Neural Networks in the Financial markets

Inputs as Prices and Outputs as Trading Signals

The inputs for the neural network in the python script can be varied, more features can be added and they usually are passed in as a matrix. For just now say we have an array of prices.

$$[P_0, P_1, P_2, ..., P_{n-2}, P_{n-1}]$$

We also know that given this array there is a sequence of actions which are hold, buy or sell. Performing these actions in order dependning on the price movements will generate optimal profits.

$$[B_0, H_1, H_2, S_3, .., B_{n-2}, S_{n-1}]$$

Where $Actions = B_i, S_i, H_i$ and i is the position at which the actions are performed. In the code examples each action is encoded in the following manner:

$0 = H_i$ $1 = B_i$ $2 = S_i$

So essentially we will have a vector like:

$$[1, 0, 0, 2, ..., 1, 2]$$

The restrictions of this vector is that it will never have any consecutive buys, and never any consecutive sells.

So given n explanatory variables at time step i, use those n data points to predict if at the ith time step we should BUY, HOLD or SELL.

So what do optimal Buy and Sells even look like?

The algorithm utilities dynamic programming to find the optimal buy, hold and sell sequences. It is also a "hard" question on leetcode.

Our input so far is just a vector but it can be extended to a matrix, some features utilized to do this were volume and trade count too.

$$\begin{bmatrix} P_0 & P_1 & ... & P_{n-1} \\ T_0 & T_1 & ... & T_{n-1} \\ V_0 & V_1 & ... & V_{n-1} \end{bmatrix}$$

However, we aren't going to map each individual it column to a specific realization.

Initially I had only used the Price row vector for my signal predictions. Note that our observations are formed from finding the most optimal signals from index 1 to n. I would then select a window of size k. The aim of utilizing this sort of neural network was for it to find patterns within the movements of price and find any relationships it has between signals.

$$[P_0, P_1, P_2, ..., P_{n-2}, P_{n-1}]$$
$$[B_0, H_1, H_2, S_3, .., B_{n-2}, S_{n-1}]$$

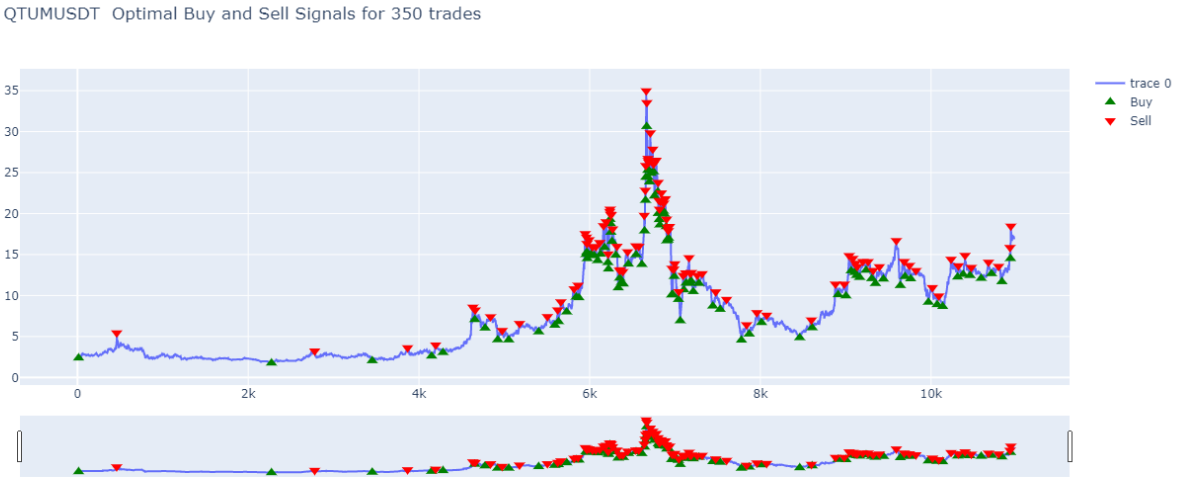


Figure 1: (A) Optimal Buy and Sell Trade Signals?

