Stock Price Predictor

Vilius Radavicius

Soft 050

Student Id: 10652211

Contents

Table of Contents

Analysis

About the project

Stock prediction software are extremely useful and convenient way to ensure that the users are making a safe investment in certain types of stocks. Although, you can make investments without the special software but the risk is significantly higher and not recommended. In the US, about 70 percent of overall trading volume is generated through algorithmic trading.[1] Thus the use of Stock prediction software is highly recommended and way more safer than basic ordinary stock trading.

The stock prediction software will be using complicated neural network algorithms, in order to actually be able to predict the fate of a stock with the given data. This is a very crucial part of the program as if without it, the program would just start predicting pure random numbers in which is highly inaccurate and would be essentially useless to be used in the stock market business.

This project will require an Api to be used in which will allow us to fetch real-world live data from the chosen stocks. Although the rest of the code will be hard coded as it will give a better customisability and will be able to have its own custom neural net architecture.

Problems

One of the problems on this project is the difficulty of making the neural net to predict the stock market values. Instead of using a simple deep neural network with just a couple of hidden layers, we will have use something called a recurrent neural network in which Is a bit more complicated and difficult to make rather than a traditional vanilla neural network.

The main disadvantage about this project is that, this stock market prediction software will not have features to actually buy/sell stocks for as this is a platform to inform the client about the predicted prices of a specific stock.

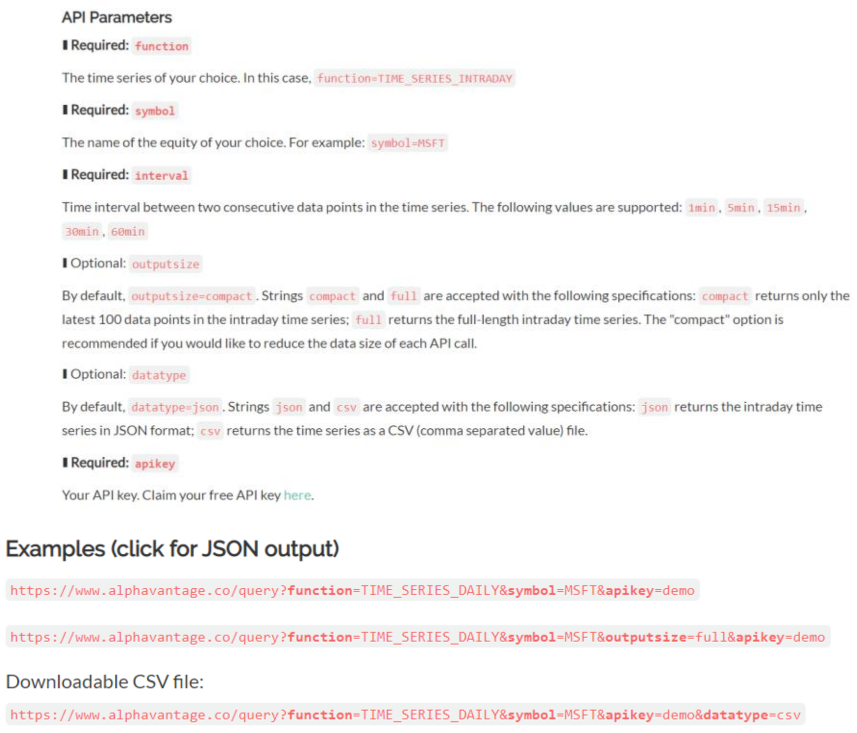
Research

Alpha vantage(API)

This api is a very effective and the most popular option as it is easy to use and has really accurate data. Alpha vantage provides in both realtime data as well as some previous historical data. Alpha vantage includes a free version in which will only allow "(5 API requests per minute; 500 API requests per day)" Whilst as the premium versions go from 30 api requests per minute all the way up to 1200 and the prices are extremely high and should be used by professional profitable stock brokers.[2]

One key advantage of using Alpha vantage is that the data can be retrieved from a JSON output as well as a CSV file.

One key disadvantage is the prices for the premium version and the fact that this will be limited to 5 api requests per minute.



API Parameters

❚ Required: `function`

The time series of your choice. In this case, `function=TIME_SERIES_INTRADAY`

❚ Required: `symbol`

The name of the equity of your choice. For example: `symbol=MSFT`

❚ Required: `interval`

Time interval between two consecutive data points in the time series. The following values are supported: `1min`, `5min`, `15min`, `30min`, `60min`

❚ Optional: `outputsize`

By default, `outputsize=compact`. Strings `compact` and `full` are accepted with the following specifications: `compact` returns only the latest 100 data points in the intraday time series; `full` returns the full-length intraday time series. The "compact" option is recommended if you would like to reduce the data size of each API call.

❚ Optional: `datatype`

By default, `datatype=json`. Strings `json` and `csv` are accepted with the following specifications: `json` returns the intraday time series in JSON format; `csv` returns the time series as a CSV (comma separated value) file.

❚ Required: `apikey`

Your API key. Claim your free API key here.

Examples (click for JSON output)

`https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=MSFT&apikey=demo`

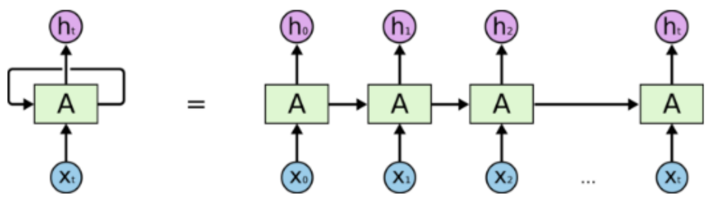
`https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=MSFT&outputsize=full&apikey=demo`

Downloadable CSV file:

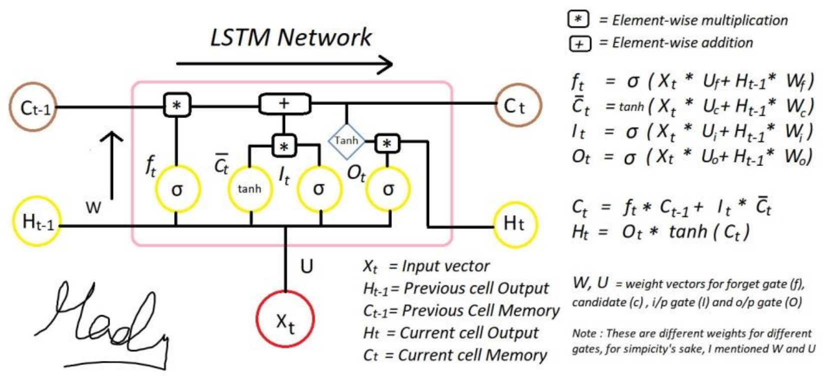
`https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=MSFT&apikey=demo&datatype=csv`

Recurrent Neural Network (RNN)

[3]As a normal vanilla recurrent neural network has a good advantage of having a memory of what it did before in which is extremely useful especially in stock market prediction as neural net needs to know if the prices is slowly descending or ascending and here's where RNN comes in, the idea behind it is that the previous hidden layer value is being merged into the current hidden layer thus giving the neural image [6]net access to the data that was the cycle before. Problem with this is that after a couple of cycles the gradient would usually explode and thus making the entire neural net not function. Fix for this is a unit called LSTM, essentially this is a very good method of to implement the whole remembering previous cycles function as well as making sure that the gradient doesn't explode. See link [4]



An unrolled recurrent neural network.



