Candidate number: 10993

EE40098 - Computational Intelligence

Stock Market Predictions

18th December 2017 stock for MGAM will be:

Up/Down? = UP

MGAM = 323.59901177

Week long confidence level = 99.3205715027 % = range from 321.4 to 345.6

Word Count = 2300

Contents

1. Introduction 3
2. Method 3
    a. The Danger of Old Trend lines 4
    b. Simulated Annealing (SA) 5
    c. Validation of trend line order 5
    d. Localised Error Concerns 6
    e. Cheating the System 7
    f. Implementation of A Neural Network (ANN) 11
    g. Sweeping Training Data 13
    h. Hidden Node Iterations 14
    i. Further System Testing 17
3. Results 18
4. Conclusion 19
5. References 19

Word Count = 2300

1. Introduction

The Stock market is a superb facilitator to bright thoughts, attracting a vast range of approaches and techniques applied by some of the most qualified people in the world, all with one goal, to make money. With such prospects of huge financial gain greed often pays its toll and stock crashes threaten entire economies. Predicting the ups and downs correctly could potentially be a gold mine, not only financially but for use in helping to prevent future stock market crashes.

Within Computational Intelligence (EE40098) the mammoth task of predicting a stock's value, 1 day in advance, was set. The stock in question to this report is MGAM - Morgan Advanced Materials PLC, a suitably volatile stock, varying **18%** within the last **6** months (July 2017 to December 2017) and drops of up to **4%** a day in recent weeks.

Code was required to be produced in Python, using at least **1** technique taught within the module (EE40098). The author had no previous knowledge of Python and as such hindered the development of the algorithm substantially.

2. Method

Prediction, by very nature, requires historical information to generate a prediction (unless a blind guess is the goal), thus the goal of the code produced was to look at historical data to produce a potential future value. The only way (in which the author could imagine) of approaching this task is to produce a constantly updating trend line (train a trend line), in which the future value is taken from an extrapolation of the trend.

Word Count = 2300

a. The Danger of Old Trend lines

Figure 1 - Stock Market Trend line prediction (1)

Figure 1 - Stock Market Trend line prediction shows a volatile stock market example with multiple "generated" trend lines; this was produced with a Long Short Term Memory neural network (1). It must be noted that trend lines are all different, with the most recent trend line clearly being completely un-related to the first, thus implying the prediction is only accurate for a limited time frame. I.e. it would be unwise to use the first trend line (green) to predict the most recent stock value.

In order to produce an "accurate" trend line continuous training is required with new data to ensure that the weightings of the trend line function are being updated (or rather the neural networks hidden nodes weightings – discussed late). A neural network trained with data from two hundred days ago is unlikely to conform with the most recent ten days characteristics. Thus a rolling average was implemented.

b. Simulated Annealing (SA)

The first approach to produce a trend line was done so by developing further on "simulated Annealing" code provide by Prof Wilson (2). The basic principle of this is to begin with an initial guess, or rather at least a format, of the potential solution and iterate with a target in order to optimise the trend lines coefficients. Initially the system is incredible volatile and has a high "temperature", this influences the feedback error allowing for large jumps to be made, as the system continues to iterate the system "cools" and such the searching becomes more localised.

Conceptually simulated annealing appeared to be a suitable approach as with such a complicated system the error associated with different trend lines is complex. The order of the system was "declared" by the author to be 5th order, using an order lower would not be able to accurately overlay on the stock data, greater than 5th order would have been advantageous however as discussed below, is far too complex.