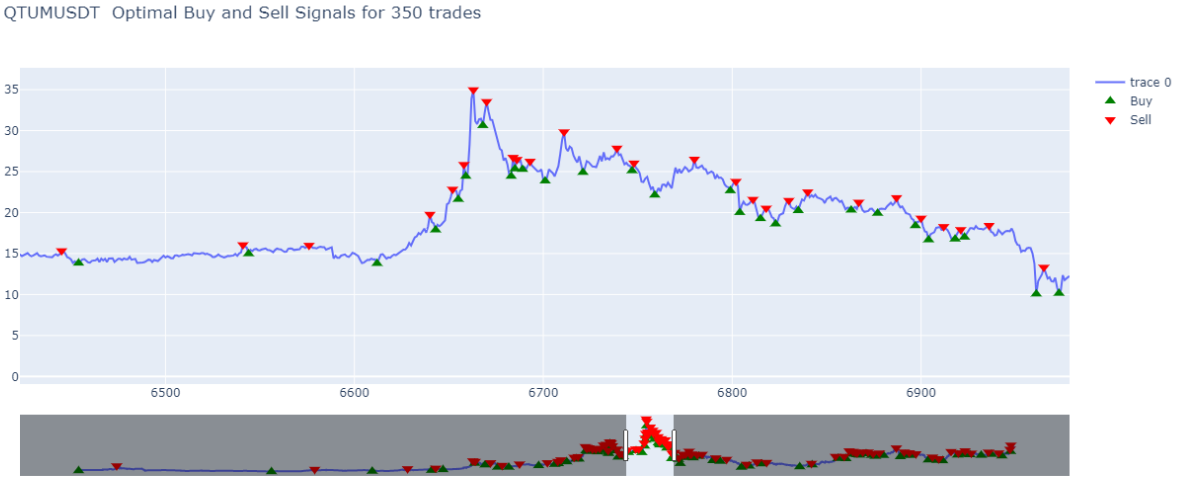


Figure 2: (A) Optimal Buy and Sell Trade Signals Zoomed in?

I select a window length of k.

$$[P_0, P_1, P_2, ..., P_{k-2}, P_{k-1}]$$

We then relate it to observation k (or k-1 index wise):

$$f([P_0, P_1, P_2, ..., P_{k-2}, P_{k-1}]) = S_{k-1}$$

In general we have:

$$f([P_i, P_{i+1}, P_{i+2}, ..., P_{k-2}, P_{k-1)}]) = S_{k-(i+1)}$$

Where $i = 0, 1, 2, 3, ..., n-k-1$ and $k = k+i-1$

So we can create a feature matrix:

$$X_i = \begin{pmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ \vdots \\ P_{k-(i+2)} \\ P_{k-(i+1)} \end{pmatrix}$$

Our observation is:

$$y_i = S_{k-1+i}$$

We can then make our feature matrix:

$$Z = [X_0, X_1, X_2, ..., X_i, ..., X_{n-k-1}]$$

So we have n-k-1 explanatory variables and only 1 feature so far.

$$A \times Z = \begin{pmatrix} y_0 \\ y_1 \\ y_3 \\ \vdots \\ y_i \\ \vdots \\ y_{n-k-1} \end{pmatrix}$$

Note that A is unknown. We want to use a Neural Network which tries to approximate such a matrix.

However, we are missing 1 step. For each explanatory instance we normalize that vector, so we normalize $X_i, i = 0, 1, ..., n-k-1$. We apply min-max normalization, this reduces the values of each X_i to in between 0 and 1.

