Final Report for Neural Network Project

Topic: Predict Daily Performance of Nasdaq Composite Index by Relative Stocks

Member name: Ke Gao

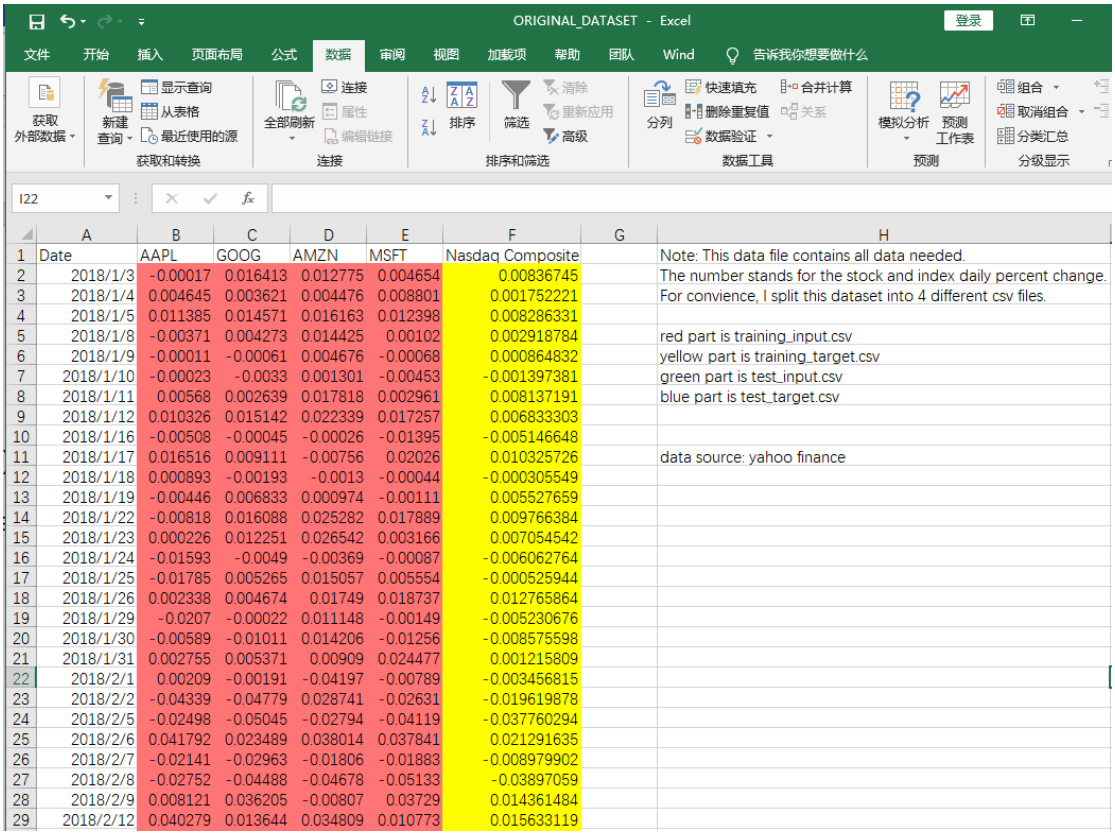
Date: 2018/12/23

1. Our goal

Our goal is to predict Nasdaq Composite Index (IXIC) daily performance by four different stocks daily performance. These four stocks are AAPL (Apple.Inc), GOOG (Google), AMZN (Amazon) and MSFT (Microsoft).

2. Our dataset:

Our data source is yahoo finance. All data needed is in ORIGINAL_DATASET.xls. The number stands for stocks and index daily percent change. For convenience, I split this dataset into 4 different csv files. Red part is training_input.csv, yellow part is training_target.csv, green part is test_input.csv and blue part is test_target.csv



	A	B	C	D	E	F	G	H
1	Date	AAPL	GOOG	AMZN	MSFT	Nasdaq Composite		Note: This data file contains all data needed.
2	2018/1/3	-0.00017	0.016413	0.012775	0.004654	0.00836745		The number stands for the stock and index daily percent change.
3	2018/1/4	0.004645	0.003621	0.004476	0.008801	0.001752221		For convience, I split this dataset into 4 different csv files.
4	2018/1/5	0.011385	0.014571	0.016163	0.012398	0.008286331		
5	2018/1/8	-0.00371	0.004273	0.014425	0.00102	0.002918784		red part is training_input.csv
6	2018/1/9	-0.00011	-0.00061	0.004676	-0.00068	0.000864832		yellow part is training_target.csv
7	2018/1/10	-0.00023	-0.0033	0.001301	-0.00453	-0.001397381		green part is test_input.csv
8	2018/1/11	0.00568	0.002639	0.017818	0.002961	0.008137191		blue part is test_target.csv
9	2018/1/12	0.010326	0.015142	0.022339	0.017257	0.006833303		
10	2018/1/16	-0.00508	-0.00045	-0.00026	-0.01395	-0.005146648		
11	2018/1/17	0.016516	0.009111	-0.00756	0.02026	0.010325726		data source: yahoo finance
12	2018/1/18	0.000893	-0.00193	-0.0013	-0.00044	-0.000305549		
13	2018/1/19	-0.00446	0.006833	0.000974	-0.00111	0.005527659		
14	2018/1/22	-0.00818	0.016088	0.025282	0.017889	0.009766384		
15	2018/1/23	0.000226	0.012251	0.026542	0.003166	0.007054542		
16	2018/1/24	-0.01593	-0.0049	-0.00369	-0.00087	-0.006062764		
17	2018/1/25	-0.01785	0.005265	0.015057	0.005554	-0.000525944		
18	2018/1/26	0.002338	0.004674	0.01749	0.018737	0.012765864		
19	2018/1/29	-0.0207	-0.00022	0.011148	-0.00149	-0.005230676		
20	2018/1/30	-0.00589	-0.01011	0.014206	-0.01256	-0.008575598		
21	2018/1/31	0.002755	0.005371	0.00909	0.024477	0.001215809		
22	2018/2/1	0.00209	-0.00191	-0.04197	-0.00789	-0.003456815		
23	2018/2/2	-0.04339	-0.04779	0.028741	-0.02631	-0.019619878		
24	2018/2/5	-0.02498	-0.05045	-0.02794	-0.04119	-0.037760294		
25	2018/2/6	0.041792	0.023489	0.038014	0.037841	0.021291635		
26	2018/2/7	-0.02141	-0.02963	-0.01806	-0.01883	-0.008979902		
27	2018/2/8	-0.02752	-0.04488	-0.04678	-0.05133	-0.03897059		
28	2018/2/9	0.008121	0.036205	-0.00807	0.03729	0.014361484		
29	2018/2/12	0.040279	0.013644	0.034809	0.010773	0.015633119		