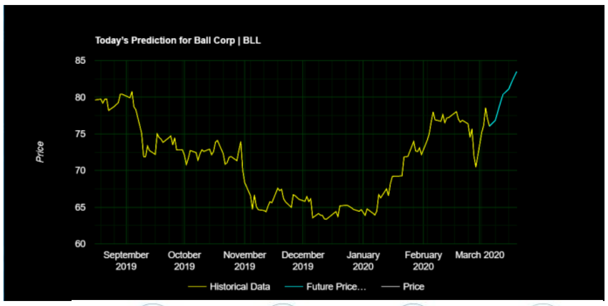
LSTM Network
C_{t-1}
C_t
H_{t-1}
H_t
W
U
X_t
f_t
$\bar{C}_t$
I_t
O_t
σ
tanh
Tanh
∗ = Element-wise multiplication
+ = Element-wise addition
$f_t = \sigma(X_t * U_f + H_{t-1} * W_f)$
$\bar{C}_t = \tanh(X_t * U_c + H_{t-1} * W_c)$
$I_t = \sigma(X_t * U_i + H_{t-1} * W_i)$
$O_t = \sigma(X_t * U_o + H_{t-1} * W_o)$
$C_t = f_t * C_{t-1} + I_t * \bar{C}_t$
$H_t = O_t * \tanh(C_t)$
X_t = Input vector
H_{t-1} = Previous cell Output
C_{t-1} = Previous Cell Memory
H_t = Current cell Output
C_t = Current cell Memory
W, U = weight vectors for forget gate (f), candidate (c), i/p gate (I) and o/p gate (O)
Note : These are different weights for different gates, for simpicity's sake, I mentioned W and U

The feed forward of LSTM is fairly simple to implement, however back propagation on an LSTM layer is challenging, but luckily there's a great example showing how to perform back-propagation on LSTM layer.[5]

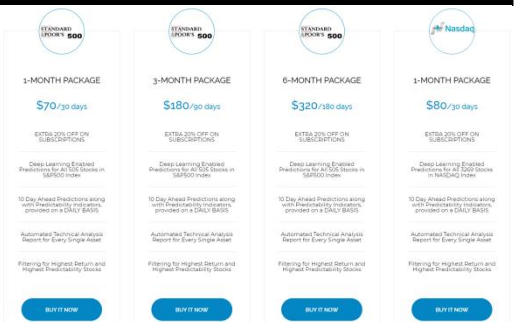
Already existing projects

FinBrain Technologies:

One very popular stock price prediction program is called FinBrain Technologies



As you may see, they produce a graph of historical data as well as the predicted data. This specific software offers a lot of different stocks, from crypto currencies to Forex and Nasdaq.



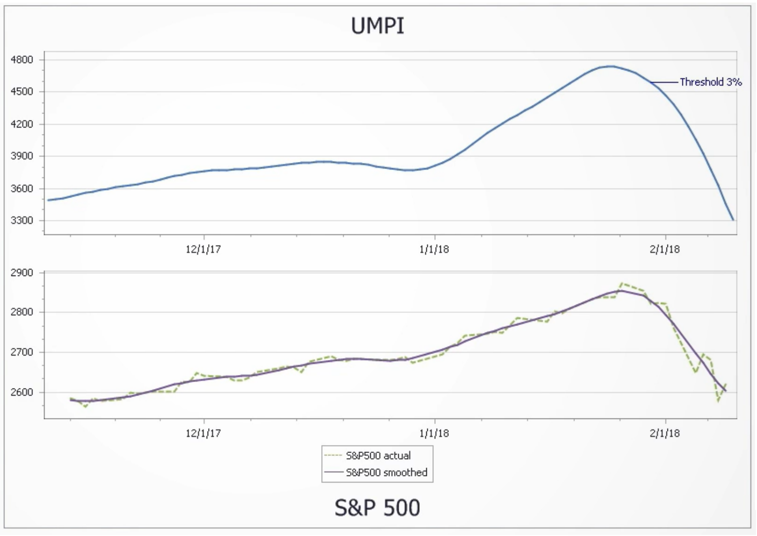
STANDARD & POOR'S 500	STANDARD & POOR'S 500	STANDARD & POOR'S 500	Nasdaq
1-MONTH PACKAGE	3-MONTH PACKAGE	6-MONTH PACKAGE	1-MONTH PACKAGE
$70/30 days	$180/90 days	$320/180 days	$80/30 days
EXTRA 20% OFF ON SUBSCRIPTIONS	EXTRA 20% OFF ON SUBSCRIPTIONS	EXTRA 20% OFF ON SUBSCRIPTIONS	EXTRA 20% OFF ON SUBSCRIPTIONS
Deep Learning Enabled Predictions for All 505 Stocks in S&P500 Index	Deep Learning Enabled Predictions for All 505 Stocks in S&P500 Index	Deep Learning Enabled Predictions for All 505 Stocks in S&P500 Index	Deep Learning Enabled Predictions for All 3269 Stocks in NASDAQ Index
10 Day Ahead Predictions along with Predictability Indicators, provided on a DAILY BASIS	10 Day Ahead Predictions along with Predictability Indicators, provided on a DAILY BASIS	10 Day Ahead Predictions along with Predictability Indicators, provided on a DAILY BASIS	10 Day Ahead Predictions along with Predictability Indicators, provided on a DAILY BASIS
Automated Technical Analysis Report for Every Single Asset	Automated Technical Analysis Report for Every Single Asset	Automated Technical Analysis Report for Every Single Asset	Automated Technical Analysis Report for Every Single Asset
Filtering for Highest Return and Highest Predictability Stocks	Filtering for Highest Return and Highest Predictability Stocks	Filtering for Highest Return and Highest Predictability Stocks	Filtering for Highest Return and Highest Predictability Stocks
BUY IT NOW	BUY IT NOW	BUY IT NOW	BUY IT NOW

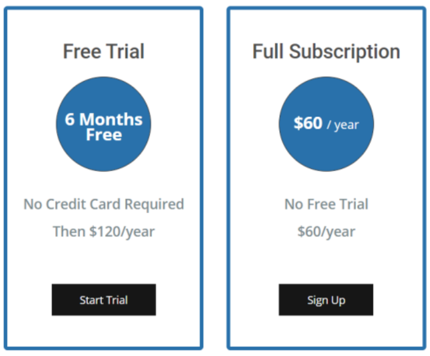
The main downside of this program is the over expensive price for the software. There for I will make this project completely for free other than if the person would like to purchase the better api key from alpha vantage.

https://finbrain.tech/#/home

Umpindex: https://www.umpindex.com/

This is yet another stock prediction software and as you may see it is mostly accurate but this one is giving out a free 6month membership but yet again, they will charge you a lot of money, although this is a much cheaper option than the previous one.