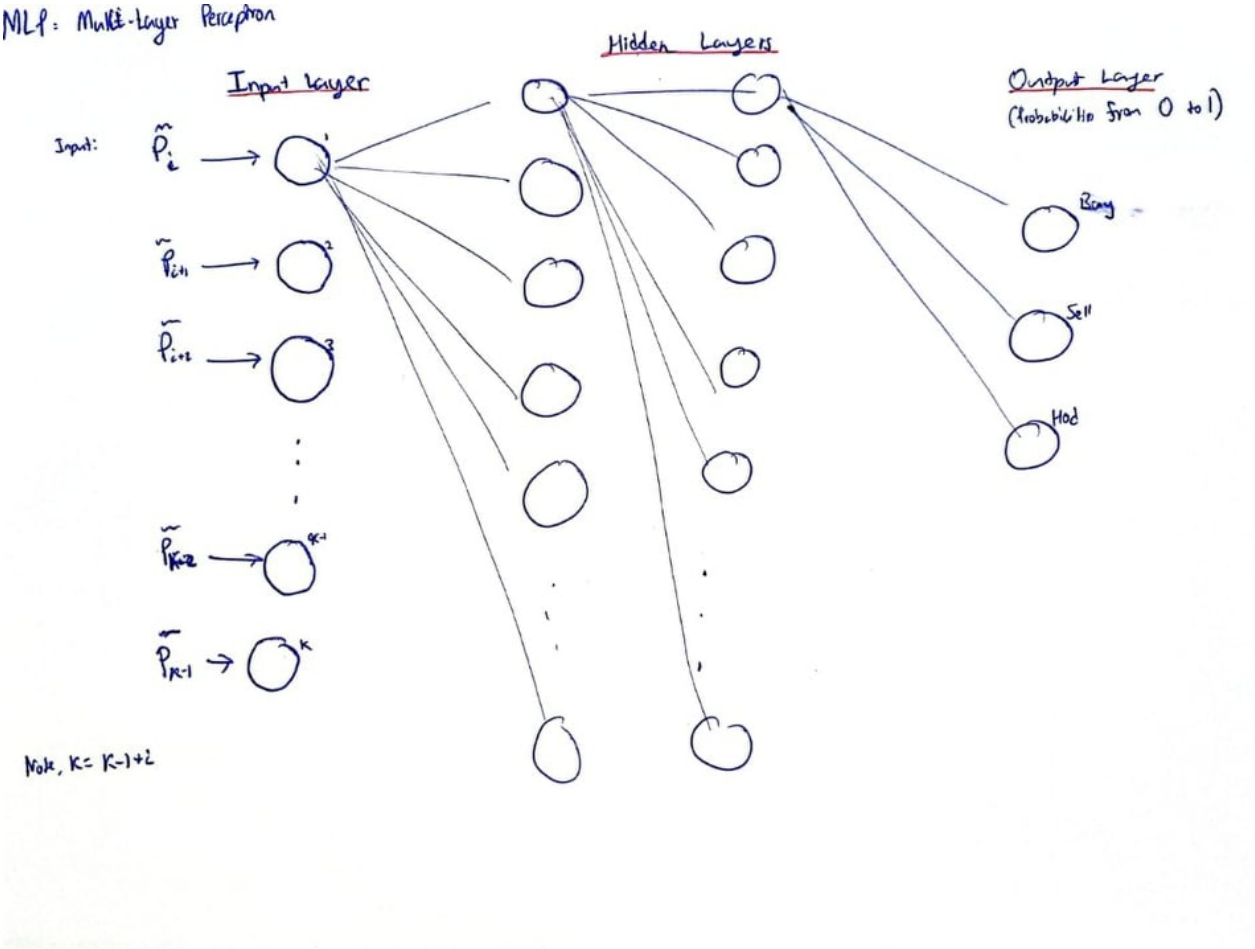
I want to maintain the overall shape and geometry of the prices whilst keeping them relative to each other. I opted to do this, instead of looking at log returns or percentage change.

We also know that there are $k!$ many ways that the prices have unique ordering.

Is there a relationship between each unique ordering of prices and there relative magnitudes with trading signals?

We need to determine how many transactions we would like to take when looking at the most optimal signals.

Here is a sketch of what the most basic version of the MLP looks like:



It is a fairly simple structure, the neural network is dynamic and its parameters and overall structure depends on the window size we select.

Above is just one example, of the neural networks being used. Say we had a number of different explanatory variables we wanted to use.

$$\begin{pmatrix} X_{0,0} & X_{0,1} & \dots & X_{0,n-k-1} \\ X_{1,0} & X_{1,1} & \dots & X_{1,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m,0} & X_{m,1} & \dots & X_{m,n-k-1} \end{pmatrix}$$

So now we have:

$$X_{row,column} = X_{j,i}$$

where row identifies the feature and column identifies the index we are using.

Now lets select column 0. Column is 0 is our first selection of inputs. If we unpack the vectors in column 0 it would look something like this, note we are going to transpose the vectors for readability too.

