

Part I: Project Information

Project Name: using neuron network to predict the forex USDCAD rate

Team Name: Zhen Qian (solo)

Project design and user manual:

Part I: Data normalization (train.xlsx)

Input eight variables: 1 WTI oil price, 2 Brent oil price, 3 SP500 index, 4 SPTSX index (SP Toronto), 5 US10Y yield, 6 CAD10Y yield, 7 Gold Spot price, 8 dollar index

Normalization method: feature scaling.

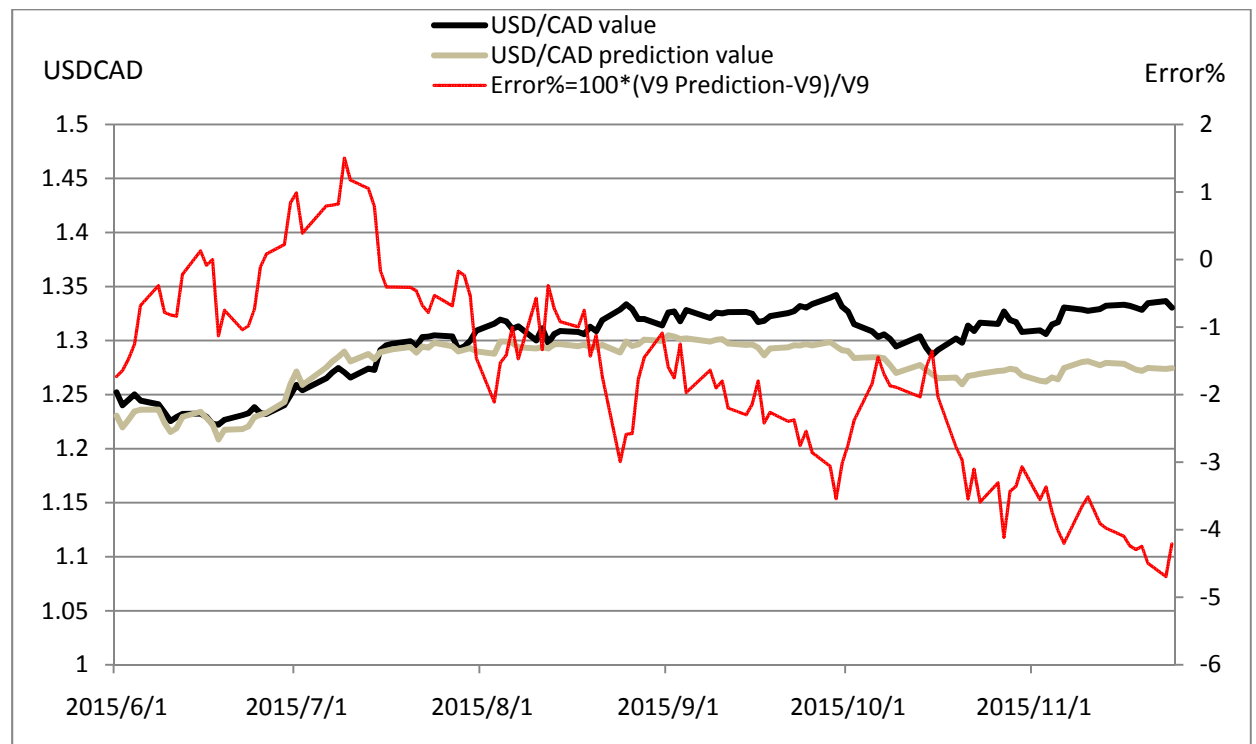
Part II: use neural network program for training

Input the training data file- train01.txt and get the weights

Part III: use neural network program for USDCAD prediction

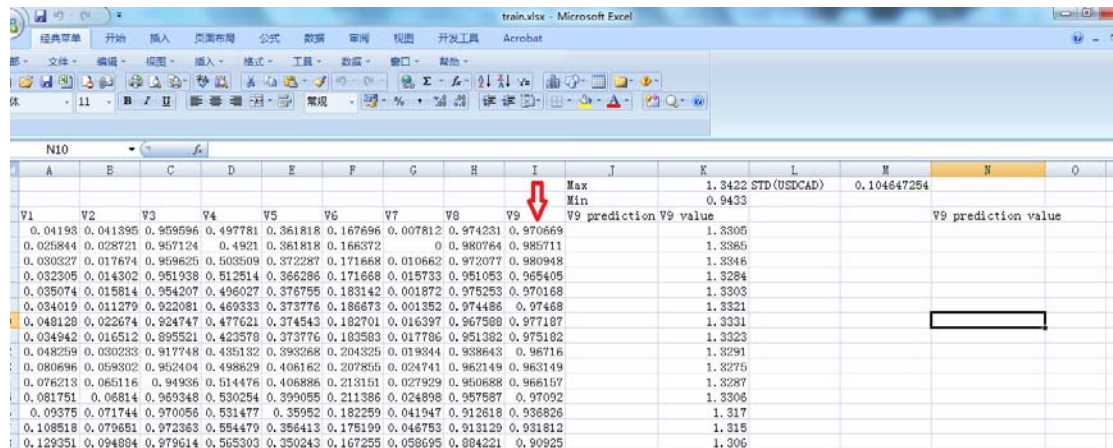
Input the test data file-test01.txt and get the results. Then go back to the train.xlsx file and use reverse feature scaling method to get the predicted USDCAD numbers.

The results are satisfied and see blow graph:



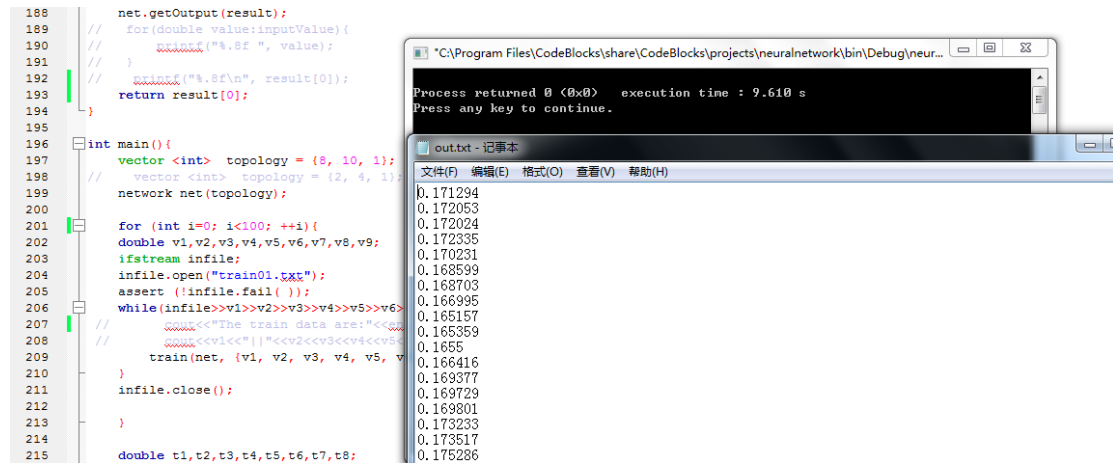
Part II: Program test results

Test Data:



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|----|----------|----------|----------|----------|----------|----------|----------|----------|----------|------------------------|--------|--------------|-------------|---------------------|---|
| | | | | | | | | | | Max | 1.3422 | STD (USDCAD) | 0.104647254 | | |
| | | | | | | | | | | Min | 0.9433 | | | | |
| | | | | | | | | | | V9 prediction V9 value | | | | V9 prediction value | |
| V1 | 0.04193 | 0.041395 | 0.959596 | 0.497781 | 0.361818 | 0.167696 | 0.007812 | 0.974231 | 0.970669 | | 1.3305 | | | | |
| V2 | 0.025844 | 0.028721 | 0.957124 | 0.4921 | 0.361818 | 0.166372 | 0.0 | 0.980764 | 0.985711 | | 1.3365 | | | | |
| V3 | 0.030327 | 0.017674 | 0.959625 | 0.503509 | 0.372287 | 0.171668 | 0.010662 | 0.972077 | 0.980948 | | 1.3346 | | | | |
| V4 | 0.032305 | 0.014302 | 0.951938 | 0.512514 | 0.366286 | 0.171668 | 0.015733 | 0.951053 | 0.965405 | | 1.3284 | | | | |
| V5 | 0.035074 | 0.015814 | 0.954207 | 0.496027 | 0.376755 | 0.183142 | 0.001872 | 0.975253 | 0.970168 | | 1.3303 | | | | |
| V6 | 0.034019 | 0.011279 | 0.922081 | 0.469333 | 0.373776 | 0.186673 | 0.001352 | 0.974486 | 0.97468 | | 1.3321 | | | | |
| V7 | 0.048128 | 0.022674 | 0.924747 | 0.477621 | 0.374543 | 0.182701 | 0.016397 | 0.967588 | 0.977187 | | 1.3331 | | | | |
| V8 | 0.034942 | 0.016512 | 0.895521 | 0.423578 | 0.373776 | 0.183583 | 0.017796 | 0.951382 | 0.975182 | | 1.3323 | | | | |
| V9 | 0.048259 | 0.030233 | 0.917748 | 0.435132 | 0.393268 | 0.204325 | 0.019344 | 0.938643 | 0.96716 | | 1.3291 | | | | |
| | 0.080696 | 0.058302 | 0.952404 | 0.498629 | 0.406162 | 0.207855 | 0.024741 | 0.962149 | 0.963149 | | 1.3275 | | | | |
| | 0.076213 | 0.065116 | 0.94938 | 0.514476 | 0.406886 | 0.213161 | 0.027929 | 0.950688 | 0.966197 | | 1.3287 | | | | |
| | 0.081751 | 0.06814 | 0.949348 | 0.530254 | 0.399055 | 0.211386 | 0.024898 | 0.957587 | 0.97092 | | 1.3306 | | | | |
| | 0.09375 | 0.071744 | 0.970056 | 0.531477 | 0.35952 | 0.182259 | 0.041947 | 0.912618 | 0.936826 | | 1.317 | | | | |
| | 0.108518 | 0.079651 | 0.972363 | 0.554479 | 0.356413 | 0.175199 | 0.046753 | 0.913129 | 0.931812 | | 1.315 | | | | |
| | 0.129351 | 0.094894 | 0.979614 | 0.565303 | 0.350243 | 0.167255 | 0.058695 | 0.884221 | 0.90925 | | 1.306 | | | | |

Running Test 1: i=100, time=9.610s



```

188 net.setOutput(result);
189 for(double value:inputValue){
190     printf("%.8f ", value);
191 }
192 printf("\n", result[0]);
193 return result[0];
194 }
195
196 int main(){
197     vector<int> topology = {8, 10, 1};
198     vector<int> topology = {2, 4, 1};
199     network net(topology);
200
201     for (int i=0; i<100; ++i){
202         double v1,v2,v3,v4,v5,v6,v7,v8,v9;
203         ifstream infile;
204         infile.open("train01.txt");
205         assert (!infile.fail());
206         while(infile>>v1>>v2>>v3>>v4>>v5>>v6>>v7>>v8>>v9){
207             printf("The train data are:");
208             printf("%d<<v1<<v2<<v3<<v4<<v5<<v6<<v7<<v8<<v9<<endl);
209             train(net, {v1, v2, v3, v4, v5, v6, v7, v8, v9});
210         }
211         infile.close();
212     }
213 }
214
215 double t1,t2,t3,t4,t5,t6,t7,t8;
    
```

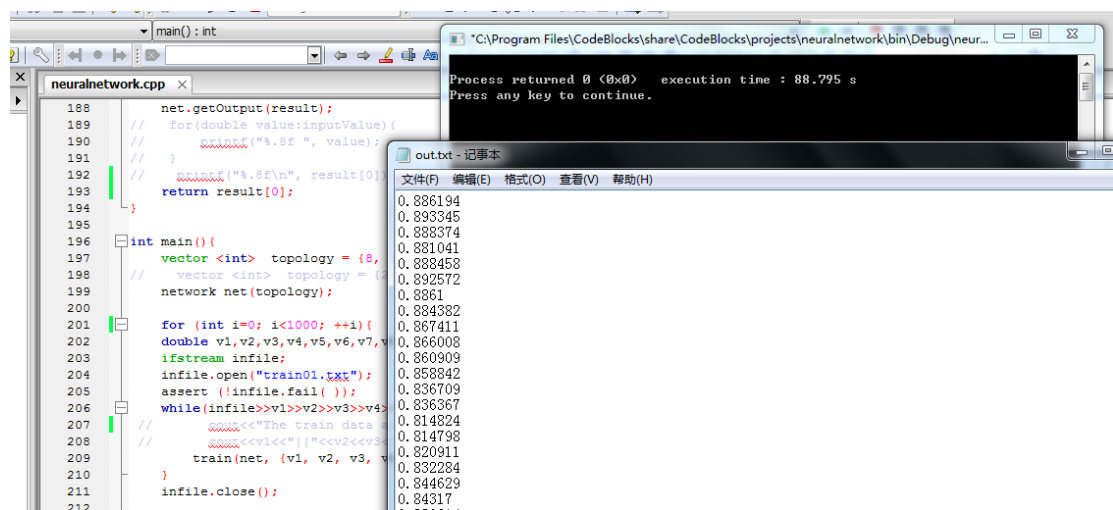
Process returned 0 (0x0) execution time : 9.610 s
Press any key to continue.

out.txt - 记事本

```

0.171294
0.172053
0.172024
0.172335
0.170231
0.168599
0.168703
0.166995
0.165157
0.165359
0.1655
0.1655
0.166416
0.169377
0.169729
0.169801
0.173233
0.173517
0.175286
    
```

Running Test 1: i=1000, time=88.795s



```

188 net.setOutput(result);
189 for(double value:inputValue){
190     printf("%.8f ", value);
191 }
192 printf("\n", result[0]);
193 return result[0];
194 }
195
196 int main(){
197     vector<int> topology = {8, 10, 1};
198     vector<int> topology = {2, 4, 1};
199     network net(topology);
200
201     for (int i=0; i<1000; ++i){
202         double v1,v2,v3,v4,v5,v6,v7,v8,v9;
203         ifstream infile;
204         infile.open("train01.txt");
205         assert (!infile.fail());
206         while(infile>>v1>>v2>>v3>>v4>>v5>>v6>>v7>>v8>>v9){
207             printf("The train data are:");
208             printf("%d<<v1<<v2<<v3<<v4<<v5<<v6<<v7<<v8<<v9<<endl);
209             train(net, {v1, v2, v3, v4, v5, v6, v7, v8, v9});
210         }
211         infile.close();
212     }
213 }
214
215 double t1,t2,t3,t4,t5,t6,t7,t8;
    
```

Process returned 0 (0x0) execution time : 88.795 s
Press any key to continue.

out.txt - 记事本

```

0.886194
0.893345
0.888374
0.881041
0.888458
0.892572
0.8861
0.884382
0.867411
0.866008
0.860909
0.858842
0.836709
0.836367
0.814824
0.814798
0.820911
0.832284
0.844629
0.843117
    
```

Running Test 1: i=5000, time=439.437s

```

188 net.getOutput(result);
189 // for(double value:inputValue){
190 //     printf("%.8f ", value);
191 // }
192 // printf("%.8f\n", result[0]);
193 return result[0];
194 }
195
196 int main(){
197     vector<int> topology = {8, 10, 1};
198     // vector<int> topology = {2, 4, 1};
199     network net(topology);
200
201     for (int i=0; i<5000; ++i){
202         double v1,v2,v3,v4,v5,v6,v7,v8,v9;
203         ifstream infile;
204         infile.open("train01.txt");
205         assert (!infile.fail());
206         while(infile>>v1>>v2>>v3>>v4>>v5>>v6>>v7>>v8>>v9){
207             // cout<<"The train data are:"<<endl;
208             // cout<<v1<<"| "<<v2<<v3<<v4<<v5<<v6<<v7<<v8<<v9<<endl;
209             train(net, {v1, v2, v3, v4, v5, v6, v7, v8, v9});
210         }
211         infile.close();
212     }
213 }
    
```

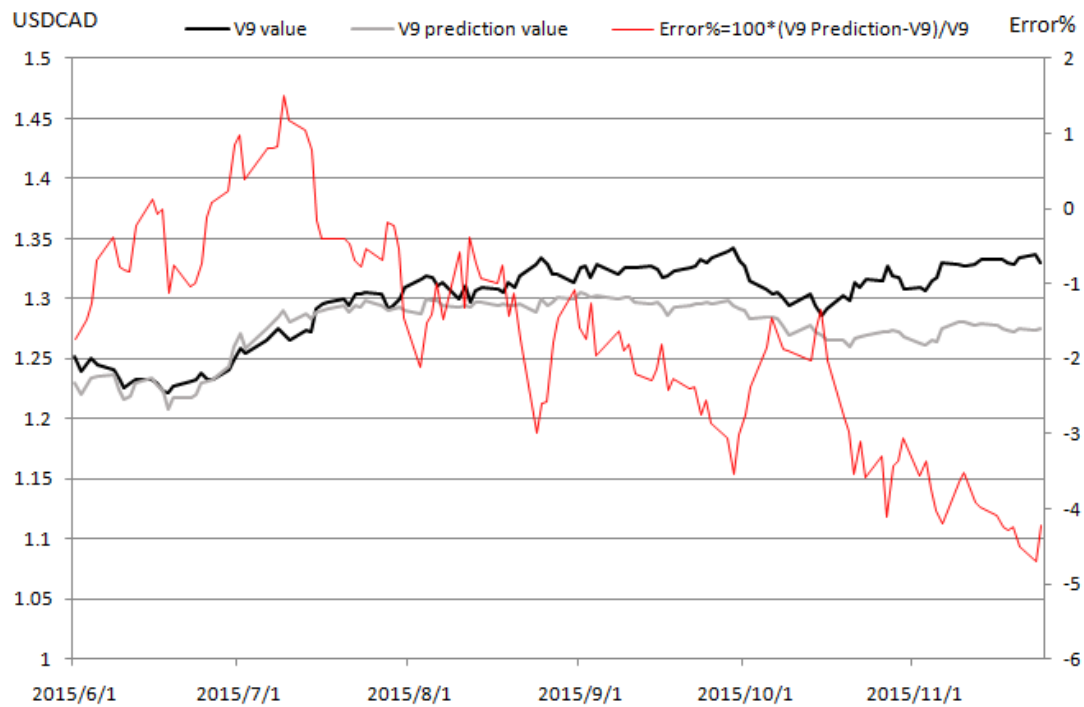
Process returned 0 (0x0) execution time : 439.437 s
Press any key to continue.

out.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

0.830228
0.828425
0.830539
0.824008
0.826992
0.833138
0.840254
0.842373
0.836983
0.846216
0.844301
0.830749
0.804375
0.808778
0.799005
0.801115
0.813679
0.82645
0.828657
0.824816
0.823595

Summary Result (I=5000)



```
//This program is to use neural network algorithm to predict USDCAD rate.

//Input files part: I: train.xlsx, input variables are high correlated with USDCAD rate.

//It includes original data sets: 1 WTI oil price, 2 Brent oil price, 3 SP500 index, 4
SPTSX index (SP Toronto)

// 5 US10Y yield, 6 CAD10Y yield, 7 Gold Spot price, 8 dollar index

// All data sets are normalized by feature scaling method.

//About 90% data sets are used for training, 10% for test.

//The neuralnetwork.cpp is the main program, input train01.txt and test01.txt.

//The output results (i=5000, iteration loop for the deep learning)are showed in
train.xlsx.

//This program is free to use, you can redistribute it

//and modify it under the software license.

//Program Author: Zhen Qian (Martin), Rutgers University

//Email:qianzhen77@hotmail.com, or zhen.qian@rutgers.edu.com

//This program is distributed in the hope that it will be useful,

//but without any warranty for a particular purpose.

//This program is passed by Code::Blocks.

// Copyright (c) Dec 2015


#include <iostream>

#include <fstream>

#include <vector>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cassert>


using namespace std;


struct neuron;
```

//define the weight file, deltaWeight is used for activation function derivative term

```
struct connection{
```

```
    double weight;
```

```
    double deltaWeight;
```

```
};
```

```
typedef vector<neuron> Layer;
```

```
typedef vector<connection> connections;
```

```
//=====class neuron=====
```

```
class neuron{
```

```
public:
```

```
    neuron(int numOutput, int index_);
```

```
    void setOutputValue(double value){outputValue = value;}
```

```
    double getOutputValue() const { return outputValue; }
```

```
    void feedForward(const Layer& prevLayer);
```

```
    void calcOutputGradients(double targetValue);
```

```
    void calcHiddenGradients(const Layer& nextLayer);
```

```
    void updateWeights(Layer& prevLayer);
```

```
private:
```

```
    static double eta;
```

```
    static double alpha;
```

```
    int index;
```

```
    connections outputWeights;
```

```
    double gradient;
```

```
    double outputValue;
```

```
//use the tanh[-1...1] for optimization
```

```
//can try other functions (Sigmoid function) for optimization.
```

```
//Derivative function is used to calculate the delta weight change

double activationFunction(double x) { return tanh(x); }

double activationFunctionDer(double x) { return 1.0 - x * x; }

//random number [0...1]

double randomWeight() { return rand() / double(RAND_MAX); }

double sumDOW(const Layer& nextLayer) const;

};

//eta:learning rate, [0...1]is the gradient decent contribution
//alpha: momentum term, [0...1] that keeps a moving average
//of gradient descent weight change contribution, thus smooth the overall weight
changes

double neuron::eta = 0.10;
double neuron::alpha = 0.30;

neuron::neuron(int numOutput, int index_){
    for(int i = 0; i < numOutput; ++i){
        outputWeights.push_back(connection());
    }
    //Initialization the weights by random numbers.
    outputWeights.back().weight=randomWeight();
}

index = index_;
}

//calculation the neuron's output by  $f(x)=weight[i]*input[i]$ 
void neuron::feedForward(const Layer& prevLayer){
    double sum = 0.0;
    for(int n = 0; n < prevLayer.size(); ++n){
        sum+=
prevLayer[n].getOutputValue()*prevLayer[n].outputWeights[index].weight;
    }
}
```

```
        setOutputValue(activationFunction(sum));
    }

//Output layer gradient descent
void neuron::calcOutputGradients(double targetValue){
    double delta=targetValue-outputValue;
    gradient=delta*activationFunctionDer(outputValue);
}

//hidden layer gradient descent
void neuron::calcHiddenGradients(const Layer& nextLayer){
    double dow=sumDOW(nextLayer);
    gradient=dow*activationFunctionDer(outputValue);
}

//sum the neuron's error
double neuron::sumDOW(const Layer& nextLayer) const{
    double sum=0.0;
    for(int i = 0; i < nextLayer.size() - 1; ++i){
        sum+=outputWeights[i].weight*nextLayer[i].gradient;
    }
    return sum;
}

//update the weights, old weight+ deltaWeight
void neuron::updateWeights(Layer& prevLayer){
    for(int i = 0; i < prevLayer.size(); ++i){
        neuron& neuron = prevLayer[i];
        double oldDeltaWeight=neuron.outputWeights[index].deltaWeight;
```

```
        double
newDeltaWeight=eta*neuron.outputValue*gradient+alpha*oldDeltaWeight;

        neuron.outputWeights[index].deltaWeight=newDeltaWeight;

        neuron.outputWeights[index].weight+=newDeltaWeight;

    }
}
```

```
//=====define the network
class=====
```

```
class network{
public:
    network(const vector <int> & topology);
    void feedForward(const vector <double> & inputValue);
    void backProp(const vector <double> & targetValue);
    void getOutput(vector <double> & outputValue) const;

private:
    typedef vector<Layer> Layers;
    Layers layers;
    double rmse;
    double recentAvgError;
    double recentAvgSmoothingFactor;
};
```

```
//set up layers and fill up with neurons
```

```
network::network(const vector <int> & topology){
    int layerNum_=topology.size();
    for(int layerNum=0; layerNum<layerNum_;++layerNum){
        layers.push_back(Layer());
        int numOutput=layerNum==layerNum_-1 ? 0:topology[layerNum+1];
        Layer& currentLayer = layers.back();
```



```
        for(int neuronNum=0; neuronNum<=topology[layerNum]; ++neuronNum){
            currentLayer.push_back(neuron(numOutput,neuronNum));
        }
//set up the bias term value 1.0 for input and hidden layer
        currentLayer.back().setOutputValue(1.0);
    }
}
```

```
void network::feedForward(const vector <double> & inputValue) {
    // assert(inputValue.size() == layers[0].size() - 1);
    for(int i = 0; i < inputValue.size(); ++i){
        layers[0][i].setOutputValue(inputValue[i]);
    }
    for(int layerNum = 1; layerNum < layers.size(); ++layerNum){
        Layer& layer = layers[layerNum];
        Layer& prevLayer = layers[layerNum-1];
        for(int n = 0; n < layer.size()-1; ++n){
            layer[n].feedForward(prevLayer);
        }
    }
}
```

```
void network::backProp(const vector <double> & targetValue) {
    // Calculate rmse
    Layer& outputLayer = layers.back();
    rmse=0.0;
    for(int i = 0; i < targetValue.size(); ++i){
        double delta = targetValue[i] - outputLayer[i].getOutputValue();
        rmse+=delta*delta;
    }
}
```

```
    rmse=sqrt(rmse/targetValue.size());  
  
    // Calculate a recent average rmse  
  
    recentAvgError=(recentAvgError*recentAvgSmoothingFactor+rmse)/(recentAvgSmoothingFactor+1.0);  
  
    // Calculate output gradient  
  
    for(int i = 0; i < outputLayer.size() - 1; ++i){  
        outputLayer[i].calcOutputGradients(targetValue[i]);  
    }  
  
    // Calculate hidden gradients  
  
    for(int i=layers.size()-2; i > 0; --i){  
        Layer& hiddenLayer=layers[i];  
        Layer& nextLayer=layers[i+1];  
        for(int j = 0; j < hiddenLayer.size(); ++j){  
            hiddenLayer[j].calcHiddenGradients(nextLayer);  
        }  
    }  
  
    // Update weights  
  
    for(int i=layers.size() - 1; i > 0; i--){  
        Layer& layer= layers[i];  
        Layer& prevLayer= layers[i-1];  
        for(int j = 0; j < layer.size()-1; ++j){  
            layer[j].updateWeights(prevLayer);  
        }  
    }  
};
```

```
void network::getOutput(vector <double> & outputValue) const{  
    outputValue.clear();  
    const Layer& outputLayer = layers.back();  
    for(int i = 0; i < outputLayer.size() - 1; ++i){
```

```
        outputValue.push_back(outputLayer[i].getOutputValue());
    }
}

//=====training
part=====

void train(network& net, vector <double> && inputValue, vector <double> &&
targetValue){
    net.feedForward(inputValue);
    net.backProp(targetValue);

}

//=====test part=====

double test(network& net, vector <double> && inputValue){
    net.feedForward(inputValue);
    vector <double> result;
    net.getOutput(result);
    // for(double value:inputValue){
    //     printf("%.8f ", value);
    // }
    // printf("%.8f\n", result[0]);
    return result[0];
}

int main(){

//=====build the 8 input neurons, 9 hidden neurons and 1 output neuron.
    vector <int> topology = {8, 10, 1};
    // vector <int> topology = {2, 4, 1};
    network net(topology);
```

```
//=====reading the training data sets from train01.txt

for (int i=0; i<5000; ++i){

double v1,v2,v3,v4,v5,v6,v7,v8,v9;

ifstream infile;

infile.open("train01.txt");

assert (!infile.fail( ));

while(infile>>v1>>v2>>v3>>v4>>v5>>v6>>v7>>v8>>v9){

//    cout<<"The train data are:"<<endl;

//    cout<<v1<<"||"<<v2<<v3<<v4<<v5<<v6<<v7<<v8<<"||"<<v9<<endl;

    train(net, {v1, v2, v3, v4, v5, v6, v7, v8},{v9});

}

infile.close();

}

//=====test prediction program by test01.txt file and get the
results

double t1,t2,t3,t4,t5,t6,t7,t8;

ifstream in_file;

ofstream outfile;

in_file.open("test01.txt");

outfile.open("out.txt");

assert (!in_file.fail());

assert (!outfile.fail());

while(in_file>>t1>>t2>>t3>>t4>>t5>>t6>>t7>>t8){

//    cout<<"The test data are :"<<endl;

//    cout<<t1<<"||"<<t2<<t3<<t4<<t5<<t6<<t7<<t8<<"||"<<endl;

    outfile<<test(net, {t1, t2, t3, t4, t5, t6, t7, t8})<<endl;

}
```

```
in_file.close();
outfile.close();

//  for(int i = 0; i < 50000; ++i){
//      train(net, {0, 0}, {0});
//      train(net, {0, 1}, {1});
//      train(net, {1, 0}, {1});
//      train(net, {1, 1}, {0});
//  }
//  test(net, {0.1, 0.2,0.3,0.4,0.5,0.6,0.7,0.8});
//  test(net, {0.2, 0.3,0.4,0.1,0.6,0.6,0.7,0.7});
//  test(net, {0.8, 1});
//  test(net, {1, 1});
}
```