdf.
[7] http://www.histdata.com/f-a-q/data-files-detailed-specification/
[8] http://www.deeplearningbook.org/contents/mlp.html - Goodfellow, Bengio, Courville - section 6.3 "Hidden Units"
[9] http://www.deeplearningbook.org/contents/mlp.html - Goodfellow, Bengio, Courville- section 6.2.2 "Output Units"
[10] Adam: a method for stochastic optimization - Kingma & Ba - https://arxiv.org/abs/1412.6980
[11] Early Stopping: https://en.wikipedia.org/wiki/Early_stopping
[12] Dropout: Improving neural networks by preventing co-adaptation of feature detectors - Hinton et al. https://arxiv.org/abs/1207.0580
[13] Repository of code and Jupyter notebooks used during research and report preparation: https://github.com/andrewwilson/mlnd_capstone
[14] https://en.wikipedia.org/wiki/Mean_reversion_(finance)