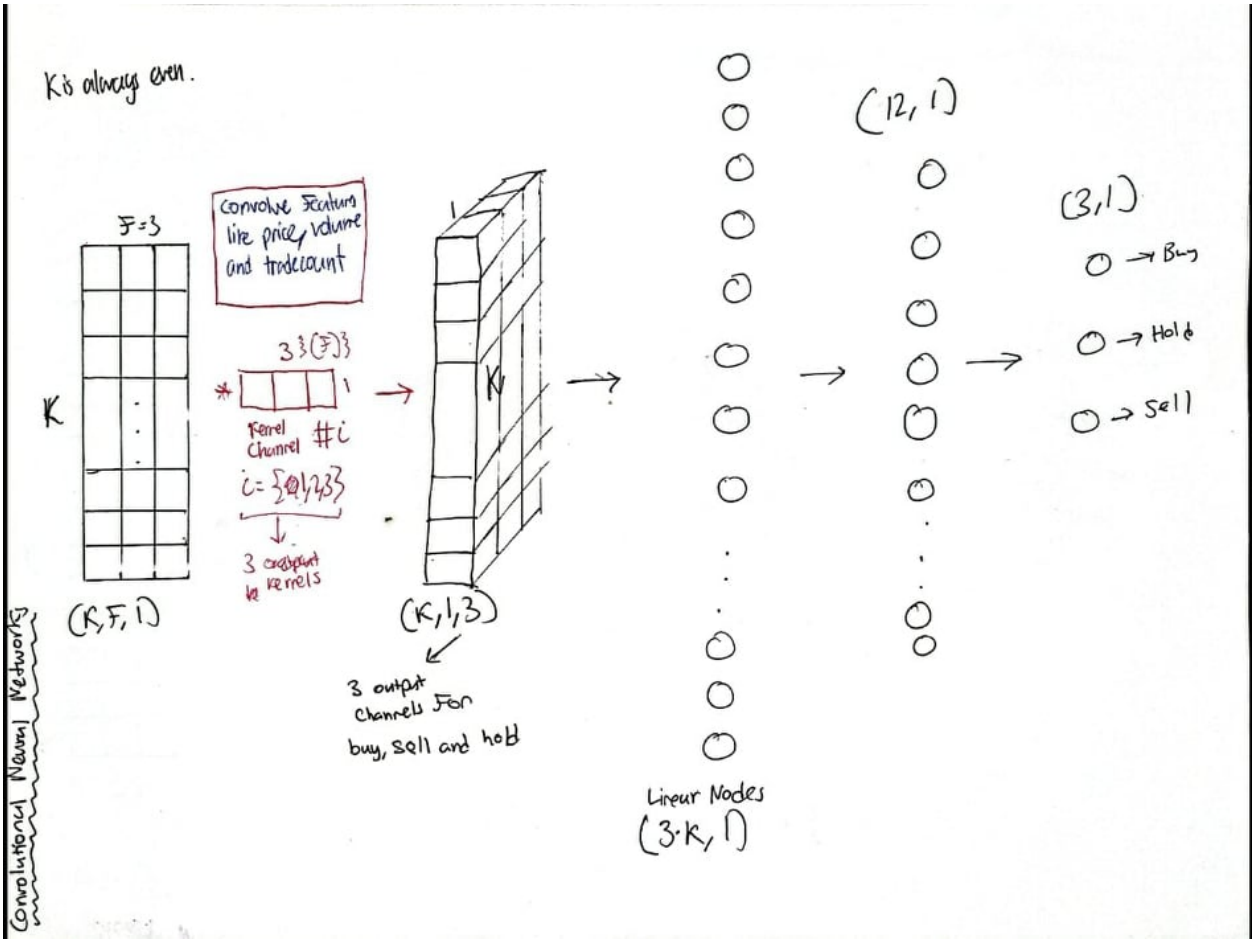
$$F = \begin{pmatrix} x_{0,0} & x_{0,1} & \dots & x_{0,k-1} \\ x_{1,0} & x_{1,1} & \dots & x_{1,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,0} & x_{m,1} & \dots & x_{m,k-1} \end{pmatrix}$$

Note:

$$X_{j,i} = \begin{pmatrix} x_{j,i} \\ x_{j,i+1} \\ \vdots \\ x_{j,k+i-1} \end{pmatrix}$$

So far we have only looked at the dynamic neural network design (note the actual hyper parameters have not been discussed but they are basic). Now, lets introduce the concept of a convolution. We apply feature-wise convolution, we then get a vector out of our F matrix. The convolution occur feature wise.

Here is a brief illustration



Above, we run the convolution which is of dimension (1,3). We also expand the number of channels from 1 to 3.

Immediately are there any problems I recognize?

1. Given $[P_{1]}, P_2, ..., P_n]$ and $[P_2, P_3, ..., P_{n+1}]$ if we normalize either vectors how different are they going to be such that the neural network recognizes there exists a pattern which informs them to make different

decisions? (We can run tests on the features to look at this). Does this mean we should choose very short windows, as there would be a higher probability that larger windows won't "change" as much as smaller windows.

2. Another problem is, can the neural network recognize that it should sell only after it buys and it should buy only after it sells?

3. There are problems with the Optimal trading signals given max k transactions algorithms too, the first one being is that if the data is used on say crypto coins, coins initially were very low in price and then they increased exponentially. This means that if we used the training data from say time step 0 to 6000 and let the neural network trade from 6000 to 10000, most of the profitable trading signals would be in between 6000 to 10000 to as during this point of time they would have more volatility and maturity, so cyclical ups and downs could be exploited, however the period from 0 to 6000 would most likely not have these sort of behaviors too. So, I think choosing better assets is also an important factor when trying to gain insight from financial markets through machine learning and artificial intelligence. If we have a look at QTUMs optimal trading signals as shown above, it is pretty clear that most of the trading occurs towards the 2nd half of the data, we can even count the number of signals that occurred during the 1st half.

4. We might need more complex features, the market is influenced by many things that we do not yet know. It can be influenced by the pricing of other assets, it can be influenced by global news and many more things.

5. Min Max scaling actually prevented the training accuracy from increasing, however standard scaling solved this issue at least.

6. In order to avoid issues and to even prevent the need of using scaling, should we be looking at returns instead? I will also implement, a version which does not standardize the feature vectors instead it just looks at the changes in the values of the features.