Free Trial

6 Months Free

No Credit Card Required

Then $120/year

Start Trial

Full Subscription

$60 / year

No Free Trial

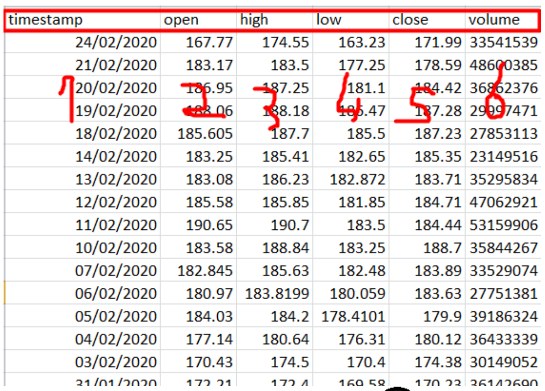
$60/year

Sign Up

Methods and algorithms

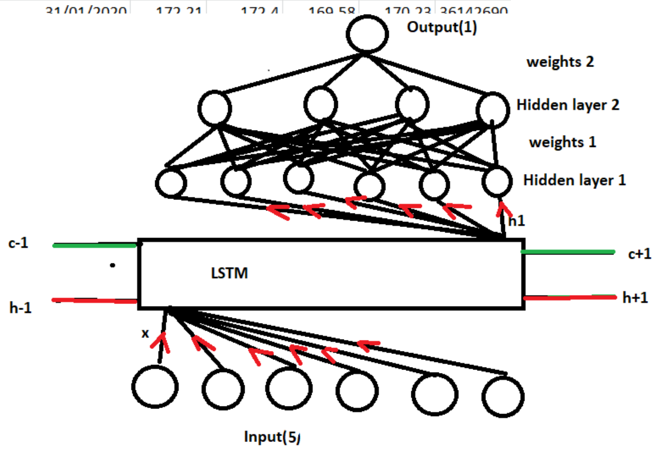
The entire project will be programmed in c# language and with the use of visual studio 2019 as the ide, the reason behind this is because, visual studio offers a great debugging tool for discovering problems within the code and c# has a lot of potential to be used in a large variety of projects.

As stated on page above, I will be using a specific type of neural net (rnn) and I will be recreating the lstm layer for the rnn to prevent the gradient from exploding and breaking the entire program. The project however will contain a custom model design.



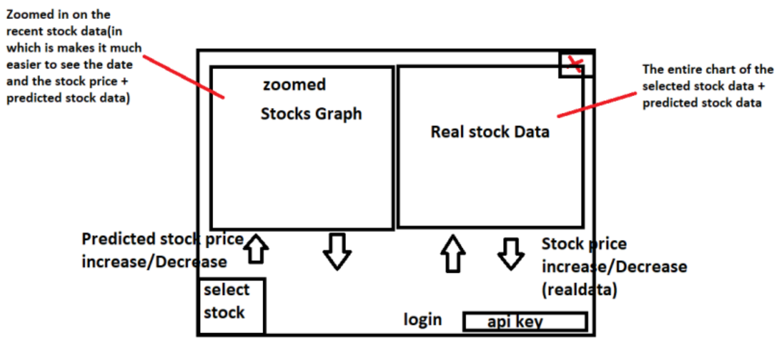
timestamp	open	high	low	close	volume
24/02/2020	167.77	174.55	163.23	171.99	33541539
21/02/2020	183.17	183.5	177.25	178.59	48600385
20/02/2020	186.95	187.25	181.1	184.42	36862376
19/02/2020	[illegible]	188.18	[illegible]	187.28	[illegible]
18/02/2020	185.605	187.7	185.5	187.23	27853113
14/02/2020	183.25	185.41	182.65	185.35	23149516
13/02/2020	183.08	186.23	182.872	183.71	35295834
12/02/2020	185.58	185.85	181.85	184.71	47062921
11/02/2020	190.65	190.7	183.5	184.44	53159906
10/02/2020	183.58	188.84	183.25	188.7	35844267
07/02/2020	182.845	185.63	182.48	183.89	33529074
06/02/2020	180.97	183.8199	180.059	183.63	27751381
05/02/2020	184.03	184.2	178.4101	179.9	39186324
04/02/2020	177.14	180.64	176.31	180.12	36433339
03/02/2020	170.43	174.5	170.4	174.38	30149052
31/01/2020	172.21	172.4	169.58	170.23	36142690

The best way to figure out what the model will look like, it's to first take a look at the inputs and the way that this AI will work is that it will be trained on 5 different inputs as you can see in the screenshot of real data from Alpha vantage, this should give the neural net enough information to be trained from. Although the original plan was to use 6 inputs but after some realisation it would complicate the project and there isn't much use for feeding the neural net with the date. You may see in the



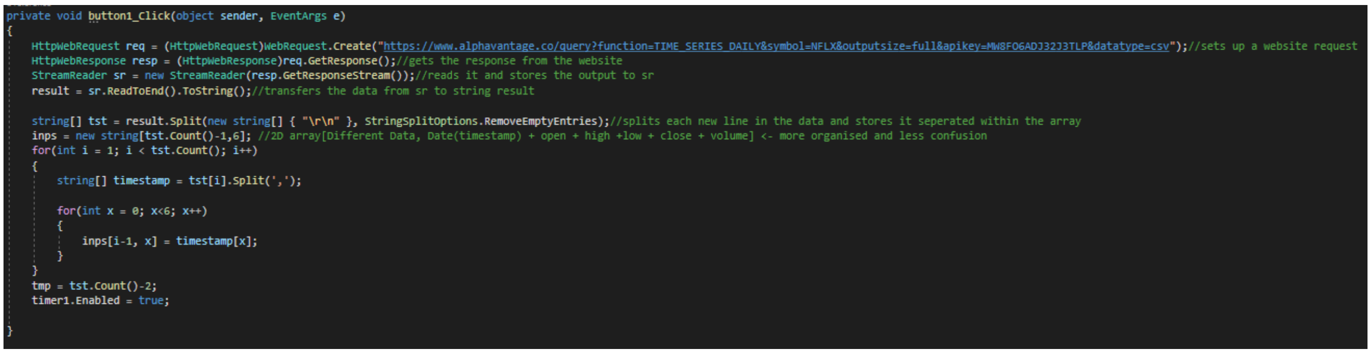
example of how the data will be selected from the file, it will start by taking the last element (oldest stock) and feeding it through the neural net then later the pointer will move up by one and work out the cost using the newer results and then later use the newer results for feed forwarding and repeat till the pointer reaches the top in which the neural net should be updated with the pattern of the stock.

