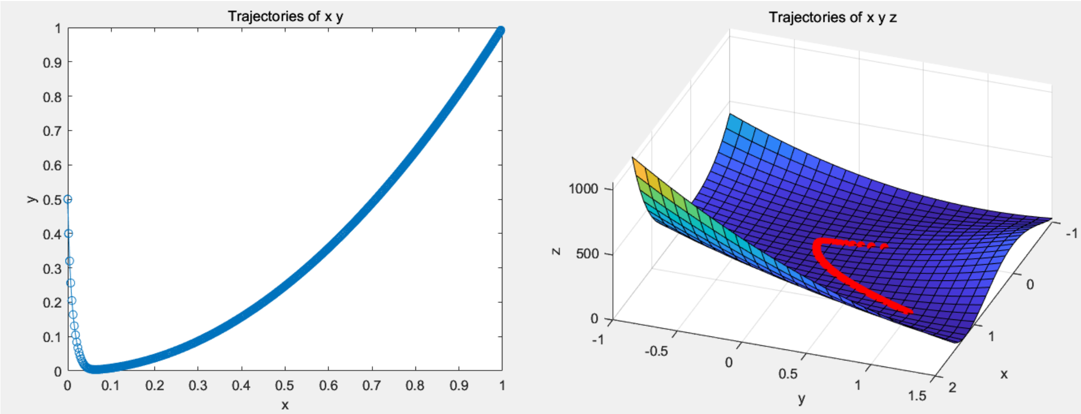
Q1:

a) Steepest (Gradient) descent method

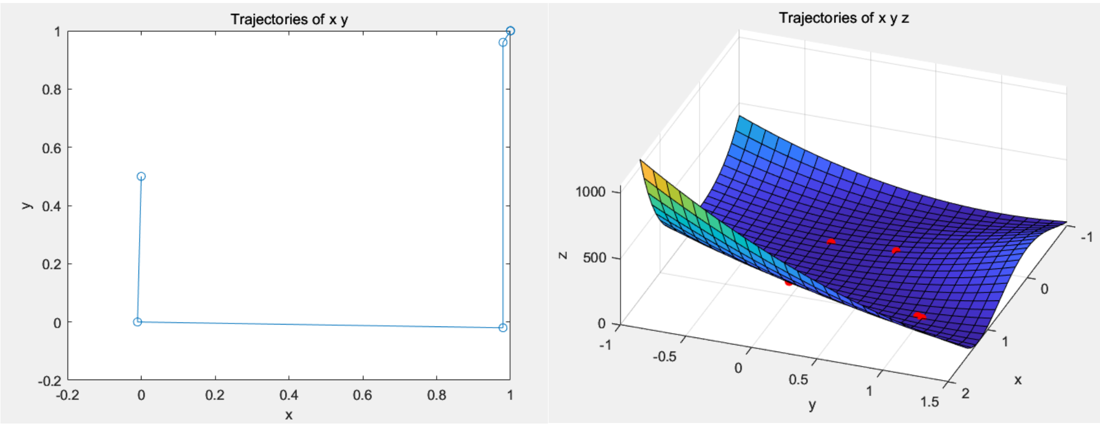
Learning rate=0.001; Iterations number=11419;




```
Minimum value: 0.000010
Number of iterations: 11419
```

When the learning rate is set to be 0.2, the function didn't converge anymore.

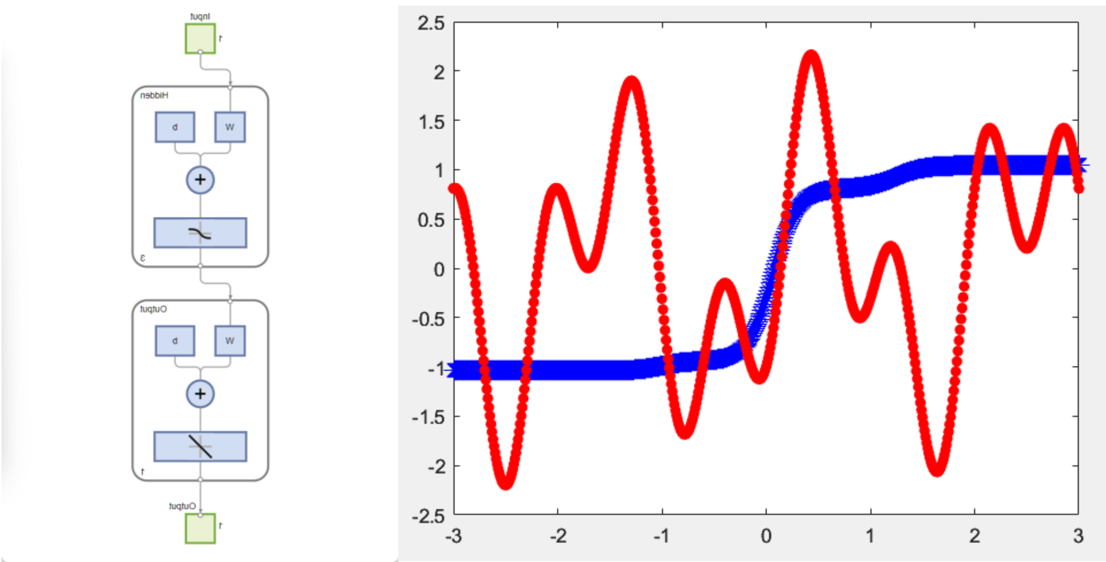
b) Newton's method



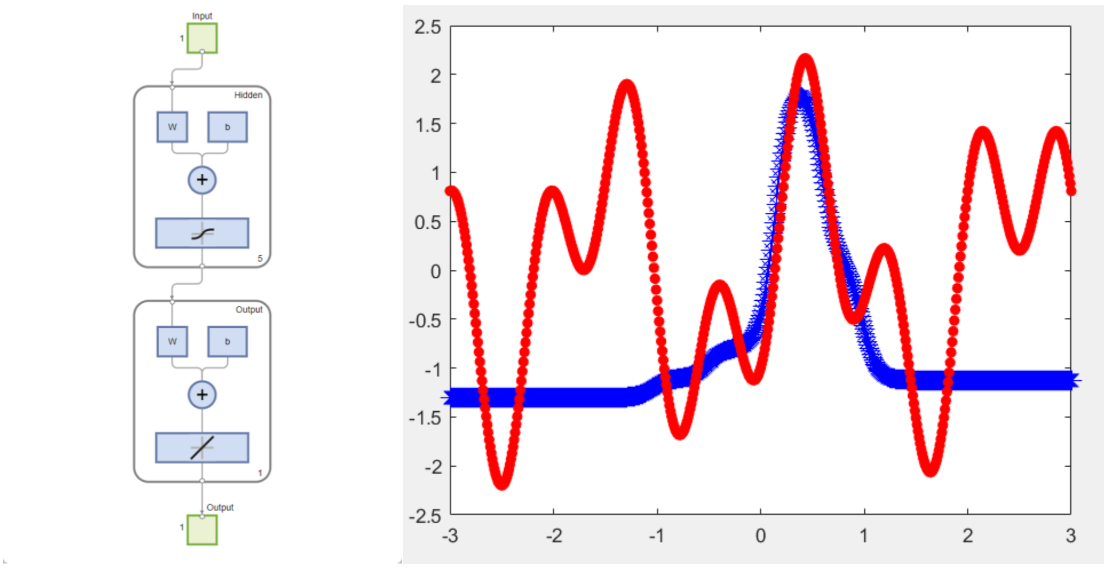

```
Minimum value: 0.000000
Number of iterations: 6
```

Q2:

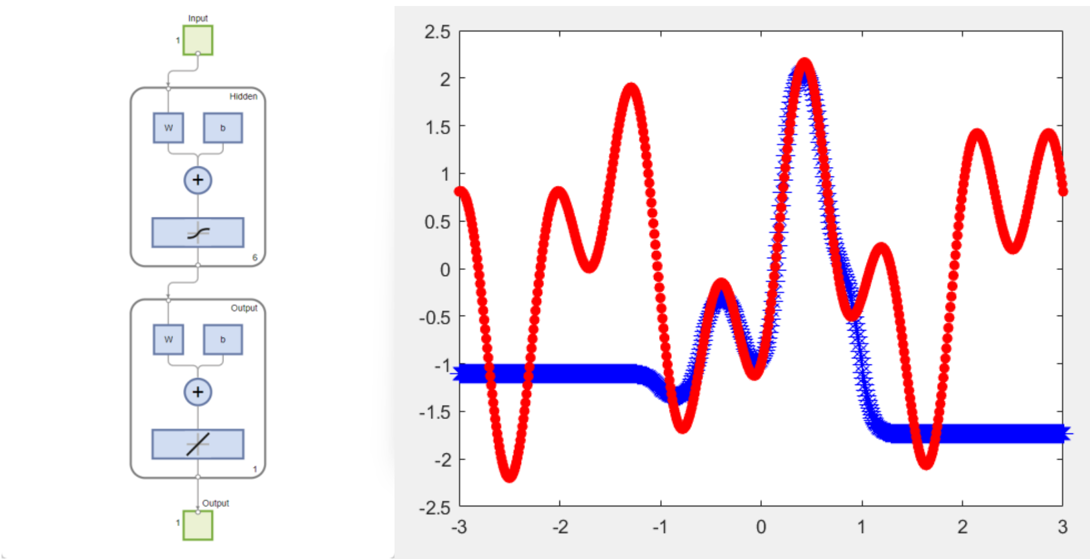
a) using the sequential mode with BP algorithm