c. Validation of trend line order

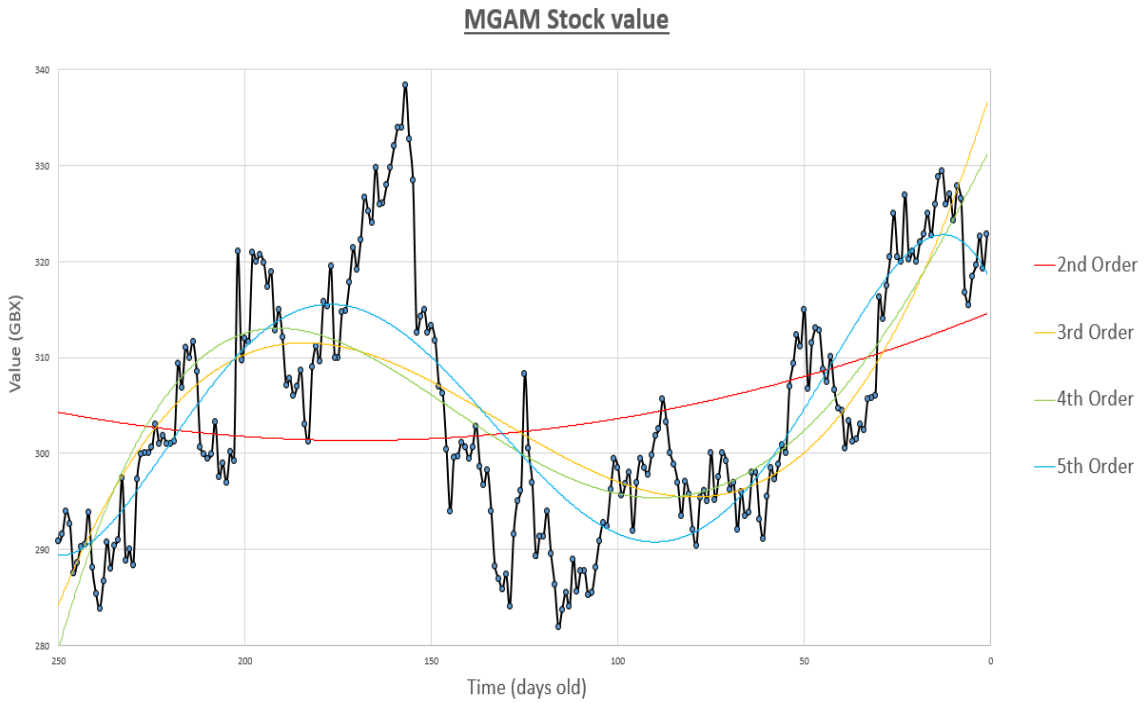

Figure 2 - MGAM, SA trend line orders

The Simulated Annealing code was designed to optimise the coefficients of a trend line equation. Figure 2 - MGAM, SA trend line orders shows the effect of increasing the order of the trend lines polynomial. Initially this declaration seems somewhat arbitrary, almost

common knowledge, however has more significant implications. Increasing the order beyond 1st order introduces potential infinite values (as comes with an ax^2 function). With a SA algorithm –especially a volatile one – it could potentially begin by making significant leaps, jumping into solutions that produce infinite values. Infinite values cannot be stored in python (or rather not that the author is aware of) and thus a stack overflow could occur (and did so on multiple occasions) crashing the program.

d. Localised Error Concerns

Initial large jumps in coefficients are necessary in order to jump out of local minima in the aim to find the global minima. Figure 4 - Error in 3d , is a "generic" 3d graph, the purpose of which is to illustrate various minima. Each axis represents a potential coefficient with total result showing the error term, however as the order increases the number of dimensions will also correlate, 5 dimensions is graphically challenging to show (hence the 3d example).

Further issues occur within SA algorithms are "relative jumps". A "perceived "large jump by the program could be enormous, completely jumping out of the "working area" (producing infinite solutions as previous discuss), or the jumps, in which the systems believes to be significant, are actually minuscule. This therefore implies that the initial "guess", or rather, starting conditions of the system must be reasonably accurate and close to the expected true value. This is problematic with such a high order system.

$$y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$

Figure 3 - 5th Order Trend line Format

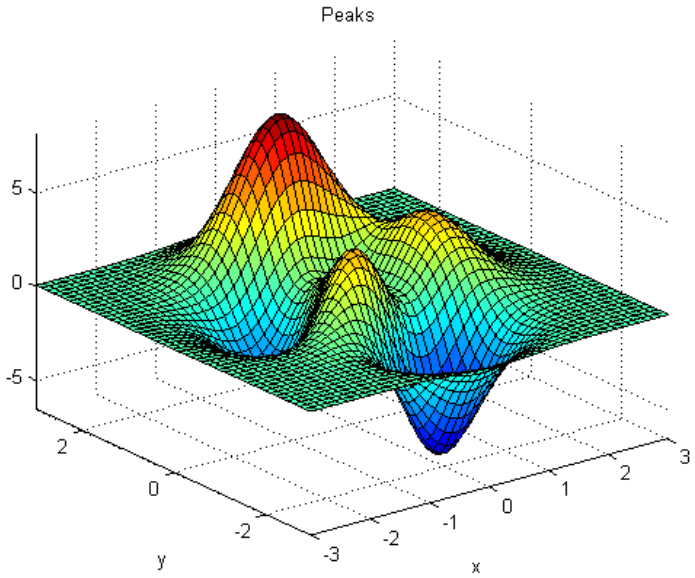

Figure 4 - Error in 3d (3)