User interface:

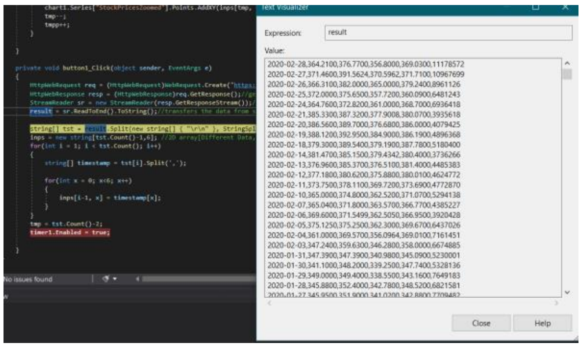

Zoomed in on the recent stock data(in which is makes it much easier to see the date and the stock price + predicted stock data)
zoomed Stocks Graph
Real stock Data
The entire chart of the selected stock data + predicted stock data
Predicted stock price increase/Decrease
Stock price increase/Decrease (realdata)
select stock
login
api key

Development

The way that I wanted to obtain the data is through the use of alpha vantage api. As the user initialises the stocks by clicking a button, a http request is sent to alphavantage to retrieve a csv file and then to be stored into a long string to be processed into a manageable array.



```
private void button1_Click(object sender, EventArgs e)
{
    HttpWebRequest req = (HttpWebRequest)WebRequest.Create("https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=NFLX&outputsize=full&apikey=MW8FO6ADJ32J3TLP&datatype=csv");//sets up a website request
    HttpWebResponse resp = (HttpWebResponse)req.GetResponse();//gets the response from the website
    StreamReader sr = new StreamReader(resp.GetResponseStream());//reads it and stores the output to sr
    result = sr.ReadToEnd().ToString();//transfers the data from sr to string result

    string[] tst = result.Split(new string[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);//splits each new line in the data and stores it seperated within the array
    inps = new string[tst.Count()-1,6]; //2D array[Different Data, Date(timestamp) + open + high +low + close + volume] <- more organised and less confusion
    for(int i = 1; i < tst.Count(); i++)
    {
        string[] timestamp = tst[i].Split(',');

        for(int x = 0; x<6; x++)
        {
            inps[i-1, x] = timestamp[x];
        }
    }
    tmp = tst.Count()-2;
    timer1.Enabled = true;

}
```



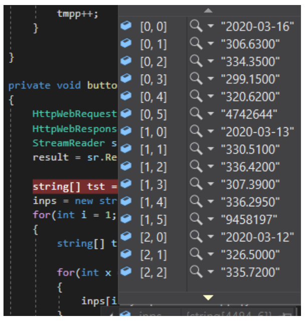
Expression: result

Value:

```
2020-02-28,364.2100,376.7700,356.8000,369.0300,11178572
2020-02-27,371.4600,391.5624,370.5962,371.7100,10967699
2020-02-26,366.3100,382.0000,365.0000,379.2400,8961126
2020-02-25,372.0000,375.6500,357.7200,360.0900,6481243
2020-02-24,364.7600,372.8200,361.0000,368.7000,6936418
2020-02-21,385.3300,387.3200,377.9008,380.0700,3935618
2020-02-20,386.5600,389.7000,376.6800,386.0000,4079425
2020-02-19,388.1200,392.9500,384.9000,386.1900,4896368
2020-02-18,379.3000,389.5400,379.1900,387.7800,5180400
2020-02-14,381.4700,385.1500,379.4342,380.4000,3736266
2020-02-13,376.9600,385.3700,376.5100,381.4000,4485383
2020-02-12,377.1800,380.6200,375.8800,380.0100,4624772
2020-02-11,373.7500,378.1100,369.7200,373.6900,4772870
2020-02-10,365.0000,374.8000,362.5200,371.0700,5294138
2020-02-07,365.0400,371.8000,363.5700,366.7700,4385227
2020-02-06,369.6000,371.5499,362.5050,366.9500,3920428
2020-02-05,375.1250,375.2500,362.3000,369.6700,6437026
2020-02-04,361.0000,369.5700,356.0964,369.0100,7161451
2020-02-03,347.2400,359.6300,346.2800,358.0000,6674885
2020-01-31,347.3900,347.3900,340.9800,345.0900,5230001
2020-01-30,341.1000,348.2000,339.2500,347.7400,5328136
2020-01-29,349.0000,349.4000,338.5500,343.1600,7649183
2020-01-28,345.8800,352.4000,342.7800,348.5200,6821581
2020-01-27,345.9500,351.9000,341.0200,342.8800,7709482
```

As I did some testing, it clearly stored all the data needed into a single string, so the next step after that was to split up the information so accessing different sections of the stock data would be more convenient.

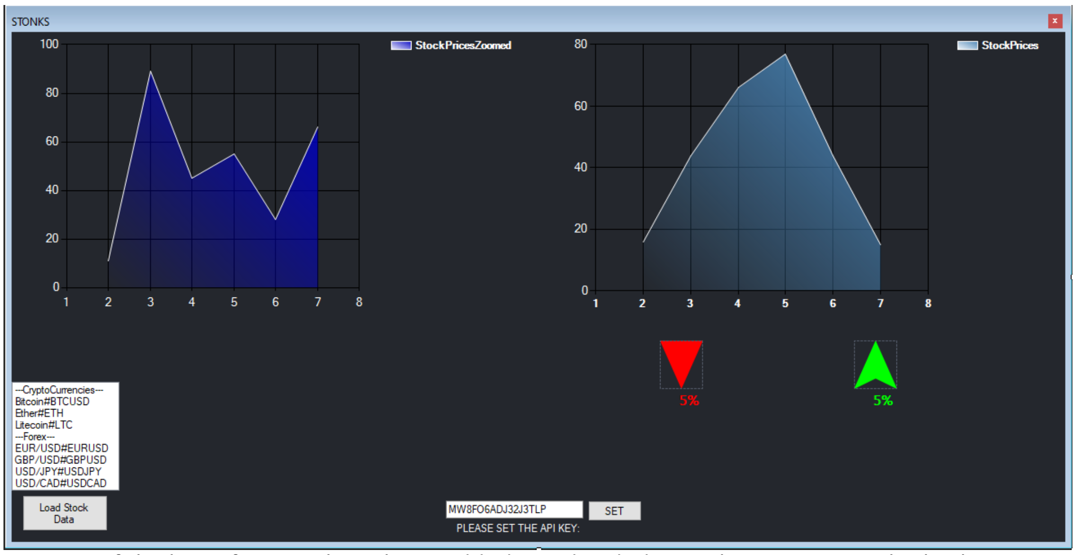
After the data is split up, it looked like this, each different information about the stocks are now



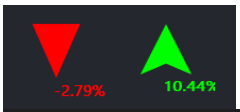
```
[0, 0]  "2020-03-16"
[0, 1]  "306.6300"
[0, 2]  "334.3500"
[0, 3]  "299.1500"
[0, 4]  "320.6200"
[0, 5]  "4742644"
[1, 0]  "2020-03-13"
[1, 1]  "330.5100"
[1, 2]  "336.4200"
[1, 3]  "307.3900"
[1, 4]  "336.2950"
[1, 5]  "9458197"
[2, 0]  "2020-03-12"
[2, 1]  "326.5000"
[2, 2]  "335.7200"
```

able to be directly accessed in which will definitely make it much easier to use the data and to feed the neural net with.

