

Homework3

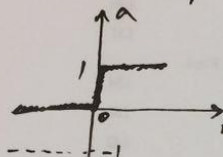
Xiongming Dai

Problem 1

Homework 3.

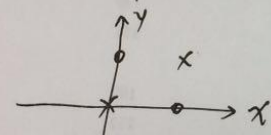
Problem 1. from the figure, we could find the the output of the first layer: $y_1 = x_1 \cdot (c+1) + x_2 \cdot (c+1) - 1.5$ ①

the activation function is hardlimit:



the output $y_2 = \text{hardlimit}[\text{hardlimit}(y_1) \cdot (-2) + x_1 \cdot (c+1) + x_2 \cdot (c+1) - 0.5]$ ②

The XOR problem as follows



We define two classes, A and B, respectively

	A	B
(0,0)	(1,1)	(0,1) (1,0)

③

from ①, ②, ③, we find $y_2(0,0) = y_2(1,1) = 0$,
 $y_2(1,0) = y_2(0,1) = 1$ Therefore, this network can solve XOR problem.

Problem2

The code of bp2.m is filled as follows:

[illegible]

The code of bp2val.m file is completed as follows:

```
[n,Q]=size(P);  
[m,M]=size(W2);  
  
% Forward step code (10 pts)  
y=zeros(m,Q);  
%%%%%%%%%%%%%%  
%% Type your code here %%  
for i=1:Q
```

```

X1=[P(:,i) 1]*W1';
X1=beta*X1;
Y1=tanh(X1);
X2=[Y1 1]*W2';
Y2=purelin(X2);
y(:,i)=Y2;
end

```

The code of q2p1.m file is completed as follows:

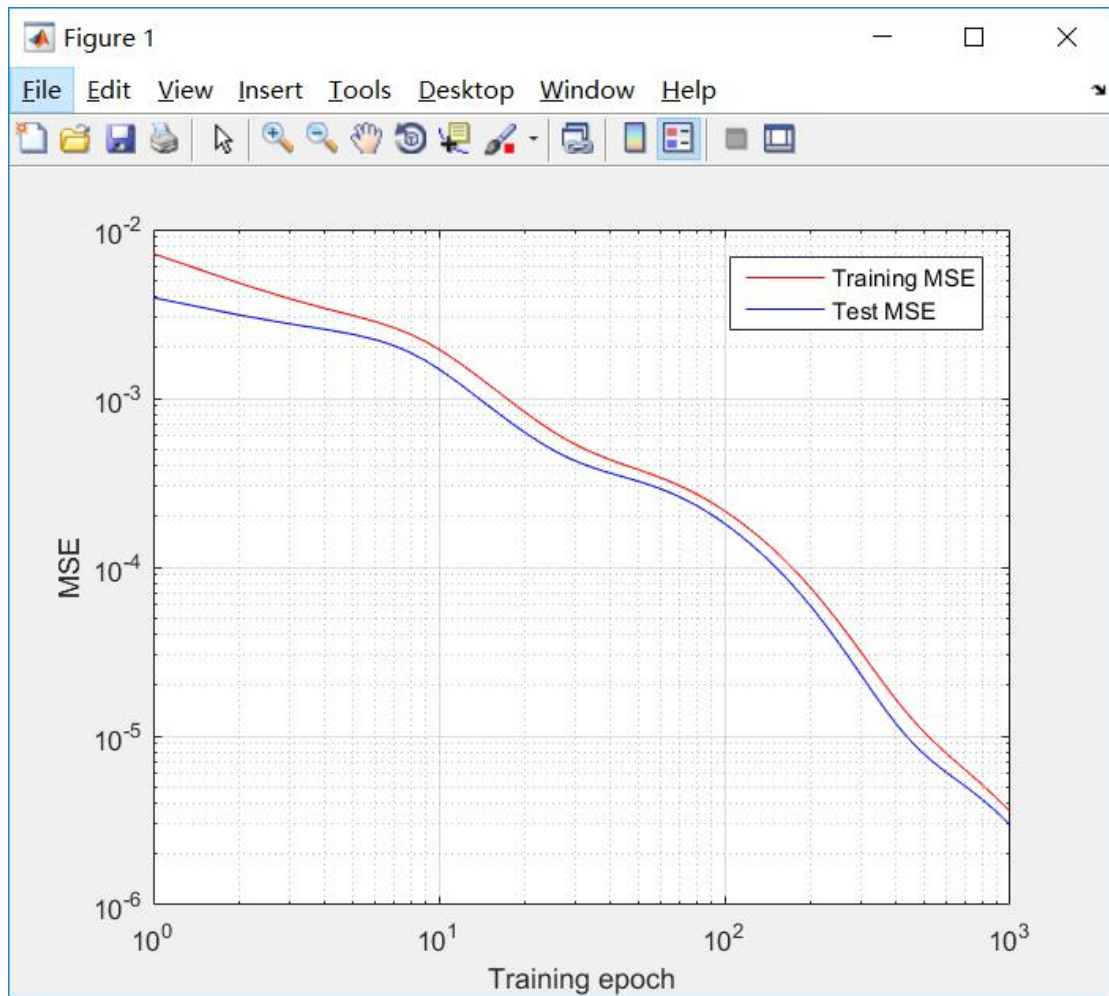
```

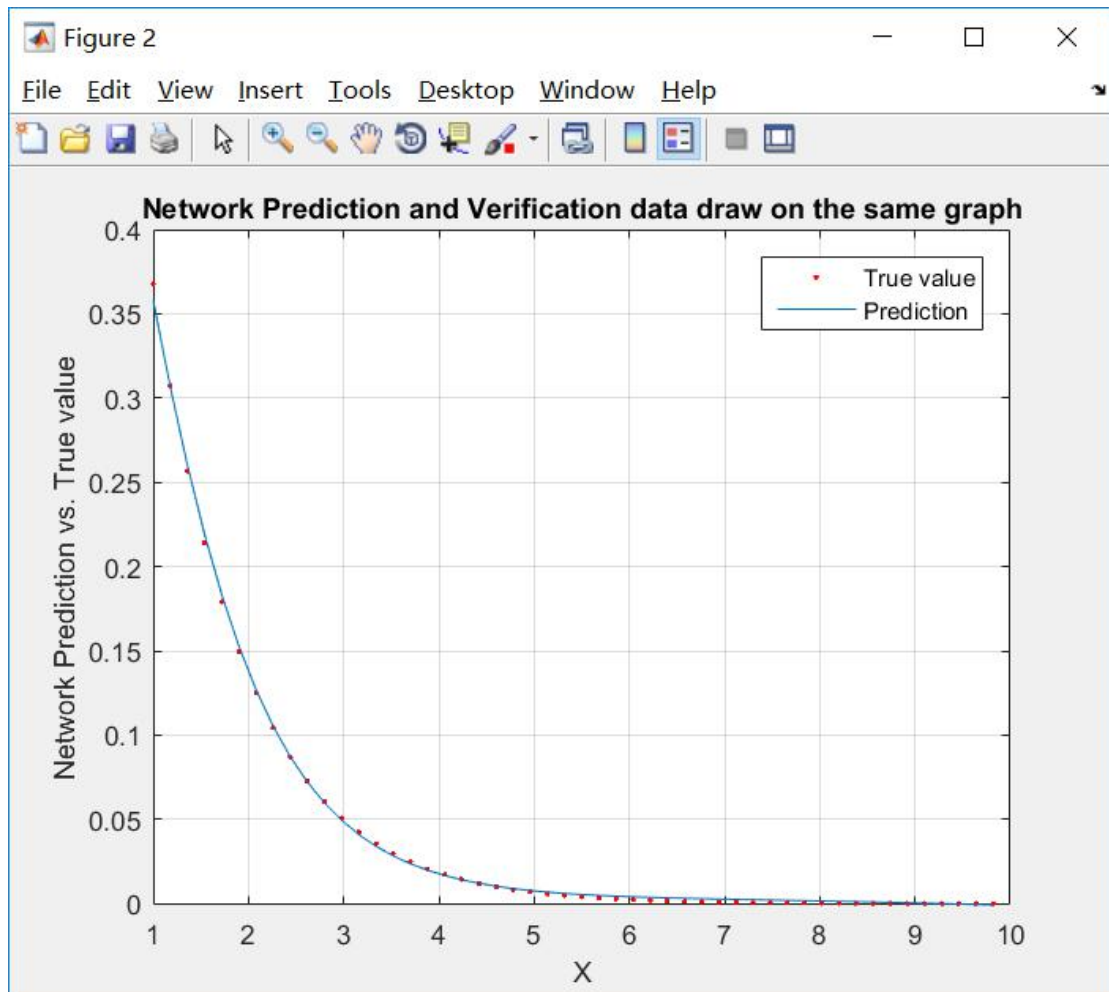
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Type your code here %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Error_matrix=Tverif-Yverif;
N=length(Tverif);
mse=norm(Error_matrix)^2/N;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

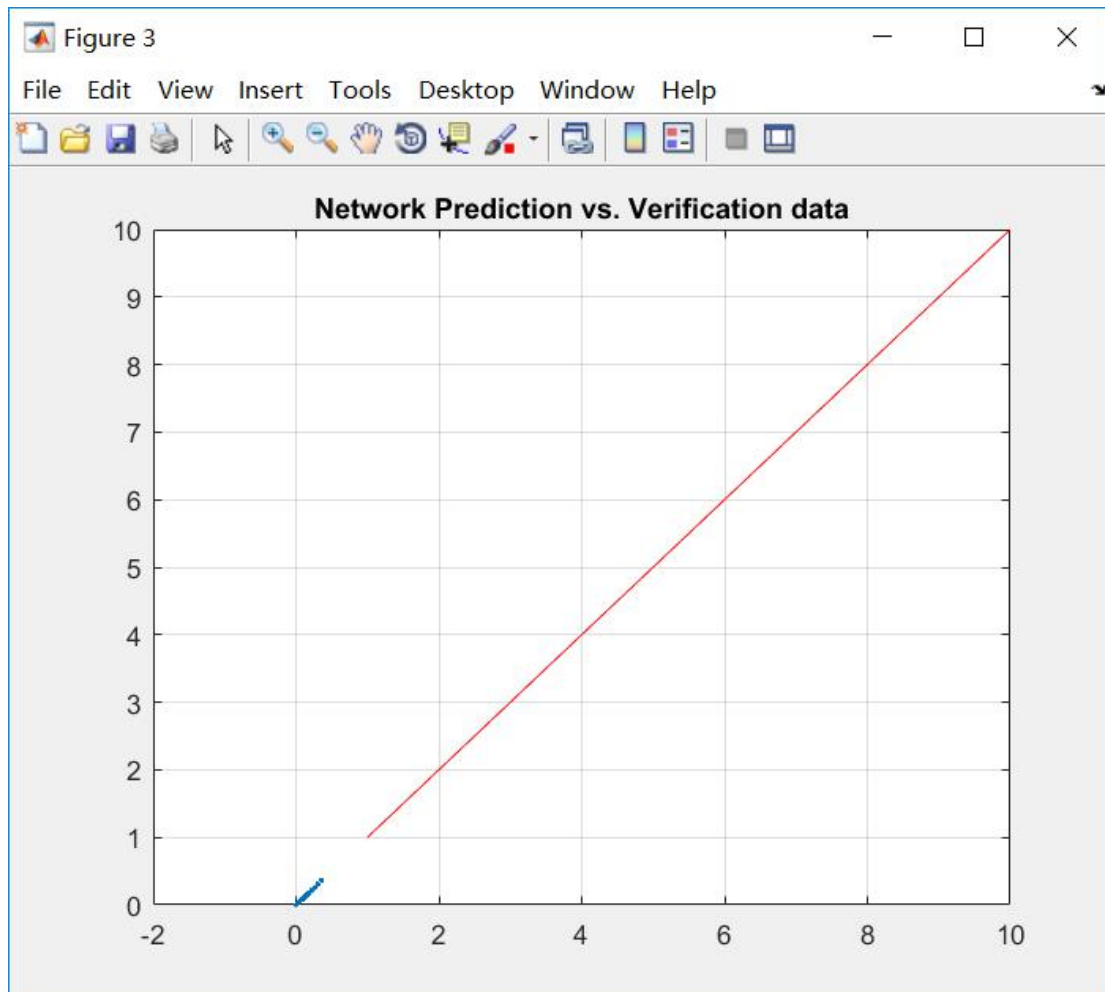
```