	A	B	C	D	E	F	G	H
202	2018/10/18	-0.02337	-0.02485	-0.03331	-0.01996	-0.020615758		
203	2018/10/19	0.01523	0.007804	-0.00378	0.001475	-0.004824272		
204	2018/10/22	0.00611	0.004287	0.014325	0.008927	0.002631228		
205	2018/10/23	0.009427	0.002297	-0.01151	-0.01396	-0.004162724		
206	2018/10/24	-0.0343	-0.048	-0.05908	-0.05347	-0.044253898		
207	2018/10/25	0.021898	0.042695	0.070887	0.058444	0.029534065		
208	2018/10/26	-0.01592	-0.022	-0.0782	-0.01237	-0.020650842		
209	2018/10/29	-0.01877	-0.04796	-0.06326	-0.02908	-0.016313171		
210	2018/10/30	0.004994	0.015812	-0.0055	-0.00116	0.015795075		
211	2018/10/31	0.026067	0.039143	0.044164	0.029692	0.020142007		
212	2018/11/1	0.015352	-0.00629	0.042253	-0.00833	0.017542008		
213	2018/11/2	-0.06633	-0.01141	0	0.002266	-0.010367124		
214	2018/11/5	-0.02839	-0.01673	-0.02265	0.012717	-0.003824952		
215	2018/11/6	0.010814	0.015114	0.009221	0.001953	0.006428002		
216	2018/11/7	0.030328	0.035593	0.06859	0.039361	0.026408771		
217	2018/11/8	-0.00695	-0.01005	-0.00033	-0.00188	-0.005266336		
218	2018/11/9	-0.01928	-0.01501	-0.02421	-0.01951	-0.016462881		
219	2018/11/12	-0.05037	-0.02581	-0.04414	-0.02464	-0.027815927		
220	2018/11/13	-0.00999	-0.00248	-0.00347	0.000655	0		
221	2018/11/14	-0.02825	0.007345	-0.01972	-0.01842	-0.008954471		
222	2018/11/15	0.024679	0.020169	0.012777	0.022006	0.01718511		
223	2018/11/16	0.011076	-0.00302	-0.01607	0.009415	-0.00153735		
224	2018/11/19	-0.03963	-0.03909	-0.05091	-0.03389	-0.030269601		

I select data from 2018/01/03 to 2018/10/31 as training set (210 days), and data from 2018/11/01 to 2018/12/21 as test set (35 days).

Explanation of data				
color	Red	Yellow	Green	Blue
filename	training_input.csv	training_target.csv	test_input.csv	test_target.csv
Counts	840	210	140	35

3.Our neural network

We use 3 different topologies of neural networks.

Type 1: (4,4,1)
(4 neurons in input layer, 4 neurons in hidden layer, 1 neuron in output layer)

Type 2: (4,4,3,1)
(4 neurons in input layer, 4 neurons in hidden layer I, 3 neurons in hidden layer II, 1 neuron in output layer)

Type 3: (4,4,3,2,1)
(4 neurons in input layer, 4 neurons in hidden layer I, 3 neurons in hidden layer II, 2 neurons in hidden layer III, 1 neuron in output layer)

Our activation function is bipolar sigmoid function:

$$f(x) = \frac{2}{1+e^{-x}} - 1$$

$$f'(x) = 0.5 * \big(1 + f(x)\big) * (1 - f(x))$$

Our error function is

$$Error = 0.5 * (output - target)^2$$

4.Our programming

Our program contains class Neuron, Layer, NeuralNetwork and Matrix.

a).vector<int> topology stands for the topology of neural network. If topology is (4,4,3,2,1), which means neural network have 4 neurons in input layer, 4 neurons in hidden layer I, 3 neurons in hidden layer II, 2 neurons in hidden layer III, 1 neuron in output layer.

```
//set topology of neural network
vector<int> topology;
topology.push_back(4);        //size of input layer = 4
topology.push_back(4);        //size of hidden layer I = 4
topology.push_back(3);        //size of hidden layer II = 3
topology.push_back(2);        //size of hidden layer III = 2
topology.push_back(1);        //size of output layer = 1
```

b). vector_of_inputs and vector_of_targets stand for the input and target data. They read data from input and target.csv which mentioned above.

```
vector<vector<double>> vector_of_inputs;
vector<vector<double>> vector_of_targets;
//read training file from csv
vector_of_inputs = ReadCSV("training_input.csv");
vector_of_targets = ReadCSV("training_target.csv");
```

c). default parameters are initialized here.

```
//set parameters for neural network
int cycle = 0;
int max_cycle = 100;
double error = 1;              //error = 0.5*(output - target)^2
double derived_error = 1;  //derived error = output - target
double average_error = 1;  //average of error
double tolerance_error = 0.0000001;
double learning_rate = 1;   //learning rate
```

d). Training part.

If cycle > max_cycle or average_error < tolerance_error, the while loop breaks.

```
//training
while (cycle< max_cycle && average_error > tolerance_error) {
    average_error = 0;
    for (int i = 0; i < vector_of_inputs.size(); i++) {
        nn.SetInputValue(vector_of_inputs[i]);
        nn.SetTargetValue(vector_of_targets[i]);
        nn.ForwardPropagation();
        nn.BackPropagation();
        error = nn.GetTotalError();
        average_error += error / vector_of_inputs.size();
    }
    cycle += 1;
    cout << "cycle = " << cycle << "  average error = " << average_error << endl;
}
```

e). Test part.

First, we read test_input.csv and test_target.csv.

Then we only do forward propagation to calculate predict_value and derived error = (predict_value -target_value)

In the end, we output result to txt file.

```
    //testing
    vector_of_inputs = ReadCSV("test_input.csv");
    vector_of_targets = ReadCSV("test_target.csv");
    cout << "Predict=    \tTarget=    \tPredict - Target =    "<< endl;
    output_file << "Predict=    \tTarget=    \tPredict - Target =    " << endl;
    for (int i = 0; i < vector_of_inputs.size(); i++) {
        nn.SetInputValue(vector_of_inputs[i]);
        nn.SetTargetValue(vector_of_targets[i]);
        nn.ForwardPropagation();
        cout.width(8);
        cout.setf(ios::showpoint);
        cout << setprecision(6);
        cout << nn.GetPredictValue(0) << "    \t" << nn.GetTargetValue(0)<< "    \t" << nn.GetTotalDerivedError()<< endl;
        output_file << nn.GetPredictValue(0) << "    \t" << nn.GetTargetValue(0) << "    \t" << nn.GetTotalDerivedError() << endl;
    }
    output_file.close();
    return 0;
}
```

f). ForwardPropagation()

We take bias into account in forward propagation.

```
71  void NeuralNetwork::ForwardPropagation() {
72      for (int i = 0; i < GetSizeNeuralNetwork()-1; i++) {
73          Matrix a(GetNeuronValueMatrix(i));                  //if i=0, which means input layer
74          if (i != 0) {
75              a=GetNeuronActivatedValueMatrixPlusBias(i);         //if i!=0, which means hidden layer
76          }
77          Matrix b=GetWeightMatrix(i);
78          Matrix c(a*b);
79          vector<double> result(c.MatrixToVector());          //make result matrix a vector
80          for (int j = 0; j < result.size(); j++) {
81              SetValueNeuralNetwork(i + 1, j, result[j]);         //push value to next layer neurons (+bias)
82          }
83      }
84      SetErrors();      //renew errors vector
85  };
```

g). SetErrors()

We calculate errors given the result value of forward propagation.

```
void NeuralNetwork::SetErrors() {
    if (target_value.size() == 0){
        cout << "No target value for neural network!" << endl;
    }
    if (target_value.size() != layers[layers.size()-1].GetLayerSize()) {
        cout << "Target size do not match with output layer size!" << endl;
    }
    total_error = 0.0;
    total_derived_error = 0.0;
    int output_layer_index = layers.size() - 1;
    vector<Neuron> output_neurons = layers[output_layer_index].GetNeuronLayer();
    vector<double> result_errors;
    vector<double> result_derived_errors;
    for (int i = 0; i < target_value.size(); i++) {
        double temp_error = 0.5*(output_neurons[i].GetActivatedValue() - target_value[i])*(output_neurons[i].GetActivatedValue() - target_value[i]);
        double temp_derived_error = (output_neurons[i].GetActivatedValue() - target_value[i]);
        result_errors.push_back(temp_error);
        result_derived_errors.push_back(temp_derived_error);
        total_error += temp_error;
        total_derived_error += temp_derived_error;
    }
    errors = result_errors;
    derived_errors = result_derived_errors;

    total_error = total_error / topology[topology.size() - 1];
    total_derived_error = total_derived_error / topology[topology.size() - 1];
    historical_total_errors.push_back(total_error);
};
```

h). BackPropagation()

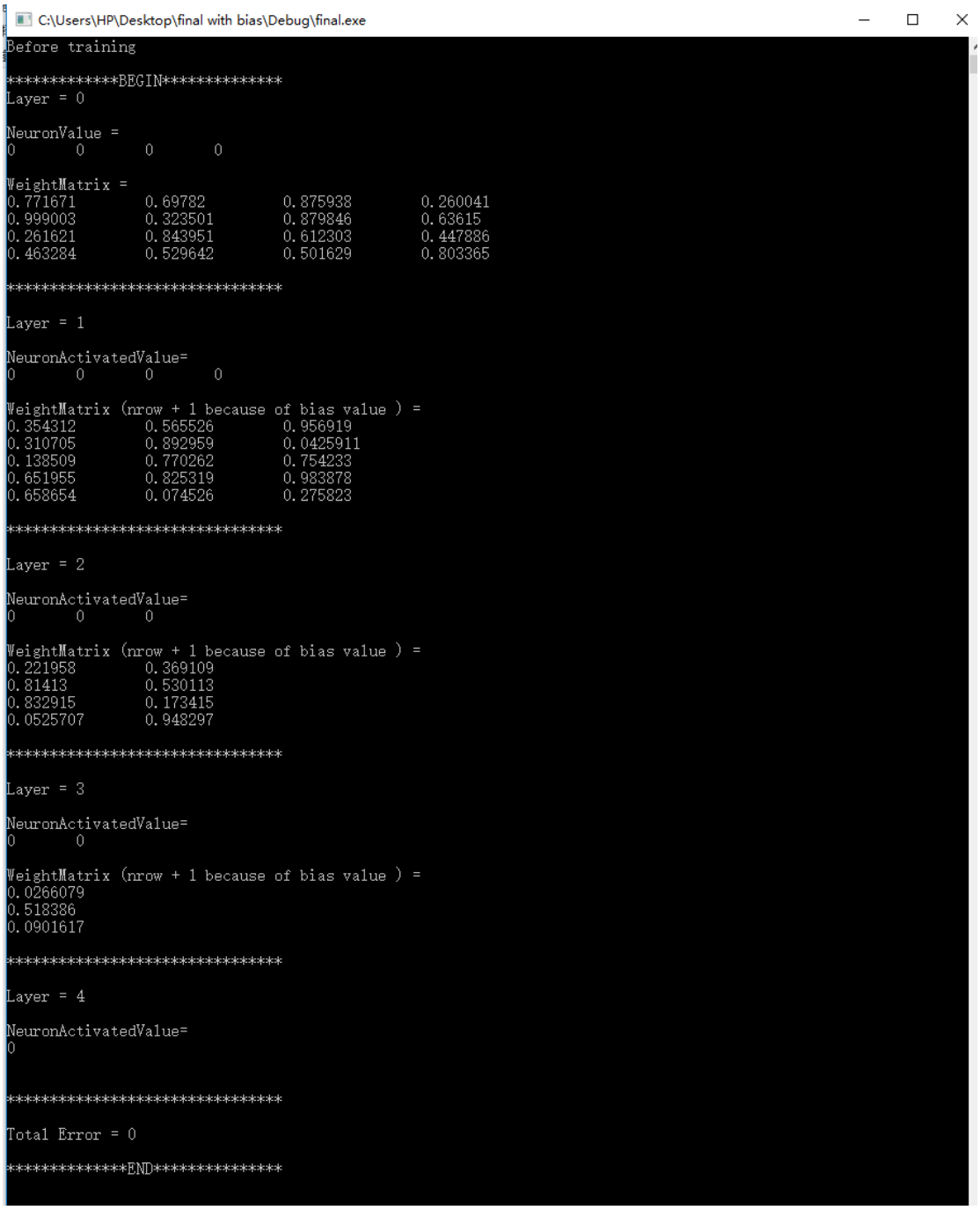
We take bias into account in back propagation.

```
87  void NeuralNetwork::BackPropagation() {
88      int output_layer_index = layers.size() - 1;
89      int last_hidden_layer_index = output_layer_index - 1;
90
91      //part 1: handle output layer to hidden layer.
92      Matrix derived_output_to_hidden(layers[output_layer_index].MatrixifyDerivedValues());    //create derived values matrix
93      Matrix error_output_to_hidden(1, errors.size(), derived_errors);
94      Matrix gradient_output_to_hidden(derived_output_to_hidden.PointwiseProduct(error_output_to_hidden));     // gradient matrix
95      //calculate gradient matrix output to hidden
96      Matrix delta_output_to_hidden((layers[last_hidden_layer_index].MatrixifyActivatedValuesPlusBias().TransposeMatrix())*gradient_output_to_hidden);
97      //refresh weight matrices.
98      weight_matrices[last_hidden_layer_index] = weight_matrices[last_hidden_layer_index] - delta_output_to_hidden*learning_rate;
99
100
101     //hidden layer a: left side              hidden layer b: right side
102     //gradient value of layer b
103     Matrix gradient_hidden(gradient_output_to_hidden);
104
105     //part 2: calcuate gradient matrix from hidden layer to input layer
106     for (int i = last_hidden_layer_index; i > 0; i--) {
107         //derived hidden value of layer b
108         Matrix derived_hidden(layers[i].MatrixifyDerivedValues());
109         //weight matrix from a to b
110         Matrix weight_matrix_b;
111         weight_matrix_b = weight_matrices[i].EraseLastRow();
112         //calculate new gradient value of layer b
113         gradient_hidden = derived_hidden.PointwiseProduct(gradient_hidden*(weight_matrix_b.TransposeMatrix()));
114         Matrix neuron_value_a(layers[i - 1].MatrixifyActivatedValuesPlusBias());
115         if (i == 1) {  //which means the layer a is input layer
116             neuron_value_a = layers[0].MatrixifyValues();
117         }
118         Matrix delta_hidden(neuron_value_a.TransposeMatrix()*gradient_hidden);
119         weight_matrices[i-1] = weight_matrices[i-1] - delta_hidden*learning_rate;
120     }
121 };
```

5.Our Output:

We take Type 3: (4,4,3,2,1) neuron network as a demo to show how does the program works. (This result is different from the final result I submitted, because initial weight matrices are randomly generated, my final result is in folder *Prediction&WeightMatrices Results*)

PS: Weight matrices has one more row due to bias value.

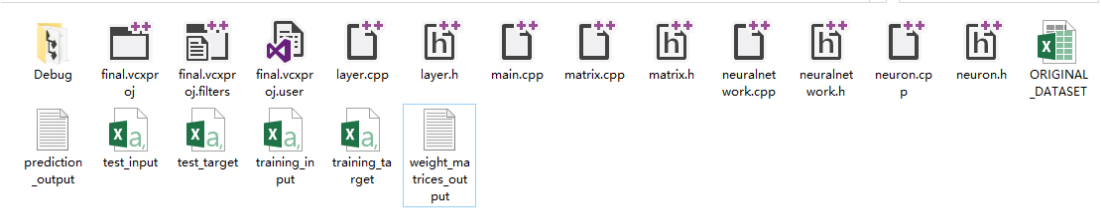


C:\Users\HP\Desktop\final with bias\Debug\final.exe

```
After trainining

**************BEGIN***************
Layer = 0

NeuronValue =
0.0260666       0.0391427       0.0441643       0.0296924

WeightMatrix =
0.792998        0.714343        0.896998        0.287421
1.02485         0.343508        0.905356        0.669327
0.281717        0.859722        0.632145        0.473843
0.498622        0.55674         0.536509        0.848529

***********************************

Layer = 1

NeuronActivatedValue=
0.0439881       0.0432561       0.0512891       0.0398854

WeightMatrix (nrow + 1 because of bias value ) =
0.373355        0.612044        0.988833
0.32835         0.935865        0.0718626
0.159847        0.822235        0.789772
0.669706        0.868823        1.01383
0.630842        0.0310458       0.243647

***********************************

Layer = 2

NeuronActivatedValue=
0.334768        0.087414        0.183496

WeightMatrix (nrow + 1 because of bias value ) =
0.249573        0.386986
0.898509        0.647973
0.905498        0.267121
0.0256584       0.84321

***********************************

Layer = 3

NeuronActivatedValue=
0.175128        0.492389

WeightMatrix (nrow + 1 because of bias value ) =
0.531258
0.557531
-0.313456

***********************************

Layer = 4

NeuronActivatedValue=
0.0270454


***********************************

Total Error = 2.38286e-05

**************END***************



Predict=        Target=         Predict - Target =
0.00870872      0.0175420       -0.00883329
-0.0119144      -0.0103671      -0.00154728
-0.00765358     -0.00382495     -0.00382863
```

```
**********

Total Error = 2.38286e-05

**********END**********


Predict=        Target=         Predict - Target =
0.00870872      0.0175420       -0.00883329
-0.0119144      -0.0103671      -0.00154728
-0.00765358     -0.00382495     -0.00382863
0.00928079      0.00642800      0.00285279
0.0322549       0.0264088       0.00584610
-0.00147189     -0.00526634     0.00379444
-0.0116476      -0.0164629      0.00481524
-0.0237878      -0.0278159      0.00402816
-0.000425519            0.000000        -0.000425519
-0.00756597     -0.00895447     0.00138850
0.0168966       0.0171851       -0.000288527
0.00295761      -0.00153735     0.00449496
-0.0274256      -0.0302696      0.00284404
-0.0119187      -0.0170250      0.00510630
0.00920047      0.00918104      1.94304e-05
-0.00671224     -0.00477178     -0.00194047
0.0235665       0.0205895       0.00297696
0.00221823      0.000120039     0.00209819
0.0330875       0.0294929       0.00359453
-0.000522220            -0.00253851     0.00201629
0.00520117      0.00790036      -0.00269920
0.0201717       0.0151380       0.00503365
-0.0315174      -0.0380406      0.00652326
0.00784895      0.00416706      0.00368189
-0.0243287      -0.0304677      0.00613900
0.00994567      0.00735660      0.00258907
0.00567267      0.00161100      0.00406167
0.00794866      0.00945415      -0.00150549
0.00411747      -0.00394178     0.00805925
-0.0193735      -0.0225831      0.00320956
-0.0168448      -0.0227084      0.00586362
0.0121495       0.00446867      0.00768080
-0.0106246      -0.0216807      0.0110561
-0.0124503      -0.0163361      0.00388585
-0.0258764      -0.0299323      0.00405586
```

Our program will generate two txt files, prediction_output.txt, which contains prediction and error calculation of testset.



prediction_output - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Predict=	Target=	Predict - Target =
0.00870872	0.017542	-0.00883329
-0.0119144	-0.0103671	-0.00154728
-0.00765358	-0.00382495	-0.00382863
0.00928079	0.006428	0.00285279
0.0322549	0.0264088	0.0058461
-0.00147189	-0.00526634	0.00379444
-0.0116476	-0.0164629	0.00481524
-0.0237878	-0.0278159	0.00402816
-0.000425519	0	-0.000425519
-0.00756597	-0.00895447	0.0013885
0.0168966	0.0171851	-0.000288527
0.00295761	-0.00153735	0.00449496
-0.0274256	-0.0302696	0.00284404
-0.0119187	-0.017025	0.0051063
0.00920047	0.00918104	1.94304e-05
-0.00671224	-0.00477178	-0.00194047
0.0235665	0.0205895	0.00297696
0.00221823	0.000120039	0.00209819
0.0330875	0.0294929	0.00359453
-0.00052222	-0.00253851	0.00201629
0.00520117	0.00790036	-0.0026992
0.0201717	0.015138	0.00503365
-0.0315174	-0.0380406	0.00652326
0.00784895	0.00416706	0.00368189
-0.0243287	-0.0304677	0.006139
0.00994567	0.0073566	0.00258907
0.00567267	0.001611	0.00406167
0.00794866	0.00945415	-0.00150549
0.00411747	-0.00394178	0.00805925
-0.0193735	-0.0225831	0.00320956

And Weight_matrices_output.txt contains the topology of neural network and weight matrices.

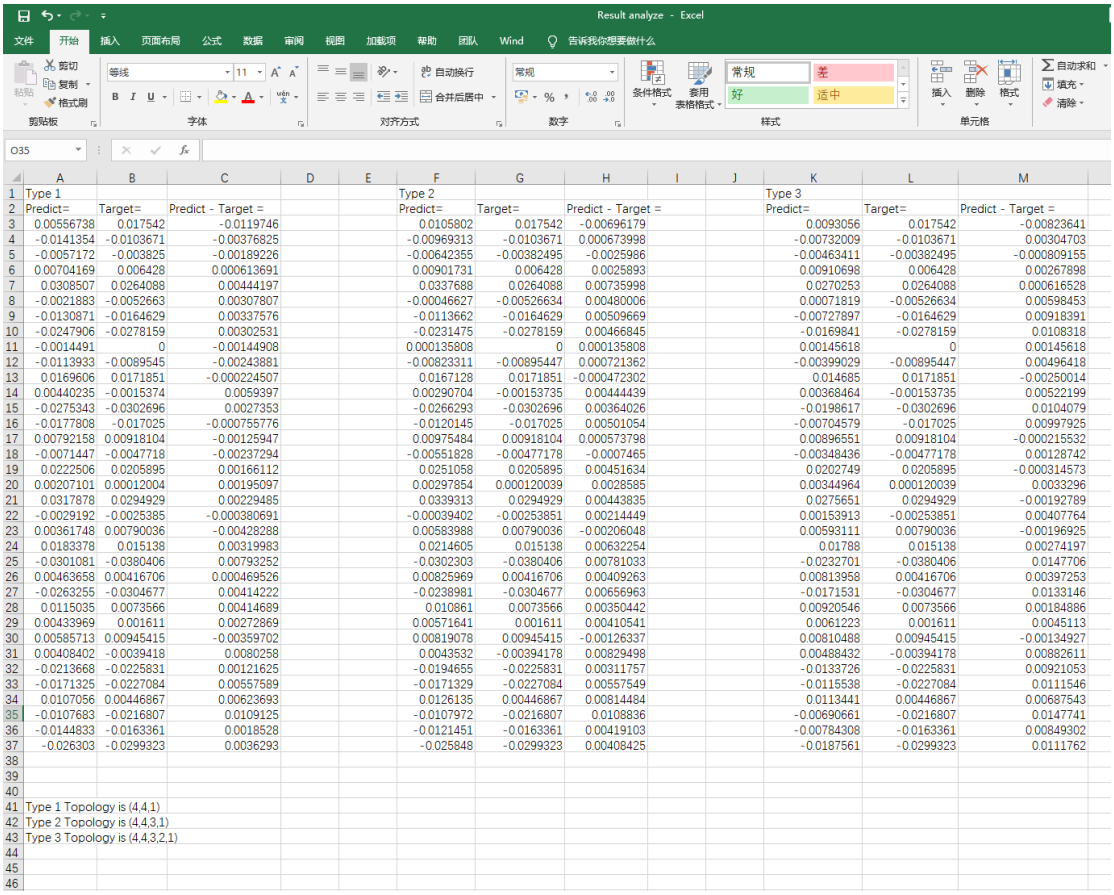
weight_matrices_output - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
**********BEGIN**********
Layer = 0

NeuronValue =
0.0260666       0.0391427       0.0441643       0.0296924

WeightMatrix =
0.792998  0.714343  0.896998  0.287421
1.02485   0.343508  0.905356  0.669327
0.281717  0.859722  0.632145  0.473843
0.498622  0.55674   0.536509  0.848529

********************************

Layer = 1

NeuronActivatedValue=
0.0439881       0.0432561       0.0512891       0.0398854

WeightMatrix (nrow + 1 because of bias value ) =
0.373355  0.612044  0.988833
0.32835   0.935865  0.0718626
0.159847  0.822235  0.789772
0.669706  0.868823  1.01383
0.630842  0.0310458         0.243647

********************************

Layer = 2

NeuronActivatedValue=
0.334768  0.087414  0.183496
```

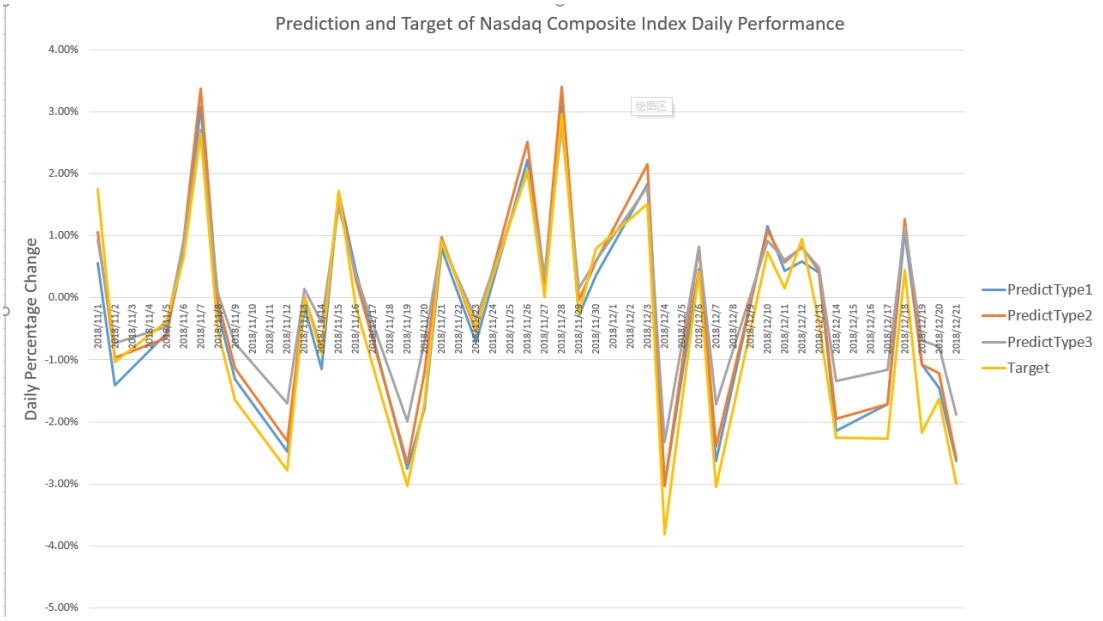
6.Result Analyze

This is the excel file merged 3 txt files
(prediction_output_type1.txt, prediction_output_type2.txt, prediction_output_type3.txt)



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Type 1					Type 2					Type 3		
2	Predict=	Target=	Predict - Target =			Predict=	Target=	Predict - Target =			Predict=	Target=	Predict - Target =
3	0.00556738	0.017542	-0.0119746			0.0105802	0.017542	-0.00696179			0.0093056	0.017542	-0.00823641
4	-0.0141354	-0.0103671	-0.00376825			-0.00969313	-0.0103671	0.000673998			-0.00732009	-0.0103671	0.00304703
5	-0.0057172	-0.003825	-0.00189226			-0.00642355	-0.00382495	-0.0025986			-0.00463411	-0.00382495	-0.000809155
6	0.00704169	0.006428	0.000613691			0.00901731	0.006428	0.0025893			0.00910698	0.006428	0.0026789
7	0.0308507	0.0264088	0.00444197			0.0337688	0.0264088	0.00735998			0.0270253	0.0264088	0.000616528
8	-0.0021883	-0.0052663	0.00307807			-0.00046627	-0.00526634	0.0048006			0.00071819	-0.00526634	0.00598453
9	-0.0130871	-0.0164629	0.00337576			-0.0113662	-0.0164629	0.00509669			-0.00727897	-0.0164629	0.00918391
10	-0.0247906	-0.0278159	0.00302531			-0.0231475	-0.0278159	0.00466845			-0.0169841	-0.0278159	0.0108318
11	-0.0014491	0	-0.00144908			0.000135808	0	0.000135808			0.00145618	0	0.00145618
12	-0.0113933	-0.0089545	-0.00243881			-0.00823311	-0.00895447	0.000721362			-0.00399029	-0.00895447	0.00496418
13	0.0169606	0.0171851	-0.000224507			0.0167128	0.0171851	-0.000472302			0.014685	0.0171851	-0.00250014
14	0.00440235	-0.0015374	0.0059397			0.00290704	-0.00153735	0.00444439			0.00368464	-0.00153735	0.00522199
15	-0.0275343	-0.0302696	0.0027353			-0.0266293	-0.0302696	0.00364026			-0.0198617	-0.0302696	0.0104079
16	-0.0177808	-0.017025	-0.000755776			-0.0120145	-0.017025	0.00501054			-0.00704579	-0.017025	0.00997925
17	0.00792158	0.00918104	-0.00125947			0.00975484	0.00918104	0.000573798			0.00896551	0.00918104	-0.000215532
18	-0.0071447	-0.0047718	-0.00237294			-0.00551828	-0.00477178	-0.0007465			-0.00348436	-0.00477178	0.00128742
19	0.0222506	0.0205895	0.00166112			0.0251058	0.0205895	0.00451634			0.0202749	0.0205895	-0.000314573
20	0.00207101	0.00012004	0.00195097			0.00297854	0.000120039	0.0028585			0.00344964	0.000120039	0.0033296
21	0.0317878	0.0294929	0.00229485			0.0339313	0.0294929	0.00443835			0.0275651	0.0294929	-0.00192789
22	-0.0029192	-0.0025385	-0.000380691			-0.00039402	-0.00253851	0.00214449			0.00153913	-0.00253851	0.00407764
23	0.00361748	0.00790036	-0.00428288			0.00583988	0.00790036	-0.00206048			0.00593111	0.00790036	-0.00196925
24	0.0183378	0.015138	0.00319983			0.0214605	0.015138	0.00632254			0.01788	0.015138	0.00274197
25	-0.0301081	-0.0380406	0.00793252			-0.0302303	-0.0380406	0.00781033			-0.0232701	-0.0380406	0.0147706
26	0.00463658	0.00416706	0.000469526			0.00825969	0.00416706	0.00409263			0.00813958	0.00416706	0.00397253
27	-0.0263255	-0.0304677	0.00414222			-0.0238981	-0.0304677	0.00656963			-0.0171531	-0.0304677	0.0133146
28	0.0115035	0.0073566	0.00414689			0.010861	0.0073566	0.00350442			0.00920546	0.0073566	0.00184886
29	0.00433969	0.001611	0.00272869			0.00571641	0.001611	0.00410541			0.0061223	0.001611	0.0045113
30	0.00585713	0.00945415	-0.00359702			0.00819078	0.00945415	-0.00126337			0.00810488	0.00945415	-0.00134927
31	0.00408402	-0.0039418	0.0080258			0.0043532	-0.00394178	0.00829498			0.00488432	-0.00394178	0.00882611
32	-0.0213668	-0.0225831	0.00121625			-0.0194655	-0.0225831	0.00311757			-0.0133726	-0.0225831	0.00921053
33	-0.0171325	-0.0227084	0.00557589			-0.0171329	-0.0227084	0.00557549			-0.0115538	-0.0227084	0.0111546
34	0.0107056	0.00446867	0.00623693			0.0126135	0.00446867	0.00814484			0.0113441	0.00446867	0.00687543
35	-0.0107683	-0.0216807	0.0109125			-0.0107972	-0.0216807	0.0108836			-0.00690661	-0.0216807	0.0147741
36	-0.0144833	-0.0163361	0.0018528			-0.0121451	-0.0163361	0.00419103			-0.00784308	-0.0163361	0.00849302
37	-0.026303	-0.0299323	0.0036293			-0.025848	-0.0299323	0.00408425			-0.0187561	-0.0299323	0.0111762
38													
39													
40													
41	Type 1 Topology is (4,4,1)												
42	Type 2 Topology is (4,4,3,1)												
43	Type 3 Topology is (4,4,3,2,1)												
44													
45													
46													

This is the prediction result by 3 different models and actual daily performance of Nasdaq composite index.(Target is the actual Nasdaq Composite Index performance)



We can find that modelType1 has the smallest average standard error = 0.00353092, which is the best model among 3 models. The modelType3 has the largest average standard error = 0.005773115, which is the worst.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Type 1					Type 2					Type 3			
2	Date	Predict=	Target=	Predict - Target =	standard error		Predict=	Target=	Predict - Target =	square error		Predict=	Target=	Predict - Target =	square error
3	2018/11/1	0.00556738	0.017542	-0.0119746	0.0119746		0.0105802	0.017542	-0.00696179	0.006962		0.0093056	0.017542	-0.00823641	0.00823641
4	2018/11/2	-0.0141354	-0.0103671	-0.00376825	0.00376825		-0.00969313	-0.0103671	0.000673998	0.000674		-0.00732009	-0.0103671	0.00304703	0.00304703
5	2018/11/5	-0.0057172	-0.00382495	-0.00189226	0.00189226		-0.00642355	-0.00382495	-0.0025986	0.002599		-0.00463411	-0.00382495	-0.000809155	0.000809155
6	2018/11/6	0.00704169	0.006428	0.000613691	0.00061369		0.00901731	0.006428	0.0025893	0.002589		0.00910698	0.006428	0.00267898	0.00267898
7	2018/11/7	0.0308507	0.0264088	0.00444197	0.00444197		0.0337688	0.0264088	0.00735998	0.00736		0.0270253	0.0264088	0.000616528	0.000616528
8	2018/11/8	-0.0021883	-0.00526634	0.00307807	0.00307807		-0.000466272	-0.00526634	0.0048006	0.0048		0.00071819	-0.00526634	0.00598453	0.00598453
9	2018/11/9	-0.0130871	-0.0164629	0.00337576	0.00337576		-0.0113662	-0.0164629	0.00509669	0.005097		-0.00727897	-0.0164629	0.00918391	0.00918391
10	2018/11/12	-0.0247906	-0.0278159	0.00302531	0.00302531		-0.0231475	-0.0278159	0.00466845	0.004668		-0.0169841	-0.0278159	0.0108318	0.0108318
11	2018/11/13	-0.0014491	0	-0.00144908	0.00144908		0.000135808	0	0.000135808	0.000136		0.00145618	0	0.00145618	0.00145618
12	2018/11/14	-0.0113933	-0.00895447	-0.00243881	0.00243881		-0.00823311	-0.00895447	0.000721362	0.000721		-0.00399029	-0.00895447	0.00496418	0.00496418
13	2018/11/15	0.0169606	0.0171851	-0.000224507	0.00022451		0.0167128	0.0171851	-0.000472302	0.000472		0.014685	0.0171851	-0.00250014	0.0250014
14	2018/11/16	0.00440235	-0.00153735	0.0059397	0.0059397		0.00290704	-0.00153735	0.00444439	0.004444		0.00368464	-0.00153735	0.00522199	0.00522199
15	2018/11/19	-0.0275343	-0.0302696	0.0027353	0.0027353		-0.0266293	-0.0302696	0.00364026	0.00364		-0.0198617	-0.0302696	0.0104079	0.0104079
16	2018/11/20	-0.0177808	-0.017025	-0.000755776	0.00075578		-0.0120145	-0.017025	0.00501054	0.005011		-0.00704579	-0.017025	0.00997925	0.00997925
17	2018/11/21	0.00792158	0.00918104	-0.00125947	0.00125947		0.00975484	0.00918104	0.000573798	0.000574		0.00896551	0.00918104	-0.000215532	0.000215532
18	2018/11/23	-0.0071447	-0.00477178	-0.00237294	0.00237294		-0.00551828	-0.00477178	-0.0007465	0.000747		-0.00348436	-0.00477178	0.00128742	0.00128742
19	2018/11/26	0.0222506	0.0205895	0.00166112	0.00166112		0.0251058	0.0205895	0.00451634	0.004516		0.0202749	0.0205895	-0.000314573	0.000314573
20	2018/11/27	0.00207101	0.000120039	0.00195097	0.00195097		0.00297854	0.000120039	0.0028585	0.002859		0.00344964	0.000120039	0.0033296	0.0033296
21	2018/11/28	0.0317878	0.0294929	0.00229485	0.00229485		0.0339313	0.0294929	0.00443835	0.004438		0.0275651	0.0294929	-0.00192789	0.00192789
22	2018/11/29	-0.0029192	-0.00253851	-0.000380691	0.00038069		-0.000394017	-0.00253851	0.00214449	0.002144		0.00153913	-0.00253851	0.00407764	0.00407764
23	2018/11/30	0.00361748	0.00790036	-0.00428288	0.00428288		0.00583988	0.00790036	-0.00206048	0.00206		0.00593111	0.00790036	-0.00196925	0.00196925
24	2018/12/3	0.0183378	0.015138	0.00319983	0.00319983		0.0214605	0.015138	0.00632254	0.006323		0.01788	0.015138	0.00274197	0.00274197
25	2018/12/4	-0.0301081	-0.0380406	0.00793252	0.00793252		-0.0302303	-0.0380406	0.00781033	0.00781		-0.0232701	-0.0380406	0.0147706	0.0147706
26	2018/12/6	0.00463658	0.00416706	0.000469526	0.00046953		0.00825969	0.00416706	0.00409263	0.004093		0.00813958	0.00416706	0.00397253	0.00397253
27	2018/12/7	-0.0263255	-0.0304677	0.00414222	0.00414222		-0.0238981	-0.0304677	0.00656963	0.00657		-0.0171531	-0.0304677	0.0133146	0.0133146
28	2018/12/10	0.0115035	0.0073566	0.00414689	0.00414689		0.010861	0.0073566	0.00350442	0.003504		0.00920546	0.0073566	0.00184886	0.00184886
29	2018/12/11	0.00433969	0.001611	0.00272869	0.00272869		0.00571641	0.001611	0.00410541	0.004105		0.0061223	0.001611	0.0045113	0.0045113
30	2018/12/12	0.00585713	0.00945415	-0.00359702	0.00359702		0.00819078	0.00945415	-0.00126337	0.001263		0.00810488	0.00945415	-0.00134927	0.00134927
31	2018/12/13	0.00408402	-0.00394178	0.0080258	0.0080258		0.0043532	-0.00394178	0.00829498	0.008295		0.00488432	-0.00394178	0.00882611	0.00882611
32	2018/12/14	-0.0213668	-0.0225831	0.00121625	0.00121625		-0.0194655	-0.0225831	0.00311757	0.003118		-0.0133726	-0.0225831	0.00921053	0.00921053
33	2018/12/17	-0.0171325	-0.0227084	0.00557589	0.00557589		-0.0171329	-0.0227084	0.00557549	0.005575		-0.0115538	-0.0227084	0.0111546	0.0111546
34	2018/12/18	0.0107056	0.00446867	0.00623693	0.00623693		0.0126135	0.00446867	0.00814484	0.008145		0.0113441	0.00446867	0.00687543	0.00687543
35	2018/12/19	-0.0107683	-0.0216807	0.0109125	0.0109125		-0.0107972	-0.0216807	0.0108836	0.010884		-0.00690661	-0.0216807	0.0147741	0.0147741
36	2018/12/20	-0.0144833	-0.0163361	0.0018528	0.0018528		-0.0121451	-0.0163361	0.00419103	0.004191		-0.00784308	-0.0163361	0.00849302	0.00849302
37	2018/12/21	-0.026303	-0.0299323	0.0036293	0.0036293		-0.025848	-0.0299323	0.00408425	0.004084		-0.0187561	-0.0299323	0.0111762	0.0111762
38				average standard erro	0.00353092				average standard erro	0.004128				average standard error	0.005773115
39				Av											
40															
41		Type 1 Topology is (4,4,1)													
42		Type 2 Topology is (4,4,3,1)													
43		Type 3 Topology is (4,4,3,2,1)													
44															
45															

Neural network	Type1	Type2	Type3
Average Standard Error	0.00353092	0.004128	0.005773115

7.Conclusion

1.We find that neural network type1 has the best prediction performance among three neural networks.

2.More hidden layers do not necessarily give better prediction result.

3. We need to find a learning rate that is low enough that the network converges, but high enough that we don't have to spend too much time training it.

8.Reference

https://finance.yahoo.com/quote/AAPL

https://finance.yahoo.com/quote/GOOG

https://finance.yahoo.com/quote/AMZN

https://finance.yahoo.com/quote/MSFT

https://finance.yahoo.com/quote/%5EIXIC

Professional C++, Second Edition