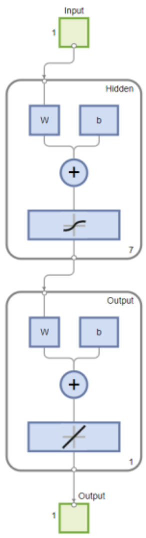
Hidden neurons = 3

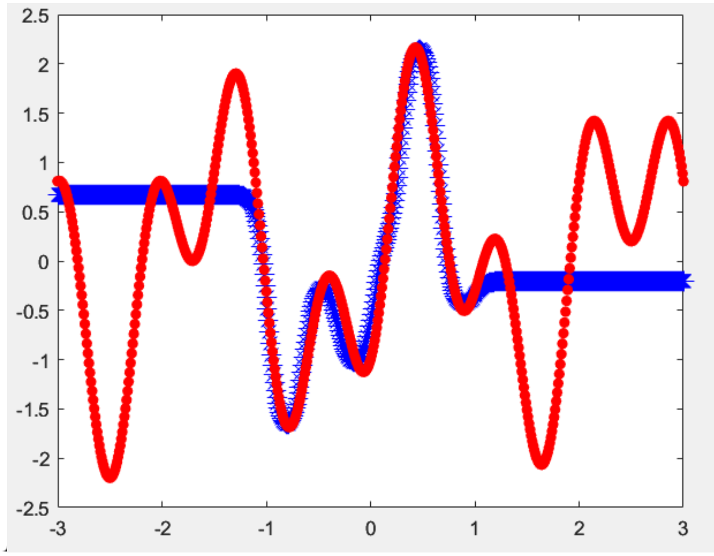


Hidden neurons = 5

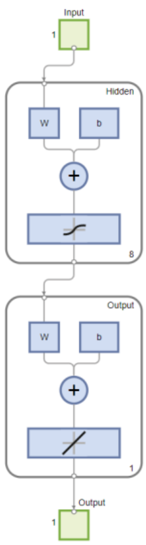


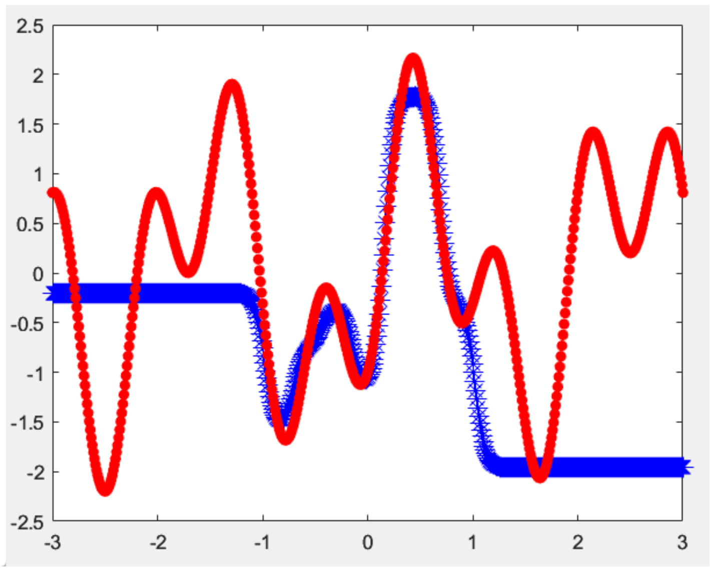
Hidden neurons = 6



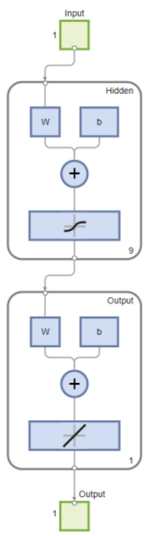


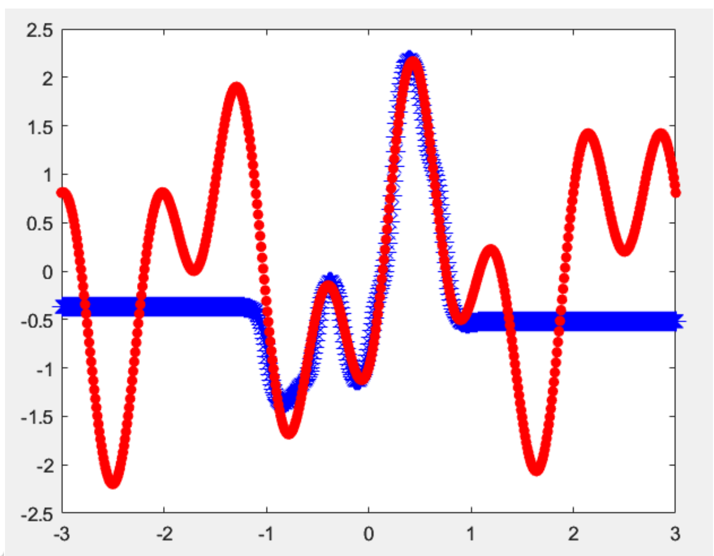
Hidden neurons = 7



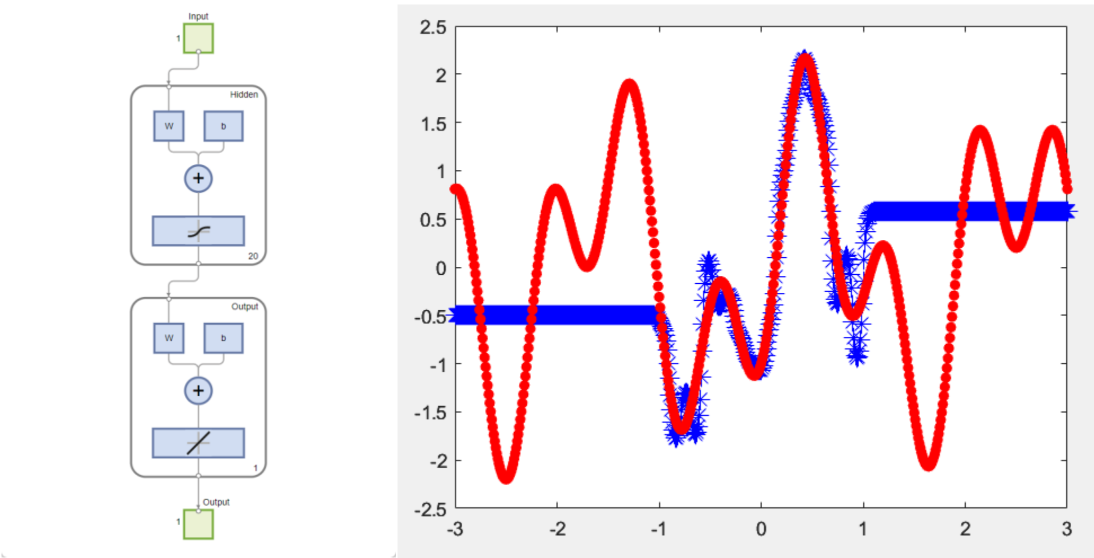


Hidden neurons = 8





Hidden neurons = 9



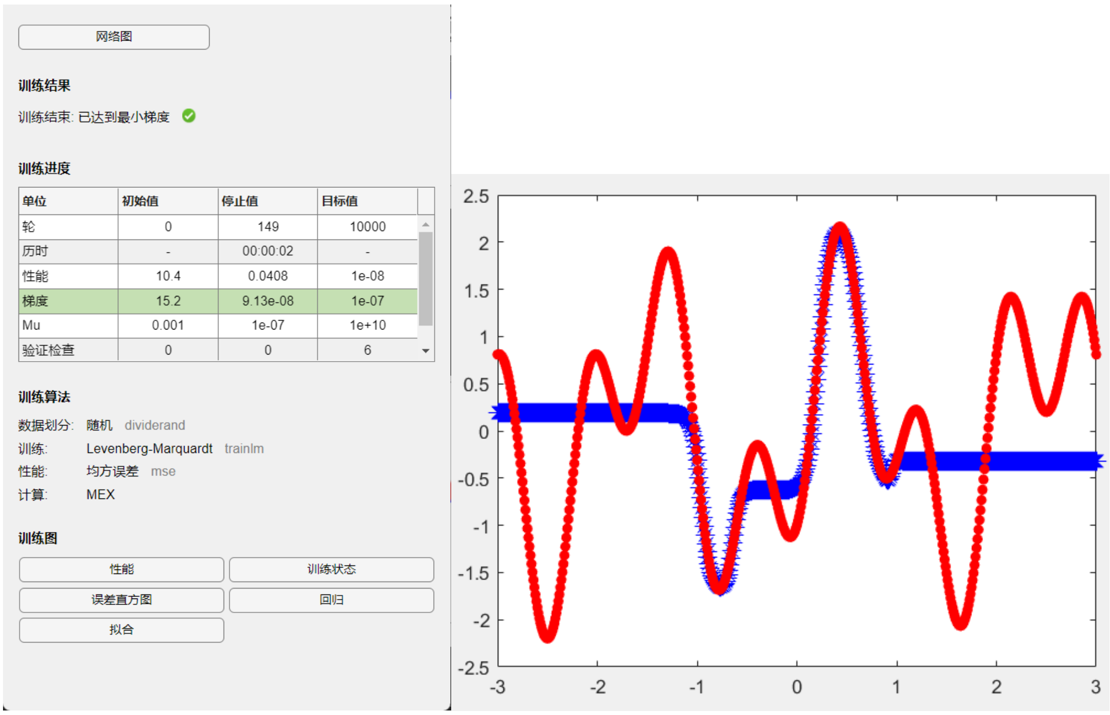
Hidden neurons = 20

Neurons	2	3	4	5	6	7	8	9	10
Performances	U	U	U	U	U	good	O	O	O
Neurons	20	30	40	50					
Performances	O	O	O	O					

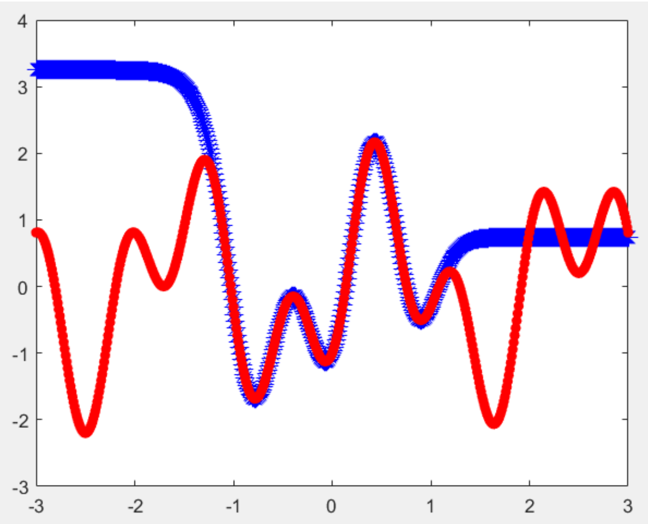
U=under-fitting; O=over-fitting.

The minimal number of hidden neurons should be 7. The number smaller than 7 caused under-fitting. And the number larger than 7 caused over-fitting. MLP could not predict the results outside the training domain. The number of neurons was consistent with the minimal wavy structure, which was in line with slides.