Word Count = 2300

e. Cheating the System

Initial guesses at the coefficients for Figure 3 - 5th Order Trend line Format's were not good enough, producing output graphs with enormous exponentials and infinite solutions – see Figure 5 - SA enormous error

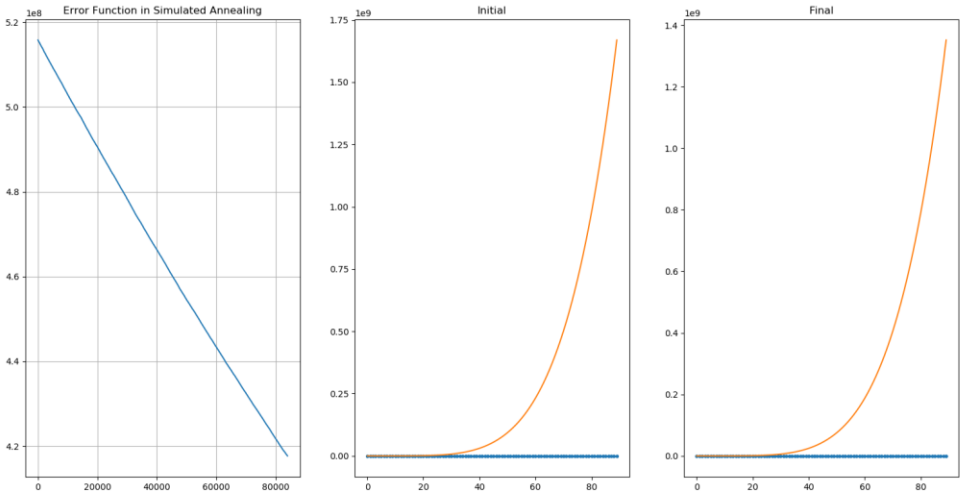

Figure 5 - SA enormous error

Even with "reasonable" manual iteration on the starting equation the program would often crash, thus a certain level of "cheating" was required. The data was plotted on Microsoft excel, with a 5th order trend line, and the equation from this was then used as the input equation. In theory this should be a reasonable guess:

Word Count = 2300

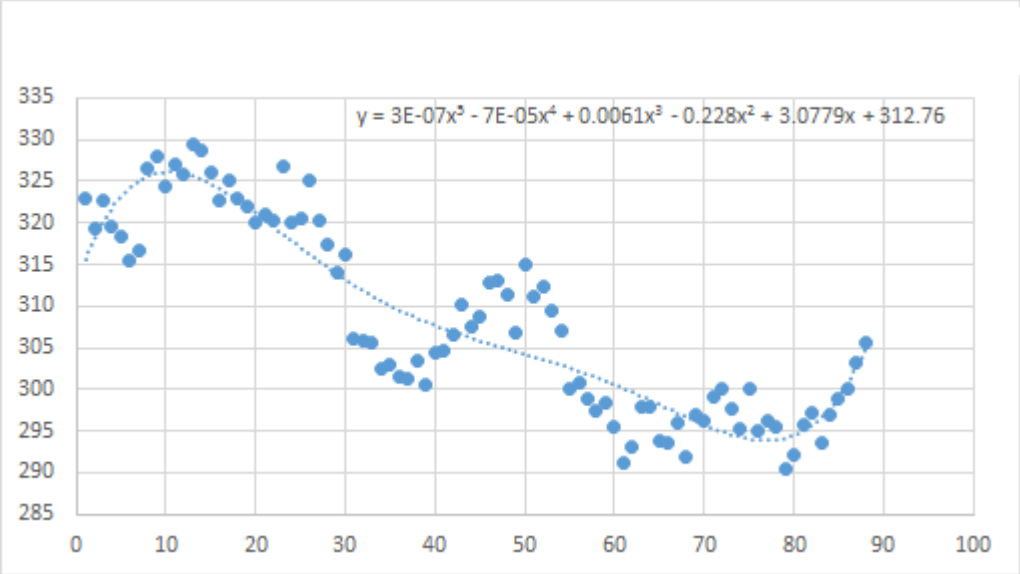

Figure 6 - Cheating

Humorously (or depressingly) the system still produced erratic results, however by increasing the cooling rate drastically helped to stabilise the system as large jumps in coefficients were less likely. Figure 7 – "Good results, with cheating" shows the output from the system when it is given the "correct" answer. The output can be seen to be "better" than the input graphically, however observing "Table 1 - Initial versus Final Parameters", the values of the equation change minimally (to be expected as we already gave it the right answer!).