The network with 15 hidden neurons, trained with 300 samples, 100 samples are validated, 50 samples are tested. From shown as follows, there is a approximate tendency between the outputs from the neuron network and the ground truth from the actual function.

We also can do many experiments by change the number of hidden neurons, and we can demonstrate the robustness of our framework. Figure 2 and 3 shows the analysis of robustness.



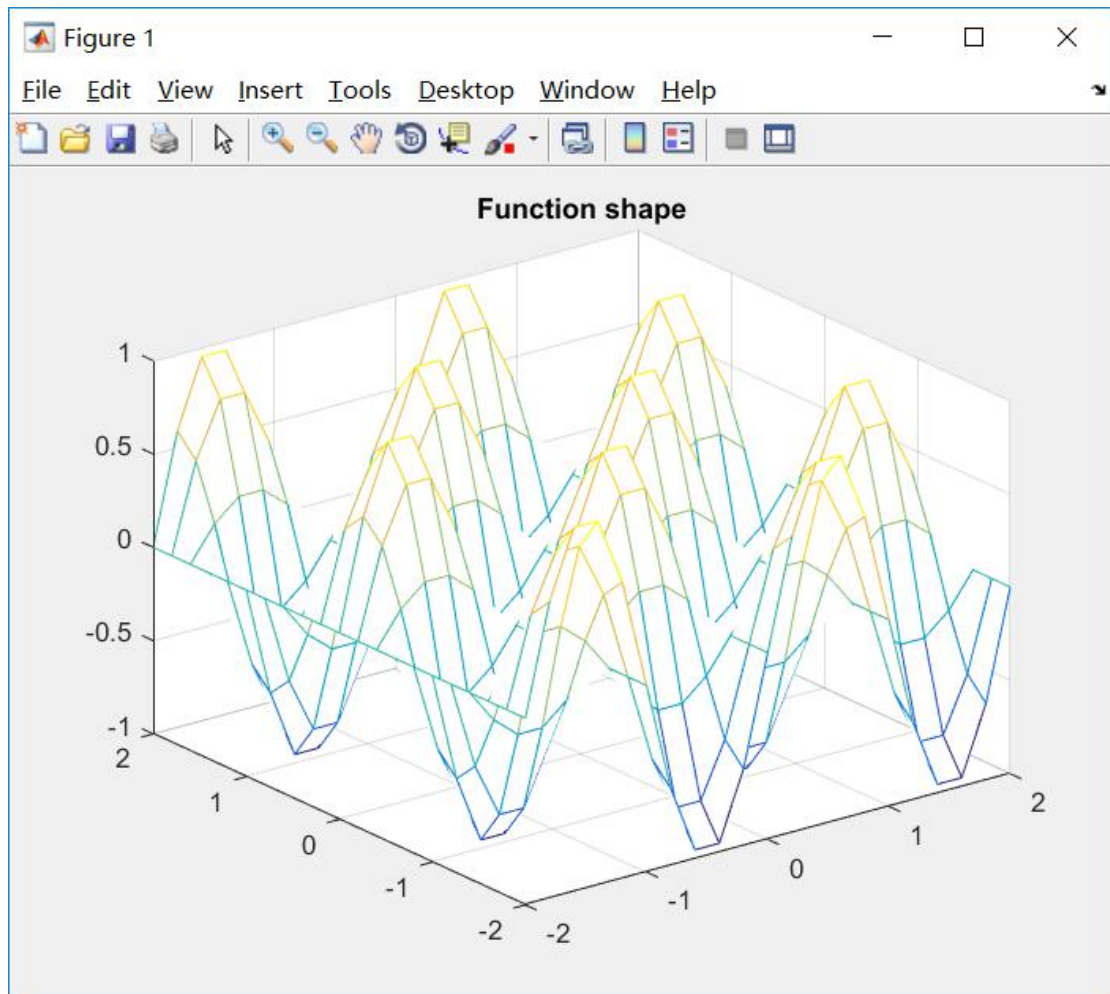


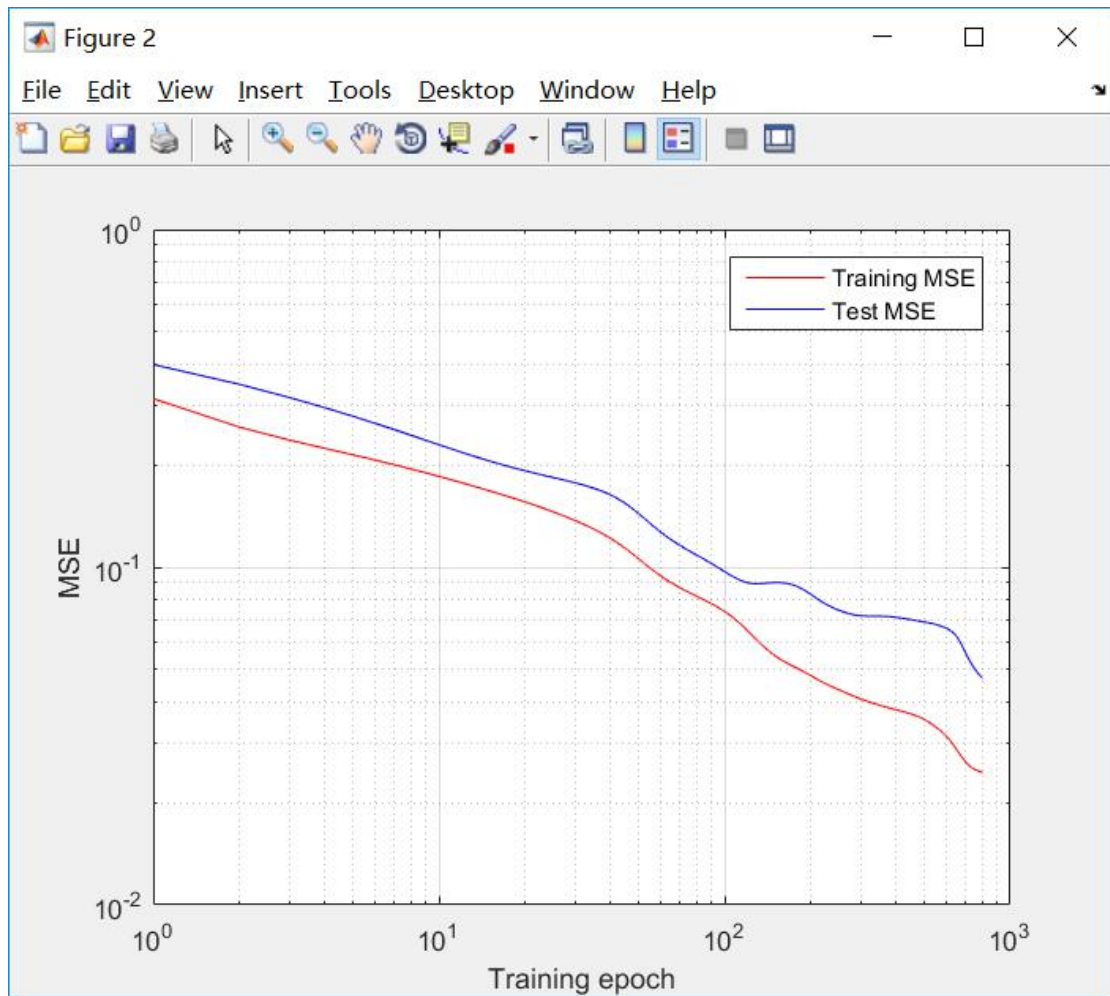


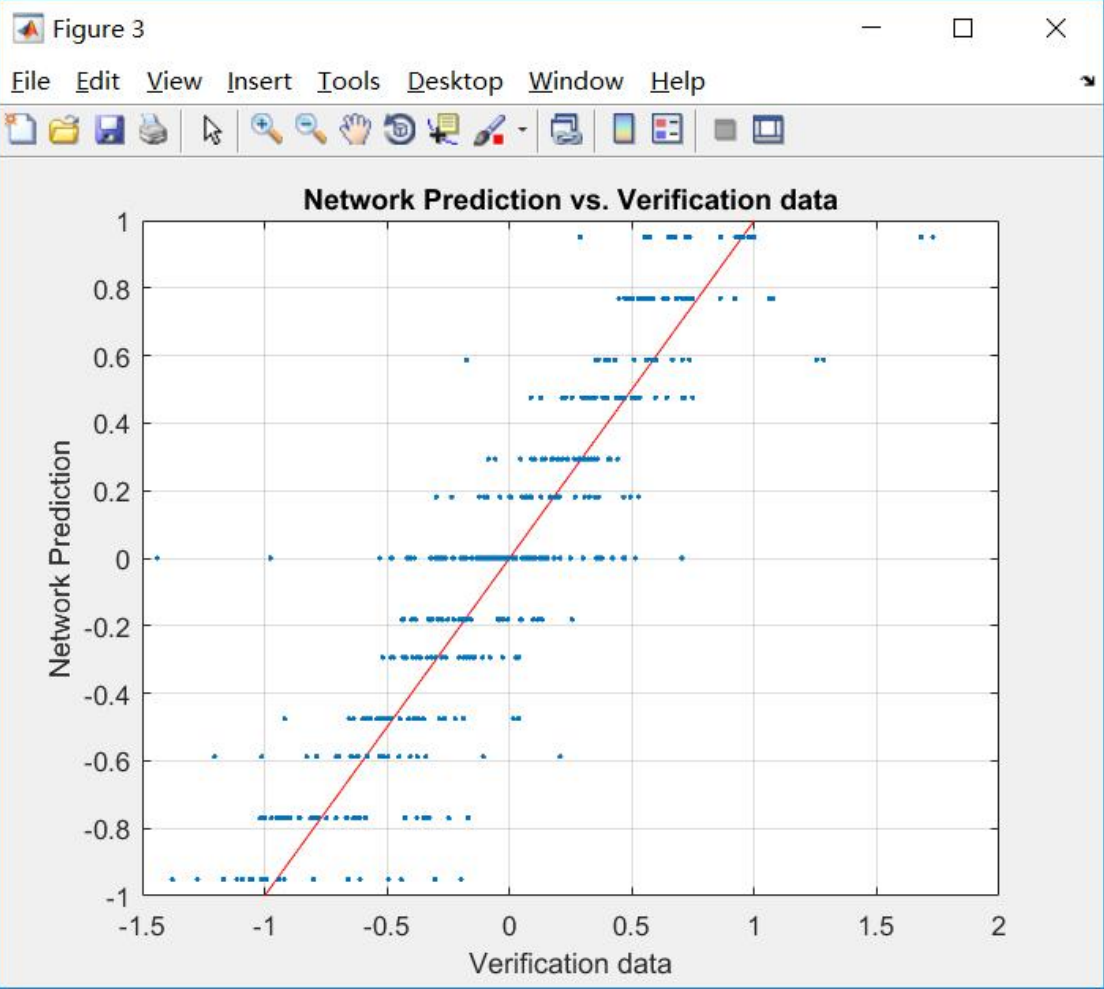
The code of q2p2.m file is completed as follows:

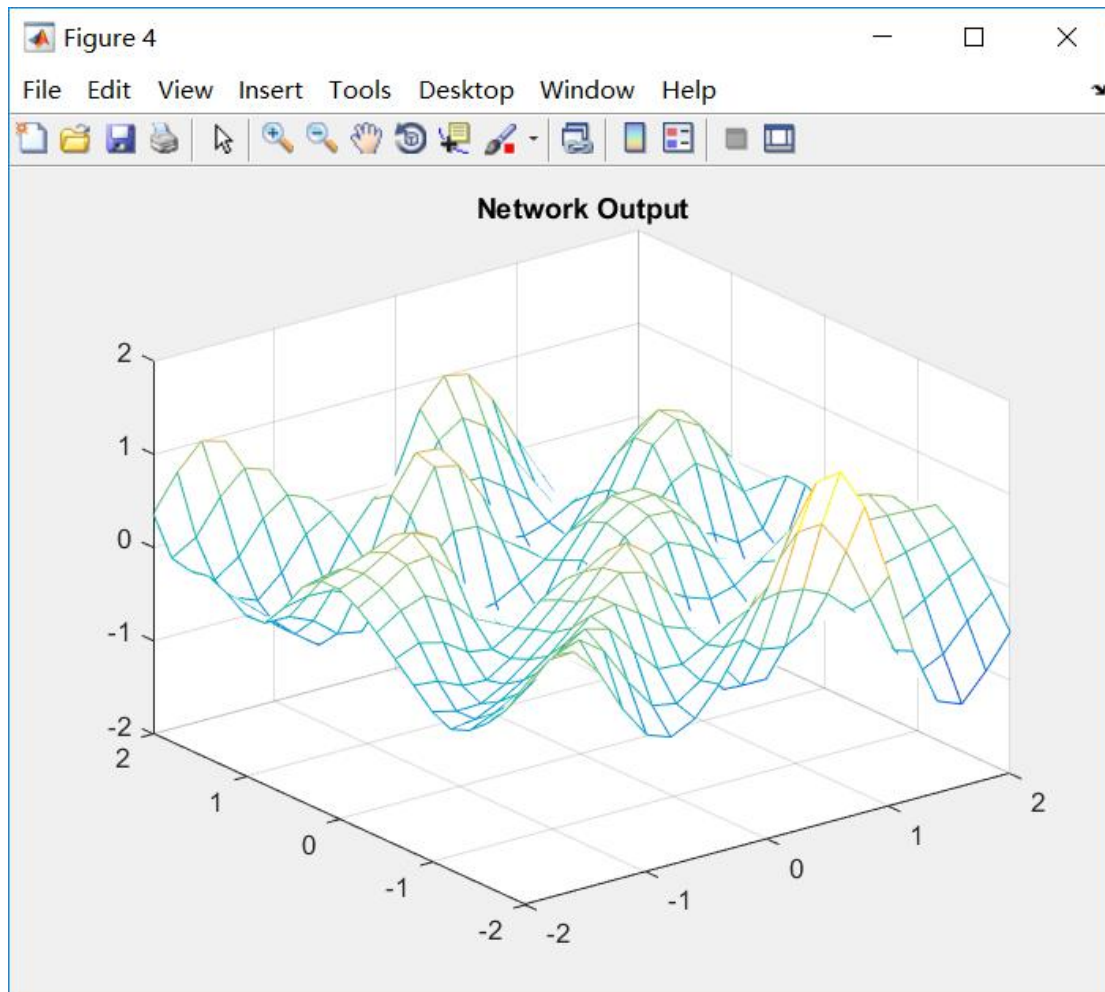
```
%%% Type your code here %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Error_matrix=Ttest-Yverif;
N=length(Yverif);
mse=norm(Error_matrix)^2/N;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

We use the Nguyen-Widrow weight initialization algorithm, and it reduce the time to converge. Figure1 is the ground truth, figure 2 is the analysis of error, Figure 3 is the validation analysis, Figure 4 is the value from the neuron network. We can do the experiments with different numbers of hidden neurons, we find the error is reduced obviously, when the hidden neurons are up to 22.









The total code is as follows:

bp2.m

```
function [W1, W2, E]=bp2(Ptrain, Ttrain, Ptest,
Ttest,M,Tp,W1,W2);
% BP2: MLP NN with one hidden layer trained using
backpropagation
%
% [W1, W2, E]=bp2(Ptrain, Ttrain,Ptest, Ttest,M,Tp,W1,W2);
% Ptrain: n by Q matrix with Q, n-dimensional training input
vectors.
% Ttrain: m by Q matrix with Q, m-dimensional training output
vectors.
% Ptest: n by q matrix with q, n-dimensional testing input
vectors.
% Ttest: m by q matrix with q, m-dimensional testing output
vectors.
% M: number of neurons in the hidden layer
```

```

% Tp:training parameter vector
% Tp(1): learning rate;
% Tp(2): maximum % number of iterations;
% Tp(3): slope of the activation function,  $f(x)=\tanh(\beta x)$ ;
% Tp(4): stopping condition; The learning is stopped if the
error for
% testing data increases by Tp(4)
% Tp(5): weights initialization option. if Tp(5)=0-random
weights
% initialization; else use Nguyen-widrow initialization;
% W1: weights from the input to the hidden layer (includes
biases)
% W2: weights from the hidden layer to the output layer
(includes bias)
% E: MSE history
[n,Q]=size(Ptrain);
[m,Q]=size(Ttrain);
q=length(Ttest);
% Parameter initialization
alpha=Tp(1); % learning rate
MaxIter=Tp(2); %maximum number of epochs
beta=Tp(3); %slope of the tanh activation function,
 $f(x)=\tanh(\beta x)$ 
Incr=Tp(4); %maximum increase of the error for the test set
NgyWid=Tp(5); %NgyWid=0: random initialization;
%NgyWid=1: Nguyen-Widrow initialization
E=[];
MSEold=realmax;
MSEnew=realmax;
iter=0;

%% Network weights initialization
if nargin<8
if NgyWid==0
%random initialization
W1=0.1*randn(M,n+1);
W2=0.1*randn(m,M+1);
else
%Nguyen-Widrow initialization
gamma1=0.7*M^(1/n); % Eq.(3.58)
gamma2=0.7*m^(1/M); % Eq.(3.58)
W1=-0.5+rand(M,n);
W2=-0.5+rand(m,M);
b1=zeros(M,1);

```

```

b2=zeros(m,1);
for i=1:M
W1(i,:) = gamma1*W1(i,:)/norm(W1(i,:)); % Eq. (3.59)
b1(i) =
(max(W1(i,:))-min(W1(i,:)))*rand(1,1)-mean(W1(i,:));
end
W1=[W1 b1];
for i=1:m
W2(i,:) = gamma2*W2(i,:)/norm(W2(i,:)); % Eq. (3.59)
b2(i) =
(max(W2(i,:))-min(W2(i,:)))*rand(1,1)-mean(W2(i,:));
end
W2=[W2 b2];
end

```

```

%% Backpropagation code (20 pts)
MSE=zeros(MaxIter,1);
while iter<MaxIter&MSEnew<=(MSEold+Incr);
    iter=iter+1;
    MSEold=MSEnew;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Type your code here %%
    for i=1:Q
        %The first layer
        X1=[Ptrain(:,i) 1]*W1';
        X1=beta*X1;
        Y1=tanh(X1);
        d1=beta*(1-Y1.^2);
        %The second layer
        X2=[Y1 1]*W2';
        Y2=purelin(X2);
        d2=1;
        %error computing
        error=Ttrain(:,i)-Y2;
        error_sum=error'*error/Q;
        MSE(iter)=MSE(iter)+error_sum;
        %partial differentiate for the parameters
        P_D2=diag(d2)*error;
        error_2=W2(1:m,1:M) '*P_D2;
        P_D1=diag(d1)*error_2;
        %compute the parameters and update the weight
        P2=P_D2*[Y1' 1];
        W2=alpha*P2+W2;
    end
end

```

```

        P1=P_D1*[Ptrain(:,i)' 1];
        W1=alpha*P1+W1;
    end
    %%%%%%%%%%%
    %%%%%%%%%%%
    %%% End of your code %%%
    Error_Matrix=Ttest-bp2val(Ptest,W1,W2,beta);
    MSEnew=norm(Error_Matrix)^2/q;
    %%%%%%%%%%%

    %%%%%%%%%%%
    E=[E;MSEnew]; % storing MSE of iteration
end

%% Generating error plot
figure;
loglog(MSE,'r');
hold on;loglog(E,'b');grid;
xlabel('Training epoch');
ylabel('MSE');
legend('Training MSE','Test MSE');
hold off;
end

```

bp2val.m

```

function y=bp2val(P,W1,W2,beta);
%BP2VAL - output of the MLP NN with one hidden layer/ predicted
values
%usage - y=bp2val(P,W1,W2,beta);
%
% P: n by Q matrix containing Q, n-dimensional input vectors
% W1: weights from the input to the hidden layer
% W2: weights from the hidden layer to the output layer
% beta: slope of the activation function, f(x)=tanh(beta*x)
% y: m by Q matrix containing Q, m-dimensional output vectors

[n,Q]=size(P);
[m,M]=size(W2);
% Forward step code (10 pts)
y=zeros(m,Q);
%%%%%%%%%%
%%% Type your code here %%%
for i=1:Q

```

```

        X1=[P(:,i) 1]*W1';
        X1=beta*X1;
        Y1=tanh(X1);
        X2=[Y1 1]*W2';
        Y2=purelin(X2);
        y(:,i)=Y2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end

```

q2p1.m

```

clear all; close all; clc;
Qq=[300 100 50]; % Number of train, validation and test samples
% Generate the training data
Ptrain=zeros(1,Qq(1));
Ptrain=9*rand(size(Ptrain))+1;
Ttrain=exp(-Ptrain);

% Generate the validation data
Pvalid=zeros(1,Qq(2));
Pvalid=9*rand(size(Pvalid))+1;
Tvalid=exp(-Pvalid);

% Generate the test data. This time choose sequential points
for plotting
Ptest=zeros(1,Qq(3));
i=1:Qq(3);
Ptest(:,i)=1+(i-1)*(9/Qq(3));
Ttest=exp(-Ptest);

% Train the network
Tp=[0.01 1000 1 0.1 0];
% Tp(1): learning rate;
% Tp(2): maximum % number of iterations;
% Tp(3): slope of the activation function, f(x)=tanh(beta*x);
% Tp(4): stopping condition; The learning is stopped if the
error for
% testing data increases by Tp(4)
% Tp(5): weights initialization option. if Tp(5)=0-random
weights
% initialization; else use Nguyen-widrow initialization;

```

```

M=15; % Number of neurons in hidden layer

% Function call for training
[W1,W2,E]=bp2(Ptrain,Ttrain,Pvalid,Tvalid,M,Tp);

% generate the plot for predicted data
Yverif=bp2val(Ptest,W1,W2,1);
figure;
axis([1 10 1 10]);
plot(Ptest, Ttest, 'r.', Ptest, Yverif, '-');grid;
xlabel('X');
ylabel('Network Prediction vs. True value');
legend('True value', 'Prediction');
title('Network Prediction and Verification data draw on the
same graph');
figure;
xlabel('Prediction');
ylabel('Verification data');
axis([1 10 1 10]);
i=1:0.1:10;
plot(i,i, 'r-', Yverif, Ttest, '.'); grid;
title('Network Prediction vs. Verification data');
% Calculate MSE for testing data
%%% Type your code here %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Error_matrix=Tverif-Yverif;
N=length(Tverif);
mse=norm(Error_matrix)^2/N;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

q2p2.m

```

clear all;close all;
Qq=[200 100 50];% Number of train, validation and test samples

% Generate the training data
Ptrain=zeros(2,Qq(1));
Ptrain=4*rand(size(Ptrain))-2;
Ttrain=sin(pi*Ptrain(1,:)).*cos(pi*Ptrain(2,:));

% Generate the validation data
Pvalid=zeros(2,Qq(2));

```

```

Pvalid=4*rand(size(Pvalid))-2;
Tvalid=sin(pi*Pvalid(1,:)).*cos(pi*Pvalid(2,:));

% Generate the test data. This time choose sequential points
for
% plotting
[X,Y] = meshgrid(-2:0.2:2);
Z=sin(pi*X).*cos(pi*Y);
mesh(X,Y,Z);
title('Function shape');
Ptest=[reshape(X,1,441);reshape(Y,1,441)];
Ttest=reshape(Z,1,441);

% Train the network
Tp=[0.02 800 1 0.1 1];
% Tp(1): learning rate;
% Tp(2): maximum % number of iterations;
% Tp(3): slope of the activation function, f(x)=tanh(beta*x);
% Tp(4): stopping condition; The learning is stopped if the
error for
% testing data increases by Tp(4)
% Tp(5): weights initialization option. if Tp(5)=0-random
weights
% initialization; else use Nguyen-widrow initialization;
M=20;% Number of neurons in hidden layer

[W1,W2,E]=bp2(Ptrain,Ttrain,Pvalid,Tvalid,M,Tp);

% generate the plots for predicted data
Yverif=bp2val(Ptest,W1,W2,1);
figure;
axis([-1 1 -1 1]);
i=-1:0.02:1;
plot(i,i,'r-',Yverif, Ttest, '.'); grid;
xlabel('Verification data');
ylabel('Network Prediction');
title('Network Prediction vs. Verification data');
figure;
X2=reshape(Ptest(1,:),21,21);
Y2=reshape(Ptest(2,:),21,21);
Z2=reshape(Yverif,21,21);
mesh(X2,Y2,Z2);
title('Network Output');

```



```

% Calculate MSE for testing data
%%% Type your code here %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Error_matrix=Ttest-Yverif;
N=length(Yverif);
mse=norm(Error_matrix)^2/N;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Problem 3:

1.The MATLAB code is shown below

```

%%% Homework 3 Problem 3 part a %%%
clear all;
clc;
% load data
IRIS3data=xlsread('IRIS 3 class dataset');
% three-dimensional input features
x=IRIS3data(:,1:4)';
% target class
t=IRIS3data(:,5)';

% build network
net = feedforwardnet(10,'traingdm');
% learning rate and momentum constant
net.trainParam.lr=0.001;
net.trainParam.mc = 0.9;
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 67/100;
net.divideParam.valRatio = 0/100;
net.divideParam.testRatio = 33/100;

net = train(net,x,t);
view(net)
y = net(x);
perf=perform(net,y,t)

```

And the performance of network is described by mean square error and shown in the following table.

η	α				
	0	0.1	0.5	0.8	0.99
0.001	3.1253	0.4957	0.4957	0.4886	0.4886
0.01	0.4869	0.4869	0.4869	0.4869	0.5543
0.1	0.2816	0.0864	0.0845	0.0847	0.0847
0.25	0.0849	0.0977	0.0806	0.0804	0.0804
0.99	0.1339	0.1339	0.1339	0.0774	0.0774

For this random initial weights, when the $(\eta, \alpha) = (0.5, 0.8)$ or $(0.5, 0.99)$ achieve the best classification accuracy.

2.The MATLAB code is

```

%%% Homework 3 Problem 3_2 %%%
clear all;
clc;
%% load data
IRIS3data=xlsread('IRIS 3 class dataset');
% three-dimensional input features
x=IRIS3data(:,1:4)';
% target class
t=IRIS3data(:,5)';

%% build network
net = feedforwardnet(10,'trainlm');
setdemorandstream(491218382);
% learning rate and momentum constant
eta=[0.001 0.01 0.1 0.25 0.5];
alpha=[0 0.1 0.5 0.8 0.99];
% eta=0.5;
% alpha=0.1;
for i=1:length(eta)
    for j=1:length(alpha)
        net.trainParam.lr=eta(i);
        net.trainParam.mc = alpha(j);
        % Set up Division of Data for Training, Validation, Testing
        net.divideParam.trainRatio = 66/100;
        net.divideParam.valRatio = 1/100;
        net.divideParam.testRatio = 33/100;

        net = train(net,x,t);
    end
end

```

```

    % view(net)
    y = net(x);
    perf(i,j)=perform(net,y,t);
end
end

```

And the performance of network is described by mean square error and shown in the following table.

η	α				
	0	0.1	0.5	0.8	0.99
0.001	0.0948	0.0669	0.0581	0.0581	0.0581
0.01	0.0809	0.0809	0.0165	0.0171	0.0079
0.1	0.0079	0.3816	0.3816	0.3816	0.3816
0.25	0.3816	0.3816	0.3816	0.3816	0.3816
0.5	0.3816	0.3816	0.3816	0.3816	0.3816

For this random initial weights, when the $(\eta, \alpha) = (0.01, 0.99)$ or $(0.1, 0)$ achieve the best classification accuracy. And the Levenberg-Marquardt variant of backpropagation has better performance with respected to classical backpropagation.

3.The MATLAB code is

```

%%% Homework 3 Problem 3_3 %%%
clear all;
clc;
%% load data
IRIS3data=xlsread('IRIS 3 class dataset');
% three-dimensional input features
x=IRIS3data(:,1:4)';
% target class
t=IRIS3data(:,5)';

%% build network
net = feedforwardnet(10,'trainlm');
% setdemorandstream(491218382);

```

```

% learning rate and momentum constant
eta=[0.001 0.01 0.1 0.25 0.5];
alpha=[0 0.1 0.5 0.8 0.99];
% eta=0.5;
% alpha=0.1;
for i=1:length(eta)
    for j=1:length(alpha)
        net.trainParam.lr=eta(i);
        net.trainParam.mc = alpha(j);
        % Set up Division of Data for Training, Validation, Testing
        net.divideParam.trainRatio = 66/100;
        net.divideParam.valRatio = 1/100;
        net.divideParam.testRatio = 33/100;
        % Nguyen-Widrow technique for weight initialization
        net.initFcn = 'initlay';
        net.layers{1}.initFcn = 'initnw';
        net.layers{2}.initFcn = 'initnw';
        % net = train(net,x,t);
        net2=init(net);
        net2= train(net2,x,t);

        % view(net)
        y = net2(x);
        perf(i,j)=perform(net,y,t);
    end
end
end

```

And the performance of network is described by mean square error and shown in the following table.

η	α				
	0	0.1	0.5	0.8	0.99
0.001	0.3157	0.0682	0.0626	0.0498	0.1156
0.01	0.1133	0.1033	0.1024	0.0560	0.0833
0.1	0.0881	0.0757	0.0786	0.1103	3.4369
0.25	0.0898	0.0958	0.0533	0.0828	0.0702
0.5	0.1023	0.0776	0.0541	0.0577	0.0607

Using the Nguyen-Widrow technique for weight initialization, $(\eta, \alpha) = (0.25, 0.5)$ or $(0.1, 0)$ achieve the best classification accuracy. And

the Nguyen-Widrow technique has better performance with respected to the one using random weight initialization.

4.The MATLAB code is

```
%%% Homework 3 Problem 3_4 %%%
clear all;
clc;
%% load data
IRIS3data=xlsread('IRIS 3 class dataset');
% three-dimensional input features
x=IRIS3data(:,1:4)';
% target class
t=IRIS3data(:,5)';
% shuffle data: random Select 80% as training data, 20% as
the test data
[n,m]=size(x); % n: number of inputs ; m: number
of samples
index=randsample(m,m);
k_fold=3; % three_fold cross validation
num_sub=m*0.8/k_fold; % number of subsets used for
training
%% build network
net = feedforwardnet(10,'trainlm');
% setdemorandstream(491218382);
% learning rate and momentum constant
eta=[0.001 0.01 0.1 0.25 0.5];
alpha=[0 0.1 0.5 0.8 0.99];
% eta=0.5;
% alpha=0.1;
for i=1:length(eta)
    for j=1:length(alpha)
        for k=1:k_fold
            % define learning rate and momentum constant
            net.trainParam.lr=eta(i);
            net.trainParam.mc = alpha(j);
            % Set up Division of Data for Training, Validation,
Testing
            net.divideFcn='divideind';
            if k==1
                net.divideParam.trainInd=index(41:120);
            elseif k==2
```

```

        net.divideParam.trainInd=[index(1:40) '
index(81:120) '];
        elseif k==3
            net.divideParam.trainInd=index(1:80);
        end

net.divideParam.valInd=index((k-1)*num_sub+1:k*num_sub);
net.divideParam.testIn=index(121:150);

% Nguyen-Widrow technique for weight
initilaization
    net.initFcn = 'initlay';
    net.layers{1}.initFcn = 'initnw';
    net.layers{2}.initFcn = 'initnw';
    % net = train(net,x,t);
    net2=init(net);
    net2= train(net2,x,t);
    % view(net)
    y = net2(x);
    perf(k)=perform(net,y,t);
end
performance(i,j)=mean(perf);
end
end

```

And the performance of network is described by mean square error and shown in the following table.

η	α				
	0	0.1	0.5	0.8	0.99
0.001	0.0716	0.0688	0.0778	0.0747	0.0640
0.01	0.0678	0.0650	0.0666	0.0731	0.0647
0.1	0.0731	0.0721	0.0660	0.0599	0.0713
0.25	0.0749	0.0880	0.0728	0.0626	0.0638
0.5	0.0747	0.0665	0.0665	0.0659	0.0794

Using the 3-fold cross validation, $(\eta, \alpha) = (0.1, 0.8)$ achieve the best classification accuracy. And the 3-fold cross validation can effectively improve the network's performance based on the results

shown in the table.

(End)