

Stock Prediction using the Back-Propagation Neural Network.

The implementation of the learning algorithm called back propagation neural network that adjust the weight for the input value as per the training set and predict the value using the 3 layer hidden network.

Datasets:

For the experimental purpose the data- set is taken manually from the site:

<https://www.nasdaq.com/symbol/goog/historical>

The data for the mini-project is taken from the site:

http://www.kibot.com/api/historical_data_api_sdk.aspx

Machine Learning Algorithms used:

Back-Propagation Neural Network:

1. Steps of the algorithm

After choosing the weights of the network randomly, the backpropagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps:

- i) Feed-forward computation
- ii) Backpropagation to the output layer
- iii) Backpropagation to the hidden layer
- iv) Weight updates

The sigmoid function used in the project is $\tanh(x)$ which is better than standard $1/(1+e^{-x})$

The better description is given in the site: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

Calculations involved in the projects are:

1. Rolling average, rolling minimum and rolling maximum using the basic sliding window algorithm.

2. Scaling Time Series Data:

Built a meaningful training set and figure out how to scale the closing price. To do so, a simple function was built which normalizes the closing stock price between -1.0 and 1.0:

```
def normalize(price, minimum, maximum):  
  
    return ((2*price*(maximum + minimum)) / (maximum - minimum))
```

Along with a roughly inverse method for de-normalizing the data:

```
def denormalize(normalized, minimum, maximum):

    return (((normalized*(maximum - minimum)) / 2) + (maximum +
minimum)) / 2
```

With this, we can properly represent our training data as an array of tuples in the form of [[rolling average, rolling minimum, rolling maximum], [normalized price]].

The **error** in the stock price was found to within $\pm 2\%$ in all the data.

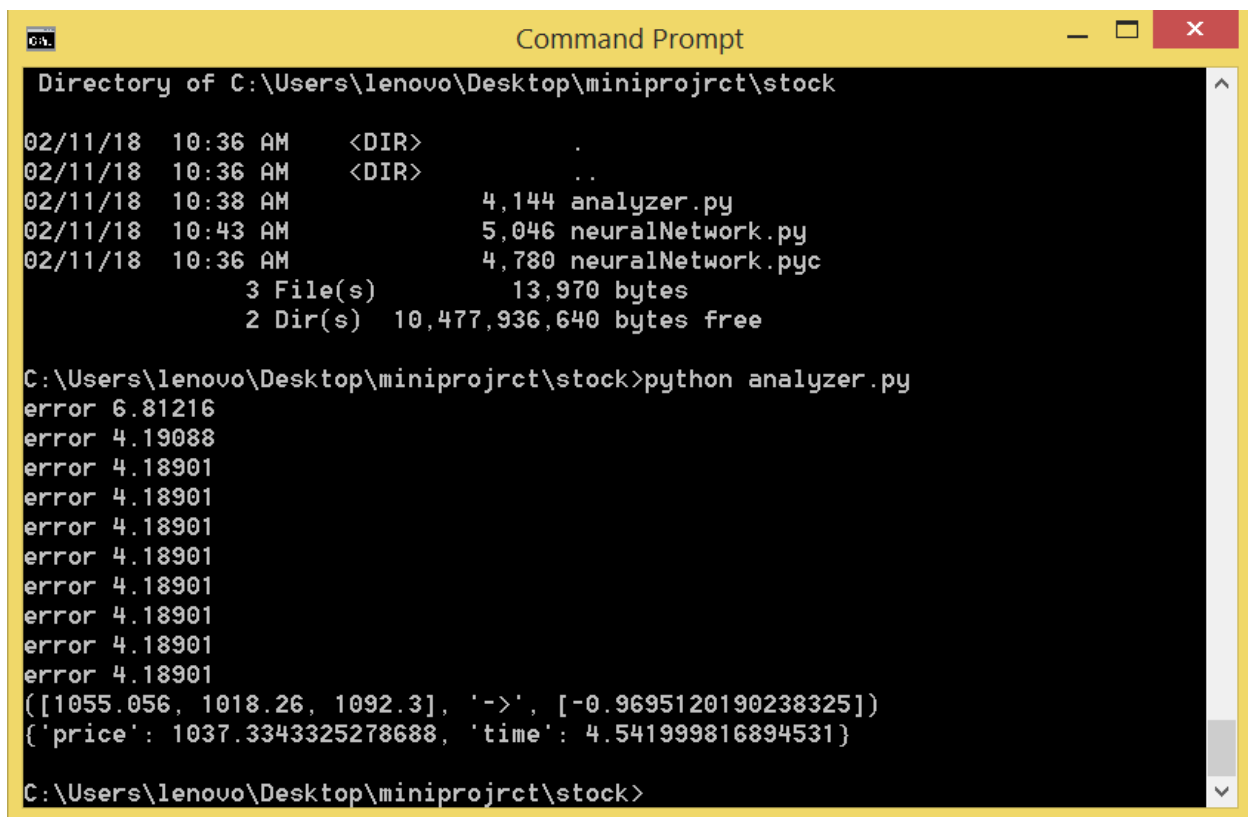
Working code: Total algorithm is hand-coded from the scratch. No use of any inbuilt library.