Prices are not normally used, instead log returns or simply stock returns are
used. The reason behind this, is that returns is generally more stationary then prices. For example, if I were to to apply linear regression on prices over a long period of time we'd encounter something called spurious regression, which provides no useful insights and forecasts as generally as time proceeds usually prices increases. (This includes stocks which are selected due to suvivorship biases etc).

Concepts of stationarity:

1. A stochastic process R_t is called strictly stationary if it's joint distribution function does not depend on time.

2. A stochastic process R_t is called weakly stationary if its first and second moments do not depend on t and thus are finite.

Usually in financial literature, it is common to make an assumption that asset returns are weakly stationary.

How do we check if the prices we standardized are stationary or not, and what does it mean to have a vector of standardized prices which are considered stationary?

$$\begin{pmatrix} P_{0,0} & P_{1,1} & \dots & P_{n-k-1,n-k-1} \\ P_{1,0} & P_{2,1} & \dots & P_{n-k,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ P_{k-1,0} & P_{k,1} & \dots & P_{n,n-k-1} \end{pmatrix}$$

Now, $P_{i,j} = P_{index,observation}$.

Note that each column vector is also standardized. We have $n - k - 1$ observations, note that index value references the actual index of the array of prices or feature array.

So for each observation, I want to check for the stationarity of the elements row-wise. So in the 1st,2nd ane etc row there are n-k-1 elements and thus we want to check for the stationarity of each element in the same row position for each observation.

So for each column I applied two stationarity tests on each row of data.

1. The Augmented Dickey Fuller ("ADF") test

This test determines the presence of unit root in the time series, thus it can indicate whether the data is stationary or not.

Null Hypothesis: The series has a unit root.

Alternate Hypothesis: The series has no unit root.

If the p-value is less than 0.05 than the null hypothesis is rejected, if not it is accepted.

The unit root is a feature of the stochastic process which can cause problems when making inferences. It is considered a non-stationary process.

2. The Kwiatkowsky-Phillips-Schmidt-Shin ("KPSS") test.

Null Hypothesis: The process is "trend" stationary.

Alternate Hypothesis : The series has a unit root (series is not stationary).

If the p-value is less than 0.05 than the null hypothesis is rejected, if not it is accepted.

Info from: https://www.statsmodels.org/devel/examples/notebooks/generated/stationarity_detrending_adf_kpss.html

I applied this to Volume, TradeCount and Price data all of which there windowed vectors were standardised. The window length was 6.

```
##   X ADF.p.values KPSS.p.values
## 1 0            0    0.05455437
## 2 1            0    0.03997104
## 3 2            0    0.10000000
## 4 3            0    0.10000000
## 5 4            0    0.07466947
## 6 5            0    0.03727858
```

```
##   X ADF.p.values KPSS.p.values
## 1 0 8.213410e-30           0.1
## 2 1 2.009483e-29           0.1
## 3 2 1.258790e-29           0.1
## 4 3 2.364373e-30           0.1
## 5 4 2.160855e-30           0.1
## 6 5 0.000000e+00           0.1
```

```
##   X ADF.p.values KPSS.p.values
## 1 0 9.154996e-30           0.1
## 2 1 0.000000e+00           0.1
## 3 2 3.213378e-30           0.1
## 4 3 2.475231e-30           0.1
## 5 4 3.747290e-30           0.1
## 6 5 6.967004e-30           0.1
```

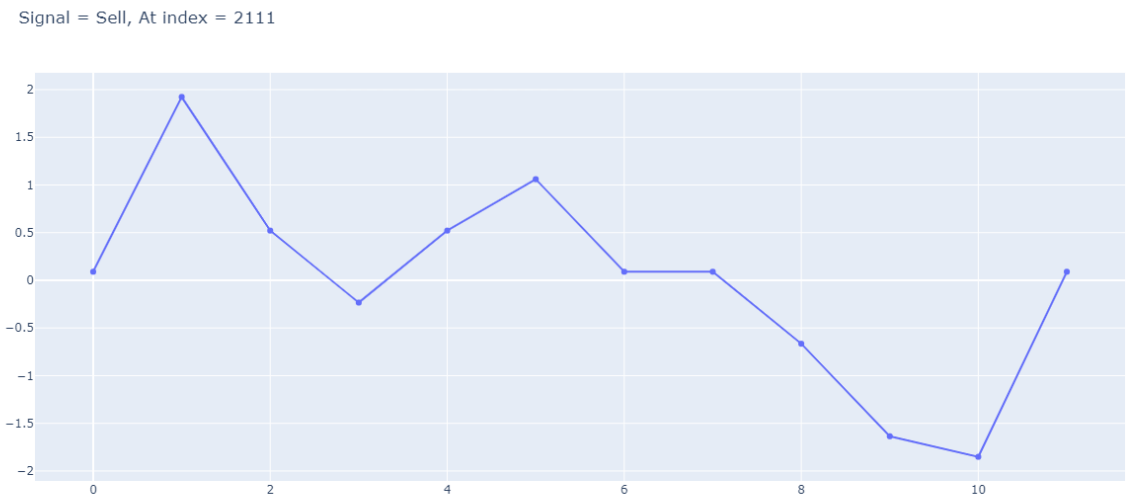
Looking at the above we can tell there is a strong indication that the sequence of vectors is indeed a stationary feature.