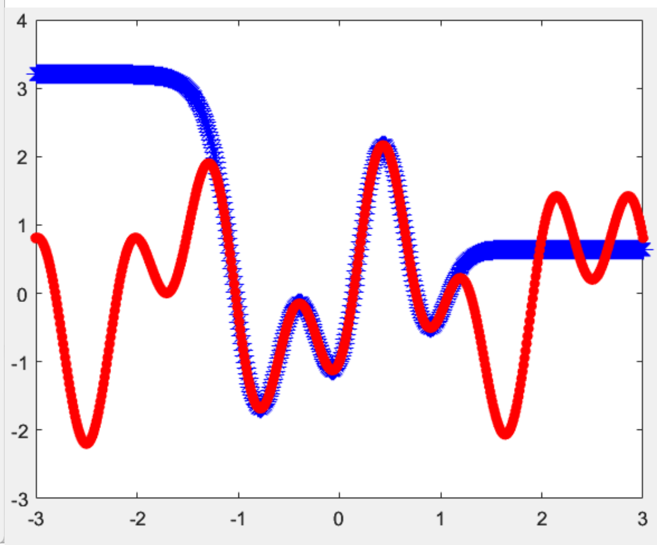
b) Trainlm mode/batch mode



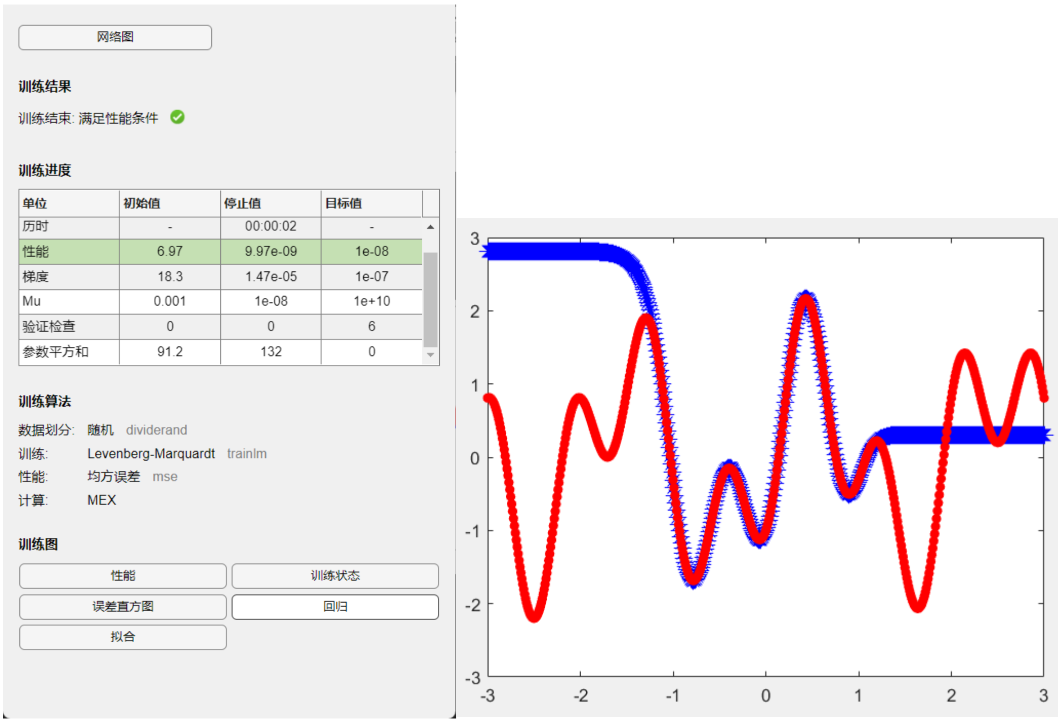
Trainlm mode: Hidden neurons = 5



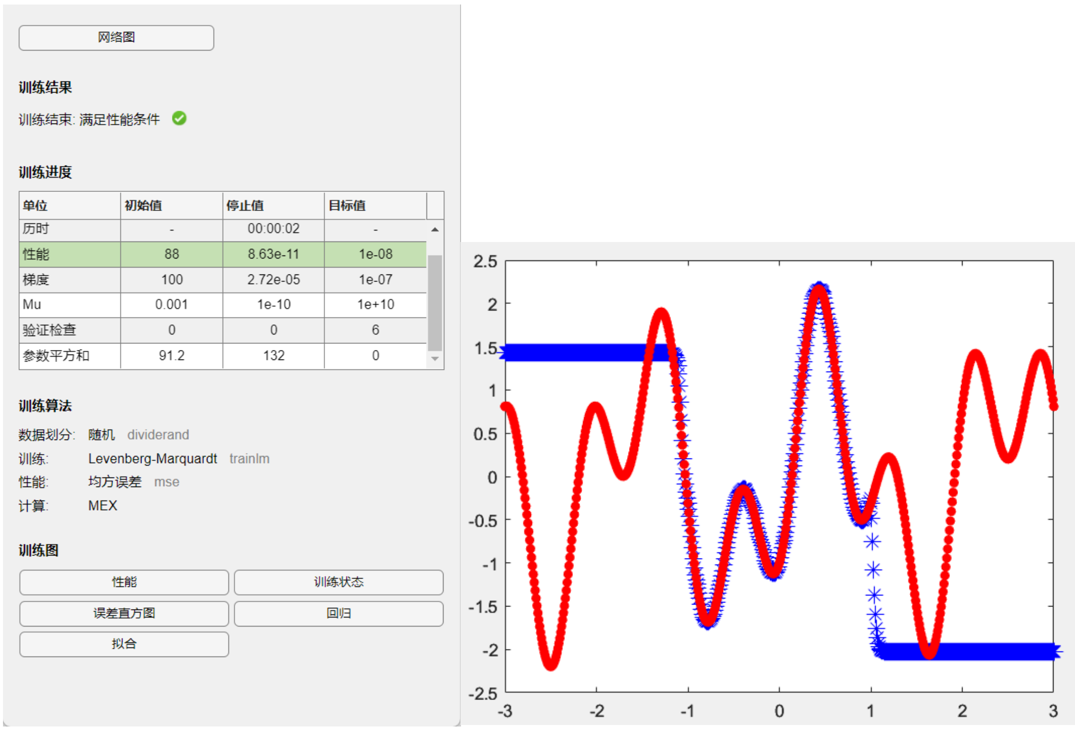
Trainlm mode: Hidden neurons = 6



Trainlm mode: Hidden neurons = 7



Trainlm mode: Hidden neurons = 10



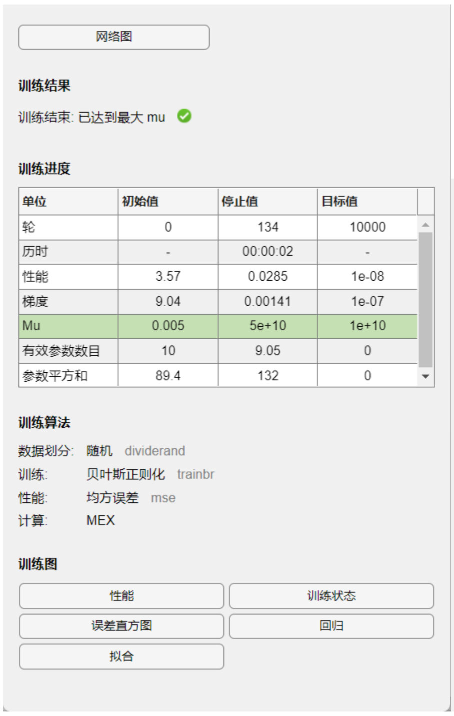
Trainlm mode: Hidden neurons = 20

Neurons	2	3	4	5	6	7	8	9	10
Performances	U	U	U	U	good	good	good	good	good
Neurons	20	30	40	50					
Performances	O	O	O	O					

U=under-fitting; O=over-fitting.

The minimal number of neurons working in the batch + trainlm mode was 6.

c) Trainbr mode/batch mode


网络图

训练结果

训练结束: 已达到最大 mu

训练进度

单位	初始值	停止值	目标值
轮	0	134	10000
历时	-	00:00:02	-
性能	3.57	0.0285	1e-08
梯度	9.04	0.00141	1e-07
Mu	0.005	5e+10	1e+10
有效参数数目	10	9.05	0
参数平方和	89.4	132	0

训练算法

数据划分: 随机 dividerand

训练: 贝叶斯正则化 trainbr

性能: 均方误差 mse

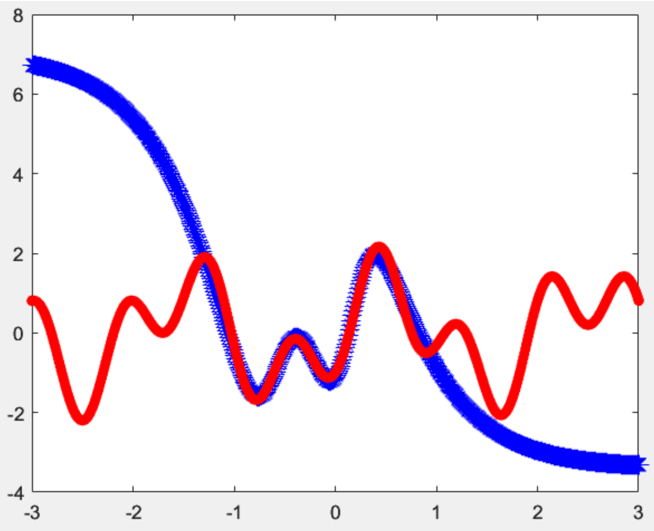
计算: MEX

训练图

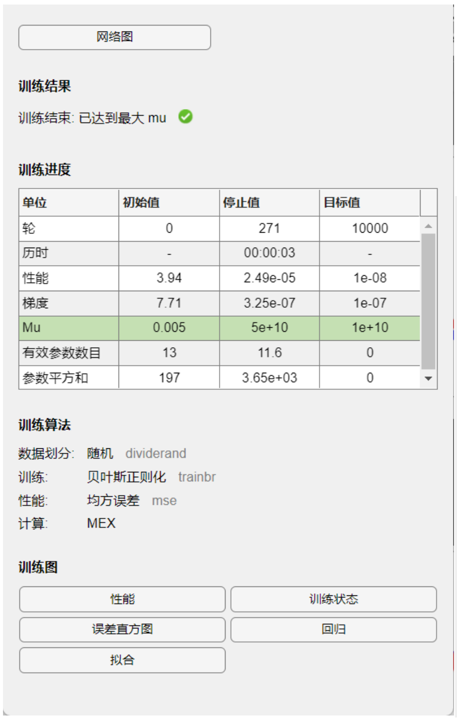
性能 训练状态

误差直方图 回归

拟合



Trainbr mode: Hidden neurons = 3


网络图

训练结果

训练结束: 已达到最大 mu

训练进度

单位	初始值	停止值	目标值
轮	0	271	10000
历时	-	00:00:03	-
性能	3.94	2.49e-05	1e-08
梯度	7.71	3.25e-07	1e-07
Mu	0.005	5e+10	1e+10
有效参数数目	13	11.6	0
参数平方和	197	3.65e+03	0

训练算法

数据划分: 随机 dividerand

训练: 贝叶斯正则化 trainbr

性能: 均方误差 mse

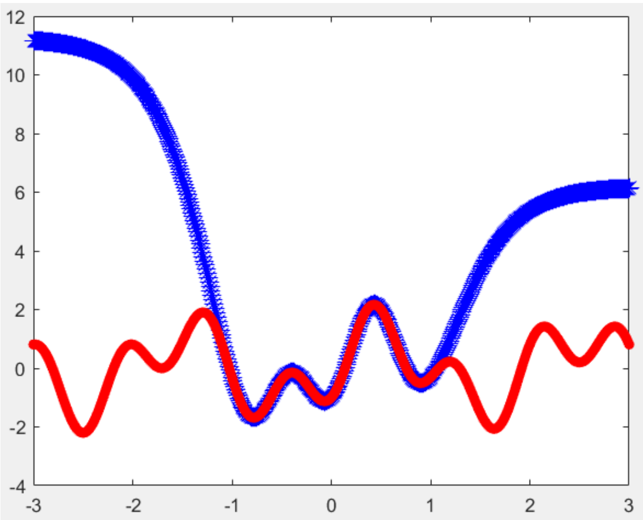
计算: MEX

训练图

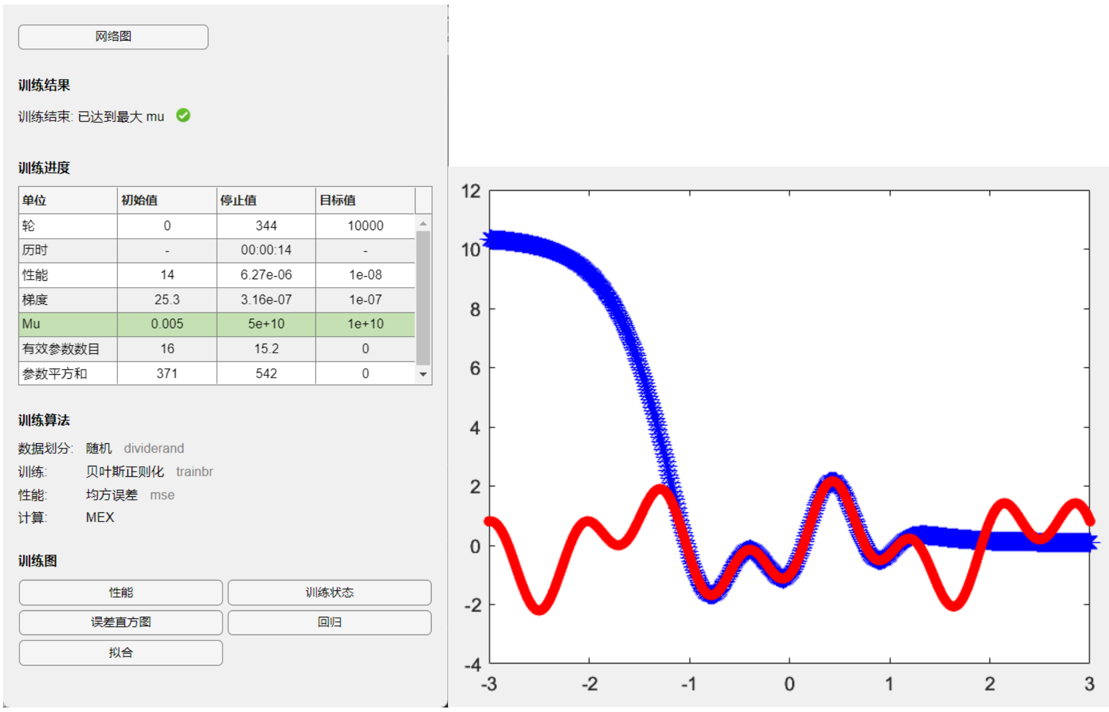
性能 训练状态

误差直方图 回归

拟合

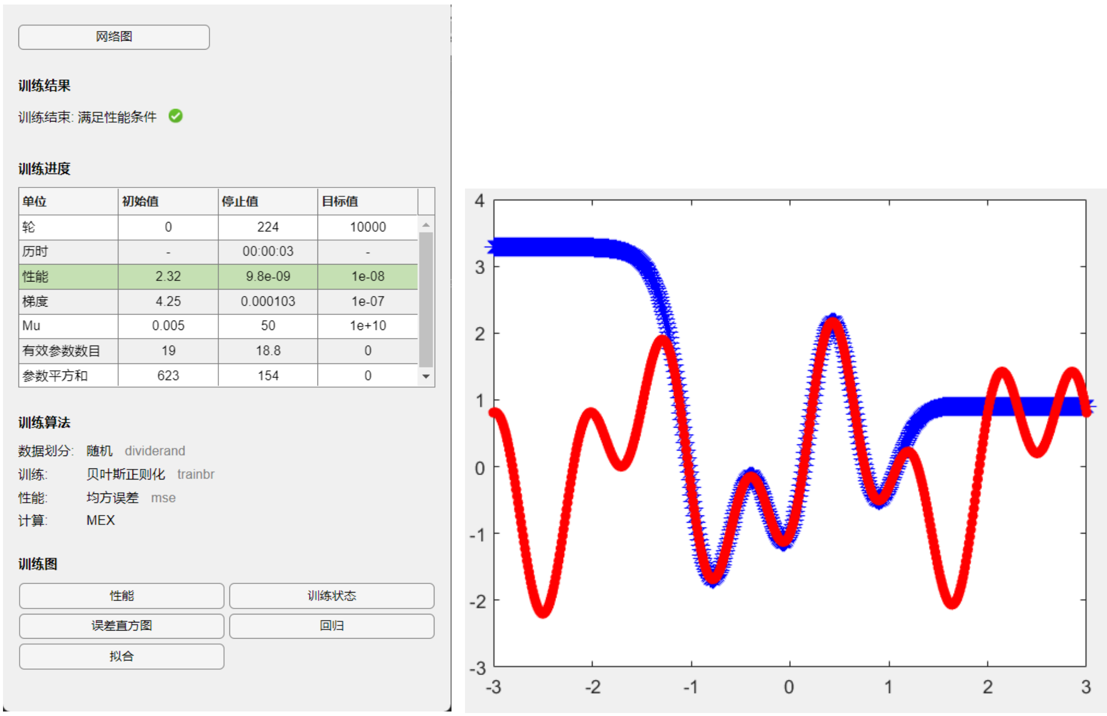


Trainbr mode: Hidden neurons = 4



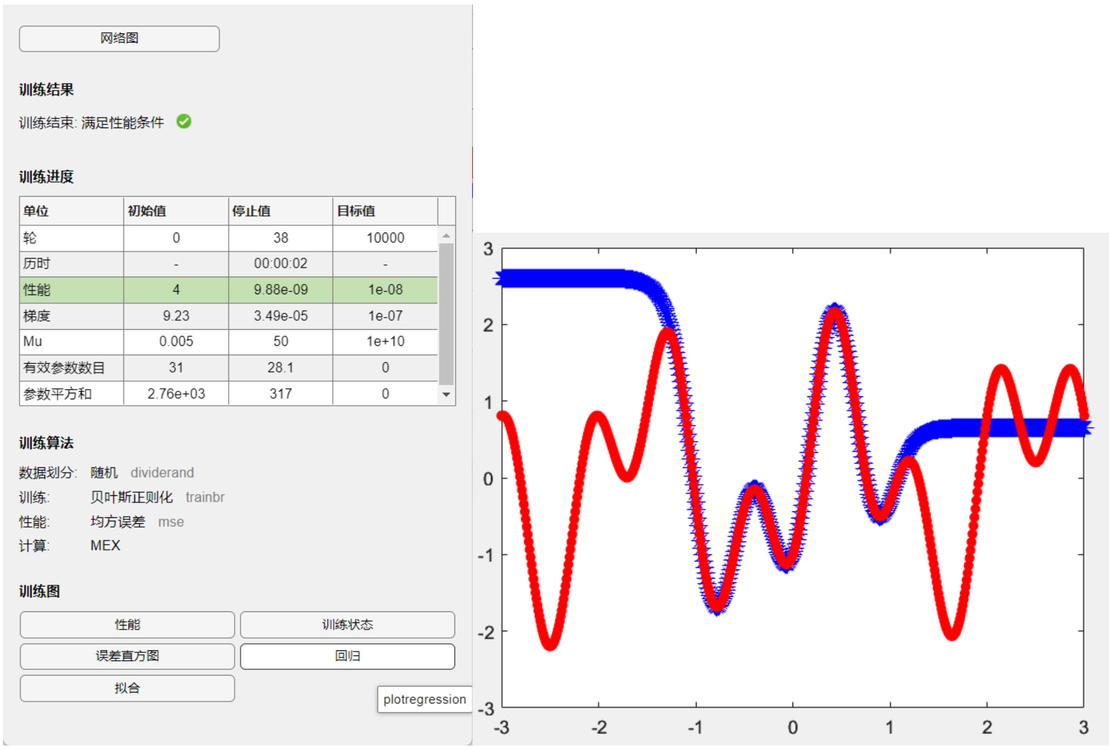
单位	初始值	停止值	目标值
轮	0	344	10000
历时	-	00:00:14	-
性能	14	6.27e-06	1e-08
梯度	25.3	3.16e-07	1e-07
Mu	0.005	5e+10	1e+10
有效参数数目	16	15.2	0
参数平方和	371	542	0

Trainbr mode: Hidden neurons = 5

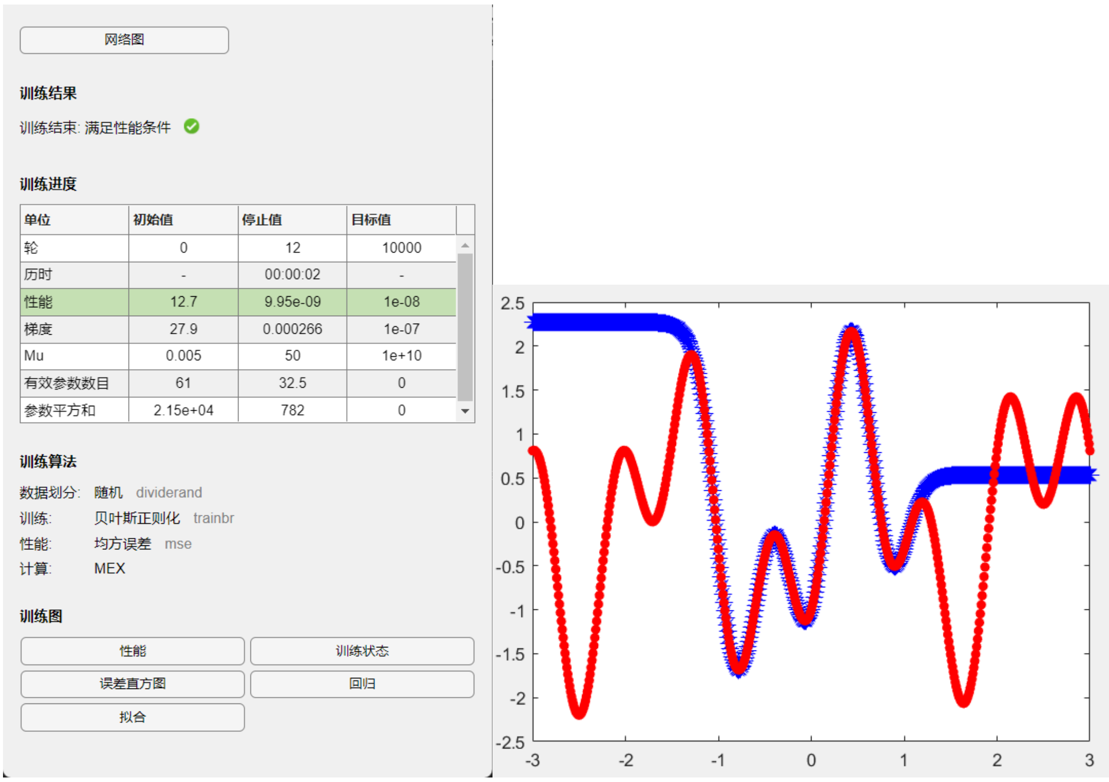


单位	初始值	停止值	目标值
轮	0	224	10000
历时	-	00:00:03	-
性能	2.32	9.8e-09	1e-08
梯度	4.25	0.000103	1e-07
Mu	0.005	50	1e+10
有效参数数目	19	18.8	0
参数平方和	623	154	0

Trainbr mode: Hidden neurons = 6



Trainbr mode: Hidden neurons = 10



Trainbr mode: Hidden neurons = 20

Neurons	2	3	4	5	6	7	8	9	10
Performances	U	U	good	good	good	good	good	good	good
Neurons	20	30	40	50					
Performances	good	good	good	good					

U= under-fitting.

Trainbr mode plus batch mode would help improve the fitting performance when the number of neurons was larger than 3. There were no obvious under-fitting or over-fitting problems when neurons number was larger than 3. The minimal neuron number is 4.

Q3:

a)

Rosenblatt's perceptron

```
============
============test
Accuracy 0.65868
there are 17 successful predictions of man-made scenes in 167 input cases
there are 93 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.67992
there are 118 successful predictions of man-made scenes in 503 input cases
there are 224 successful predictions of natural secenes in 503 input cases
============
```

For the testing dataset, there are only 110 successful predictions or classifications, and the accuracy is only 65%. For the training dataset, there are only 342 successful predictions or classifications, and the accuracy is only 67%. The performance was not very good and the processing time was very long. In this case , I already had applied SVD algorithm to help process the image.

b)

When I applied the SVD compressing algorithm and shrieked the image size to 128*128, the performances were best. Perceptron trained from 32*32 images almost couldn't distinguish two scenes.

svd() + Resizing	128*128	64*64	32*32
Successful predictions for test	117	110	135
Testing accuracy	70%	66%	60%
Epochs	50	70	70
	Picture 1	Picture 2	Picture 3

```
============
============test
Accuracy 0.7006
there are 27 successful predictions of man-made scenes in 167 input cases
there are 90 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.63022
there are 105 successful predictions of man-made scenes in 503 input cases
there are 212 successful predictions of natural secenes in 503 input cases
============
```

Picture 1

```
============
============test
Accuracy 0.65868
there are 19 successful predictions of man-made scenes in 167 input cases
there are 91 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.60437
there are 100 successful predictions of man-made scenes in 503 input cases
there are 204 successful predictions of natural secenes in 503 input cases
============
```

Picture 2

```
============
============test
Accuracy 0.60479
there are 1 successful predictions of man-made scenes in 167 input cases
there are 100 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.58847
there are 96 successful predictions of man-made scenes in 503 input cases
there are 200 successful predictions of natural secenes in 503 input cases
============
```

Picture 3

c)

Applying the svd() algorithm and resizing the image to help improve the performance. The energy threshold was set to be 0.9 in the svd() configuration. The regularization parameter was set to be 0.1. I followed the principle that in the case of similar performances, I chose the minimal number of neurons.

svd() + Resizing	128*128	64*64	32*32
Successful predictions for test	135	137	135
Testing accuracy	80%	82%	80%
Neurons	8	8	8
Epochs	1500	1500	1500
	Picture 4	Picture 5	Picture 6

```
============test
Accuracy 0.80838
there are 47 successful predictions of man-made scenes in 167 input cases
there are 88 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.99801
there are 195 successful predictions of man-made scenes in 503 input cases
there are 307 successful predictions of natural secenes in 503 input cases
============
```

Picture 4

```
============test
Accuracy 0.82036
there are 49 successful predictions of man-made scenes in 167 input cases
there are 88 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.99602
there are 193 successful predictions of man-made scenes in 503 input cases
there are 308 successful predictions of natural secenes in 503 input cases
============
```

Picture 5

```
============test
Accuracy 0.80838
there are 50 successful predictions of man-made scenes in 167 input cases
there are 85 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.99602
there are 193 successful predictions of man-made scenes in 503 input cases
there are 308 successful predictions of natural secenes in 503 input cases
============
```

Picture 6

When I compressed the original images, it seemed to have no ill effects on the accuracy, but at the same time improved the training speed. So, it was true that the original data contained much redundant information which should be omitted in the following processing. For the training dataset, the accuracy had reached almost 100%.

d) When the neuron number was set to be larger than 8 or the epoch number is larger than 1500, the classification efficiency decreased and these changes didn't contribute to the improvement of the performances. When I applied the regularization parameter (0.1), the performances improved a little. So, in the group 1 assignment, it could be a challenge for MLP to accurately classify two scenes.

Best performances	neurons	regularization	epochs	accuracy
128*128 (0.9 SVD)	8	0.1	1500	82%

e)

Sequential mode took quite a long time for each epoch because of its intrinsic large calculations. So, the epoch number was limited to a small number. The performances were quite good because of its successful predictions in both scenes. However, when it was compared to the results got from batch model, it was inferior. The prediction accuracy was always around 77%.

My recommendations: We should increase the samples in the dataset to give more information to machine. We should also improve the computer's GPU or CPU to get the faster results and do more calculations.

svd() + Resizing	256*256	128*128	64*64
Successful predictions in testing	125	127	128
Testing accuracy	75%	76%	77%

Successful predictions in training	386	412	425
Training accuracy	77%	82%	84%
Neurons	8	8	8
Epochs	10	20	40
	Picture 7	Picture 8	Picture 9

```
============test
Accuracy 0.7485
there are 41 successful predictions of man-made scenes in 167 input cases
there are 84 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.7674
there are 139 successful predictions of man-made scenes in 503 input cases
there are 247 successful predictions of natural secenes in 503 input cases
============
```

Picture 7

```
============test
Accuracy 0.76048
there are 45 successful predictions of man-made scenes in 167 input cases
there are 82 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.81909
there are 155 successful predictions of man-made scenes in 503 input cases
there are 257 successful predictions of natural secenes in 503 input cases
============
```

Picture 8

```
============test
Accuracy 0.76647
there are 45 successful predictions of man-made scenes in 167 input cases
there are 83 successful predictions of natural secenes in 167 input cases
============train
Accuracy 0.84493
there are 160 successful predictions of man-made scenes in 503 input cases
there are 265 successful predictions of natural secenes in 503 input cases
============
```

Picture 9

f)

1. We need to collect more data and more pictures. Some specific scenes are difficult for machine to detect. It need more experience.
2. We need to magnify the output categories. Man-made scenes and natural scenes could still be divided into many tiny and specific scenes. It would help to improve the accuracy and predictions.

Attachment1

```
clear all
clc

% Define the Rosenbrock function
```

```
rosenbrock = @(x,y) 100 * (y - x^2)^2 + (1 - x)^2;

% Define the gradient of the Rosenbrock function
rosenbrock_grad = @(x,y) [-400*x*(y-x^2)-2*(1-x); 200*(y-x^2)];

% Set the initial point
x=0;
y=0.5;

% Set the tolerance level
tol = 1e-5;

% Set the step size (also known as the learning rate)
alpha = 0.001;

% Initialize the iteration counter
k = 1;

% Initialize the trajectory matrix
x_trajectory=[];
y_trajectory=[];
z_trajectory=[];

% Start the steepest descent algorithm
while rosenbrock(x,y) > tol

    % Record the trajectory
    x_trajectory(k,1) = x;
    y_trajectory(k,1) = y;
    z_trajectory(k,1) = rosenbrock(x,y);

    % Compute the gradient
    p = rosenbrock_grad(x,y);

    % Update the current point
    x = x - alpha*p(1);
    y = y - alpha*p(2);

    % Increment the iteration counter
    k = k + 1;

end
    % Record the last trajectory
    x_trajectory(k,1) = x;
```

```
    y_trajectory(k,1) = y;
    z_trajectory(k,1) = rosenbrock(x,y);

% Plot the trajectories
figure;
plot3(x_trajectory,y_trajectory,z_trajectory,'o', 'MarkerSize', 6,
'MarkerFaceColor', 'red','MarkerEdgeColor', 'red');
grid on;
hold on;

% Plot the valley function
[m,n] = meshgrid(-1:0.1:1.5);
q = (1-m).^2 + 100*(n-m.^2).^2;
surf(m,n,q);
view(110,50);

% Set the figure
xlabel('x');
ylabel('y');
zlabel('z');
title('Trajectories of x y z');

% Plot the x y in 2D
figure;
plot(x_trajectory,y_trajectory, '-o');
xlabel('x');
ylabel('y');
title('Trajectories of x y');

% Display the minimum value and the number of iterations
fprintf('Minimum value: %f\n', rosenbrock(x,y));
fprintf('Number of iterations: %d\n', k);
```

Attachment2

```
clear all
clc

% Define the Rosenbrock function
rosenbrock = @(x,y) 100 * (y - x^2)^2 + (1 - x)^2;

% Define the gradient of the Rosenbrock function
rosenbrock_grad = @(x,y) [-400*x*(y-x^2)-2*(1-x); 200*(y-x^2)];
```

```
% Define the Hessian matrix of the Rosenbrock function
rosenbrock_hess = @(x,y) [1200*x^2 - 400*y + 2, -400*x; -400*x, 200];

% Set the initial point
x=0;
y=0.5;

% Set the tolerance level
tol = 1e-5;

% Initialize the iteration counter
k = 1;

% Initialize the trajectory matrix
x_trajectory=[];
y_trajectory=[];
z_trajectory=[];

% Start the Newton method algorithm
while rosenbrock(x,y) > tol

    % Record the trajectory
    x_trajectory(k,1) = x;
    y_trajectory(k,1) = y;
    z_trajectory(k,1) = rosenbrock(x,y);

    % Compute the search direction using the inverse of the Hessian matrix
    p = inv(rosenbrock_hess(x,y))*rosenbrock_grad(x,y);

    % Update the current point
    x = x - p(1);
    y = y- p(2);

    % Increment the iteration counter
    k = k + 1;
end

    % Record the last trajectory
    x_trajectory(k,1) = x;
    y_trajectory(k,1) = y;
    z_trajectory(k,1) = rosenbrock(x,y);

% Plot the trajectories
```

```
figure;
plot3(x_trajectory,y_trajectory,z_trajectory,'o', 'MarkerSize', 6,
'MarkerFaceColor', 'red','MarkerEdgeColor', 'red');
grid on;
hold on;

% Plot the valley function
[m,n] = meshgrid(-1:0.1:1.5);
q = (1-m).^2 + 100*(n-m.^2).^2;
surf(m,n,q);
view(110,50);

% Set the figure
xlabel('x');
ylabel('y');
zlabel('z');
title('Trajectories of x y z');

% Plot the x y in 2D
figure;
plot(x_trajectory,y_trajectory, '-o');
xlabel('x');
ylabel('y');
title('Trajectories of x y');

% Display the minimum value and the number of iterations
fprintf('Minimum value: %f\n', rosenbrock(x,y));
fprintf('Number of iterations: %d\n', k);
```

Attachment3

```
clear all
clc

load('traindata.mat');
load('testdata.mat');
load('testdata2.mat');

% Construct and configure the MLP
epochs = 1000;
train_num = size(traindata,1);

% Set the train dataset
```

```
x=traindata(:,1)';
t=traindata(:,2)';

% specify the structure and learning algorithm for MLP
net = fitnet(20,'trainlm');
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'purelin';
net = configure(net,x,t);
view(net);


 % Train the network in sequential mode
 for i = 1 : epochs
 display(['Epoch: ', num2str(i)])
 idx = randperm(train_num); % shuffle the input
 net = adapt(net, x(:,idx), t(:,idx));
 end

% Generate the test input data
input=testdata2(:,1)';
desiredout=testdata2(:,2)';

% Feed the input
pred = net(input);
perf = perform(net, desiredout, pred);

% Plot the fitting results and compare
figure;
plot(input',pred','*','MarkerFaceColor', 'b', 'MarkerEdgeColor',
'b','MarkerSize',10);% Plot the fitting dataset
hold on;
plot(testdata2(:,1),testdata2(:,2),'o','MarkerFaceColor', 'r',
'MarkerEdgeColor', 'r','MarkerSize',5);% Plot the original test dataset
```

Attachment4

```
clear all
clc

load('traindata.mat');
load('testdata.mat');
load('testdata2.mat');

% Set the train dataset
```

```
x=traindata(:,1)';
t=traindata(:,2)';

% Specify the structure and learning algorithm for MLP
net = fitnet(7,'trainbr');
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'purelin';
net = configure(net,x,t);
net.trainparam.lr=0.01;
net.trainparam.epochs=10000;
net.trainparam.goal=1e-8;
net.divideParam.trainRatio=1.0;
net.divideParam.valRatio=0.0;
net.divideParam.testRatio=0.0;

% Train the network
net = train(net, x, t);
view(net)

% Generate the test input data
input=testdata2(:,1)';
desiredout=testdata2(:,2)';

% Feed the input
pred = net(input);
perf = perform(net, desiredout, pred);

% Plot the fitting results and compare
figure;
plot(input',pred','*','MarkerFaceColor', 'b', 'MarkerEdgeColor',
'b','MarkerSize',10);% Plot the fitting dataset
hold on;
plot(testdata2(:,1),testdata2(:,2),'o','MarkerFaceColor', 'r',
'MarkerEdgeColor', 'r','MarkerSize',5);% Plot the original test dataset
```

Attachment5

```
clear all
clc

% Read directory
files1=dir('C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\train');
folder1 = 'C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\train';
```

```
% Define the magnifying/shrinking coefficent
coeff = 0.125;

% Read directory
files2=dir('C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\test');
folder2 = 'C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\test';

% Read images and processing
[train_images,trainDesired]=loadimage(files1,folder1,coeff);
[test_images,testDesired]=loadimage(files2,folder2,coeff);

% Initial weights
w = rand(size(train_images,2),1);
b = rand();
train_pred = rand(1,length(files1)-2);

% Learning rate
eta =5;

% Counting number
numi=0;

while true
    % Update the weights
    for i = 1:length(files1)-2
        train_pred(i) = sign(train_images(i,:)*w + b);
        if (train_pred(i) ~= trainDesired(i))
            w = w + eta * (trainDesired(i) - train_pred(i)) *
train_images(i,:)';
            b = b + eta * (trainDesired(i) - train_pred(i));
        end

    end

    % make the judgement
    if isequal(train_pred,trainDesired)
        break;
    elseif numi > 70
        break;
    end

    % Trajectories
    numi=numi+1;
    disp(numi);
```

```
end

disp("============");


% Identify the successful predictions of input
for k = 1:length(files2)-2
    test_pred_output(k) = sign(test_images(k,:)*w + b);
end

disp("============test");
calacc_classification(testDesired,test_pred_output,length(files2)-2);
disp("============train");
calacc_classification(trainDesired,train_pred,length(files1)-2);
disp("============");

function calacc_classification(Desired,pred_output,length)

% Record the successful prediction number
k1=0;
k2=0;

% Identify the successful predictions of input
for i = 1:length
    if pred_output(i)==0
        if Desired(i) == pred_output(i)
            k1=k1+1;
        end
    end
    if pred_output(i)==1
        if Desired(i) == pred_output(i)
            k2=k2+1;
        end
    end
end

% Calculate the accuracy and Display the accuracy
accuracy = (k1+k2)/length;
Z=['Accuracy ',num2str(accuracy)];
X=['there are ',num2str(k1), ' successful predictions of man-made scenes in
',num2str(length),' input cases'];
Y=['there are ',num2str(k2), ' successful predictions of natural secenes in
',num2str(length),' input cases'];
```

```
disp(Z);
disp(X);
disp(Y);
end

function [output]=sign(x)

    if x<0
        output=0;
    elseif x>0
        output=1;
    else
        output=0.5;
    end
end

function [I_reconstructed] = svd_process(I)

    % Perform SVD on the image
    [U, S, V] = svd(I);

    % Calculate the total energy of the singular values
    energy = sum(diag(S).^2);

    % Calculate the cumulative energy of the singular values
    cumulative_energy = cumsum(diag(S).^2);

    % Calculate the percentage of the information to keep
    threshold = 0.9;
    n = find(cumulative_energy >= threshold*energy, 1, 'first');

    % Use only the first n singular values to reconstruct the image
    I_reconstructed = U(:,1:n) * S(1:n,1:n) * V(:,1:n)';

end

function [image,label]=loadimage(files,folder,coeff)
for i=3:length(files)
    % Get the label
    tmp1 = strsplit(files(i).name, {'_', '.'});
    label(i-2)= str2num(tmp1{2});
    % Read train images and store
    filename = fullfile(folder, files(i).name);
    % Change the format
```

```
    images = double(imread(filename));
    % Compress the pictures
    resizeImage = imresize(images,coeff);
    % SVD compressing
    svdimage=svd_process(resizeImage);
    % Put the data into columns
    V = svdimage(:);
    image(i-2,:)=V;
end
end
```

Attachment6

```
clear all
clc

% Read directory
files1=dir('C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\train');
folder1 = 'C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\train';

% Define the magnifying/shrinking coefficent
coeff = 0.125;

% Read directory
files2=dir('C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\test');
folder2 = 'C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\test';

% Read images and processing
[train_images,trainDesired]=loadimage(files1,folder1,coeff);
[test_images,testDesired]=loadimage(files2,folder2,coeff);


% Set the epoch number
epochs = 2000;

% Construct and configure the MLP
net = patternnet(8);
net.divideFcn = 'dividetrain'; % input for training only
net.performParam.regularization = 0; % regularization strength
net.trainFcn = 'traingdx'; % 'trainrp' 'traingdx'
net.trainParam.epochs = epochs;


% Training the network
net=train(net,train_images,trainDesired);
```

```matlab
view(net);

% Feed the testing or the training images
test_pred_output = round(net(test_images));
train_pred_ouput = round(net(train_images));

disp("============test");
calacc_classification(testDesired,test_pred_output,length(files2)-2);
disp("============train");
calacc_classification(trainDesired,train_pred_ouput,length(files1)-2);
disp("============");

function calacc_classification(Desired,pred_output,length)

% Record the successful prediction number
k1=0;
k2=0;

% Identify the successful predictions of input
for i = 1:length
    if pred_output(i)==0
        if Desired(i) == pred_output(i)
            k1=k1+1;
        end
    end
    if pred_output(i)==1
        if Desired(i) == pred_output(i)
            k2=k2+1;
        end
    end
end

% Calculate the accuracy and Display the accuracy
accuracy = (k1+k2)/length;
Z=['Accuracy ',num2str(accuracy)];
X=['there are ',num2str(k1), ' successful predictions of man-made scenes in
',num2str(length),' input cases'];
Y=['there are ',num2str(k2), ' successful predictions of natural secenes in
',num2str(length),' input cases'];
disp(Z);
disp(X);
disp(Y);
end
```

```
function [I_reconstructed] = svd_process(I)

    % Perform SVD on the image
    [U, S, V] = svd(I);

    % Calculate the total energy of the singular values
    energy = sum(diag(S).^2);

    % Calculate the cumulative energy of the singular values
    cumulative_energy = cumsum(diag(S).^2);

    % Calculate the percentage of the information to keep
    threshold = 0.9;
    n = find(cumulative_energy >= threshold*energy, 1, 'first');

    % Use only the first n singular values to reconstruct the image
    I_reconstructed = U(:,1:n) * S(1:n,1:n) * V(:,1:n)';

end

function [image,label]=loadimage(files,folder,coeff)
for i=3:length(files)
    % Get the label
    tmp1 = strsplit(files(i).name, {'_', '.'});
    label(1,i-2)= str2num(tmp1{2});
    % Read train images and store
    filename = fullfile(folder, files(i).name);
    % Change the format
    images = double(imread(filename));
    % Compress the pictures
    resizeImage = imresize(images,coeff);
    % SVD compressing
    svdimage=svd_process(resizeImage);
    % Put the data into columns
    V = svdimage(:);
    image(:,i-2)=V;
end
end
```

Attachment7

```
clear all
clc
```

```matlab
% Read directory
files1=dir('C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\train');
folder1 = 'C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\train';

% Define the magnifying/shrinking coefficent
coeff = 0.25;

% Read directory
files2=dir('C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\test');
folder2 = 'C:\Users\sunsh\Desktop\tem\tem matlab photo\group_1\test';

% Read images and processing
[train_images,trainDesired]=loadimage(files1,folder1,coeff);
[test_images,testDesired]=loadimage(files2,folder2,coeff);


epochs = 40;
train_num = length(files1)-2;


% 1. Change the input to cell array form for sequential training
 images_c = num2cell(train_images, 1);
 labels_c = num2cell(trainDesired, 1);

 % 2. Construct and configure the MLP
 net = patternnet(8);
 net.divideFcn = 'dividetrain'; % input for training only
 net.performParam.regularization = 0.1; % regularization strength
 net.trainFcn = 'traingdx'; % 'trainrp' 'traingdx'
 net.trainParam.epochs = epochs;


 % 3. Train the network in sequential mode
 for i = 1 : epochs

 display(['Epoch: ', num2str(i)])

 idx = randperm(train_num); % shuffle the input

 net = adapt(net, images_c(:,idx), labels_c(:,idx));

 end

% Feed the testing or the training images
```

```
test_pred_output = round(net(test_images));
train_pred_ouput = round(net(train_images));

disp("============test");
calacc_classification(testDesired,test_pred_output,length(files2)-2);
disp("============train");
calacc_classification(trainDesired,train_pred_ouput,length(files1)-2);
disp("============");

function calacc_classification(Desired,pred_output,length)

% Record the successful prediction number
k1=0;
k2=0;

% Identify the successful predictions of input
for i = 1:length
    if pred_output(i)==0
        if Desired(i) == pred_output(i)
            k1=k1+1;
        end
    end
    if pred_output(i)==1
        if Desired(i) == pred_output(i)
            k2=k2+1;
        end
    end
end

% Calculate the accuracy and Display the accuracy
accuracy = (k1+k2)/length;
Z=['Accuracy ',num2str(accuracy)];
X=['there are ',num2str(k1), ' successful predictions of man-made scenes in
',num2str(length),' input cases'];
Y=['there are ',num2str(k2), ' successful predictions of natural secenes in
',num2str(length),' input cases'];
disp(Z);
disp(X);
disp(Y);
end

function [I_reconstructed] = svd_process(I)

    % Perform SVD on the image
```

```
    [U, S, V] = svd(I);

    % Calculate the total energy of the singular values
    energy = sum(diag(S).^2);

    % Calculate the cumulative energy of the singular values
    cumulative_energy = cumsum(diag(S).^2);

    % Calculate the percentage of the information to keep
    threshold = 0.9;
    n = find(cumulative_energy >= threshold*energy, 1, 'first');

    % Use only the first n singular values to reconstruct the image
    I_reconstructed = U(:,1:n) * S(1:n,1:n) * V(:,1:n)';

end

function [image,label]=loadimage(files,folder,coeff)
for i=3:length(files)
    % Get the label
    tmp1 = strsplit(files(i).name, {'_', '.'});
    label(1,i-2)= str2num(tmp1{2});
    % Read train images and store
    filename = fullfile(folder, files(i).name);
    % Change the format
    images = double(imread(filename));
    % Compress the pictures
    resizeImage = imresize(images,coeff);
    % SVD compressing
    svdimage=svd_process(resizeImage);
    % Put the data into columns
    V = svdimage(:);
    image(:,i-2)=V;
end
end
```