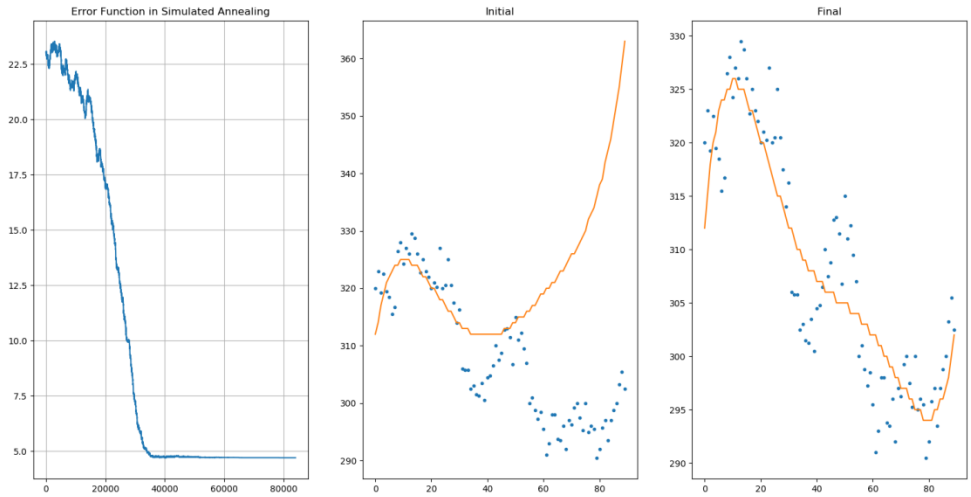

Figure 7 – "Good results, with cheating"

Word Count = 2300

$$y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$

Coefficient	a	b	c	d	e	f
Initial	3E-07	-7E-05	6.1E-03	-2.28E-01	3.0779	312.76
Final	2.99E-07	-7.036E-05	6.07E-03	-2.29E-01	3.0775	312.23

Table 1 - Initial versus Final Parameters

The final SA Parameters were:

 # Simulated Annealing Parameters

alpha=0.8

iterations = 200

var = 0.00001

initial_parameters = [312, 3.0779 , -0.228, 0.0061 , -0.00007 , 0.0000003]

- Code attached

From this, it was concluded that SA was not suitable (or too difficult to implement) and a different approach was required to predict the stock data. An overview of the SA can be seen in Figure 8 - SA Block Diagram