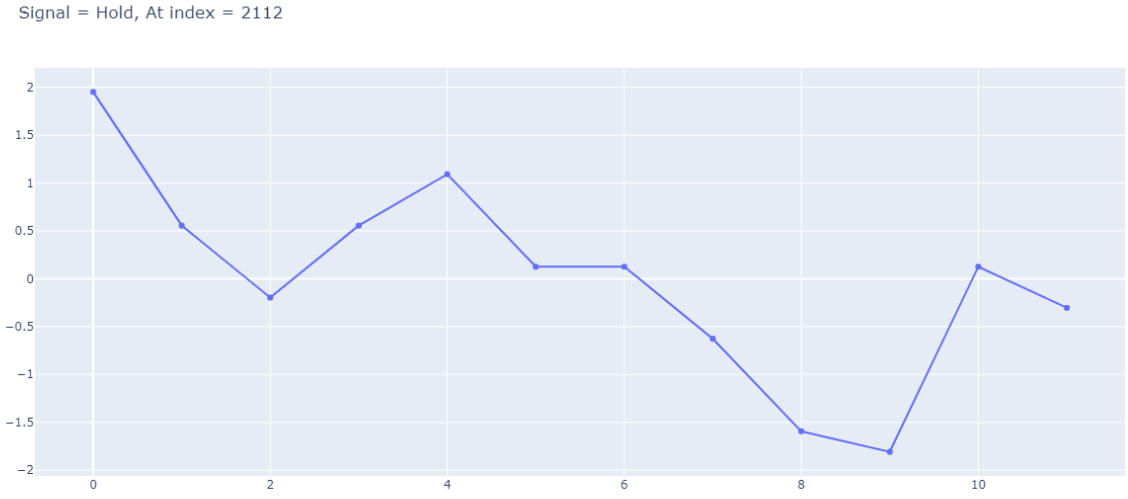
When looking at log returns however, the p value associated with that when using the ADF test is 1.118244e-28 which is very small and the p-value of the KPSS test is 0.1 which is the max it can be. Both indicating that the log returns is indeed stationary.

** I am not sure if the above is correct in regards to looking at the stationarity of multi-dimensional vectors**

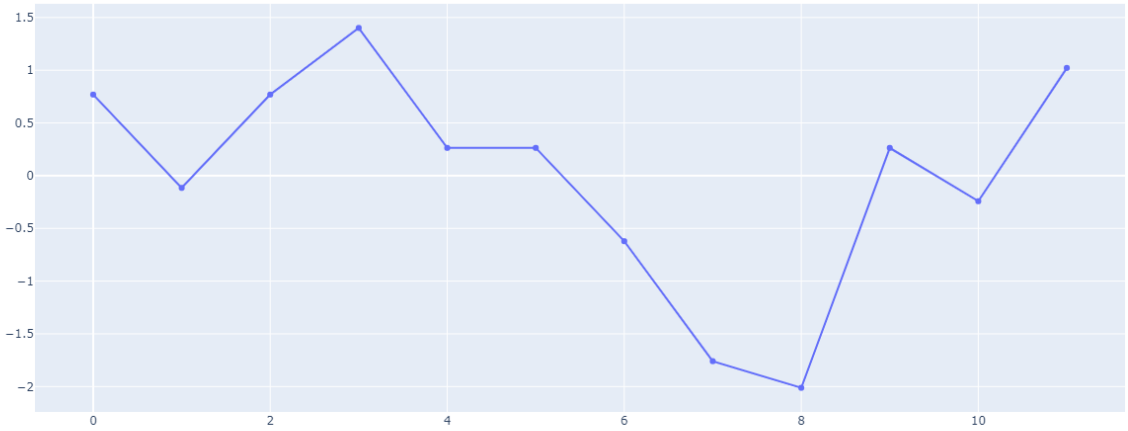
Despite this idea sounding reasonable in the first place, it may not be so reasonable. I think the reason why this NN won't work well as I would of thought initially, is because of the above points.