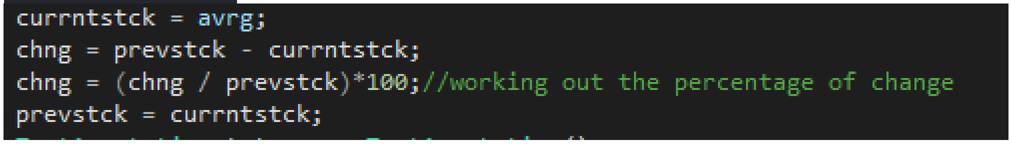
Version 1.0



As Most of the basic features have been added in, I decided to make my program look a bit neater and added in a couple of useful features, in which they're the following:



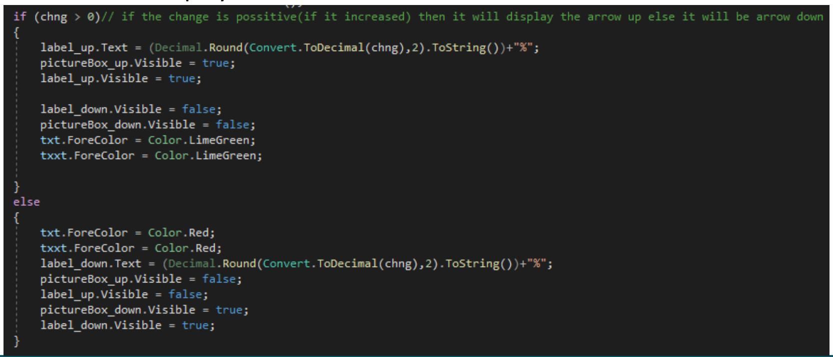
The main two arrows, are the percentage of change that I decided to add, this will give the user an idea of the current situation of the stock market.



```
currntstck = avrg;
chng = prevstck - currntstck;
chng = (chng / prevstck)*100;//working out the percentage of change
prevstck = currntstck;
```

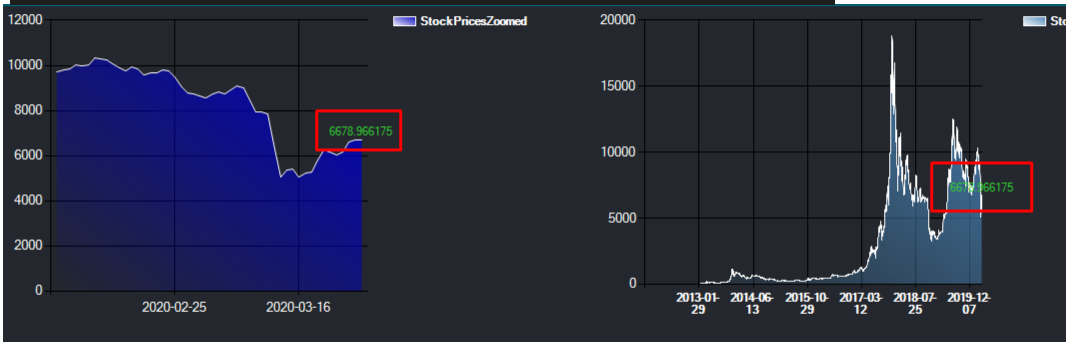
To calculate it, was pretty simple, all you do is get the current stock price and you minus the previous stock price by the current one, then you divide the change by the previous stock and times it by a 100 and this should give you the percentage of change.

After that you just simply check if the change is greater than 0 and if so then it will display the green arrow and if not it will display the red arrow.



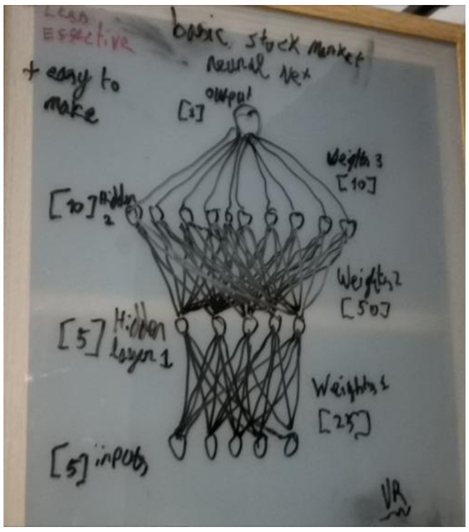
```
if (chng > 0)// if the change is possitive(if it increased) then it will display the arrow up else it will be arrow down
{
    label_up.Text = (Decimal.Round(Convert.ToDecimal(chng),2).ToString())+"%";
    pictureBox_up.Visible = true;
    label_up.Visible = true;

    label_down.Visible = false;
    pictureBox_down.Visible = false;
    txt.ForeColor = Color.LimeGreen;
    txxt.ForeColor = Color.LimeGreen;

}
else
{
    txt.ForeColor = Color.Red;
    txxt.ForeColor = Color.Red;
    label_down.Text = (Decimal.Round(Convert.ToDecimal(chng),2).ToString())+"%";
    pictureBox_up.Visible = false;
    label_up.Visible = false;
    pictureBox_down.Visible = true;
    label_down.Visible = true;
}
```

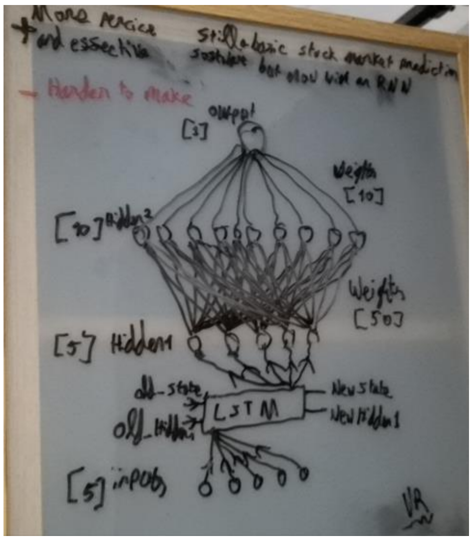
Another useful function is this label in which shows the actual value of the current sale and it stays on the same x-axis but it moves up and down on the same size as the current stock. Depending if the change is positive or negative the label will change its colour as well; this is for helping out the user and making it as comfortable and possible.



Version 1.1(testing and conclusion, final design)

Version 1.1 will include an Rnn(reccurent neural net) to predict the stock prices.





As a starting point in creating a recurrent neural net, I needed to work out exactly how many weights I needed. For a basic neural net it was practically easy as all you need to do is draw out your model and then just count how many weights are needed, depending on the sizes of hidden layers, output layers and even input layers; Whilst for the RNN neural net it was a different story, since an RNN needs to use either an LSTM or GRU, as they prevent gradients from exploding/imploding during run time.

The following websites provided with a really detailed explanation on how to perform back propagation on an LSTM unit.

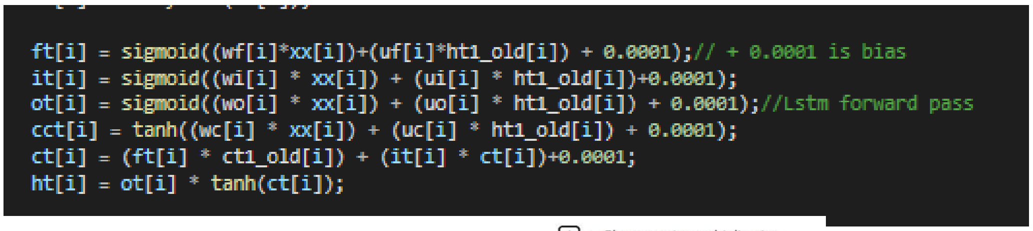
https://mc.ai/backpropagation-through-lstm-a-differential-approach/

https://towardsdatascience.com/back-to-basics-deriving-back-propagation-on-simple-rnn-lstm-feat-aidan-gomez-c7f286ba973d

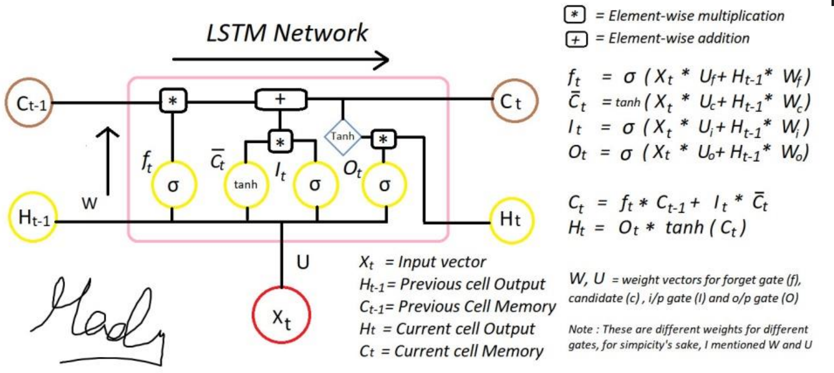
https://medium.com/@dovanhuyen2018/back-propagation-in-long-short-term-memory-lstm-a13ad8ae7a57

https://www.linkedin.com/pulse/chapter-101-deepnlp-lstm-long-short-term-memory-math-sanjeevi-mady-

For the Lstm since it was at the front of the neural net I needed to start on that first so I coded



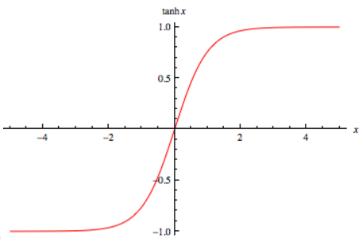
```
ft[i] = sigmoid((wf[i]*xx[i])+(uf[i]*ht1_old[i]) + 0.0001);// + 0.0001 is bias
it[i] = sigmoid((wi[i] * xx[i]) + (ui[i] * ht1_old[i])+0.0001);
ot[i] = sigmoid((wo[i] * xx[i]) + (uo[i] * ht1_old[i]) + 0.0001);//Lstm forward pass
cct[i] = tanh((wc[i] * xx[i]) + (uc[i] * ht1_old[i]) + 0.0001);
ct[i] = (ft[i] * ct1_old[i]) + (it[i] * ct[i])+0.0001;
ht[i] = ot[i] * tanh(ct[i]);
```



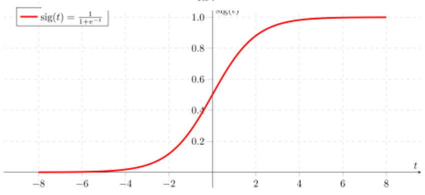
LSTM Network
$\boxed{*}$ = Element-wise multiplication
$\boxed{+}$ = Element-wise addition
C_{t-1}
C_t
H_{t-1}
H_t
W
U
X_t
f_t
$\bar{C}_t$
I_t
O_t
σ
tanh
Tanh
$f_t = \sigma(X_t * U_f + H_{t-1} * W_f)$
$\bar{C}_t = \tanh(X_t * U_c + H_{t-1} * W_c)$
$I_t = \sigma(X_t * U_i + H_{t-1} * W_i)$
$O_t = \sigma(X_t * U_o + H_{t-1} * W_o)$
$C_t = f_t * C_{t-1} + I_t * \bar{C}_t$
$H_t = O_t * \tanh(C_t)$
X_t = Input vector
H_{t-1} = Previous cell Output
C_{t-1} = Previous Cell Memory
H_t = Current cell Output
C_t = Current cell Memory
W, U = weight vectors for forget gate (f), candidate (c), i/p gate (I) and o/p gate (O)
Note : These are different weights for different gates, for simpicity's sake, I mentioned W and U

Diagram from[6]. In this line I simply coded in the feed forward function of the lstm cell. As you may see in the code above I've added in functions "sigmoid and tanh" which are known as activation functions, and they do a simple role of squashing down numbers.

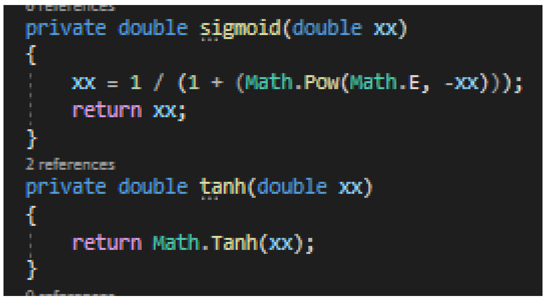
for example Tanh caps down the number between 1 and -1 whilst the sigmoid function caps it down between 1 and 0.



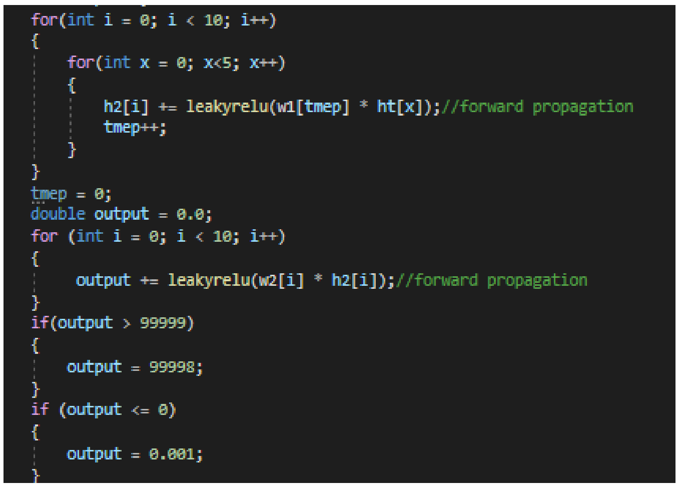
[8]



[7]



```
2 references
private double sigmoid(double xx)
{
    xx = 1 / (1 + (Math.Pow(Math.E, -xx)));
    return xx;
}
2 references
private double tanh(double xx)
{
    return Math.Tanh(xx);
}
```

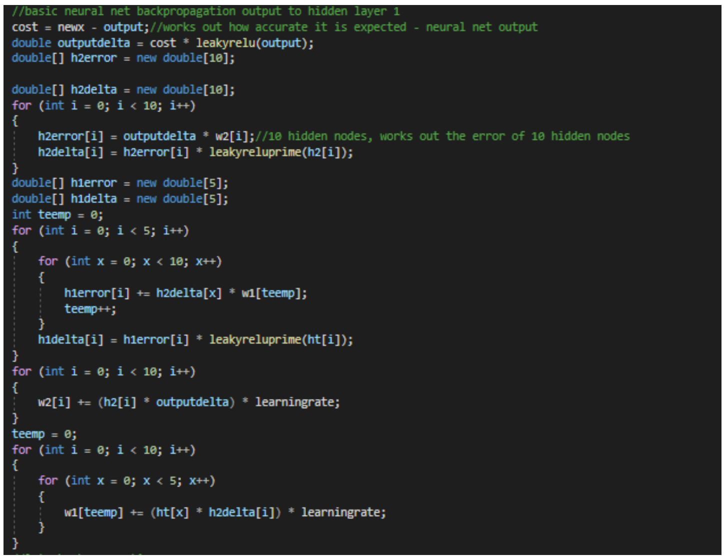


```
for(int i = 0; i < 10; i++)
{
    for(int x = 0; x<5; x++)
    {
        h2[i] += leakyrelu(w1[tmep] * ht[x]);//forward propagation
        tmep++;
    }
}
tmep = 0;
double output = 0.0;
for (int i = 0; i < 10; i++)
{
     output += leakyrelu(w2[i] * h2[i]);//forward propagation
}
if(output > 99999)
{
    output = 99998;
}
if (output <= 0)
{
    output = 0.001;
}
```

Later after making implementing the Lstm cell I was finally able to create the rest of the neural net with another hidden layer.

The reason why I had a cap that prevented the output from going over 99999 and bellow 0.001 is because in case the neural nets break it should not crash the entire program.

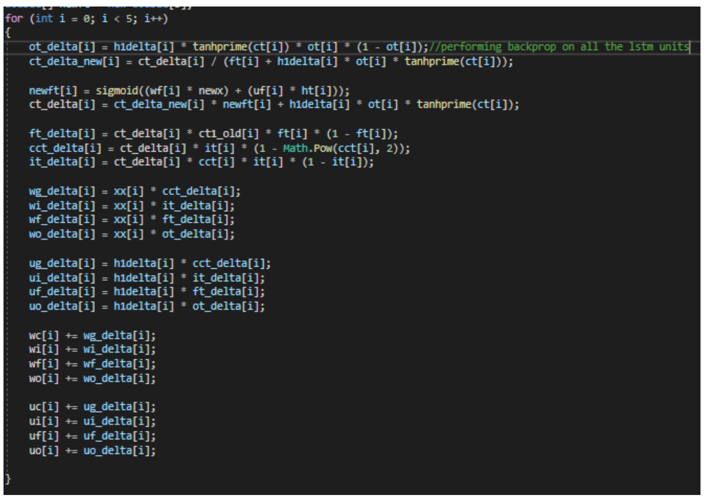
More about vanilla neural nets [9],[10]



```
//basic neural net backpropagation output to hidden layer 1
cost = newx - output;//works out how accurate it is expected - neural net output
double outputdelta = cost * leakyrelu(output);
double[] h2error = new double[10];

double[] h2delta = new double[10];
for (int i = 0; i < 10; i++)
{
    h2error[i] = outputdelta * w2[i];//10 hidden nodes, works out the error of 10 hidden nodes
    h2delta[i] = h2error[i] * leakyreluprime(h2[i]);
}
double[] h1error = new double[5];
double[] h1delta = new double[5];
int teemp = 0;
for (int i = 0; i < 5; i++)
{
    for (int x = 0; x < 10; x++)
    {
        h1error[i] += h2delta[x] * w1[teemp];
        teemp++;
    }
    h1delta[i] = h1error[i] * leakyreluprime(ht[i]);
}
for (int i = 0; i < 10; i++)
{
    w2[i] += (h2[i] * outputdelta) * learningrate;
}
teemp = 0;
for (int i = 0; i < 10; i++)
{
    for (int x = 0; x < 5; x++)
    {
        w1[teemp] += (ht[x] * h2delta[i]) * learningrate;
    }
}
```

After completing the forward propagation and working out the cost which is the future input – output, reason for why it's future input is because we are trying to get the neural net to predict the next stock price.

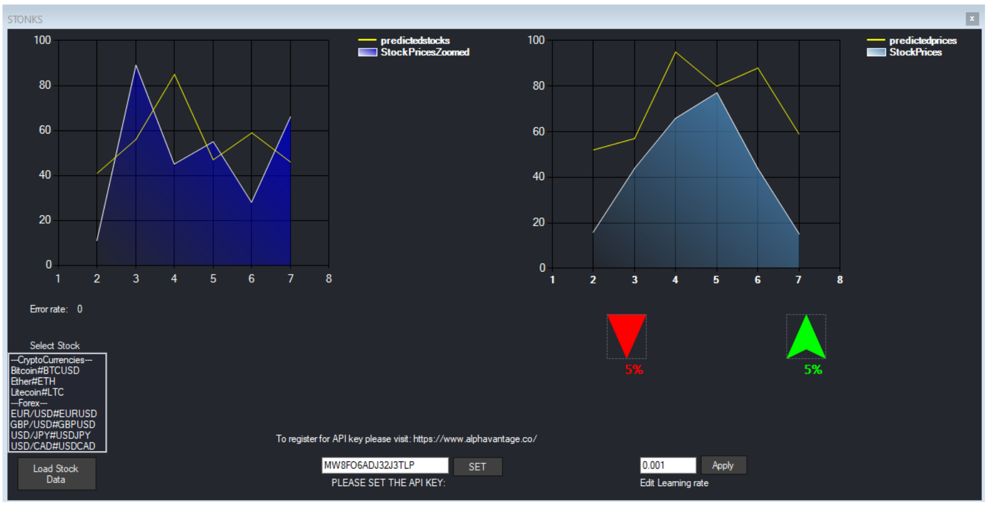
After the cost is calculated, each hidden layer error and delta is calculated and their weights are updated appropriately and also multiplied by the learning rate in which the user gets to set in the GUI.



```
for (int i = 0; i < 5; i++)
{
    ot_delta[i] = h1delta[i] * tanhprime(ct[i]) * ot[i] * (1 - ot[i]);//performing backprop on all the lstm units
    ct_delta_new[i] = ct_delta[i] / (ft[i] + h1delta[i] * ot[i] * tanhprime(ct[i]));

    newft[i] = sigmoid((wf[i] * newx) + (uf[i] * ht[i]));
    ct_delta[i] = ct_delta_new[i] * newft[i] + h1delta[i] * ot[i] * tanhprime(ct[i]);

    ft_delta[i] = ct_delta[i] * ct1_old[i] * ft[i] * (1 - ft[i]);
    cct_delta[i] = ct_delta[i] * it[i] * (1 - Math.Pow(cct[i], 2));
    it_delta[i] = ct_delta[i] * cct[i] * it[i] * (1 - it[i]);

    wg_delta[i] = xx[i] * cct_delta[i];
    wi_delta[i] = xx[i] * it_delta[i];
    wf_delta[i] = xx[i] * ft_delta[i];
    wo_delta[i] = xx[i] * ot_delta[i];

    ug_delta[i] = h1delta[i] * cct_delta[i];
    ui_delta[i] = h1delta[i] * it_delta[i];
    uf_delta[i] = h1delta[i] * ft_delta[i];
    uo_delta[i] = h1delta[i] * ot_delta[i];

    wc[i] += wg_delta[i];
    wi[i] += wi_delta[i];
    wf[i] += wf_delta[i];
    wo[i] += wo_delta[i];

    uc[i] += ug_delta[i];
    ui[i] += ui_delta[i];
    uf[i] += uf_delta[i];
    uo[i] += uo_delta[i];

}
```

Lastly the rest of back prop algorithm is added, in which it back propagates through the Lstm cell, it works in a similar way as the vanilla backdrop method just slightly more complex. https://www.linkedin.com/pulse/chapter-101-deepnlp-lstm-long-short-term-memory-math-sanjeevi-mady-

As the process of creating a neural net was done all was left was to do a lot of testing and calibrating, to how weights were generated and also tweaking the biases to achieve the best results.



Error rate: 0

--CryptoCurrencies--
Bitcoin#BTCUSD
Ether#ETH
Litecoin#LTC
--Forex--
EUR/USD#EURUSD
GBP/USD#GBPUSD
USD/JPY#USDJPY
USD/CAD#USDCAD

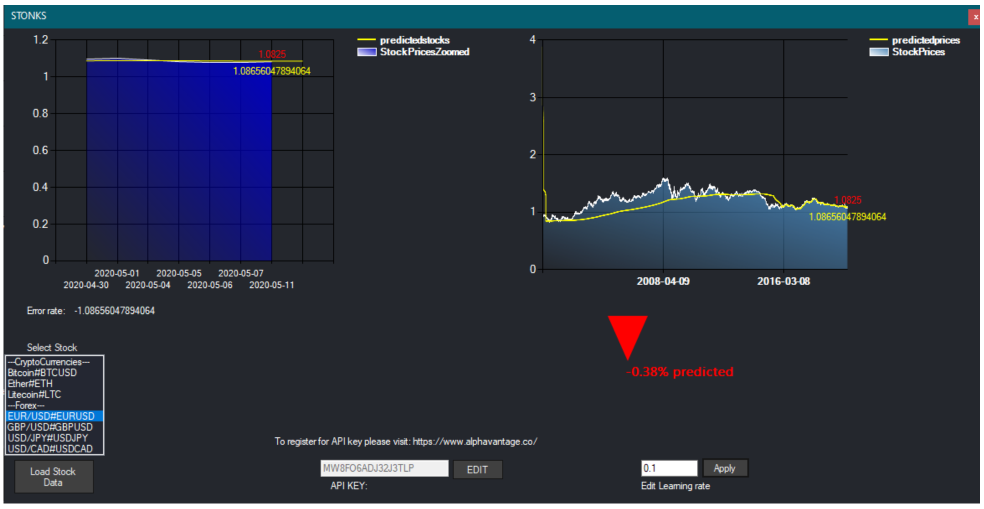
To register for API key please visit: https://www.alphavantage.co/

MW8FO6ADJ32J3TLP
PLEASE SET THE API KEY:

0.001
Edit Learning rate

The final design consisted in a new feature being added in which was a learning rate adjuster, since the weights will be different every time a user deletes it, the learning rate will have to be different every time.

Tip: the smaller the learning rate is the more stable the neural net will be but training will also be super slow (negative numbers will break the neural net).



Error rate: -1.08656047894064

--CryptoCurrencies--
Bitcoin#BTCUSD
Ether#ETH
Litecoin#LTC
--Forex--
EUR/USD#EURUSD
GBP/USD#GBPUSD
USD/JPY#USDJPY
USD/CAD#USDCAD

To register for API key please visit: https://www.alphavantage.co/

MW8FO6ADJ32J3TLP
API KEY:

0.1
Edit Learning rate

Overall I am satisfied by how my program is, it managed to predict more than one of the stocks correctly, and it works perfect

References

[1] https://www.experfy.com/blog/the-future-of-algorithmic-trading

[2] https://www.alphavantage.co/premium/

[3] https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e

[4] https://stackoverflow.com/questions/44273249/in-keras-what-exactly-am-i-configuring-when-i-create-a-stateful-lstm-layer-wi

[5] https://towardsdatascience.com/back-to-basics-deriving-back-propagation-on-simple-rnn-lstm-feat-aidan-gomez-c7f286ba973d

[6] https://www.linkedin.com/pulse/chapter-101-deepnlp-lstm-long-short-term-memory-math-sanjeevi-mady-

[7] https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

[8] https://www.i2tutorials.com/explain-hyperbolic-tangent-or-tanh-relu-rectified-linear-unit/

[9] https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-networks-process-examples-code-minus-math/

[10] https://www.youtube.com/watch?v=Ilg3gGewQ5U