Word Count = 2300

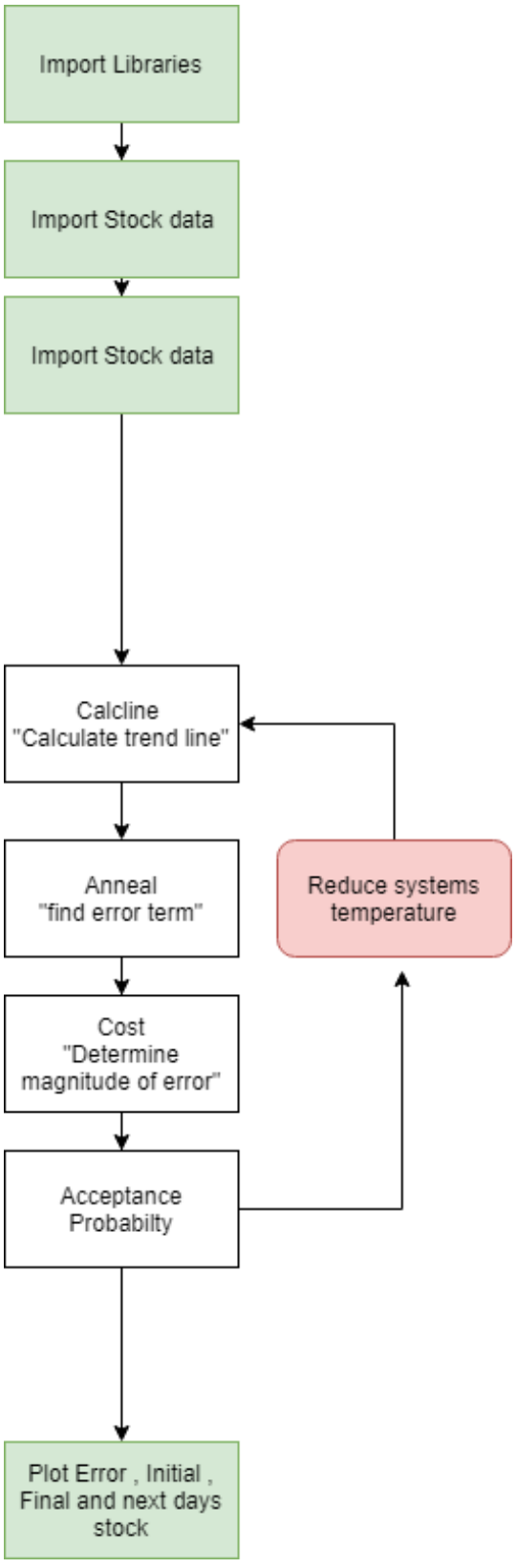

Figure 8 - SA Block Diagram

Word Count = 2300

f. Implementation of A Neural Network (ANN)

Beginning with the oldest data available (200 days old) a list of 9 days was used as inputs to a neural network to then produce a single output, the next day. As this was done so with historical data the true value of the "10th" day is known and thus a comparison of the predicted versus actual data point is achievable. The error term therefore is simply the prediction minus the actual value. This "stack" of inputs (length of 9 days) was then slid across the entire range (at 1 day increments) till eventually the end of the list (most recent data) is being reviewed, therefore producing the next future day (which does not yet exist). The oldest data therefore has the least impact on the neural network weightings.

Before discussing further the Sweep method, a view of the neural network is required – see Figure 9 - Neural Network Implementation

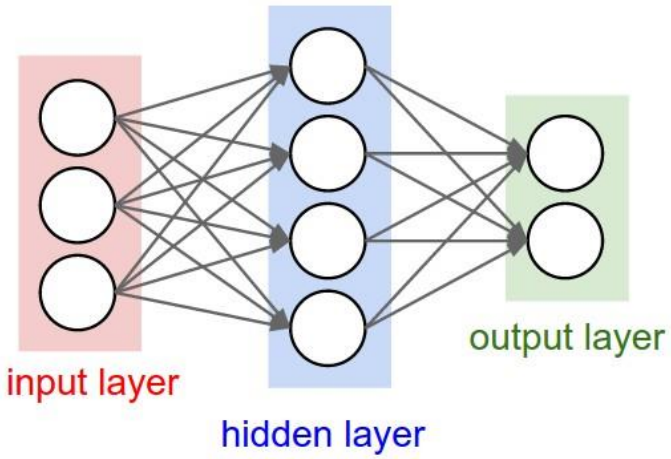

Figure 9 - Neural Network Implementation (4)

Figure 9 - Neural Network Implementation shows a basic visualisation of ANN. The input layer however contained 9 inputs; the hidden layer had 22 nodes, output layer having only 1 node. This values were found using iterative (manual) testing – discussed later

Word Count = 2300

The weightings on the hidden layers were initially random; meaning the error at the beginning of the program was significant (or fantastic once in a blue moon), however the error reduced as more training data was provided. As the program sweeps through the data the hidden layer values are constantly adjusted, therefore the oldest (first data to be used) has the least impact on the final values.

A Graph to show the influence level of input data

Figure 10 - Influence level of input data (5)

Figure 10 - Influence level of input data is a representation of the impact the training data has on the final values of the hidden nodes, the gradient of the graph will also be influenced by the "training value". The training value is a coefficient of the error term used as feedback to train the data.

Word Count = 2300

g. Sweeping Training Data

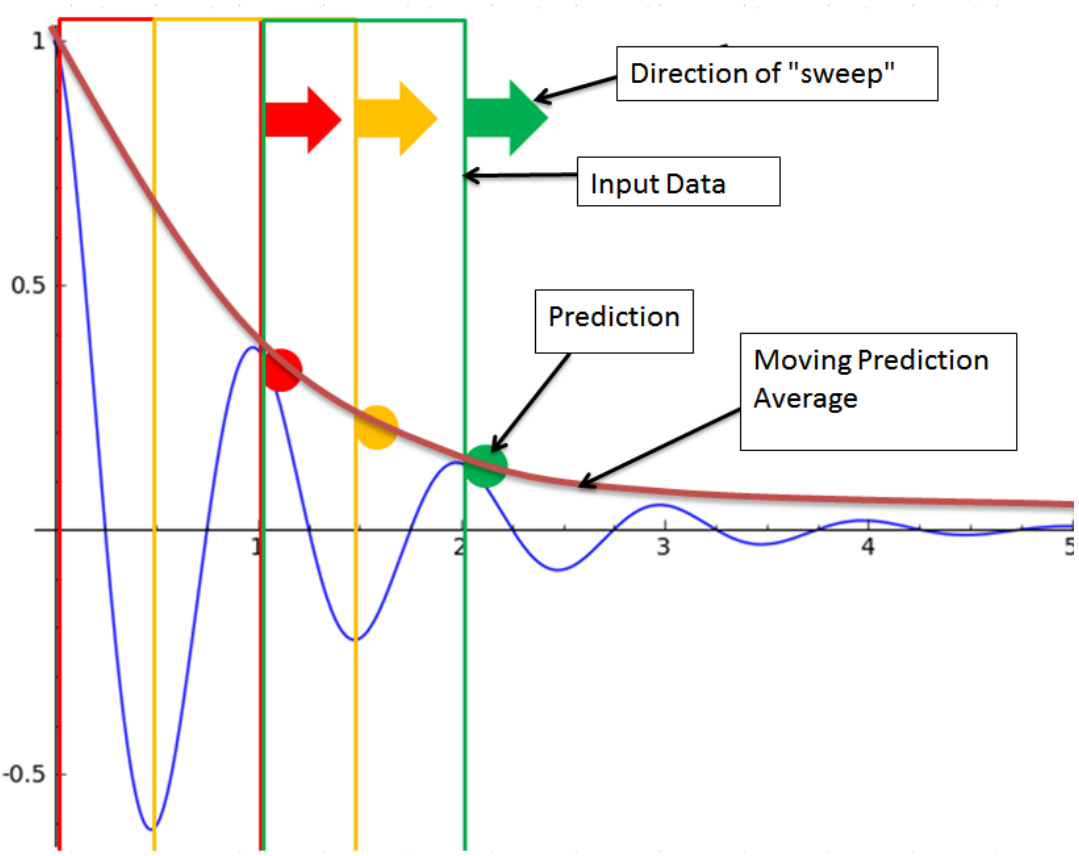

Figure 11 - Dampened Sine Sweep (6)

A more complete view of the principle behind the system is apparent in Figure 11 - Dampened Sine Sweep . In this example a dampened sine wave is the actual data, the first set of input data is provided to ANN (shown is red box) and the red circle represents the prediction, the inputs are then stepped (e.g. to orange) and the next output is given. The effective relationship of the neural network is represented by the “moving prediction average”, showing how the system constantly adjusts to new data.

This “sweep” train was implemented in to the code as follows in Figure 12 - Training Sweep

```
I = 220

for x in reversed(range(1,I)):

   inputs = (numpy.asfarray(close[(x+1):(x+10)]) / 500 * 0.99) + 0.01

   targets = (numpy.asfarray(close[x]) /500 *0.99) + 0.01

   N.train(inputs, targets)
```

Figure 12 - Training Sweep

h. Hidden Node Iterations

The number of hidden nodes is important, having too many (i.e. 1000) would require significant volumes of training data as it implies there are thousands of links/relationships between the input data. Furthermore having too many often resulted in highly volatile outputs, see Figure 13 - 1000 Hidden nodes. Although Figure - 13 demonstrates the algorithm is at least “doing something”, more often the output would latch to 500, the maximum output available.

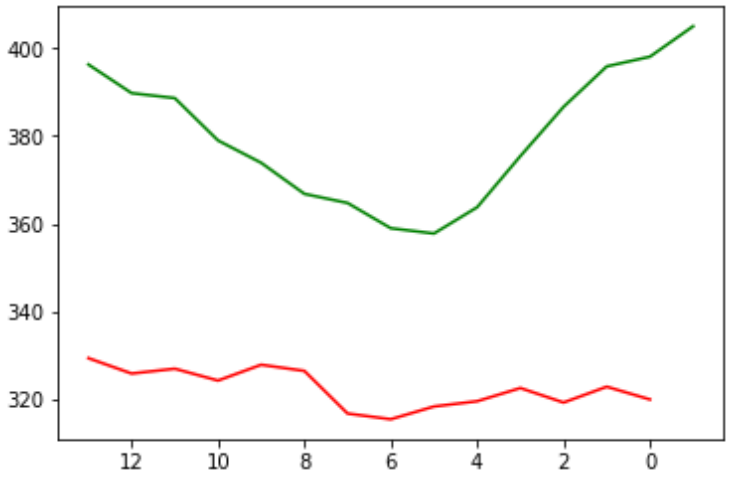

Figure 13 - 1000 Hidden nodes

Reducing the number of nodes had a positive impact on the error, up to a point. Having too few nodes resulted in a sluggish prediction, for example 1 hidden node resulted in an output that was clearly just a mean average of all the inputs. See Figure 14 - 1 Hidden Node . Red is input data, green in prediction.

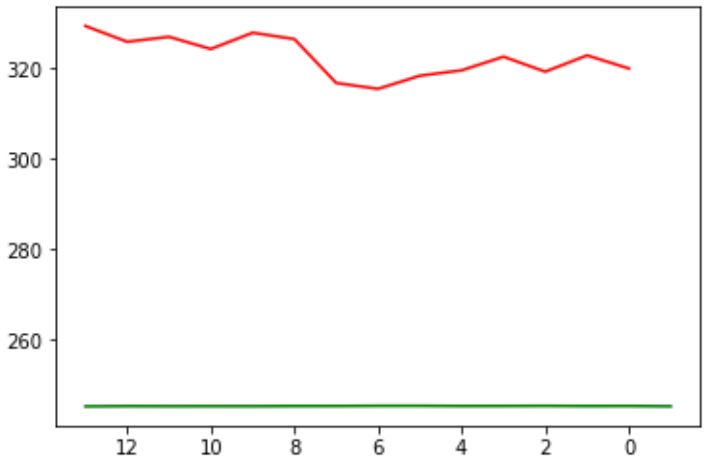

Figure 14 - 1 Hidden Node

In order to find the optimum number of hidden nodes another feedback could have been used, however was not implemented, clearly the required number of nodes is between 1 and 1000 however.

A manual iteration approach was used, 200 days of data were provided to the system (in blocks of 9), the final output was then compared to the actual stock value and the error was noted. The number of hidden nodes was then adjusted until the error appeared to no longer vary (within reason). This was slow, taking around 4 hours as each time the neural network was refreshed and re-trained. Furthermore the results must be taken with a pinch of salt, the initial weightings of the hidden nodes is set randomly each time, therefore a system run twice in succession will give two slightly different results, not ideal if the error term is trying to be compared.

A view of the neural network system, for reference, can be seen in Figure 15 - ANN block diagram.

Word Count = 2300

Figure 15 - ANN block diagram

Word Count = 2300

i. Further System Testing

Further testing on the system was performed, firstly, the task of predicting the "next day" from a perfectly straight line graph was performed. This performed the task perfectly, with the output mapping the input every time, this however was to be expected.

The second, more challenging task was to predict the next value from a Sine wave input, the period was initially set to 200 days and reduced until system failure occurred. A period of 28 days resulted in "questionable" results see Figure 16 - Sine wave predictions

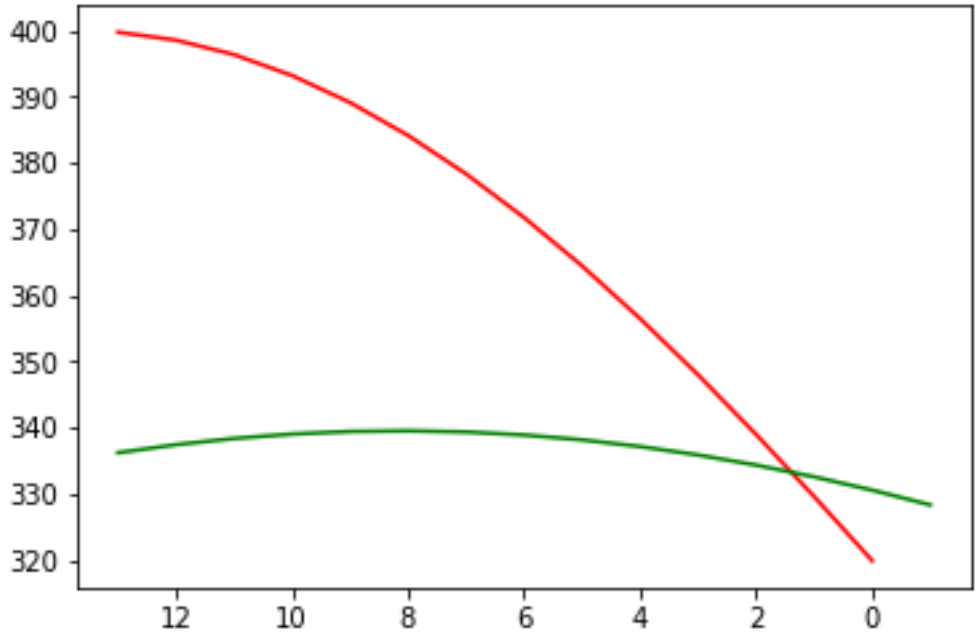

Figure 16 - Sine wave predictions

Again, the red is input, green is output, with 0 representing the present day. The prediction follows the general trend of the sine graph, however is too slow to respond, clearly there is a limitation on the neural network. In order to improve this more data points per day are provided, this however has the same effect as reducing the sine waves period. This was a significant find, as it implies that in order to produce more accurate predictions of the stock

market, more data points per day are required, rather than just the singular close value of the day.

Other stock data, such as Bitcoin value, was inputted into the neural network in order to test if the neural network produced wasn't sensitive to just MGAM stock data, the results produced were similar. Bitcoin, at the time of writing (December 2017) has just experienced abnormal exponential growth and thus was interesting to see ANN attempt to predict. However due to the "lag" of the neural network as only close values were used , the predictions were always "pessimistic" , however on the day of testing appeared to successfully predict the next day's crash (a humorous coincidence) ,but failed to predict the following spikes.

3. Results

The stock will go **Up** from Friday 15th December 2017, to Monday's Close on the 18th December 2017. See Figure 17 - Results, 18th Prediction, Figure 18 - MGAM Current Predictions

Tomorrow's stock will be:

323.59901177

Week long confidence level =

99.3205715027 %

Week long risk level =

1.29267860589 %

Figure 17 - Results, 18th Prediction

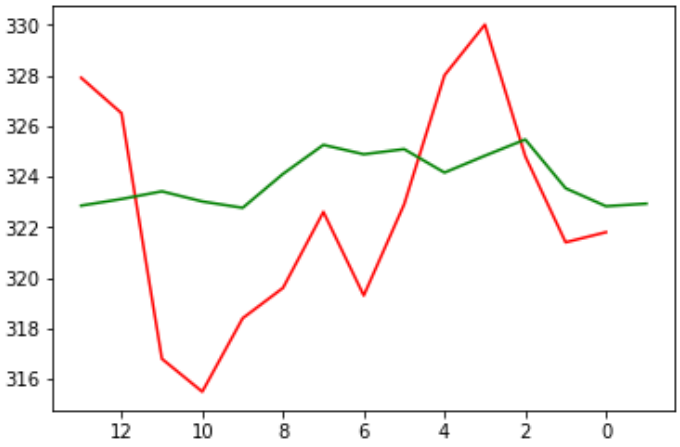

Figure 18 - MGAM Current Predictions

Word Count = 2300

4. Conclusion

In conclusion, the algorithms implemented, ANN and SA, produced interesting results. SA proved to be very volatile and sensitive to initial starting conditions when asked to predict complex trend lines. The neural network produced consistently "lagging" predictions. In order to improve the systems performance more data is necessary, and would need to be constantly self-updating and training in order to produce "valid" results.

References

1. **http://www.jakob-aungiers.com/img/article/lstm-neural-network-timeseries/stockmultseqprediction.png. [Online]**

2. Prof. P Wilson, EE40098.

3. MATHWORKS. *https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/27178/versions/6/previews/html/psPeaksExample_01.png.* [Online]

4. http://cs231n.github.io/assets/nn1/neural_net.jpeg. [Online]

5. *https://www.wolframalpha.com/input/?i=e%5Ex+%2B+1.* [Online]

6. *https://upload.wikimedia.org/wikipedia/commons/2/25/DampedSine.png.* [Online]

7. Metcalfe, Dr.B. *EE40098 Computational Intelligence - Lecture 05.*

Word Count = 2300