Here are a couple of images showcasing what essentially occurs:

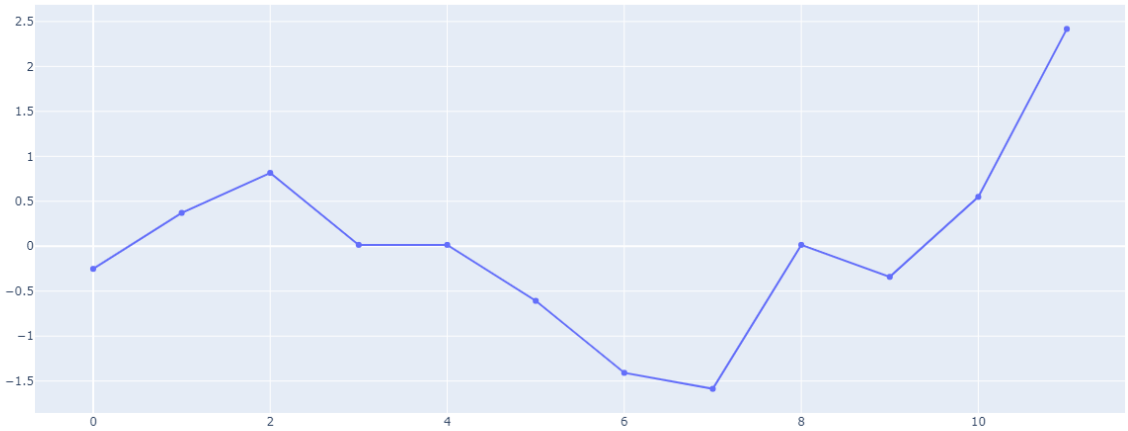


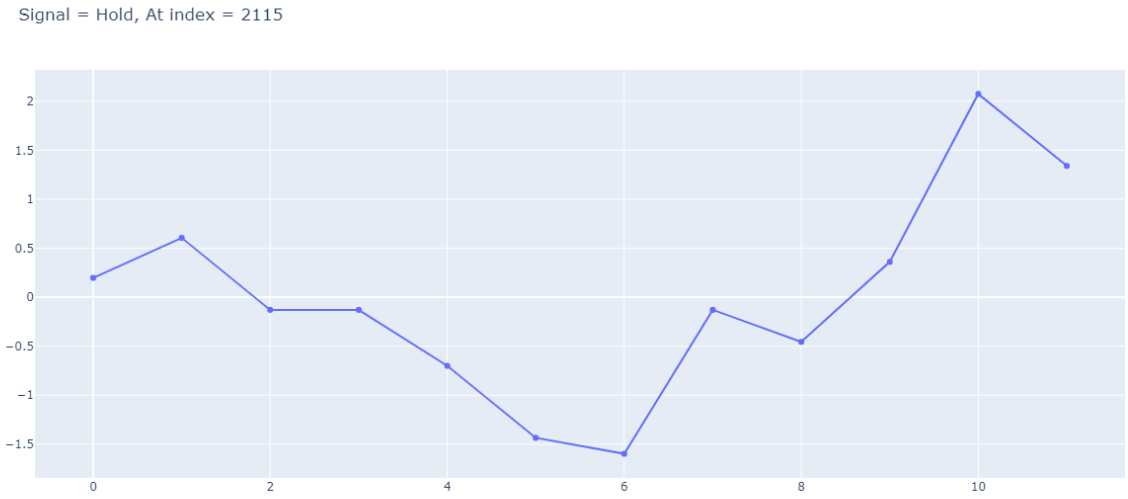


Signal = Buy, At index = 2113



Signal = Hold, At index = 2114





An advantage of standard scaling was that it allowed the neural network better to generalise and produce an approximation such that the training accuracy was very high, but the validation accuracy remained low. With min-max scaling the neural network could not even increase training accuracy. It is obvious that the neural network might have problems differentiating the movement of scaled prices in between say buy and hold when in many instances, such images seems very similar to the human eyes. For example, a human can easily most of the time differentiate between a cat and a dog, or a cat and a tiger but even for us differentiating between a buy and a hold signal is extremely tough and we rely on various external features too.

We can implement various different neural networks with varying structures but if there was an analysis of the explanatory variables used as well as the observations this idea won't seem like such a good one. Thus, we need to look at the types of variables we use and we need to understand that what we may need to predict might and should change, and for the better. Predicting trading signals might have seemed like a good idea as the neural network itself can be regarded as a complicated strategy that no one else is able to understand and thus no one else would be able to exploit the same hidden patterns it would be able to.

Cosine Similarity of vectors in between Buy and Hold, Buy and Sell and Hold and Sell.

We could have also included data such as Highs and Lows.

Insert Testing example:

A new outlook

Instead of predicting classifications, how about we generate predictions for actual values?

What to predict? What sort of values can we predict?

What relevance would past information have with future information?

Is it possible to predict how well a neural network or a hypothetical function can learn a task?