Input: The total datasets have been adopted from the Kibot's API.

The sliding window is used to create the input set of data. A window of 5 days is created and the stock price of the 6th day is calculated using the algorithm.

The input is instance [avg, min, max] for the recent days.

Output: Stock Price for the new day in terms of normalized price which will be de-normalized again.



```

C:\Users\lenovo\Desktop\miniprojct\stock
02/11/18  10:36 AM    <DIR>          .
02/11/18  10:36 AM    <DIR>          ..
02/11/18  10:38 AM                4,144 analyzer.py
02/11/18  10:43 AM                5,046 neuralNetwork.py
02/11/18  10:36 AM                4,780 neuralNetwork.pyc
                3 File(s)            13,970 bytes
                2 Dir(s)  10,477,936,640 bytes free

C:\Users\lenovo\Desktop\miniprojct\stock>python analyzer.py
error 6.81216
error 4.19088
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
([1055.056, 1018.26, 1092.3], '->', [-0.9695120190238325])
{'price': 1037.3343325278688, 'time': 4.541999816894531}

C:\Users\lenovo\Desktop\miniprojct\stock>
```

Fig: Output from the calculate.py

```
Command Prompt

| April 28th | 1031.40 | 1001.52 | 1.47 |
| April 29th | 1022.93 | 1037.78 | -0.72 |
+-----+-----+-----+-----+

C:\Users\lenovo\Desktop\experiments>python stocks-experiments.py
error 1.09757
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
([1145.0120000000002, 1129.79, 1169.97], '->', [-0.9170984455956367])
([1151.464, 1129.79, 1169.97], '->', [-0.9170984455956367])
([1159.58, 1137.51, 1170.37], '->', [-0.9170984455956367])
([1167.2459999999999, 1155.81, 1175.84], '->', [-0.9170984455956367])
([1171.2, 1164.24, 1175.84], '->', [-0.9170984455956367])
([1169.944, 1163.69, 1175.84], '->', [-0.9170984455956367])
([1171.084, 1163.69, 1175.84], '->', [-0.9170984455956367])
([1170.55, 1163.69, 1175.84], '->', [-0.9170984455956367])
([1157.762, 1111.9, 1175.58], '->', [-0.9170984455956367])
([1133.806, 1055.8, 1169.94], '->', [-0.9170984455956367])
([1117.188, 1055.8, 1169.94], '->', [-0.9170984455956367])
([1092.916, 1048.58, 1167.7], '->', [-0.9170984455956367])
([1059.6799999999998, 1001.52, 1111.9], '->', [-0.9170984455956367])
([1044.856, 1001.52, 1080.6], '->', [-0.9170984455956367])
+-----+-----+-----+-----+
| Date | Predicted | Actual | Error Percentage |
+-----+-----+-----+-----+
| April 12th | 1140.67 | 1169.97 | -1.27 |
| April 13th | 1140.67 | 1164.24 | -1.02 |
| April 14th | 1146.41 | 1170.37 | -1.03 |
| April 15th | 1161.23 | 1175.84 | -0.63 |
| April 18th | 1167.38 | 1175.58 | -0.35 |
| April 19th | 1166.98 | 1163.69 | 0.14 |
| April 20th | 1166.98 | 1169.94 | -0.13 |
| April 21th | 1166.98 | 1167.7 | -0.03 |
| April 22th | 1129.14 | 1111.9 | 0.77 |
| April 25th | 1086.70 | 1055.8 | 1.44 |
| April 26th | 1086.70 | 1080.6 | 0.28 |
| April 27th | 1080.83 | 1048.58 | 1.51 |
| April 28th | 1031.40 | 1001.52 | 1.47 |
| April 29th | 1022.93 | 1037.78 | -0.72 |
+-----+-----+-----+-----+

C:\Users\lenovo\Desktop\experiments>
```

Fig: out from the stocks-experiments.py

Running the project:

Run python stocks-experiments.py to view the prediction and error by the algorithm

Run python calculate.py to predict the new value using the dataset from API