Given:

$$[P_0, P_1, P_2, ..., P_{n-2}, P_{n-1}]$$

Say we select a window of length k:

$$[P_i, P_{i+1}, P_{i+2}, ..., P_{k-2}, P_{k-1}]$$

Say we select another window of length j, but its strictly after the window of k. We give this window a special name, called future window.

$$[P_k, P_{k+1}, P_{k+2}, ..., P_{j-2}, P_{j-1}]$$

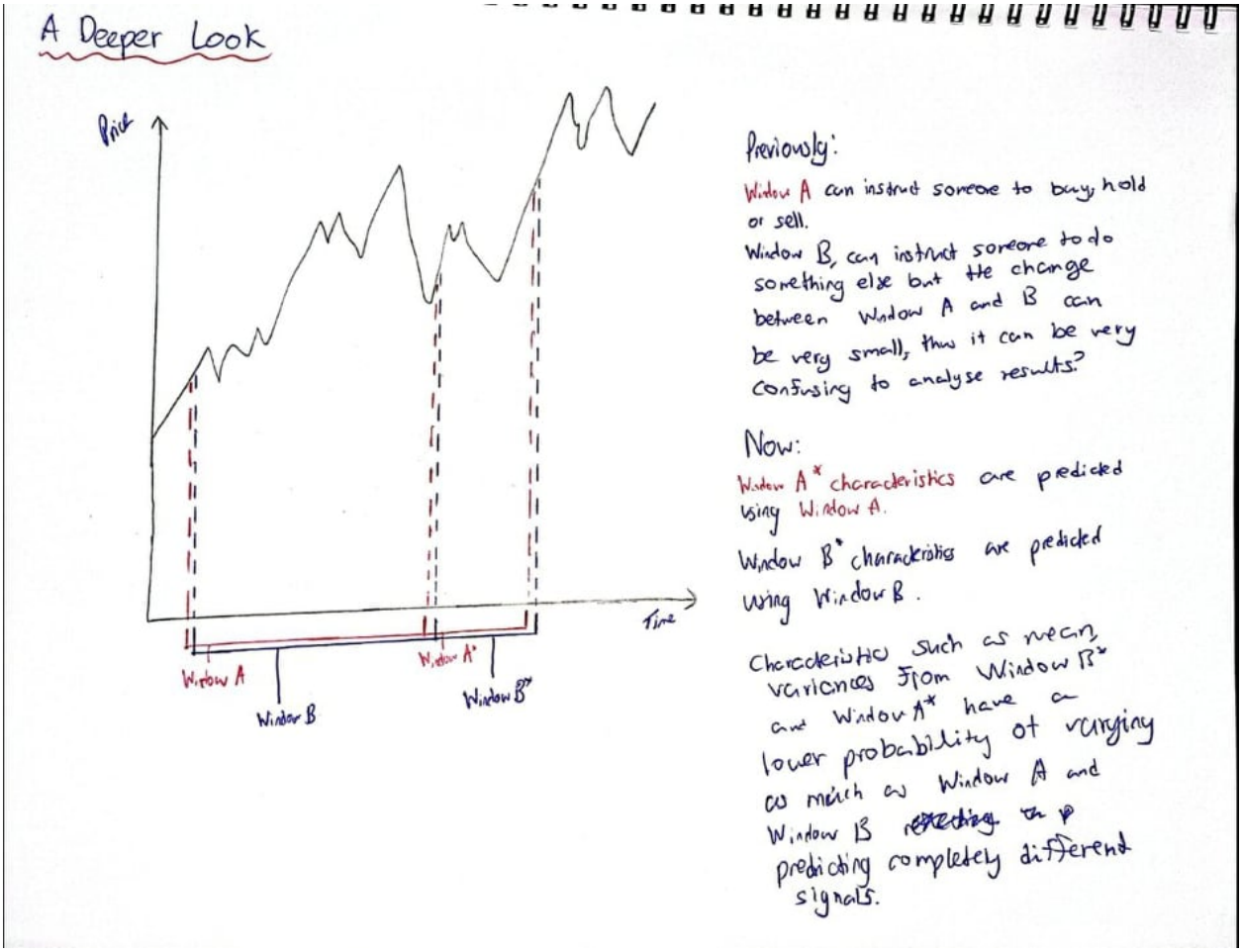
Can we use the results in the first window to predict some useful statistics in the second window?

Previously, we tried to train a neural network to predict signals based on pre-determined optimal signals. Essentially, when we buy we predict that the price will go up. When we sell we predict that the price will go down, but we predict that these things will happen within 1 to j time steps.

Instead of predicting the above, we could predict some of the following examples:

1. The Max returns if we bought now, within the future window?
2. The Minimum returns if we sold now, within the future window?
3. What the Average Returns in that period would be, within the future window?
4. The variance of returns within the future window?
5. What the linear regression slope would be if we used prices from P_{k-1} to P_{j-1}.

Here is an illustration:



How should we pre-process data ?

We use log returns of minute by minute data.

Further work is to be continued: