Task8: Neural Network

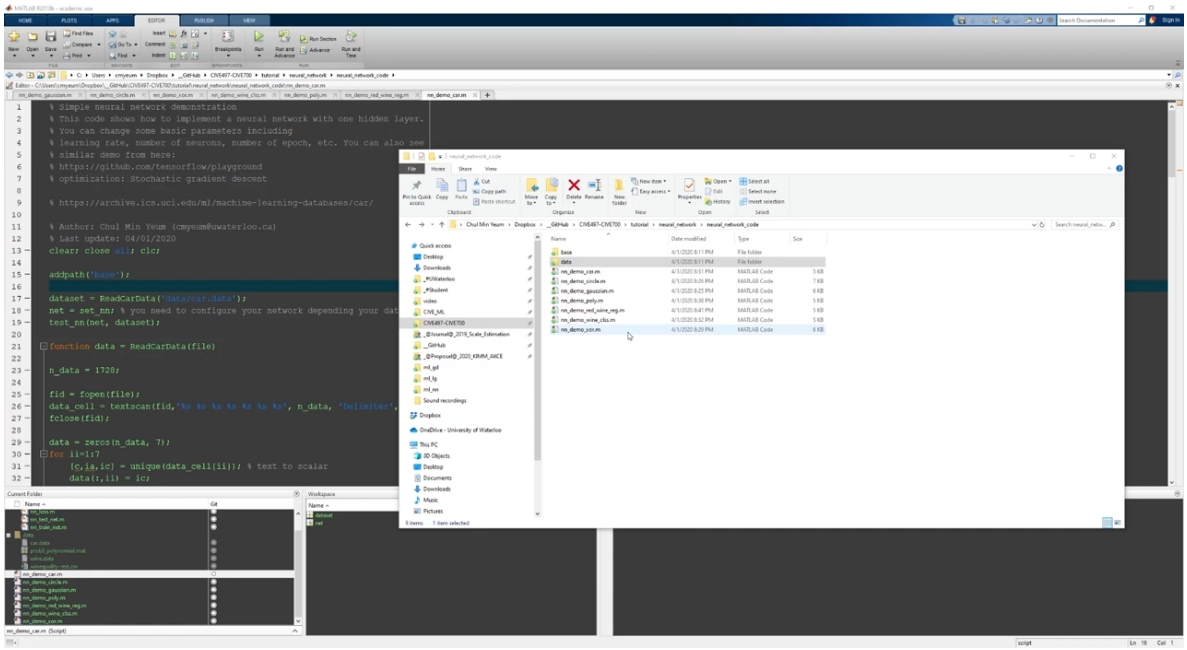
Name: Saeed Hatefi Ardakani
Degree: PH
ID: 20793159

Problem 1: (40 points)

(a) Try out a neural network playground in this link: http://playground.tensorflow.org. Run all four examples by changing features, # of hidden layers, # of neurons, activation function, and/or learning rate. Please watch a convergence speed of the loss depending on the parameter changes that you made. (You do not need to include your results in your report).

Done!

(b) Please run all demos in the tutorial. I recommend that you configure parameters to improve the results. Please watch my review video.



video download

(Sorry, I used a wrong microphone so the audio quality is not good).

In your report, briefly explain (1) an objective of each sample demo, (2) what change I made in default codes or you made (if you configure the network), and (3) a screenshot of the final plot. Basically, you introduce your model and its performance. The default parameters are the ones that I tuned but might not be the best. You can find better neural network architecture by changing # of neurons, activation functions, or loss function. You can also change the learning rate. If you achieve performance improvement, please introduce your network configuration. You can allow changing the code if you can improve the model performance.

1: "nn_demo_gaussian.m" demo:

1a) objective:

This is two-class classification neural network. The objective is classifying 2D datasets. It means that the input node has two values (x and y for each data), and then, it is introduced some neurons in the hidden layer as the mathematical functions each designed to produce an output specific to an intended result. Finally, the number of output node is 2 in this demo, meaning that the output of each data has two values (two classes). So, we can estimate the probability that the input belongs to class 1 and 2. For the final decision, the input belongs to the class of the node with the highest value/probability (softmax).

There are three steps for this code: 1) generating the data 2) setting up the network 3) testing the network.

In the first step, the dataset is generated. There are two gaussian distribution with different means (the mean value for data_set1 and data_set2 are equal to 2 and -2, respectively). In addition, it is split the data to 70% for training and 30% for testing the performance of the model. In the second step, we have training process. The training process is to find out the boundary between these two gaussian distribution in 2D by defining weights and bias in the net. Finally, we test the performance of the model by defining dataset for testing the accuracy of the network.

The optimization method used here is stochastic gradient descent method, meaning that it updates the weight and bias in the network using every single dataset.

1b) what change I made in default codes or you made (if you configure the network):

For this problem, there is no need to change the parameters because the accuracy of the results is high (0.9933), and loss function also experiences minimum state at the end of the iteration (epoch). So, the current configuration is fine for this problem:

learning rate: 0.03

number of neurons in hidden layer: 5

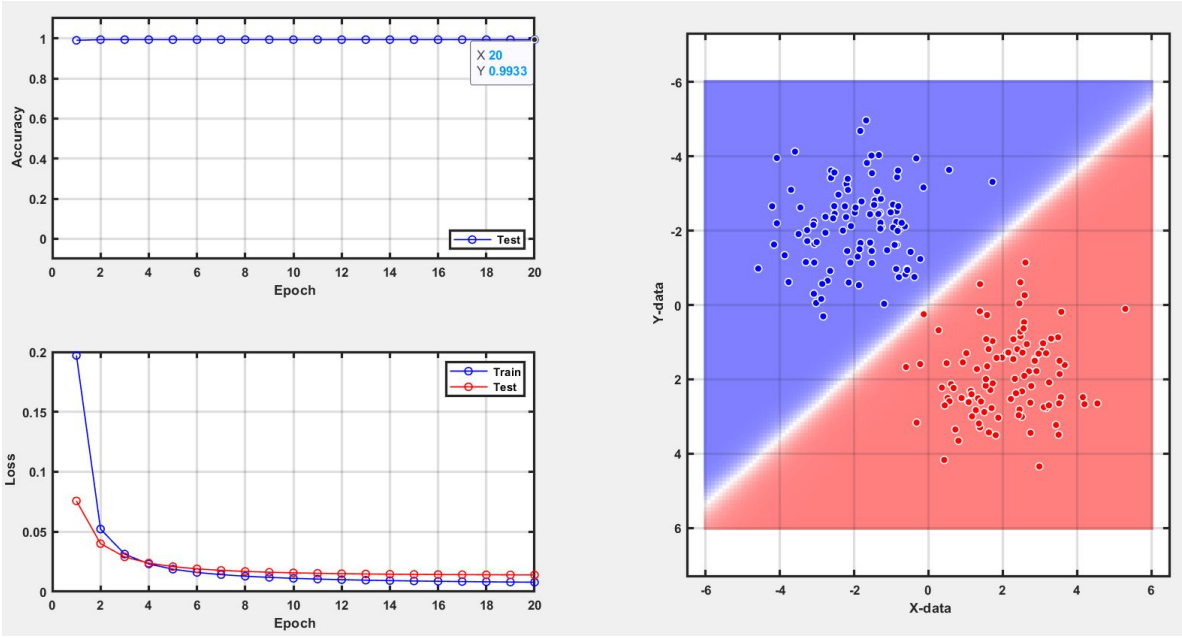
number of epoch: 20

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'

1c) screenshot of the final plot:



2: "nn_demo_circle.m" demo:

2a) objective:

This is also two-class classification neural network. The objective is classifying 2D datasets to find what data is located inside the circle while other data is in the rest of the area. It means that the input node has two values (x and y for each data), and then, it is introduced some neurons in the hidden layer as the mathematical functions each designed to produce an output specific to an intended result. Finally, the number of output node is 2 in this demo, meaning that the output of each data has two values (two classes). Class 1 is related to the data inside the circle and class 2 is related to the data outside of the circle.

So, we can estimate the probability that the input belongs to class 1 and 2. For the final decision, the input belongs to the class of the node with the highest value/probability (softmax).

Here, the neural network code is automatically finding the boundary around the data inside the circle. Here, the boundary is nonlinear. The reason that we can find a nonlinear boundary is that we are using activation function. Without activation function, this is just a linear regression, meaning that we can only find linear boundary.

Like the previous problem, there are three steps for this code: 1) generating the data 2) setting up the network 3) testing the network.

There are two types of data. One (dataset1) is the data inside the circle with the radius equal to 2. The other data (dataset2) is related to the points out of this circle. In addition, it is split the data to 85% for training and 15% for testing the performance of the model. In the second step, we have training process. The training process is to find out the nonlinear boundary around the circle data in 2D by defining weights and bias in the net. Finally, we test the performance the model by defining dataset for testing the accuracy of the network.

2b) what change I made in default codes or you made (if you configure the network):

We cannot find the good result with the default parameters because in the default code, the learning rate (0.0001) is too slow. So, we need to increase the value of epoch to improve the results.

In the below, I have plotted the result related to the default parameters. As you can see, the amount of accuracy is not enough at the end of run (epoch=30):

learning rate: 0.0001

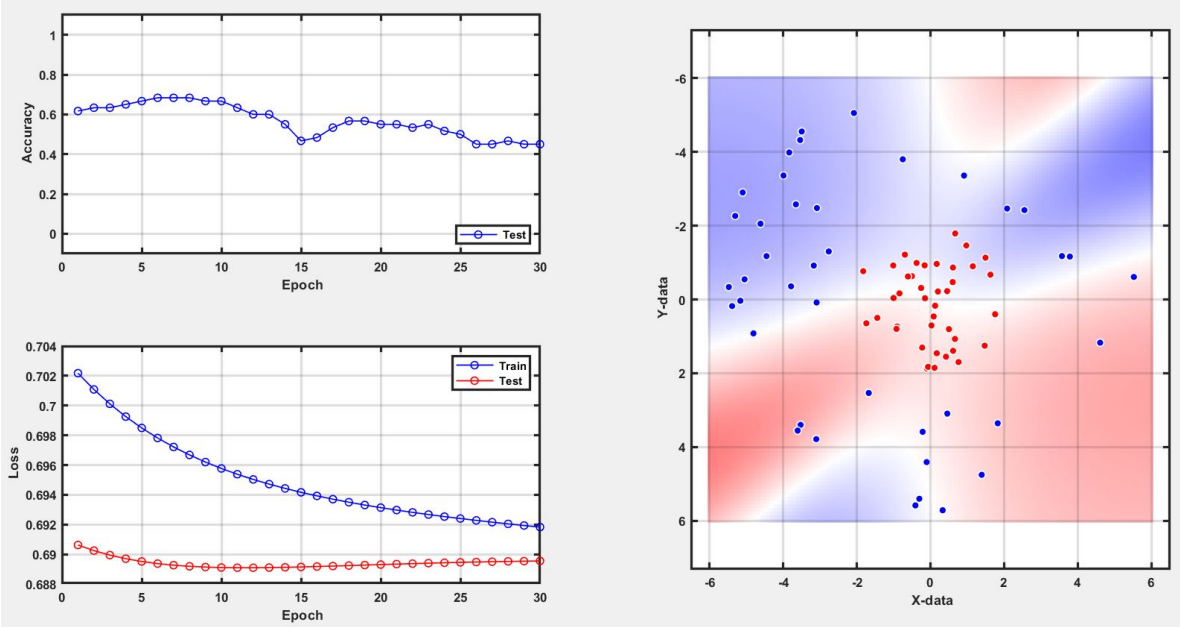
number of neurons in hidden layer: 10

number of epoch: 30

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



So, in order to improve the result, we have two options. 1: increasing the epoch value, 2: increasing the learning rate.

Option 1: Option 1 works well, but the problem is that the computational cost, run time, is so high due to the fact that we need to use too high epoch. In fact, if we use very small learning rate, the rate of converging to the local minimum of loss function is so slow, meaning that we need to increase epoch to find the local minimum of the cost function. For example, with the below configuration, we can get good result:

learning rate: 0.0001

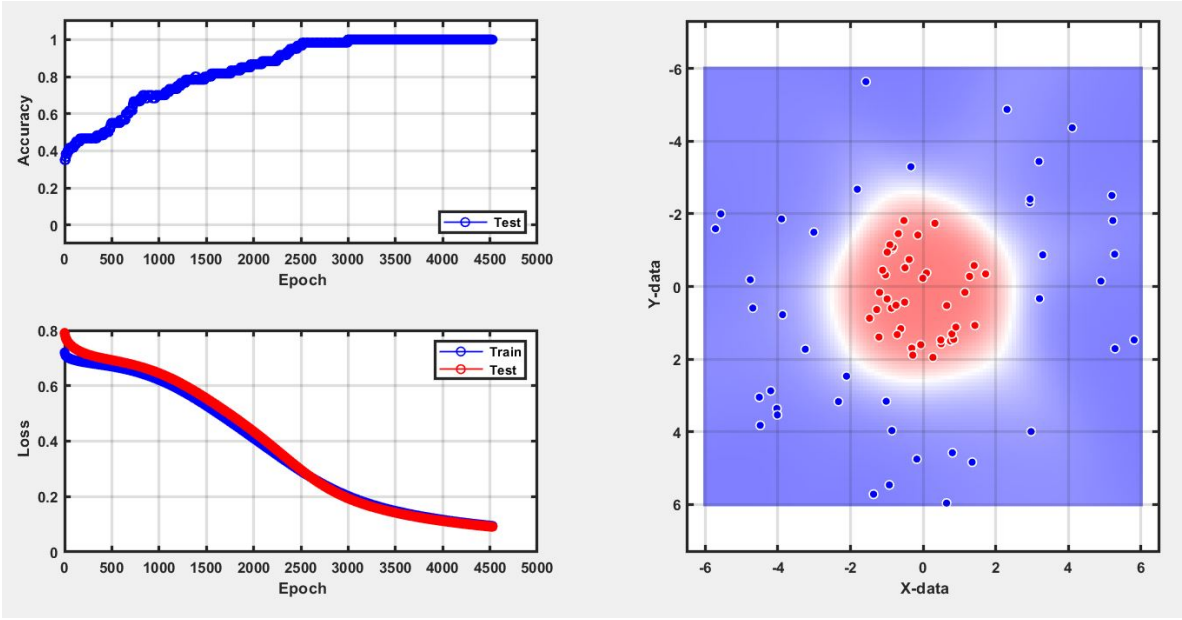
number of neurons in hidden layer: 10

number of epoch: 4500

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



Option 2 (better option): Option 2 is increasing the learning rate instead of epoch. So, I changed the learning rate to 0.02 to achieve good results:

learning rate: 0.02

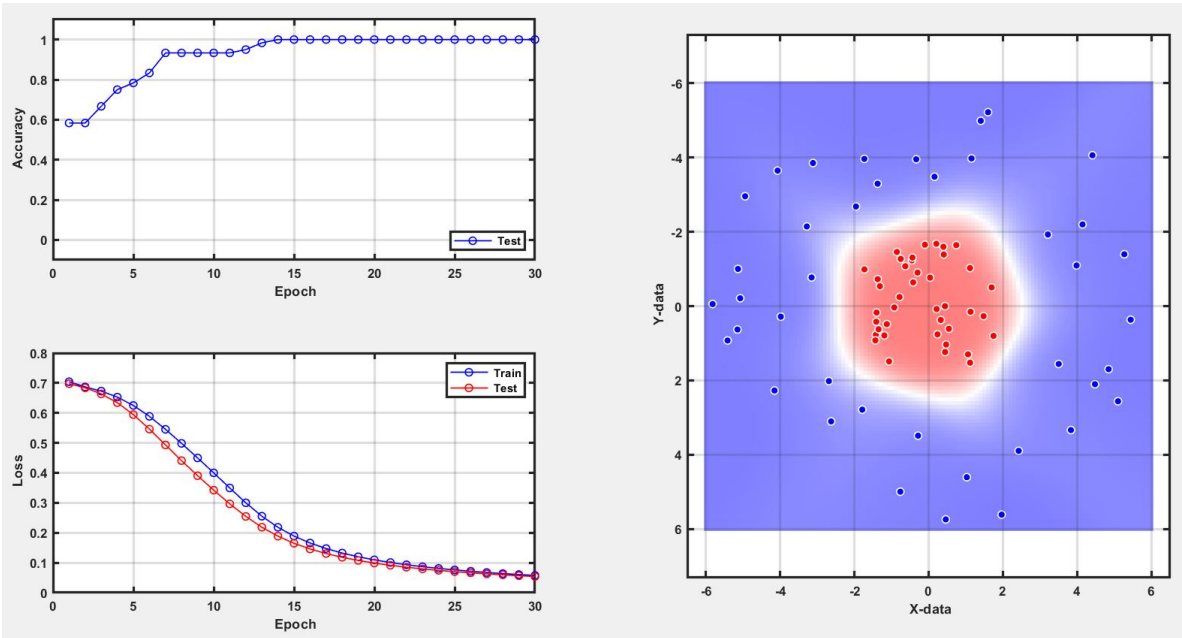
number of neurons in hidden layer: 10

number of epoch: 30

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



2c) screenshot of the final plot:

learning rate: 0.02

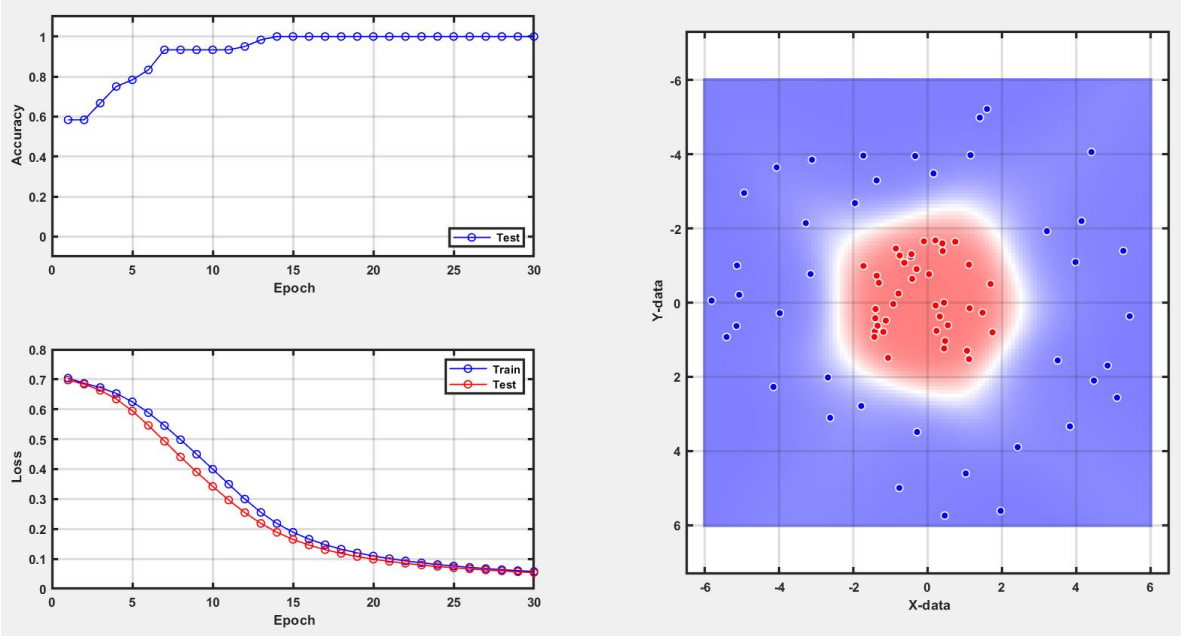
number of neurons in hidden layer: 10

number of epoch: 30

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



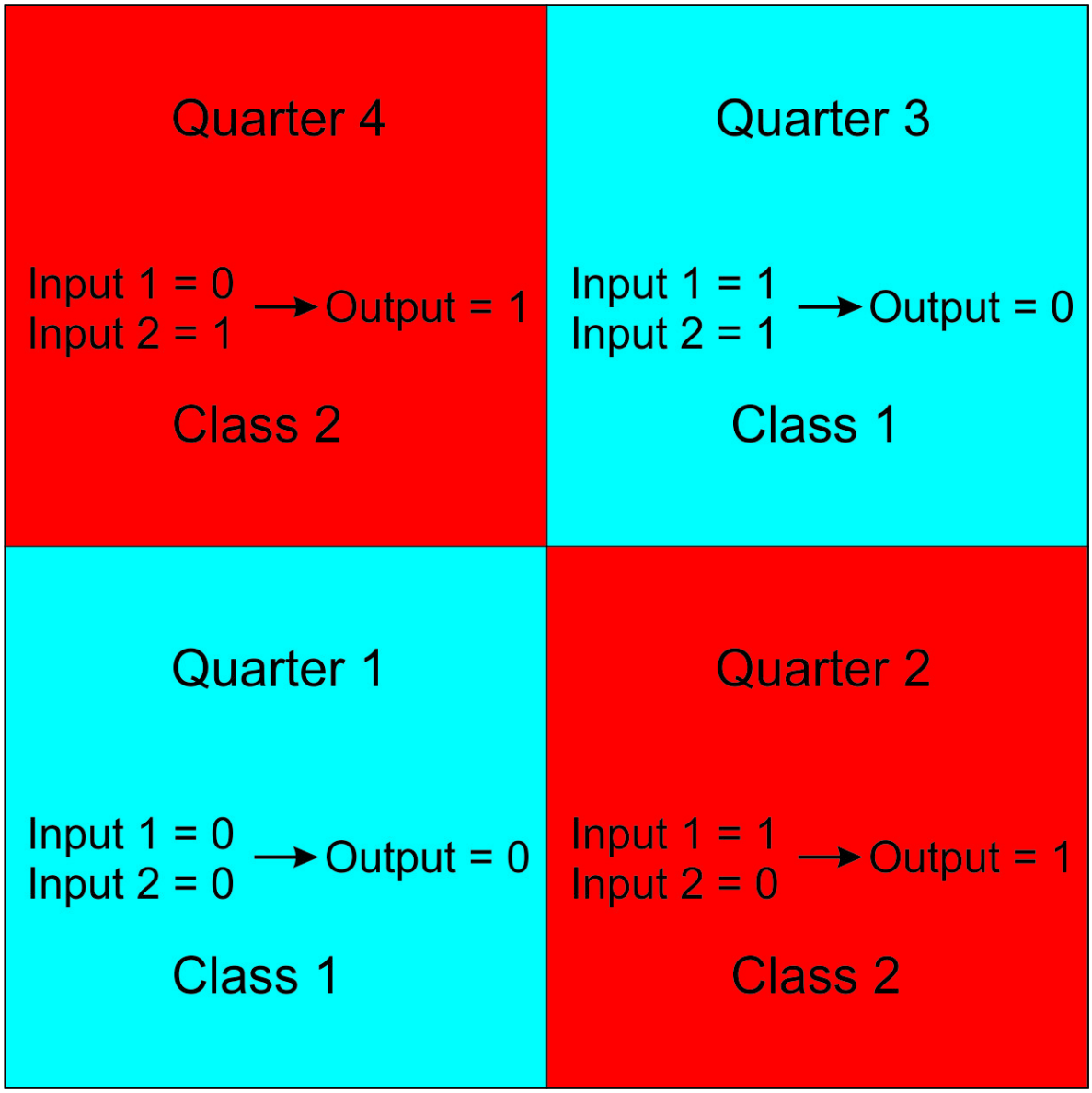
3: "nn_demo_xore.m" demo:

3a) objective:

The xor, or “exclusive or”, problem is also two-class classification neural network. It is the problem of using a neural network to predict the outputs of XOr logic gates given two binary inputs. An XOr function should return a true value if the two inputs are not equal and a false value if they are equal. All possible inputs and predicted outputs are shown in the below figure.

Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

The 2D domain between [-6 6] is divided into four sections, and then, using the definition of XOr function, it is inserted blue and red dots in each section as the below figure:



So, the objective is classifying 2D datasets to find what data is related to each of the four sections. The input node has two values (x and y for each data). The number of output node is 2 in this demo, meaning that the output of each data has two values (two classes). Class 1 is related to the data inside the blue parts (quarters 1 and 3) and class 2 is related to the data inside the red parts (quarters 2 and 4) (see the above figure).

Here, the neural network code is automatically finding the boundary of each of the four sections. Like the previous problem, there are three steps for this code: 1) generating the data 2) setting up the network 3) testing the network.

In the first step, the dataset is generated. There are two types of data. One (dataset1) is the data inside the quarters 1 and 3. The other data (dataset2) is related to the dots inside the quarters 2 and 4. In addition, it is split the data to 85% for training and 15% for testing the performance of the model.

3b) what change I made in default codes or you made (if you configure the network):

By changing the learning rate, I improve the results in such a way that the accuracy is equal to 0.9933. So, I used learning rate = 0.05 to improve the results.

So, the result related to the default parameters is:

learning rate: 0.1

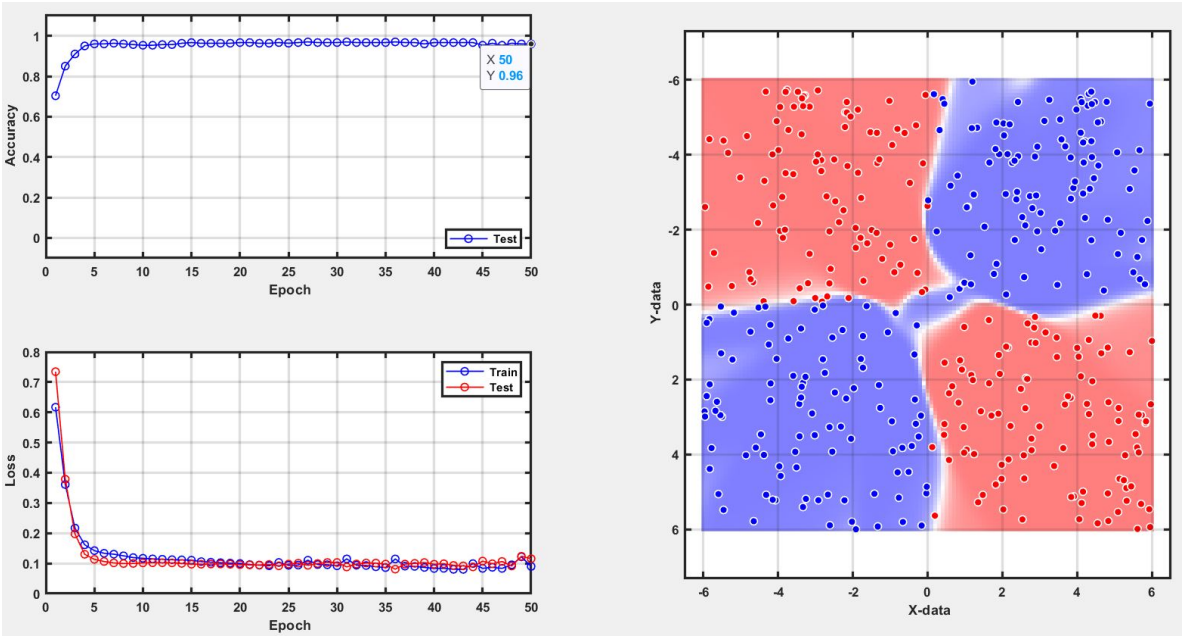
number of neurons in hidden layer: 15

number of epoch: 50

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



accuracy is equal to 0.96.

and the new improved configuration:

learning rate: 0.05

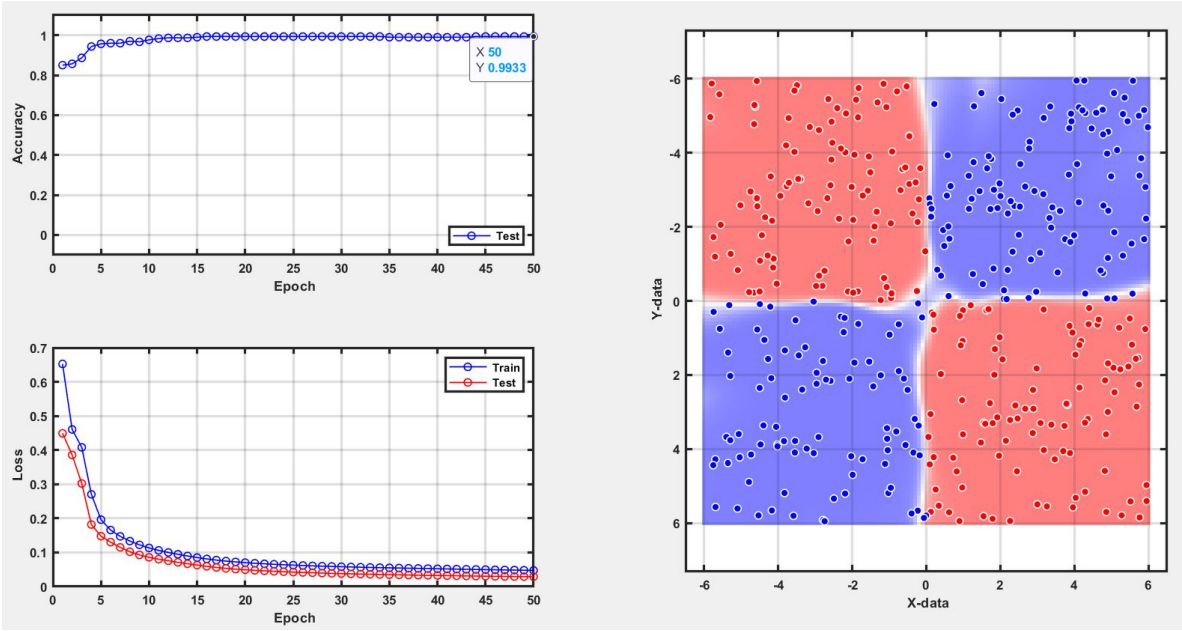
number of neurons in hidden layer: 15

number of epoch: 50

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



accuracy is equal to 0.9933.

3c) screenshot of the final plot:

The new improved configuration:

learning rate: 0.05

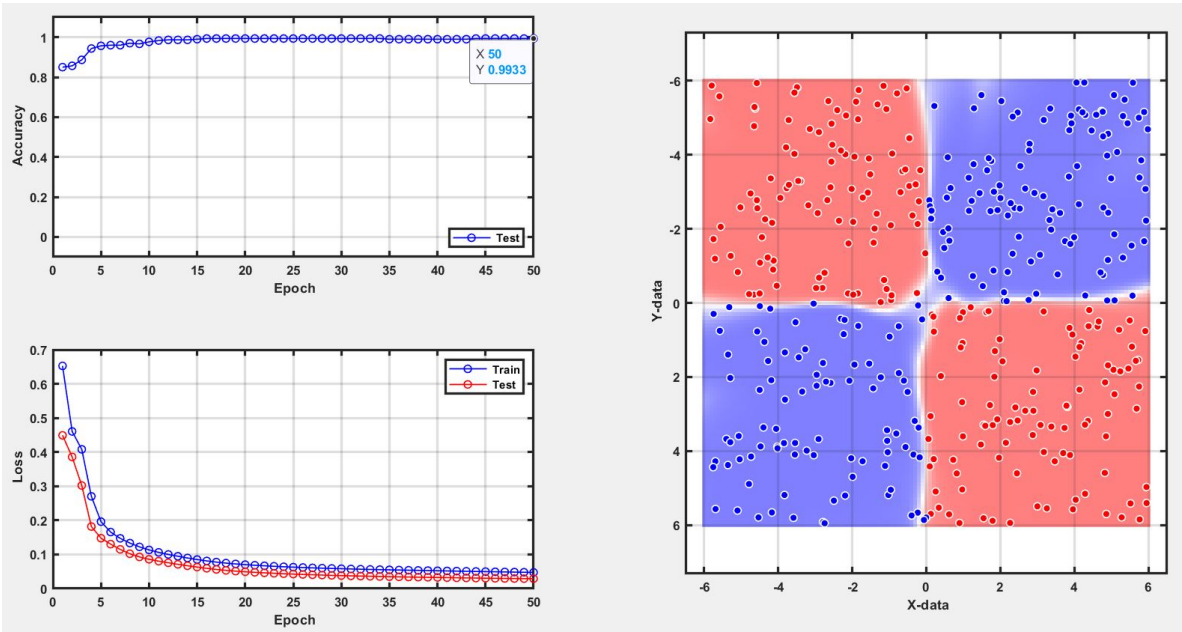
number of neurons in hidden layer: 15

number of epoch: 50

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



accuracy is equal to 0.9933.

4: "nn_demo_wine_clss.m" demo:

4a) objective:

This is a real example. The problem is also three-class classification neural network. The input dimension is equal to 13 and the number of class is equal to 3. It means that the analysis determines the quantities of 13 constituents found in each of the three types of wines.

The objective is to classify the wine using the 13 dimension vector inputs, which are the quantities of 13 constituents:

1) Alcohol

2) Malic acid

3) Ash

4) Alcalinity of ash

5) Magnesium

6) Total phenols

7) Flavanoids

8) Nonflavanoid phenols

9) Proanthocyanins

10) Color intensity

11) Hue

12) OD280/OD315 of diluted wines

13) Proline

The number of class here is 3.

One important point in this example is normalizing the input data. The reason for using normalizing the data is that the range of 13 data is between 0 and infinite number, leading to produce interference and diverge the neural network.

As a result, we can estimate the probability that the input belongs to class 1 or 2 or 3. For the final decision, the input belongs to the class of the node with the highest value/probability (softmax).

4b) what change I made in default codes or you made (if you configure the network):

For this problem, there is no need to change the parameters because the accuracy of the results is high (almost 1), and loss function also experiences minimum state at the end of iteration (epoch). So, the current configuration is fine for this problem:

learning rate: 0.01

number of neurons in hidden layer: 15

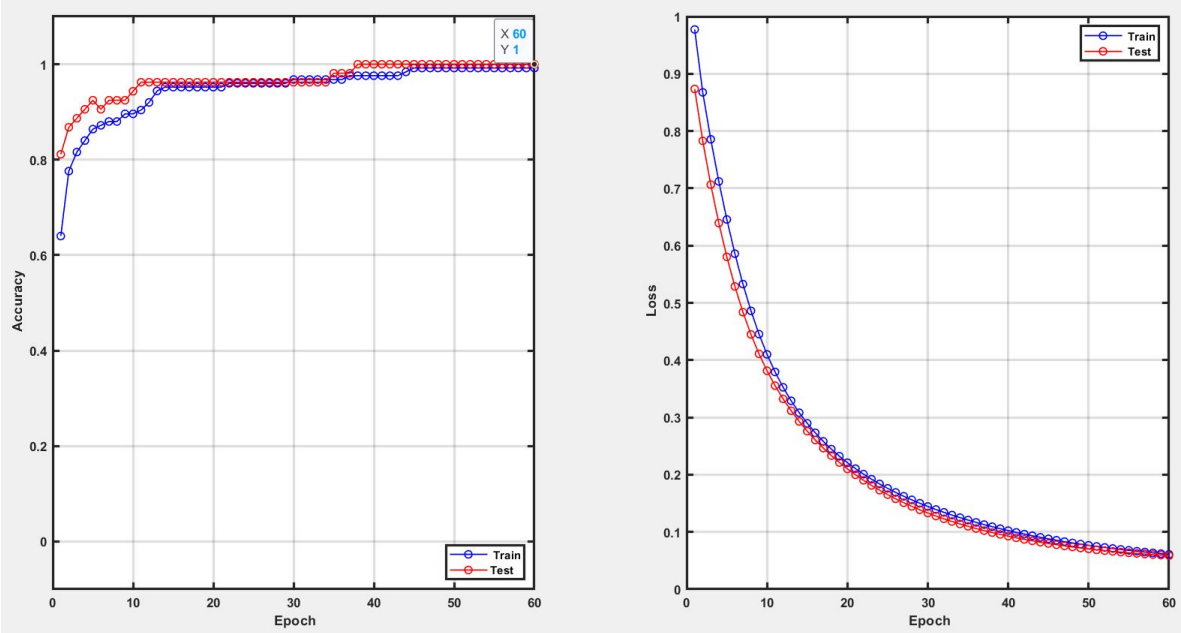
number of epoch: 60

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'

4c) screenshot of the final plot:



accuracy is equal to 1.

5: "nn_demo_poly.m" demo:

5a) objective:

This is regression problem. The objective is finding the polynomial curve that we estimated in Task 7. In fact, the neural network is used to develop a model which is fitted to our data. So, the coefficients calculated in Task 7 are used to formulate true values of the polynomial function. Also, the data is generated when x is within -3.5 to 4.

Here, the number of input is 1, x, and the number of output is 1, y.

It should be mentioned that the difference between regression model and classification model is activation function. In regression, "linear" function is utilized instead of "softmax" because "softmax" bounded the output by [0 1] while the "linear" function is just related to the real value.

For regression problem, accuracy is defined as RMSE (root mean square error):

$$\text{Root-Mean-Square (RMS) Error: } E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^{\star})/N}$$

It means that the smaller RMSE, the more accurate it is.

In addition, in this case, since the dataset includes outliers in this case, it may not be easy to find out the optimum curve. So, learning rate scheduling is used . Another point is that in the current code, every 10 epoch, the learning rate is decreasing as:

$$\theta_{j+1} \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \qquad \alpha_j = \alpha_j \gamma$$

5b) what change I made in default codes or you made (if you configure the network):

For this problem, there is no need to change the parameters because the accuracy of the results (RMSE) is enough (3.176), and loss function also experiences minimum state at the end of iteration (epoch). So, the current configuration is fine for this problem:

learning rate: 0.005

number of neurons in hidden layer: 15

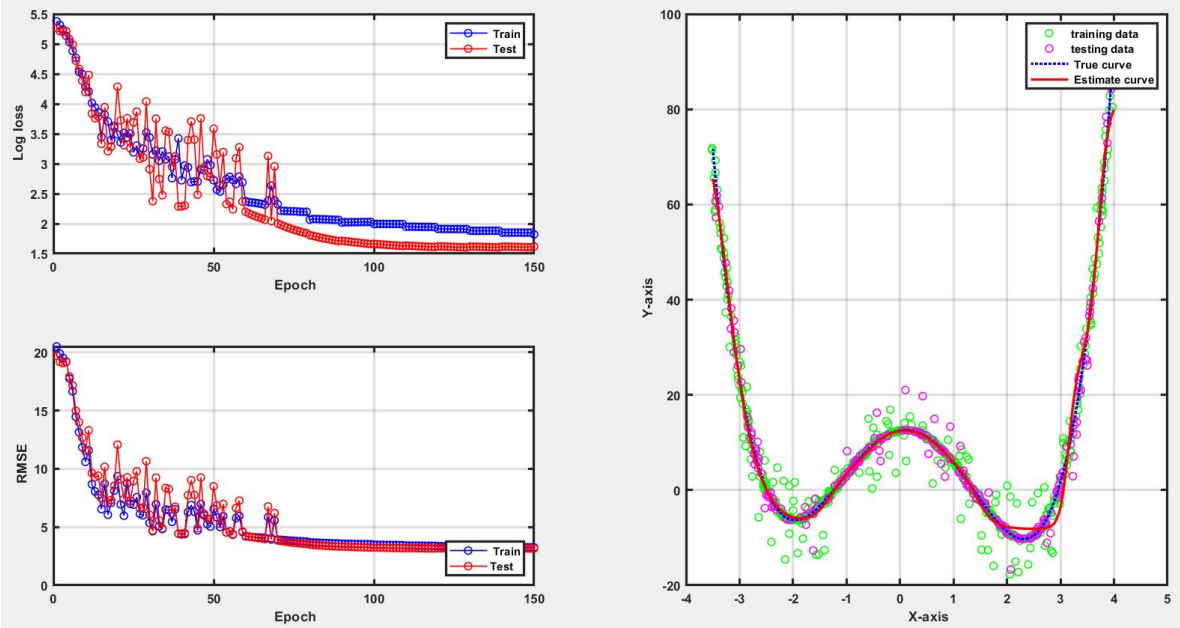
number of epoch: 150

learning rate scheduling = 0.9

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'linear'

type of loss function: 'l2_loss'



RMSE is equal to 3.176.

-> But we can improve the accuracy a little bit by changing the number of neurons in hidden layer and learning rate scheduling:

learning rate: 0.005

number of neurons in hidden layer: 25

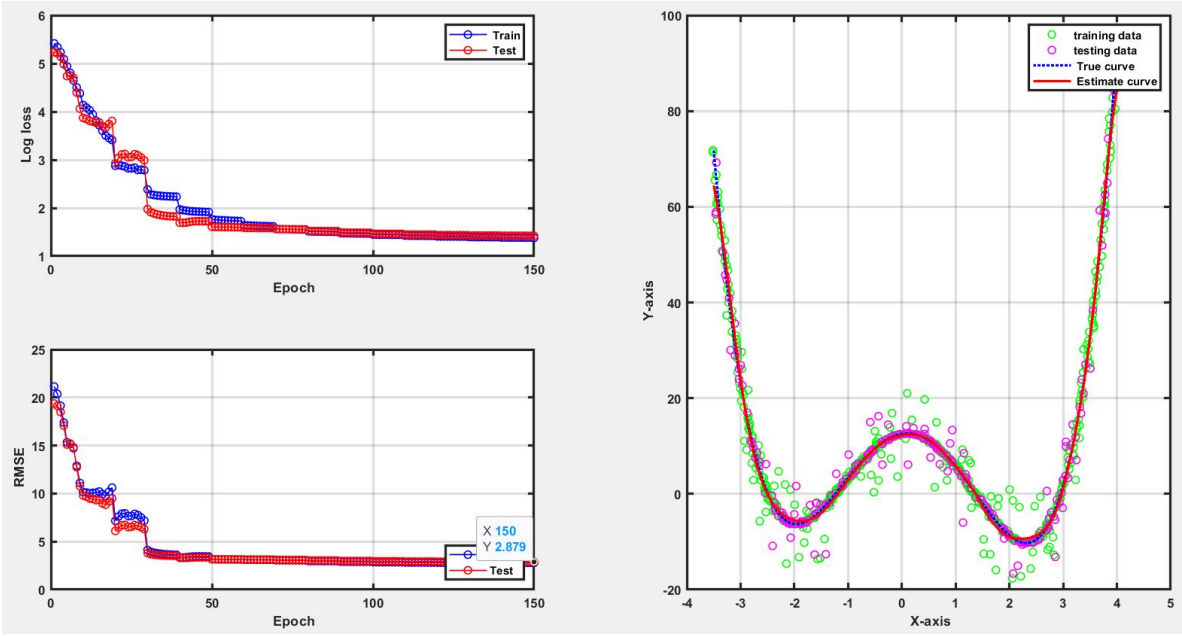
number of epoch: 150

learning rate scheduling = 0.8

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'linear'

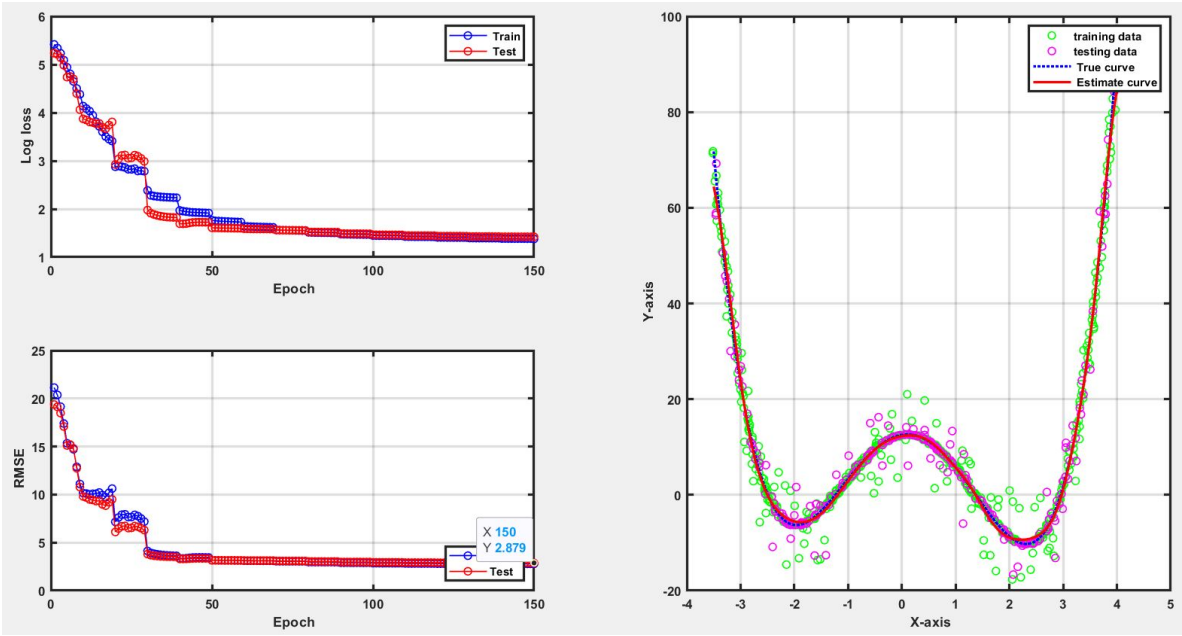
type of loss function: 'l2_loss'



RMSE is equal to 2.879.

It should be noted that increasing the number of neurons in the hidden layer may lead to overfitting phenomena, and we need to be careful about this issue.

5c) screenshot of the final plot:



RMSE is equal to 2.879.

6: "nn_demo_red_wine_reg.m" demo:

6a) objective:

This is a real example. This is a classification-regression problem for wine quality dataset because the output is the quality of the wine which a value between 0 and 10.
The input dimension is equal to 11 and the number of output is equal to 1 (quality of wine).

Since the problem is regression problem, "linear" function is utilized instead of "softmax".

The objective is finding the quality of wine using the 11 dimension vector input:

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol

Also, in this example, normalizing the input data is used. The reason for using normalizing the data is that the range of 11 data is between 0 and infinite number, leading to produce interference and diverge the neural network.

6b) what change I made in default codes or you made (if you configure the network):

For this problem, there is no need to change the parameters because the accuracy of the results (RMSE) is enough (0.62), and loss function also experiences minimum state at the end of iteration (epoch). So, the current configuration is fine for this problem:

learning rate: 0.005

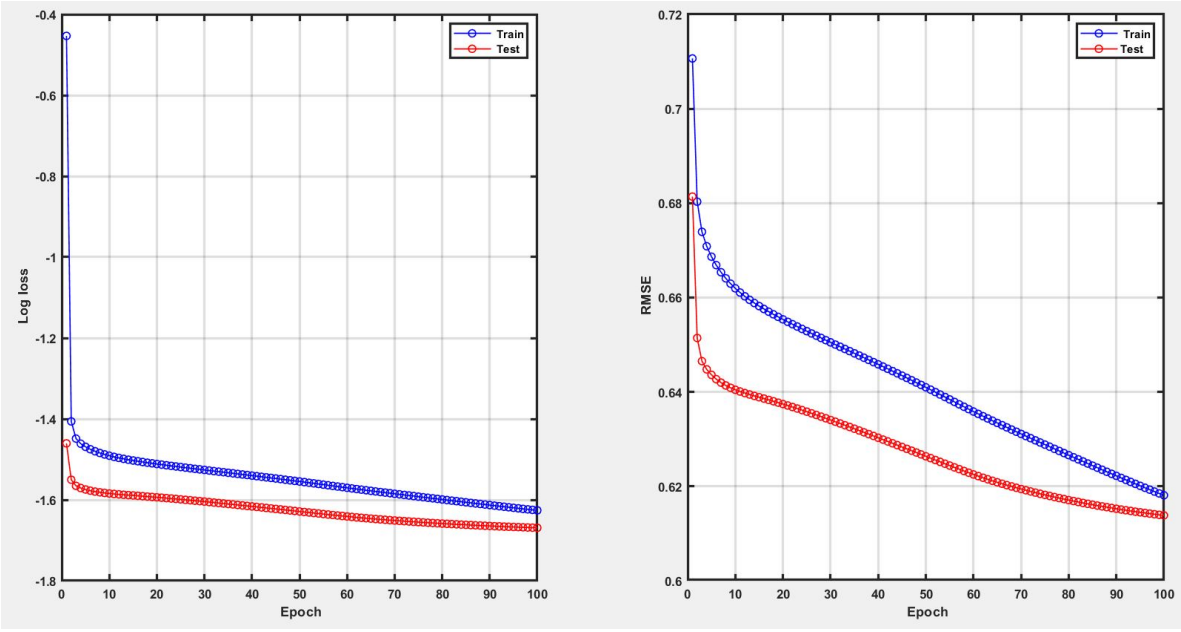
number of neurons in hidden layer: 20

number of epoch: 100

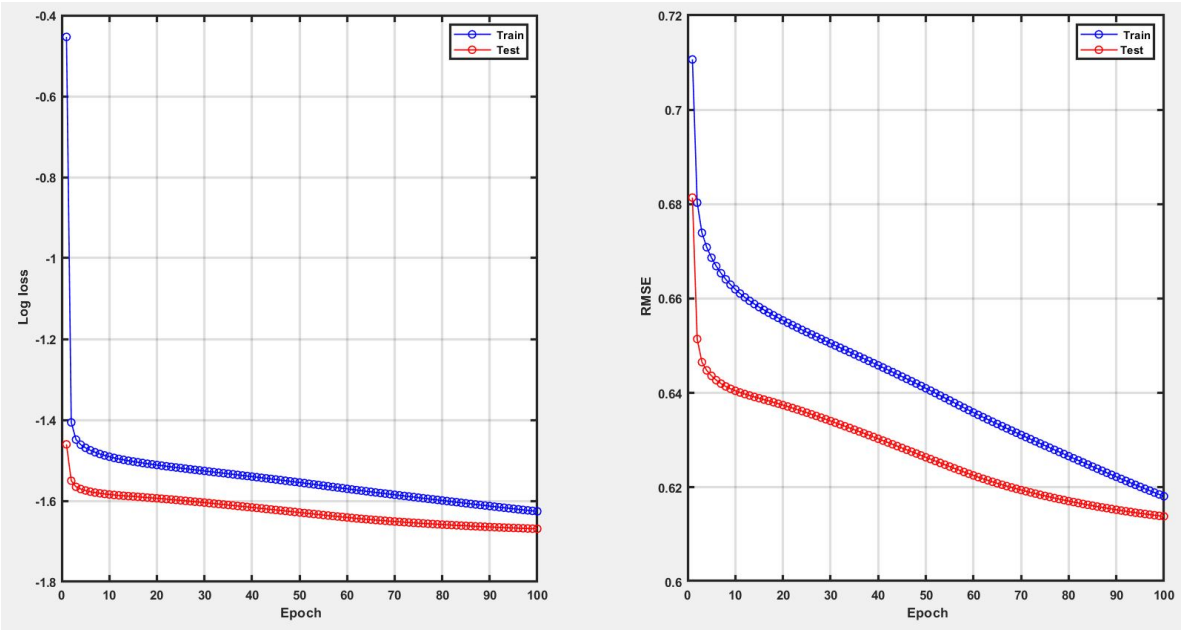
type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'linear'

type of loss function: 'l2_loss'



6c) screenshot of the final plot:



7: "nn_demo_car.m" demo:

7a) objective:

This is four-class classification neural network. It estimates the car condition base on six attributes:

Attributes:

1. buying: vhigh, high, med, low.
2. maint: vhigh, high, med, low.
3. doors: 2, 3, 4, 5more.
4. persons: 2, 4, more.
5. lug_boot: small, med, big.
6. safety: low, med, high.

So, using these six inputs, the model classifies the car in 4 classes: unacc, acc, good, vgood

7b) what change I made in default codes or you made (if you configure the network):

In this problem, the type of data is text data. So, we need to transform it to the scalar values (trainable forms). To do so, built-in function "unique" is used.

In addition, learning rate scheduling is used. Another point is that the learning rate is decreasing every 10 epoch.

As you can see from running this problem, it takes long time to converge. In the beginning, there is no improvement, but after a time, it gradually converges, meaning that the accuracy increases.

As you can see in the Loss function curve, the Loss function is still going down, meaning that the performance the network is increasing.

For this problem, there is no need to change the parameters because the accuracy of the results is enough (0.9247), and loss function also experiences minimum state at the end of iteration (epoch). So, the default configuration is fine for this problem:

learning rate: 0.1

number of neurons in hidden layer: 12

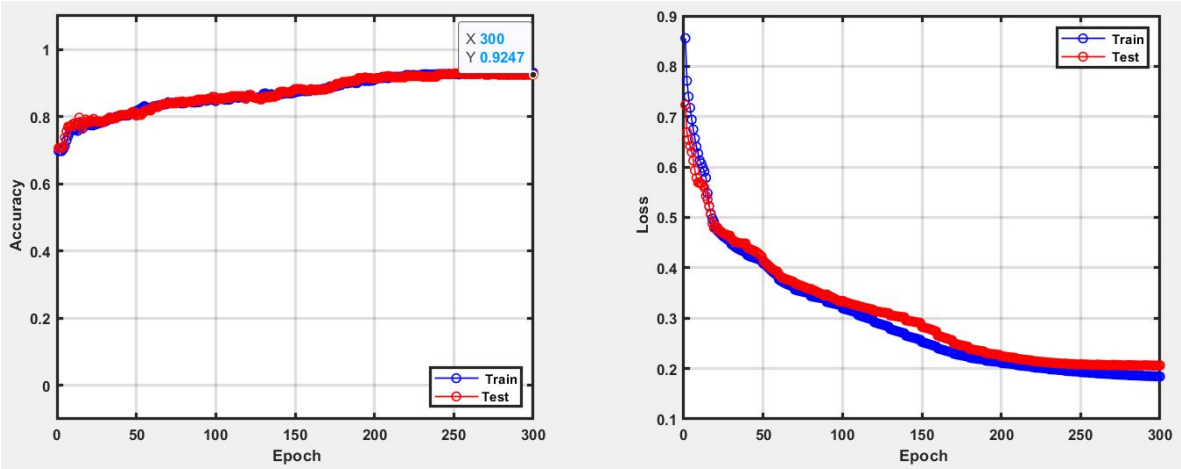
number of epoch: 300

learning rate scheduling = 0.9

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



RMSE is equal to 0.9247.

But we can improve the accuracy a little bit by changing the number of neurons in hidden layer:

learning rate: 0.1

number of neurons in hidden layer: 18

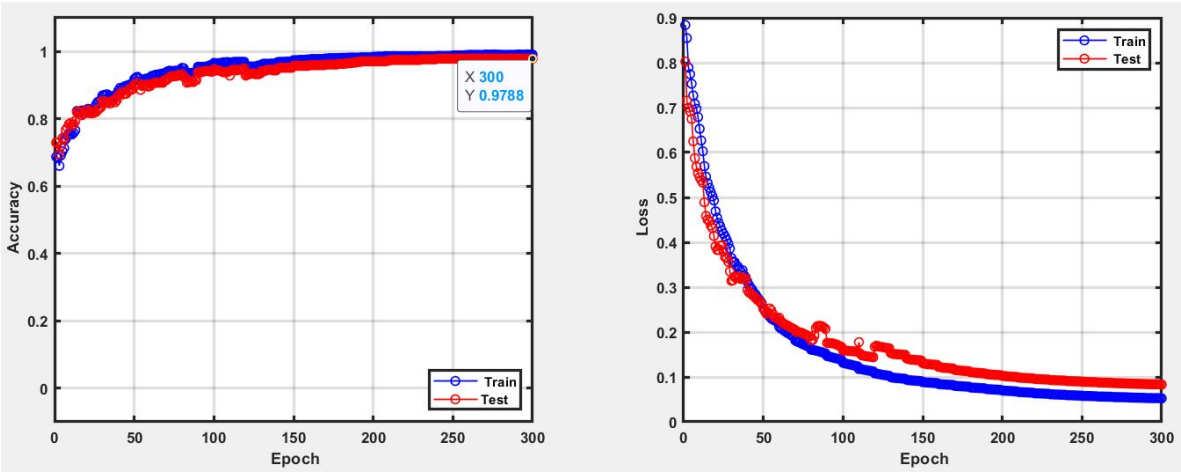
number of epoch: 300

learning rate scheduling = 0.9

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

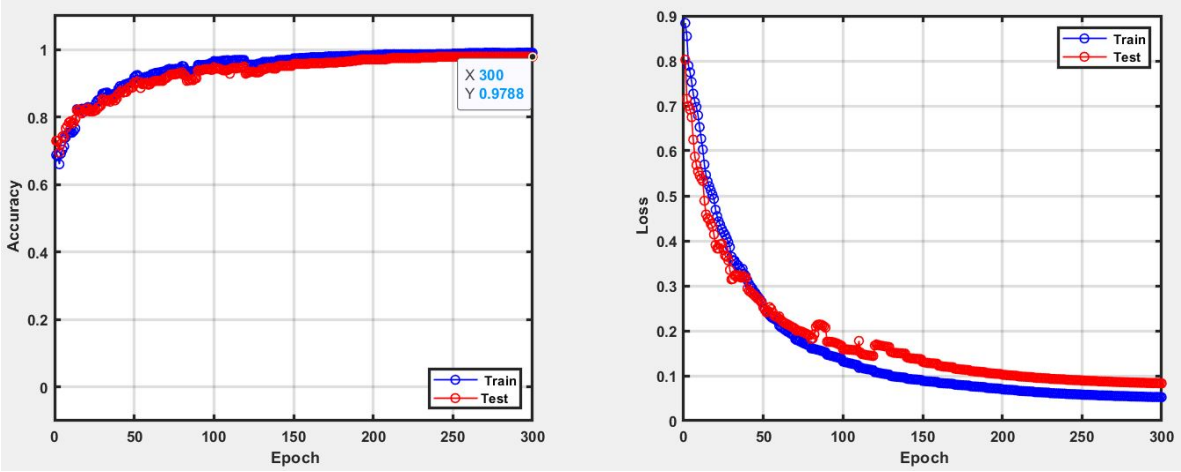
type of loss function: 'cross_entropy'



RMSE is equal to 0.9788.

It should be noted that increasing the number of neurons in the hidden layer may lead to overfitting phenomena, and we need to be careful about this issue.

7c) screenshot of the final plot:



RMSE is equal to 0.9788.

Problem 2 (80 points)

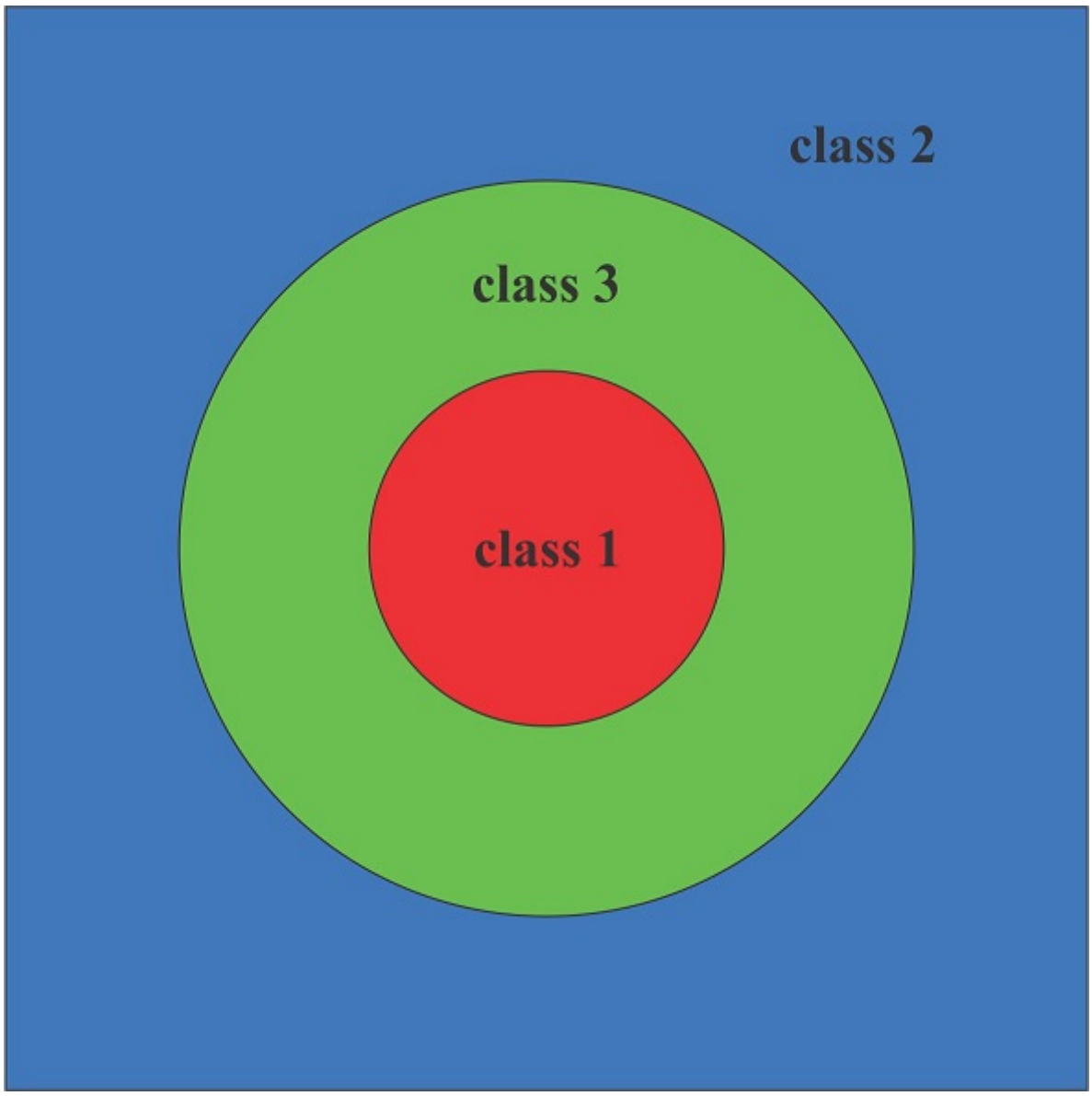
You are going to build your own model using neural network

(a) Create your own dataset and build a classifier. Please do not create the dataset similar to the examples provided. Please be creative! You will get the idea from the several demos in the tutorial. For example, like the polynomial example, think about making a sample data from a known model.

For this part, I have designed two neural network problems. One is classification problem, and another is regression problem. For these problems, I have created my own dataset.

Problem 1: classification problem:

This is three-class classification neural network. The objective is classifying 2D datasets to find what data is located inside the circle, what data is located inside the ring and what data is located in the rest of the area (see the below figure). It means that the input node has two values (x and y for each data), and then, it is introduced some neurons in the hidden layer as the mathematical functions each designed to produce an output specific to an intended result. Finally, the number of output node is 3, meaning that the output of each data has three values (three classes). So, we can estimate the probability that the input belongs to class 1 to 3. For the final decision, the input belongs to the class of the node with the highest value/probability (softmax).



As you can see, the neural network code is automatically finding the boundary around the data inside the circle, the boundary around the data inside the ring, and the boundary around the data outside of the circle and the ring. Here, the boundary is nonlinear. The reason that we can find a nonlinear boundary is that we are using activation function. Without activation function, this is just a linear regression, meaning that we can only find linear boundary.

In the first step, the dataset is generated. There are three types of data. Dataset1 is the data inside the circle with the radius equal to 2. Dataset3 is related to the points inside the middle ring (maximum and minimum radii are equal to 4 and 3, respectively). Dataset2 is related to the points out of the ring and the circle. In addition, it is split the data to 70% for training and 30% for testing the performance of the model. In the second step, we have training process. The training process is to find out the nonlinear boundary around the circle and ring data in 2D by defining weights and bias in the net. Finally, we test the performance the model by defining dataset for testing the accuracy of the network.

I used the below configuration for solving this classification problem:

learning rate: 0.01

number of neurons in input layer: 2

number of neurons in hidden layer: 20

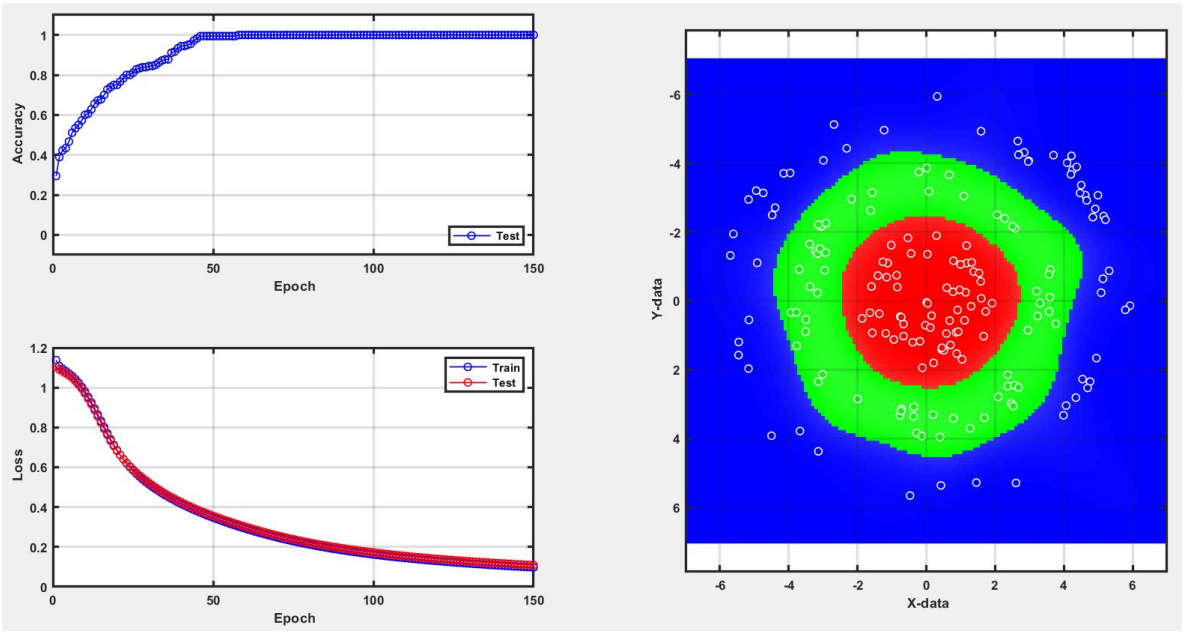
number of neurons in output layer: 3

number of epoch: 150

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'



As you can see, the nonlinear boundary around the circle and ring data has been correctly detected. Also, the accuracy of the results is high (1), and loss function also experiences minimum state at the end of iteration (epoch). So, this configuration is fine for this problem.

Matlab code:

First, I will present the parts of the code that I changed from the original code:

Function for generating the data:

```
function data = GenerateConcentricCircleData

% parameters
outRadius = [2 3 4 5 6];
numPt = 200;

% inner circle
data.set1 = zeros(numPt,2);
count = 1;
while count<=numPt
    xin = rand*outRadius(1)*2-outRadius(1);
    yin = rand*outRadius(1)*2-outRadius(1);
    if xin^2 + yin^2 < outRadius(1)^2
        data.set1(count,:) = [xin yin];
        count = count + 1;
    end
end

% outer ring
data.set2 = zeros(numPt,2);
count = 1;
while count<=numPt
    xin = rand*outRadius(5)*2-outRadius(5);
```

```
    yin = rand*outRadius(5)*2-outRadius(5);
    if (xin^2 + yin^2 < outRadius(5)^2) && (xin^2 + yin^2 > outRadius(4)^2)
        data.set2(count,:) = [xin yin];
        count = count + 1;
    end
end

% middle ring
data.set3 = zeros(numPt,2);
count = 1;
while count<=numPt
    xin = rand*outRadius(3)*2-outRadius(3);
    yin = rand*outRadius(3)*2-outRadius(3);
    if (xin^2 + yin^2 < outRadius(3)^2) && (xin^2 + yin^2 > outRadius(2)^2)
        data.set3(count,:) = [xin yin];
        count = count + 1;
    end
end

end
```

Function for data transformation for three-class classification:

```
function [x, t] = data2xt(dataset)

nSet1 = size(dataset.set1,1);
nSet2 = size(dataset.set2,1);
nSet3 = size(dataset.set3,1);

x = cat(2, dataset.set1', dataset.set2', dataset.set3');
t = zeros(3, nSet1+nSet2+nSet3);

t(1,1:nSet1) = 1;
t(2,nSet1+1:nSet1+nSet2) = 1;
t(3,nSet1+nSet2+1:end) = 1;

% shuffling
rind = randperm(nSet1+nSet2+nSet3);
x = x(:,rind);
t = t(:,rind);
end
```

Function for showing the nonlinear boundary between datasets:

```
function show_map(cur_ax, x, t, grid_info)

axes(cur_ax);

[M,I] = max(grid_info.grid_clss_prob);
M(I==1) = (10+M(I==1));
M(I==2) = (1-M(I==2));
M(I==3) = (5+M(I==3));
prob_value_plot = reshape(M, grid_info.size_mat);
```

```
% color gradient
custom_map = tri_color_gradient([0 0 1],[1 1 1],[0 1 0],[1 1 1],[1 0 0],256);

imagesc(grid_info.xrange, grid_info.yrange, prob_value_plot);
hold on; view(0,90); colormap(custom_map);

ind_set1 = find(t(1,:));
ind_set2 = find(t(2,:));
ind_set3 = find(t(3,:));

ind_set1 = ind_set1(1:grid_info.numPt_plot);
ind_set2 = ind_set2(1:grid_info.numPt_plot);
ind_set3 = ind_set3(1:grid_info.numPt_plot);
plot(x(1,ind_set1), x(2,ind_set1), 'MarkerEdgeColor', 'w', ...
    'MarkerFaceColor', [1 0 0], 'Marker', 'o', ...
    'LineStyle', 'none', 'linewidth', 1);
plot(x(1,ind_set2), x(2,ind_set2), 'MarkerEdgeColor', 'w', ...
    'MarkerFaceColor', [0 0 1], 'Marker', 'o', ...
    'LineStyle', 'none', 'linewidth', 1);
plot(x(1,ind_set3), x(2,ind_set3), 'MarkerEdgeColor', 'w', ...
    'MarkerFaceColor', [0 1 0], 'Marker', 'o', ...
    'LineStyle', 'none', 'linewidth', 1);
set(gca,'fontsize',10,'linewidth',2,'fontweight','bold')
xlabel('\bf X-data');xlim([-7 7]);
ylabel('\bf Y-data');ylim([-7 7]);
grid on;  axis('equal');
hold off;

end
```

Function for defining new colormap:

```
function grad=tri_color_gradient(c1,c2,c3,c4,c5,depth)

drgb1 =(c2-c1)/(depth/4-1);
drgb2 =(c3-c2)/(depth/4-1);
drgb3 =(c4-c3)/(depth/4-1);
drgb4 =(c5-c4)/(depth/4-1);

grad = zeros(depth, 3);

for ii=1:depth/4
    grad(ii,:) = drgb1*(ii-1) + c1;
    grad(ii+(depth/4),:) = drgb2*(ii-1) + c2;
    grad(ii+(depth/2),:) = drgb3*(ii-1) + c3;
    grad(ii+(3*depth/4),:) = drgb4*(ii-1) + c4;
end

end
```

Finally, I will present the full neural network Matlab code for this problem:

```
% Simple neural network demonstration
% This code shows how to implement a neural network with one hidden layer.
% You can change some basic parameters including
% learning rate, number of neurons, number of epoch, etc. You can also see
% similar demo from here:
% https://github.com/tensorflow/playground
% optimization: Stochastic gradient descent

% Dataset: Cencentric Circles (Three-class Classification)

% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Updated by Saeed Hatefi Ardakani: 04/18/2020
clear; close all; clc;

addpath('base');

dataset = GenerateConcentricCircleData;
net = set_nn; % you need to configure your network depending your dataset
test_nn(net, dataset);

function data = GenerateConcentricCircleData

% parameters
outRadius = [2 3 4 5 6];
numPt = 200;

% inner circle
data.set1 = zeros(numPt,2);
count = 1;
while count<=numPt
    xin = rand*outRadius(1)*2-outRadius(1);
    yin = rand*outRadius(1)*2-outRadius(1);
    if xin^2 + yin^2 < outRadius(1)^2
        data.set1(count,:) = [xin yin];
        count = count + 1;
    end
end

% outer ring
data.set2 = zeros(numPt,2);
count = 1;
while count<=numPt
    xin = rand*outRadius(5)*2-outRadius(5);
    yin = rand*outRadius(5)*2-outRadius(5);
    if (xin^2 + yin^2 < outRadius(5)^2) && (xin^2 + yin^2 > outRadius(4)^2)
        data.set2(count,:) = [xin yin];
        count = count + 1;
    end
end

% middle ring
data.set3 = zeros(numPt,2);
count = 1;
while count<=numPt
    xin = rand*outRadius(3)*2-outRadius(3);
    yin = rand*outRadius(3)*2-outRadius(3);
    if (xin^2 + yin^2 < outRadius(3)^2) && (xin^2 + yin^2 > outRadius(2)^2)
        data.set3(count,:) = [xin yin];
```

```
        count = count + 1;
    end
end

end

function net = set_nn
net.param.numepoch = 150; % number of epoch
net.param.learningrate = 0.01; % learning rate
net.param.train_test_ratio = [0.7 0.3]; % training and testing division

net.input.node = 2;
net.layer.node = 20;
net.output.node = 3;

net.layer.activation = 'sigmoid';
net.output.activation = 'softmax';
net.output.loss = 'cross_entropy';

% layer: hidden layer (input to hidden)
net.layer.weight = rand(net.input.node, net.layer.node)-1; % set random value
between -1 ~ 1
net.layer.bias = zeros(net.layer.node, 1); % set bias as zero

% output: output layer (hidden to output)
net.output.weight = rand(net.layer.node, net.output.node)-1; % -1 ~ 1
net.output.bias = zeros(net.output.node, 1);

end

function test_nn(net, dataset)

train_test_ratio = net.param.train_test_ratio;
numepoch = net.param.numepoch;  % # of iteration

[x, t] = data2xt(dataset); % data transformation

% training and testing split
[trainInd, ~, testInd] = ...
    dividerand(size(x,2),train_test_ratio(1), 0, train_test_ratio(2));

close all; clc;
figure('Name', 'Neural Network Training', 'NumberTitle','off', 'Position', [10
10 1300 600]);
h1 = subplot(2,2,1);
h2 = subplot(2,2,3);
h3 = subplot(2,2,[2 4]);

[xgrid, ygrid] = meshgrid(-7:0.1:7);
loss = zeros(2, numepoch);
accuracy = zeros(1, numepoch);

for ii=1:numepoch

    % training
    [net, train_loss] = nn_train_net(net, x(:,trainInd), t(:, trainInd));

    % testing
```

```
    out = nn_test_net(net, x(:,testInd));

    % plotting loss curve
    loss(1, ii) = train_loss;

    test_loss = mean(nn_loss(out,t(:, testInd), net.output.loss, 'forward'));
    loss(2, ii) = test_loss; % testing loss

    show_loss(h2, ii, loss);

    % plotting accuracy curve (check if the data is classified correctly)
    [~,out_clss] = max(out, [], 1); % predicted class
    [~,true_clss] = max(t(:, testInd), [], 1); % true class
    n_test = numel(out_clss);
    n_test_correct = sum(out_clss == true_clss);

    accuracy(ii) = n_test_correct/n_test;
    show_acc(h1, ii, accuracy);


    % ploting
    % visualization (testing grid coordinates)
    out_grid = nn_test_net(net, [xgrid(:)';ygrid(:)']);

    grid_info.xrange = [-7 7];
    grid_info.yrange = [-7 7];
    grid_info.size_mat = size(xgrid);
    grid_info.grid_clss_prob = out_grid;
    grid_info.numPt_plot = 0.1*size(t,2);

    show_map(h3, x, t, grid_info);
    pause(0.001);
end
end

function show_acc(cur_ax, epoch, accuracy)

axes(cur_ax);
plot(1:epoch, accuracy(1,1:epoch), '-ob', 'linewidth', 1); hold on;

legend('\bf Test', 'Location','southeast')
xlabel('\bf Epoch');
ylabel('\bf Accuracy');ylim([-0.1 1.1]);
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function show_loss(cur_ax, epoch, loss)

axes(cur_ax);
plot(1:epoch, loss(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, loss(2,1:epoch), '-or', 'linewidth', 1);

legend('\bf Train','\bf Test', 'Location','northeast');
xlabel('\bf Epoch');
ylabel('\bf Loss');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
```

```
grid on; hold off;

end

function show_map(cur_ax, x, t, grid_info)

axes(cur_ax);

[M,I] = max(grid_info.grid_clss_prob);
M(I==1) = (10+M(I==1));
M(I==2) = (1-M(I==2));
M(I==3) = (5+M(I==3));
prob_value_plot = reshape(M, grid_info.size_mat);

% color gradient
custom_map = tri_color_gradient([0 0 1],[1 1 1],[0 1 0],[1 1 1],[1 0 0],256);

imagesc(grid_info.xrange, grid_info.yrange, prob_value_plot);
hold on; view(0,90); colormap(custom_map);

ind_set1 = find(t(1,:));
ind_set2 = find(t(2,:));
ind_set3 = find(t(3,:));

ind_set1 = ind_set1(1:grid_info.numPt_plot);
ind_set2 = ind_set2(1:grid_info.numPt_plot);
ind_set3 = ind_set3(1:grid_info.numPt_plot);
plot(x(1,ind_set1), x(2,ind_set1), 'MarkerEdgeColor', 'w', ...
    'MarkerFaceColor', [1 0 0], 'Marker', 'o', ...
    'LineStyle', 'none', 'linewidth', 1);
plot(x(1,ind_set2), x(2,ind_set2), 'MarkerEdgeColor', 'w', ...
    'MarkerFaceColor', [0 0 1], 'Marker', 'o', ...
    'LineStyle', 'none', 'linewidth', 1);
plot(x(1,ind_set3), x(2,ind_set3), 'MarkerEdgeColor', 'w', ...
    'MarkerFaceColor', [0 1 0], 'Marker', 'o', ...
    'LineStyle', 'none', 'linewidth', 1);
set(gca,'fontsize',10,'linewidth',2,'fontweight','bold')
xlabel('\bf X-data');xlim([-7 7]);
ylabel('\bf Y-data');ylim([-7 7]);
grid on;  axis('equal');
hold off;

end

function [x, t] = data2xt(dataset)

nSet1 = size(dataset.set1,1);
nSet2 = size(dataset.set2,1);
nSet3 = size(dataset.set3,1);

x = cat(2, dataset.set1', dataset.set2', dataset.set3');
t = zeros(3, nSet1+nSet2+nSet3);

t(1,1:nSet1) = 1;
t(2,nSet1+1:nSet1+nSet2) = 1;
t(3,nSet1+nSet2+1:end) = 1;

% shuffling
```

```
rind = randperm(nSet1+nSet2+nSet3);
x = x(:,rind);
t = t(:,rind);
end

function grad=tri_color_gradient(c1,c2,c3,c4,c5,depth)

drgb1 =(c2-c1)/(depth/4-1);
drgb2 =(c3-c2)/(depth/4-1);
drgb3 =(c4-c3)/(depth/4-1);
drgb4 =(c5-c4)/(depth/4-1);

grad = zeros(depth, 3);

for ii=1:depth/4
    grad(ii,:) = drgb1*(ii-1) + c1;
    grad(ii+(depth/4),:) = drgb2*(ii-1) + c2;
    grad(ii+(depth/2),:) = drgb3*(ii-1) + c3;
    grad(ii+(3*depth/4),:) = drgb4*(ii-1) + c4;
end

end
```

Problem 2: regression problem:

This is regression problem. The objective is finding the hyperbolic paraboloid curve that we can estimate from the noisy data. I will explain the method for creating the data.

For this problem, I used three steps:

1) First step:

I used built-in function "awgn" for adding some noise to the true data. The equation for hyperbolic paraboloid curve is

$$z = ax^2 - by^2$$

awgn(in,snr) function adds white Gaussian noise to the vector signal "in". "snr" is signal-to-noise ratio.

By using the below Matlab code, we can add gaussian noise to the data related to the hyperbolic paraboloid curve. The amount of a and b parameters in the true equation are 1 and 1, respectively.

Code for data generation:

```
clc
clear all;
close all;

% parameters
a = 1;
b = 1;
```

```
c = 1;

% adding the gaussian noise to the signal
[xx,yy] = meshgrid(-6:0.1:6);
zz = (a*xx.^2 - b*yy.^2)/c ;
z = awgn(zz,10,'measured');
fig = figure(1);
plot3(xx, yy, z, 'ob', 'linewidth', 1);hold on;

% ploting the true equation as a surface
[xxx,yyy] = meshgrid(-6.5:0.1:6.5);
zzz = (a*xxx.^2 - b*yyy.^2)/c ;
surf(xxx,yyy,zzz,'FaceColor','y'); %Plot the surface
xlabel('\bf X-axis');
ylabel('\bf Y-axis');
zlabel('\bf Z-axis');
grid on; hold off;

% preparing the generated data for neural network code
x = [];
y = [];
for ii = 1 : size(xx,2)
    for jj = 1 : size(xx,2)
        x = [x ; [xx(ii,jj) yy(ii,jj)]];
        y = [y ; z(ii,jj)];
    end
end

% saving the "paraboloid.mat" file as an input for neural network code
filename = 'paraboloid.mat';
save(filename,'x','y');
```

In the above code, I used two values for snr in awgn function. One is 20, and another is 10. By using bigger value for snr, the created noisy data is closer to the true surface. So:

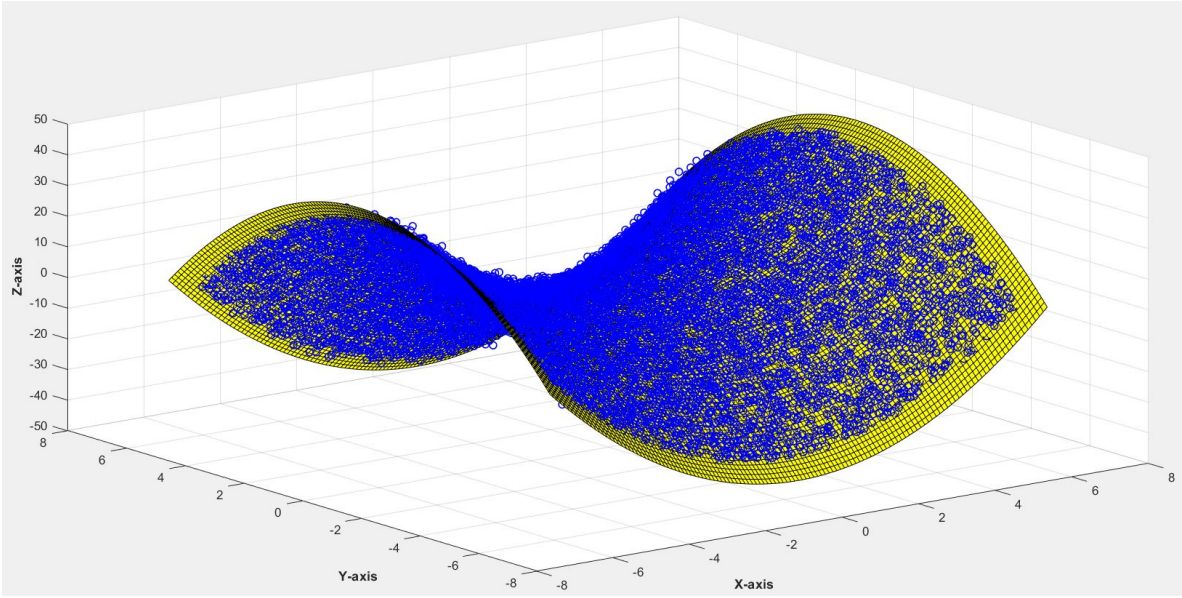
dataset1 ---> awgn(in,20)

dataset2 ---> awgn(in,10)

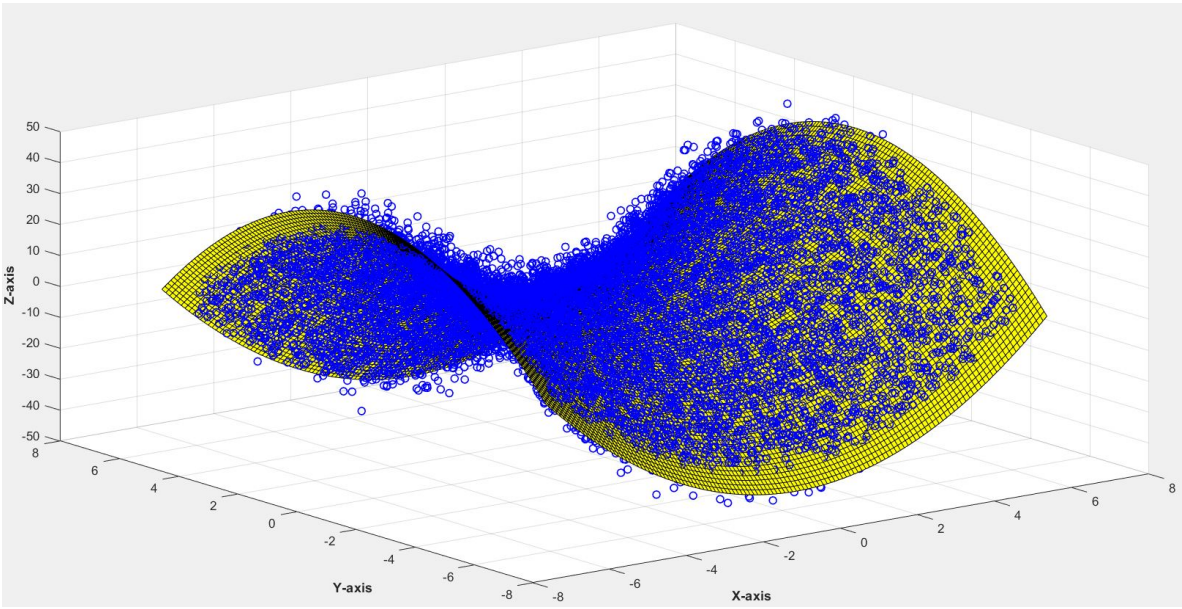
It should be mentioned that the reason for using two datasets is that I want to compare the effect of noise on the result of neural network.

The below figure is the result of the code for creating two datasets:

Creating dataset1 (snr = 20):



Creating dataset2 (snr = 10):



As you can see from the above figures, bigger value of snr makes the created data closer to the true surface.

2) Second step:

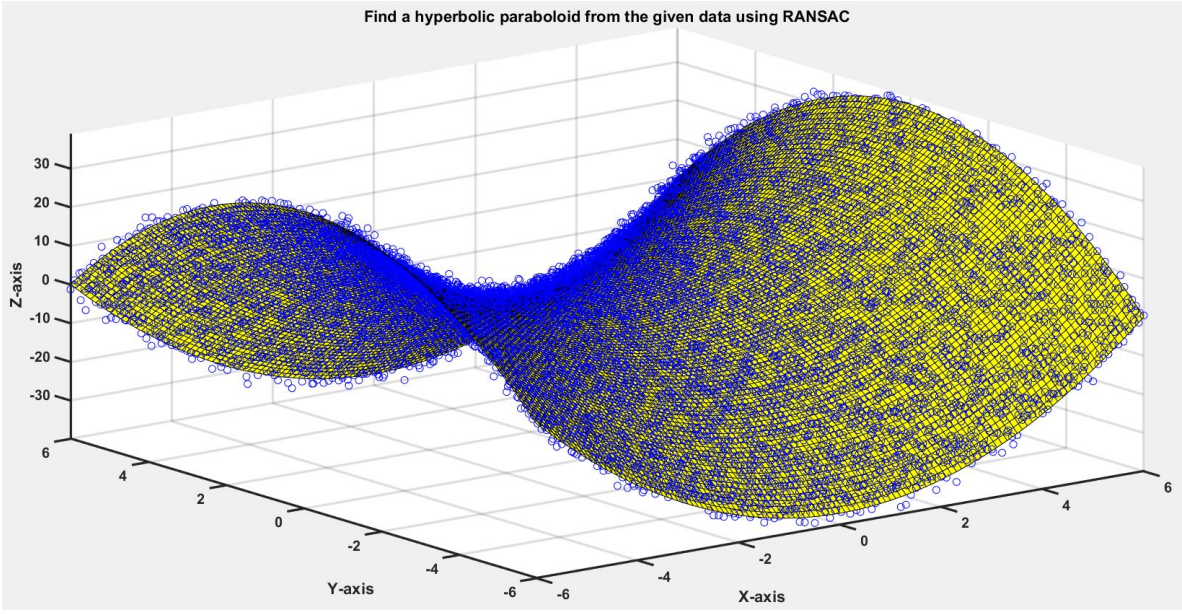
In this step, I used the Ransac code related to Task 7 to predict the surface from the noisy datasets. In fact, I would like to estimate the values of a and b in $z = ax^2 - by^2$ using Ransac:

The results of Ransac code for datasets 1 and 2 are:

For dataset1:

a = 1.0244 ≈ 1

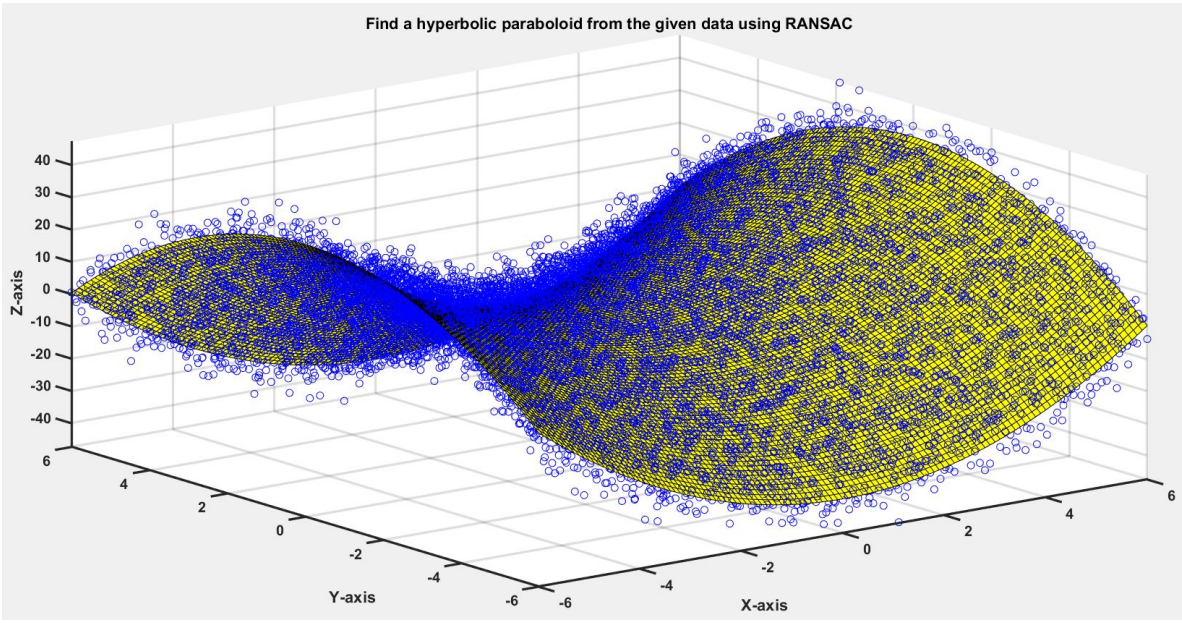
b = 1.0224 ≈ 1



For dataset2:

a = 0.9626 ≈ 1

b = 0.9598 ≈ 1



According to our expectation, the results for a and b are very close the true values that I utilized for creating noisy data. So, we will consider a=1 and b=1 in the neural network code.

Code for estimating the values a and b using Ransac:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 2a ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Fit a hyperbolic paraboloid to the given data using RANSAC %%%
% the general equation for a hyperbolic paraboloid is as follows:
% : z = a*x^2 - b*y^2
clc;
clear all;
close all;
```

```
load('paraboloid2_10.mat');
m = x(:,1)';
n = x(:,2)';
o = y(:,1)';
x = [];
y = [];
x = m;
y = n;
z = o;

nData = numel(x);
nSampLen = 2;
param_best = zeros(2,1);
max_n_trials = 10000;
num_inliner_best = 0;
dist_thr = 0.1;

for ii=1:max_n_trials

    pt_idx = ceil(nData .* rand(1, nSampLen));
    x_sub = x(pt_idx)';
    y_sub = y(pt_idx)';
    z_sub = z(pt_idx)';

    A = [x_sub.^2 -y_sub.^2];
    coef = A\z_sub; % calculating a and b

    % finding outlier points
    dist_pt = abs(z - (coef(1)*x.^2 - coef(2)*y.^2));

    num_inlier = sum(dist_pt<dist_thr);
    if num_inlier>num_inliner_best
        param_best(1) = coef(1);
        param_best(2) = coef(2);
        num_inliner_best = num_inlier;
    end
end

[xrange,yrange] = meshgrid(-6:0.1:6);
z_fit = param_best(1)*xrange.^2 - param_best(2)*yrange.^2 ;

figure(1);
plot3(x, y, z, 'ob', 'linewidth', 0.5);hold on;
surf(xrange,yrange,z_fit,'FaceColor','y') %Plot the surface
title('Find a hyperbolic paraboloid from the given data using RANSAC')
axis tight;grid on; hold off
xlabel('\bf X-axis');
ylabel('\bf Y-axis');
zlabel('\bf Z-axis');
set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')
```

3) Third step:

In this step, the neural network is used to develop a model which is fitted to our data. So, the coefficients calculated in the step 2 are used to formulate true values of the paraboloid function. Also, the data is generated when x is within -6 to 6.

Here, the number of input is 2, x and y, and the number of output is 1, z.

In addition, in this case, since the dataset includes outliers in this case, it may not be easy to find out the optimum curve. So, learning rate scheduling is used. Another point is that the learning rate is decreasing every 10 epoch.

I used the below configuration for solving this regression problem for dataset1 and dataset2:

learning rate: 0.005

learning rate scheduling: 0.8

number of neurons in input layer: 2

number of neurons in hidden layer: 20

number of neurons in output layer: 1

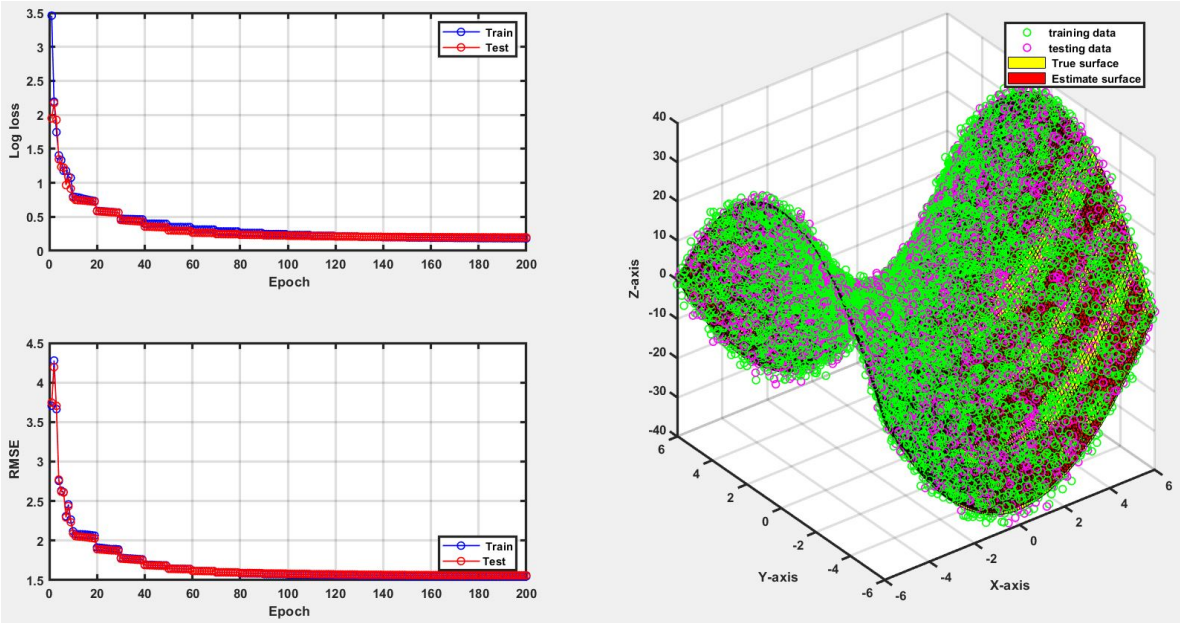
number of epoch: 200

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'linear'

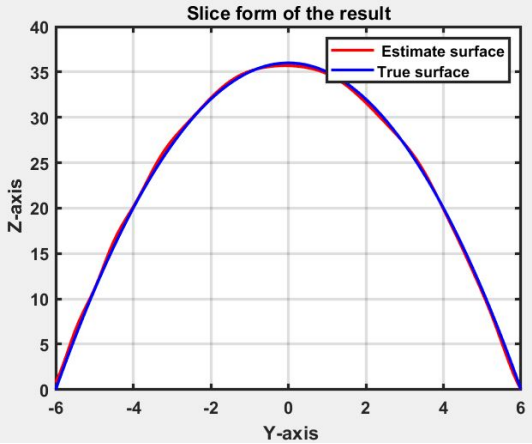
type of loss function: 'l2_loss'

Results for dataset1:

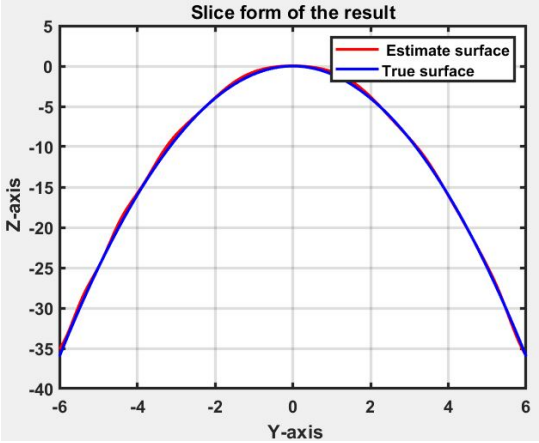


For showing the difference between the estimated surface and true surface, I made some slices at different locations of the surface, and plotted 2D curves for these slices. I used 8 slices with the following plane equations: "X=-6", "X=0", "X=3", "X=6", "Y=-6", "Y=0", "Y=3", "Y=6":

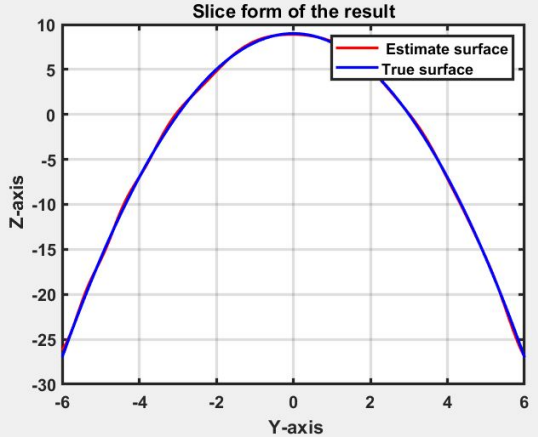
Slice 1 (intersection of the surface with x=-6)



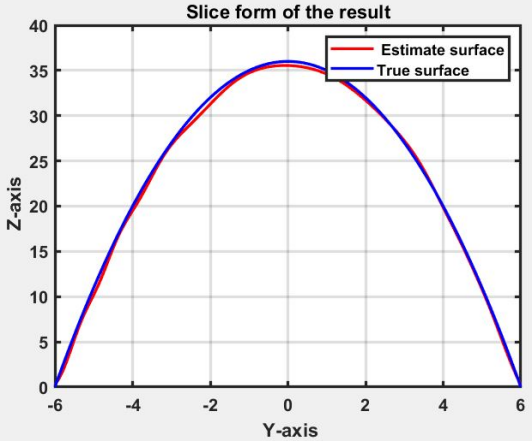
Slice 2 (intersection of the surface with x=0)



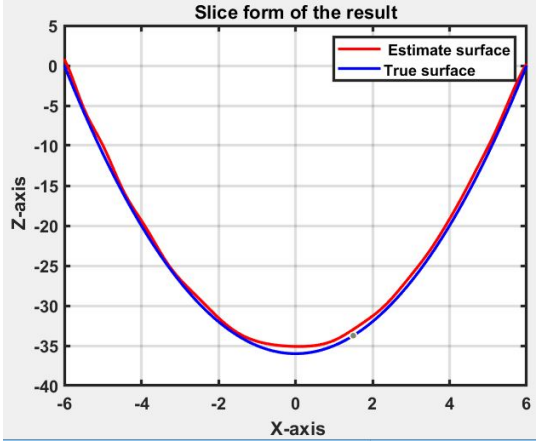
Slice 3 (intersection of the surface with x=3)



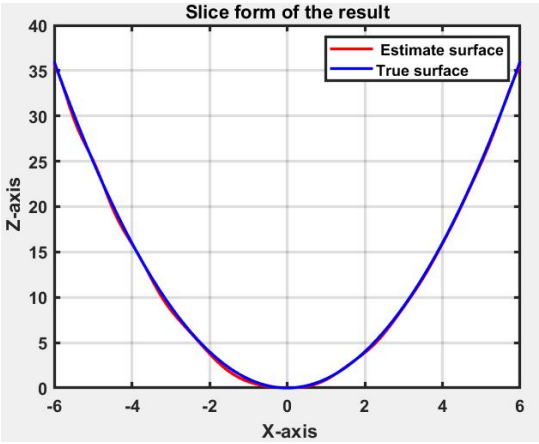
Slice 4 (intersection of the surface with x=6)

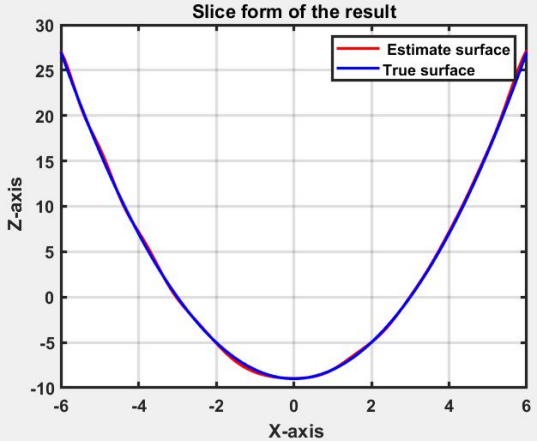
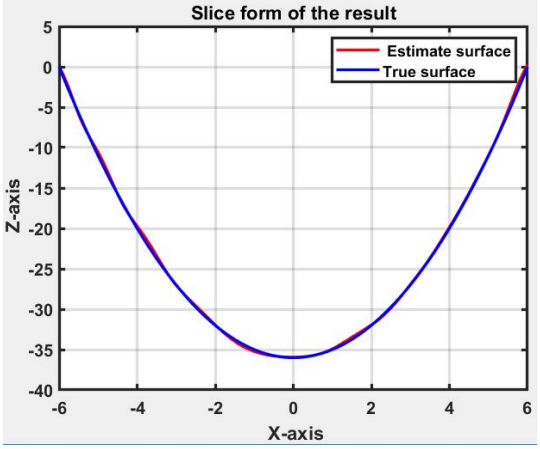


Slice 5 (intersection of the surface with y=-6)

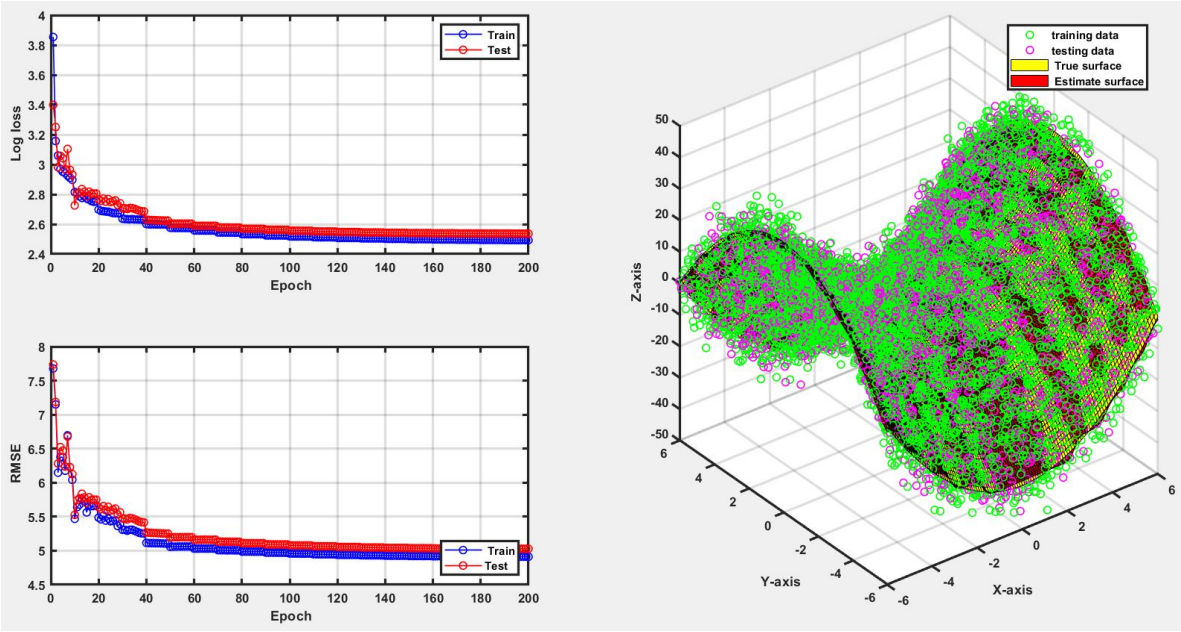


Slice 6 (intersection of the surface with y=0)



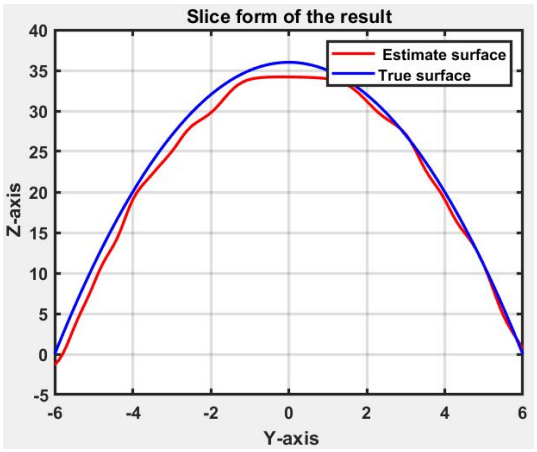
Slice 7 (intersection of the surface with y=3)	Slice 8 (intersection of the surface with y=6)
	

Results for dataset2:

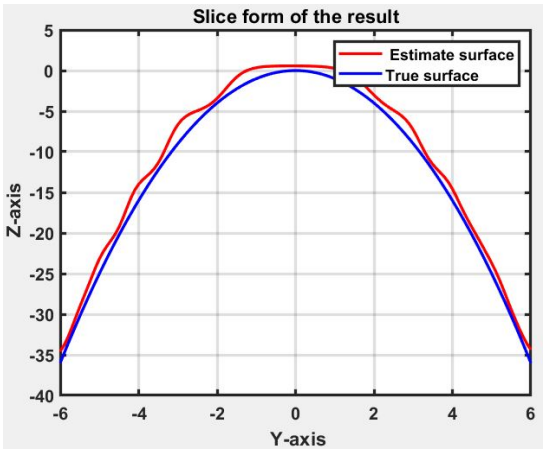


For showing the difference between the estimated surface and true surface, I made some slices at different locations of the surface, and plotted 2D curves for these slices. I used 8 slices with the following plane equations: "X=-6", "X=0", "X=3", "X=6", "Y=-6", "Y=0", "Y=3", "Y=6":

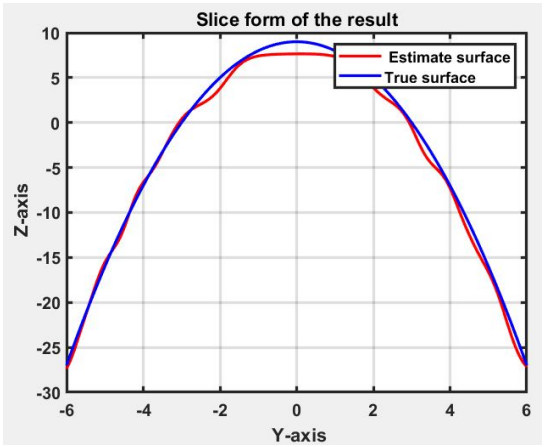
Slice 1 (intersection of the surface with x=-6)



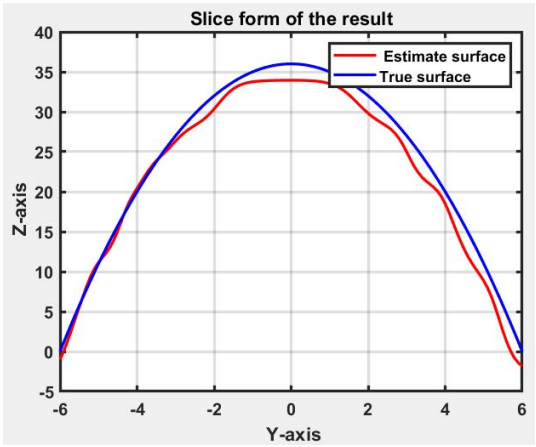
Slice 2 (intersection of the surface with x=0)



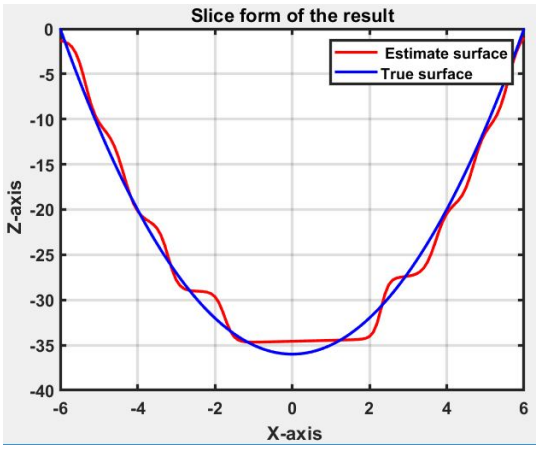
Slice 3 (intersection of the surface with x=3)



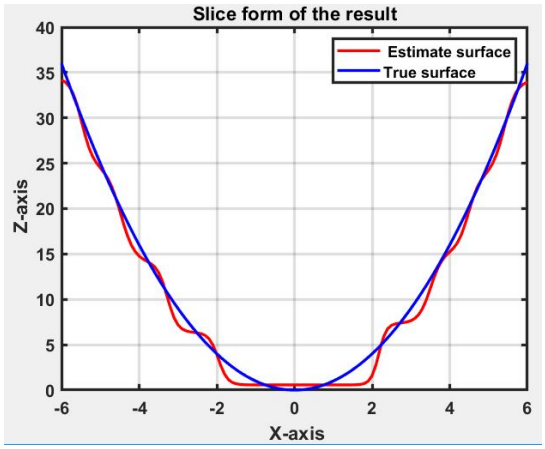
Slice 4 (intersection of the surface with x=6)

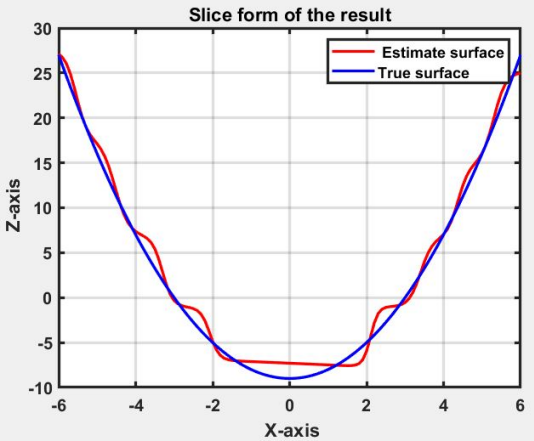
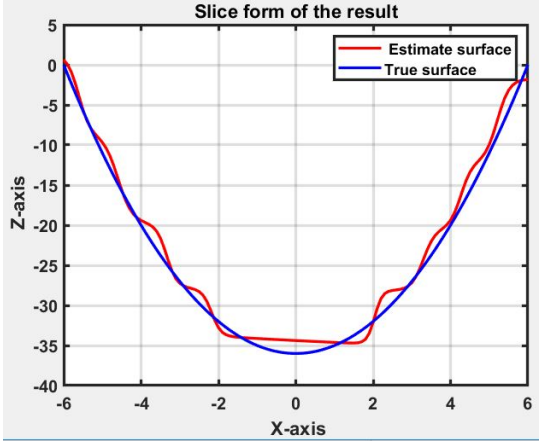


Slice 5 (intersection of the surface with y=-6)



Slice 6 (intersection of the surface with y=0)



Slice 7 (intersection of the surface with y=3)	Slice 8 (intersection of the surface with y=6)
	

By comparing the results of dataset1 and dataset2, we can observe that the neural network model for dataset1 is more accurate than the one for dataset2 (RMSE=5 for dataset2 while RMSE=1.5 for dataset1). The reason is that for the daraset1, the noisy data is closer to the true surface in comparison to the dataset2.

Neural network code for the hyperbolic paraboloid problem:

```
% Simple neural network demonstration
% This code shows how to implement a neural network with one hidden layer.
% You can change some basic parameters including
% learning rate, number of neurons, number of epoch, etc. You can also see
% similar demo from here:
% https://github.com/tensorflow/playground
% optimization: Stochastic gradient descent

% Dataset: Paraboloid dataset
% CM: It is added learning rate scheduling to fine-tune the curve.

% True curve
%{
% Hyperbolic paraboloid: z/c = a*x^2 - b*y^2
a = 1;
b = 1;
c = 1;
% The data is generated when x is within -6 to 6
%}

% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Updated by Saeed Hatefi Ardakani
% Last update: 04/19/2020
clear; close all; clc;

addpath('base');

dataset = ReadParaboloidData('data/paraboloid.mat');
net = set_nn; % you need to configure your network depending your dataset
test_nn(net, dataset);

function data = ReadParaboloidData(file)
```

```
data = load(file);
end

function net = set_nn
net.param.numepoch = 200; % number of epoch
net.param.learningrate = 0.005; % learning rate
net.learning_rate_scheduling = 0.8;

net.param.train_test_ratio = [0.7 0.3]; % training and testing division

net.input.node = 2;
net.layer.node = 20;
net.output.node = 1;

net.layer.activation = 'sigmoid';
net.output.activation = 'linear';
net.output.loss = 'l2_loss';

% layer: hidden layer (input to hidden)
net.layer.weight = rand(net.input.node, net.layer.node)-1; % set random value
between -1 ~ 1
net.layer.bias = zeros(net.layer.node, 1); % set bias as zero

% output: output layer (hidden to output)
net.output.weight = rand(net.layer.node, net.output.node)-1; % -1 ~ 1
net.output.bias = zeros(net.output.node, 1);

end

function test_nn(net, dataset)

train_test_ratio = net.param.train_test_ratio;
numepoch = net.param.numepoch;  % # of iteration

[x, t] = data2xt_Paraboloid(dataset); % data transformation

% training and testing split
[trainInd, ~, testInd] = ...
    dividerand(size(x,2),train_test_ratio(1), 0, train_test_ratio(2));

close all; clc;
figure('Name', 'Neural Network Training', 'NumberTitle','off', 'Position', [10
10 1300 600]);
h1 = subplot(2,2,1);
h2 = subplot(2,2,3);
h3 = subplot(2,2,[2 4]);

a = 1;
b = 1;
c = 1;
[xrange,yrange] = meshgrid(-6:0.1:6);
z_true = (xrange.^2/a^2 - yrange.^2/b^2)/c ;

loss = zeros(2, numepoch);
rmse = zeros(2, numepoch);
for ii=1:numepoch

    % learning rate scheduling
```

```
    if ~mod(ii,10)
        net.param.learningrate = net.param.learningrate *
net.learning_rate_scheduling;
    end
    % training
     [net, train_loss] = nn_train_net(net, x(:,trainInd), t(:, trainInd));

    % testing
    out = nn_test_net(net, x(:,testInd));

    % plotting loss curve
    loss(1, ii) = train_loss;

    test_loss = mean(nn_loss(out,t(:, testInd), net.output.loss, 'forward'));
    loss(2, ii) = test_loss; % testing loss
    show_log_loss(h1, ii, loss);

    % plotting RMSE (root mean square error)accuracy
    rmse_test = sqrt(mean((t(:, testInd)-out).^2));
    rmse(2, ii) = rmse_test;

    out_train = nn_test_net(net, x(:,trainInd));
    rmse_train = sqrt(mean((t(:, trainInd)-out_train).^2));
    rmse(1, ii) = rmse_train;

    show_rmse(h2, ii, rmse);

    % plotting: visualization
    out_graph = nn_test_net(net, x);

    axes(h3);
    plot3(x(1,trainInd)',x(2,trainInd)', t(:,trainInd)', 'og', 'linewidth', 1);
hold on;
    plot3(x(1,testInd)',x(2,testInd)', t(:,testInd)', 'om', 'linewidth', 1);

    surf(xrange,yrange,z_true,'FaceColor','y') %Plot the surface
    hold on;

    T = delaunay(x(1,:),x(2,:));
    trisurf(T,x(1,:)',x(2,:)',out_graph,'FaceColor','r');

    legend('training data', 'testing data','\bf True surface','\bf Estimate
surface',  'Location','northeast');
    xlabel('\bf X-axis');
    ylabel('\bf Y-axis');
    zlabel('\bf Z-axis');
    set(h3,'fontsize',10,'linewidth',2,'fontweight','bold')
    grid on; hold off;

end
end

function show_log_loss(cur_ax, epoch, loss)

axes(cur_ax);
plot(1:epoch, log(loss(1,1:epoch)), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, log(loss(2,1:epoch)), '-or', 'linewidth', 1);
```

```
legend('\bf Train','\bf Test', 'Location','northeast');
xlabel('\bf Epoch');
ylabel('\bf Log loss');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function show_rmse(cur_ax, epoch, accuracy)

axes(cur_ax);
plot(1:epoch, accuracy(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, accuracy(2,1:epoch), '-or', 'linewidth', 1);
legend('\bf Train', 'Test', 'Location','southeast')
xlabel('\bf Epoch');
ylabel('\bf RMSE');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function [x, t] = data2xt_Paraboloid(dataset)

t = dataset.y;
x = dataset.x;

n_data = numel(t);

% shuffling
rind = randperm(n_data);
x = x(rind,:)';
t = t';
t = t(:,rind);
end
```

Code for plotting slices (intersection of the surface with the plane equation (x=a or y=b)):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code for plotting slice (intersection of the surface with the
% plane equation (x=a or y=b))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

slice_type = 2;
% 1: intersection of the surface with the plane equation x=a_p
% 2: intersection of the surface with the plane equation y=a_p
a_p = 6;

if slice_type == 1
    % plotting estimaed curve
    TT = (find(x(slice_type,:)==a_p));
    gg = x(:,TT);
    outtt = out_graph(1,TT);
    [BB,II] = sort(gg(2,:));
    x_slice(1,:) = gg(1,II);
    x_slice(2,:) = gg(2,II);
```

```
    y_slice(1,:) = outtt(1,II);
    figure(1);
    plot(x_slice(2,:),y_slice(1,:),'-r','linewidth',2)
    hold on;

    % plotting True curve
    TT = (find(xrange(slice_type,:)==a_p));
    ss = yrange(:,TT);%xrange
    qq = z_true(:,TT);
    plot(ss,qq,'-b','linewidth',2)
    title('Slice form of the result')
    legend('\bf Estimate surface', 'True surface')
    xlabel('\bf Y-axis');
    ylabel('\bf Z-axis');
    grid on; hold off;
    set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')

elseif slice_type == 2
    % plotting estimaed curve
    TT = (find(x(slice_type,:)==a_p));
    gg = x(:,TT);
    outtt = out_graph(1,TT);
    [BB,II] = sort(gg(1,:));
    x_slice(1,:) = gg(1,II);
    x_slice(2,:) = gg(2,II);
    y_slice(1,:) = outtt(1,II);
    figure(1);
    plot(x_slice(1,:),y_slice(1,:),'-r','linewidth',2)
    hold on;

    % plotting estimaed curve
    TT = (find(xrange(slice_type,:)==a_p));
    ss = yrange(:,TT);%xrange
    qq = z_true(TT,:);
    plot(ss,qq,'-b','linewidth',2)
    title('Slice form of the result')
    legend('\bf Estimate surface', 'True surface')
    xlabel('\bf X-axis');
    ylabel('\bf Z-axis');
    grid on; hold off;
    set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')
end
```

(b) Search for real-world dataset from UCI Machine Learning data repository or other data repository. You can build either a regression or classification model. The performance of the model will not be evaluated but the model should be reasonable and acceptable. For instance, the accuracy of the binary classifier should not be less than 0.5. Please think about how to visualize your result that your model is learned from the actual data. If you download raw data from the web, you should pre-process or transform the data so that it become a trainable form. You will get some hints from the demos. You will get extra mark if you fully utilize the techniques that are implemented in the demos. Again, you can allow changing the code if you can improve the model performance.

Here, I built four models. One of them is regression problem, and the others are classification problems.

1: Iris Data Set (classification problem)

This is a classification problem for classifying three types of Iris plant. Iris is a genus of 260–300 species of flowering plants with showy flowers.

The input dimension is equal to 4 and the number of class is equal to 3.

The objective is finding the type of Iris plant using the 4 dimension vector input:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

It should be mentioned that sepal and petal are different parts of a flower. Sepal is female accessory reproductive organ while petal is male accessory reproductive organ.

The output is three class Classification as below:

- Iris Setosa
- Iris Versicolour
- Iris Virginica

As a result, we can estimate the probability that the input belongs to class 1 or 2 or 3. For the final decision, the input belongs to the class of the node with the highest value/probability (softmax).

In this example, normalizing the input data is used. The reason for using normalizing the data is that the range of 4 data is not clear, leading to produce interference in the neural network.

In this problem, the type of real output data is text data ("Iris Setosa", "Iris Versicolour", "Iris Virginica"). So, we need to transform it to the scalar values (trainable forms). To do so, built-in function "unique" is used. The below code is related to the function of the code for reading the data and transferring the output to the scalar values (1, 2, and 3):

```
function data = ReadIrisData(file)
fid = fopen(file);
data_cell = textscan(fid,'%f %f %f %f %s', 150, 'Delimiter',',');

fclose(fid);

[c,ia,ic] = unique(data_cell{5}); % text to scalar
data_cell(:,5) = {ic};

data = cell2mat(data_cell);
end
```

In addition, since we can get good result, there is no need to use learning rate scheduling for this problem.

We use the following configuration for solving the problem:

learning rate: 0.05

number of neurons in hidden layer: 6

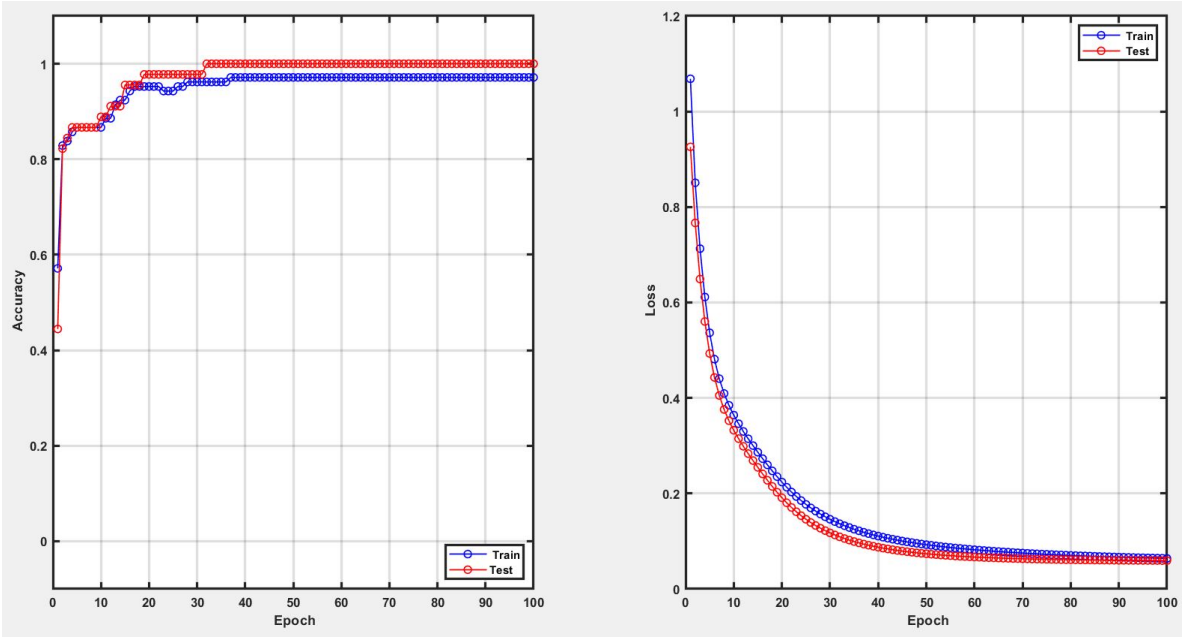
number of epoch: 100

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'

The below figure shows the final result of the code:



As you can see, the accuracy of the results is excellent (1), and loss function also experiences minimum state at the end of iteration (epoch).

Final Matlab code:

```
% Simple neural network demonstration
% This code shows how to implement a neural network with one hidden layer.
% You can change some basic parameters including
% learning rate, number of neurons, number of epoch, etc. You can also see
% similar demo from here:
% https://github.com/tensorflow/playground
% optimization: Stochastic gradient descent

% Dataset: Iris class dataset
% https://archive.ics.uci.edu/ml/datasets/Iris
% The dataset can be downloaded from the following link (data/iris.data)
% https://archive.ics.uci.edu/ml/machine-learning-databases/iris/

% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Updated by Saeed Hatefi Ardakani
% Last update: 04/21/2020
clear; close all; clc;

addpath('base');

dataset = ReadIrisData('data/iris.data');
net = set_nn; % you need to configure your network depending your dataset
test_nn(net, dataset);

function data = ReadIrisData(file)
fid = fopen(file);
data_cell = textscan(fid,'%f %f %f %f %s', 150, 'Delimiter',',');

fclose(fid);
```

```
[c,ia,ic] = unique(data_cell{5}); % text to scalar
data_cell(:,5) = {ic};

data = cell2mat(data_cell);
end

function net = set_nn
net.param.numepoch = 100; % number of epoch
net.param.learningrate = 0.05; % learning rate

net.param.train_test_ratio = [0.7 0.3]; % training and testing division

net.input.node = 4;
net.layer.node = 6;
net.output.node = 3;

net.layer.activation = 'sigmoid';
net.output.activation = 'softmax';
net.output.loss = 'cross_entropy';

% layer: hidden layer (input to hidden)
net.layer.weight = rand(net.input.node, net.layer.node)-1; % set random value
between -1 ~ 1
net.layer.bias = zeros(net.layer.node, 1); % set bias as zero

% output: output layer (hidden to output)
net.output.weight = rand(net.layer.node, net.output.node)-1; % -1 ~ 1
net.output.bias = zeros(net.output.node, 1);

end

function test_nn(net, dataset)

train_test_ratio = net.param.train_test_ratio;
numepoch = net.param.numepoch;  % # of iteration

[x, t] = data2xt_iris(dataset); % data transformation

% training and testing split
[trainInd, ~, testInd] = ...
    dividerand(size(x,2),train_test_ratio(1), 0, train_test_ratio(2));

close all; clc;
figure('Name', 'Neural Network Training', 'NumberTitle','off', 'Position', [10
10 1300 450]);
h1 = subplot(1,2,1);
h2 = subplot(1,2,2);

loss = zeros(2, numepoch);
accuracy = zeros(2, numepoch);

for ii=1:numepoch

    % training
    [net, train_loss] = nn_train_net(net, x(:,trainInd), t(:, trainInd));

    % testing
```

```
    out = nn_test_net(net, x(:,testInd));

    % plotting loss curve
    loss(1, ii) = train_loss;

    test_loss = mean(nn_loss(out,t(:, testInd), net.output.loss, 'forward'));
    loss(2, ii) = test_loss; % testing loss

    show_loss(h2, ii, loss);

    % plotting accuracy curve (check if the data is classified correctly)
    [~,out_clss] = max(out, [], 1); % predicted class
    [~,true_clss] = max(t(:, testInd), [], 1); % true class
    n_test = numel(out_clss);
    n_test_correct = sum(out_clss == true_clss);
    accuracy(2,ii) = n_test_correct/n_test;

    out_train = nn_test_net(net, x(:,trainInd));
    [~,out_train_clss] = max(out_train, [], 1); % predicted class
    [~,true_train_clss] = max(t(:, trainInd), [], 1); % true class
    n_train = numel(out_train_clss);
    n_train_correct = sum(out_train_clss == true_train_clss);
    accuracy(1, ii) = n_train_correct/n_train;

    show_acc(h1, ii, accuracy);
end
end

function show_acc(cur_ax, epoch, accuracy)

axes(cur_ax);
plot(1:epoch, accuracy(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, accuracy(2,1:epoch), '-or', 'linewidth', 1);
legend('\bf Train', 'Test', 'Location','southeast')
xlabel('\bf Epoch');
ylabel('\bf Accuracy');ylim([-0.1 1.1]);
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function show_loss(cur_ax, epoch, loss)

axes(cur_ax);
plot(1:epoch, loss(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, loss(2,1:epoch), '-or', 'linewidth', 1);

legend('\bf Train','\bf Test', 'Location','northeast');
xlabel('\bf Epoch');
ylabel('\bf Loss');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function [x, t] = data2xt_iris(dataset)

% 3-class
```

```
n_data = size(dataset,1);

out = dataset(:,end);
in = dataset(:,1:end-1);

% normalize each column
x = normalize(in, 1);
t = zeros(3, n_data);

for ii=1:n_data
    t(out(ii),ii) = 1;
end

% shuffling
rind = randperm(n_data);
x = x(rind,:)';
t = t(:,rind);
end
```

2: QSAR fish toxicity Data Set (regression problem)

This is a regression problem. The objective is developing quantitative regression QSAR models to predict acute aquatic toxicity towards the fish Pimephales promelas (fathead minnow) on a set of 908 chemicals.

Input:

The model comprised 6 molecular descriptors: MLOGP (molecular properties), CIC0 (information indices), GATS1i (2D autocorrelations), NdssC (atom-type counts), NdsCH ((atom-type counts), SM1_Dz(Z) (2D matrix-based descriptors).

Output:

The output is LC50 data (LOG(mol/L)), which is the concentration that causes death in 50% of test fish over a test duration of 96 hours. The output acts as the quantitative response.

Also, in this example, normalizing the input data is used. The reason for using normalizing the data is that the range of 6 data is between 0 and infinite number, leading to produce interference and diverge the neural network.

In addition, learning rate scheduling is used in this case. The reason is that by using learning rate scheduling, we can get more optimum result and our accuracy is improved.

We use the following configuration for solving the problem:

learning rate: 0.005

number of neurons in hidden layer: 10

number of epoch: 100

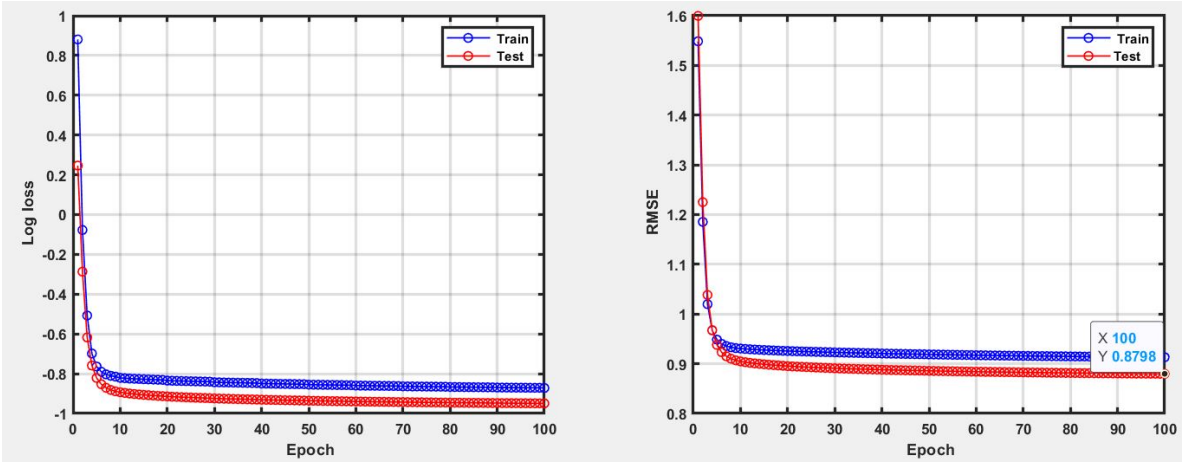
learning rate scheduling: 0.8

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'linear'

type of loss function: 'l2_loss'

The below figure shows the final result of the code:



As you can see, the accuracy of the results is good (RMSE=0.8798), and loss function also experiences minimum state at the end of iteration (epoch).

Final Matlab code:

```
% Simple neural network demonstration
% This code shows how to implement a neural network with one hidden layer.
% You can change some basic parameters including
% learning rate, number of neurons, number of epoch, etc. You can also see
% similar demo from here:
% https://github.com/tensorflow/playground
% optimization: Stochastic gradient descent

% Dataset: QSAR fish toxicity dataset
% https://archive.ics.uci.edu/ml/datasets/QSAR+fish+toxicity
% The dataset can be downloaded from the following link (qsar_fish_toxicity.csv)
% https://archive.ics.uci.edu/ml/machine-learning-databases/00504/

% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Updated by Saeed Hatefi Ardakani
% Last update: 04/21/2020

clear; close all; clc;

addpath('base');

dataset = ReadQsarData('data/qsar_fish_toxicity.csv');
net = set_nn; % you need to configure your network depending your dataset
test_nn(net, dataset);

function data = ReadQsarData(file)
fid = fopen(file);
data_cell = textscan(fid,'%f %f %f %f %f %f %f', 908, 'Delimiter',';');
data = cell2mat(data_cell);
end

function net = set_nn
net.param.numepoch = 100; % number of epoch
net.param.learningrate = 0.005; % learning rate
net.learning_rate_scheduling = 0.8;

net.param.train_test_ratio = [0.7 0.3]; % training and testing division

net.input.node = 6;
```

```
net.layer.node = 10;
net.output.node = 1;

net.layer.activation = 'sigmoid';
net.output.activation = 'linear';
net.output.loss = 'l2_loss';

% layer: hidden layer (input to hidden)
net.layer.weight = rand(net.input.node, net.layer.node)-1; % set random value
between -1 ~ 1
net.layer.bias = zeros(net.layer.node, 1); % set bias as zero

% output: output layer (hidden to output)
net.output.weight = rand(net.layer.node, net.output.node)-1; % -1 ~ 1
net.output.bias = zeros(net.output.node, 1);

end

function test_nn(net, dataset)

train_test_ratio = net.param.train_test_ratio;
numepoch = net.param.numepoch;  % # of iteration

[x, t] = data2xt_Qsar(dataset); % data transformation

% training and testing split
[trainInd, ~, testInd] = ...
    dividerand(size(x,2),train_test_ratio(1), 0, train_test_ratio(2));

close all; clc;
figure('Name', 'Neural Network Training', 'NumberTitle','off', 'Position', [10
10 1300 450]);
h1 = subplot(1,2,1);
h2 = subplot(1,2,2);

loss = zeros(2, numepoch);
rmse = zeros(2, numepoch);
for ii=1:numepoch

    % learning rate scheduling
    if ~mod(ii,10)
        net.param.learningrate = net.param.learningrate *
net.learning_rate_scheduling;
    end

    % training
     [net, train_loss] = nn_train_net(net, x(:,trainInd), t(:, trainInd));

    % testing
    out = nn_test_net(net, x(:,testInd));

    % plotting loss curve
    loss(1, ii) = train_loss;

    test_loss = mean(nn_loss(out,t(:, testInd), net.output.loss, 'forward'));
    loss(2, ii) = test_loss; % testing loss
    show_log_loss(h1, ii, loss);
```

```
    % plotting RMSE (root mean square error)accuracy
    rmse_test = sqrt(mean((t(:, testInd)-out).^2));
    rmse(2, ii) = rmse_test;

    out_train = nn_test_net(net, x(:,trainInd));
    rmse_train = sqrt(mean((t(:, trainInd)-out_train).^2));
    rmse(1, ii) = rmse_train;

    show_rmse(h2, ii, rmse);

end
end

function show_loss(cur_ax, epoch, loss)

axes(cur_ax);
plot(1:epoch, loss(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, loss(2,1:epoch), '-or', 'linewidth', 1);

legend('\bf Train','\bf Test', 'Location','northeast');
xlabel('\bf Epoch');
ylabel('\bf Loss');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function show_log_loss(cur_ax, epoch, loss)

axes(cur_ax);
plot(1:epoch, log(loss(1,1:epoch)), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, log(loss(2,1:epoch)), '-or', 'linewidth', 1);

legend('\bf Train','\bf Test', 'Location','northeast');
xlabel('\bf Epoch');
ylabel('\bf Log loss');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function show_rmse(cur_ax, epoch, accuracy)

axes(cur_ax);
plot(1:epoch, accuracy(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, accuracy(2,1:epoch), '-or', 'linewidth', 1);
legend('\bf Train', 'Test', 'Location','northeast')
xlabel('\bf Epoch');
ylabel('\bf RMSE');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function [x, t] = data2xt_Qsar(dataset)

n_data = size(dataset,1);
```

```
out = dataset(:,end);
in = dataset(:,1:end-1);

% normalize each column
x = normalize(in, 1);
t = out';

% shuffling
rind = randperm(n_data);
x = x(rind,:)';
t = t(:,rind);
end
```

3: Nursery Data Set (classification problem)

This is a classification problem for ranking applications for nursery schools and for making decision about the quality of the applications. It was used during several years in 1980's when there was excessive enrollment to these schools in Ljubljana, Slovenia, and the rejected applications frequently needed an objective explanation. The final decision depended on three subproblems: occupation of parents and child's nursery, family structure and financial standing, and social and health picture of the family.

The input dimension is equal to 8 and the number of classes is equal to 5.

Input:

1. parents: usual, pretentious, great_pret
2. has_nurs: proper, less_proper, improper, critical, very_crit
3. form: complete, completed, incomplete, foster
4. children: 1, 2, 3, more
5. housing: convenient, less_conv, critical
6. finance: convenient, inconv
7. social: non-prob, slightly_prob, problematic
8. health: recommended, priority, not_recom

Output: decision for the applications: (class attribute)

1. not_recom,
2. priority,
3. recommend,
4. spec_prior,
5. very_recom

In this problem, the type of data is text data. So, we need to transform it to the scalar values (trainable forms). To do so, built-in function "unique" is used. In addition, learning rate scheduling is used. Another point is that the learning rate is decreasing every 10 epoch.

We use the following configuration for solving the problem:

learning rate: 0.1

number of neurons in hidden layer: 18

number of epoch: 200

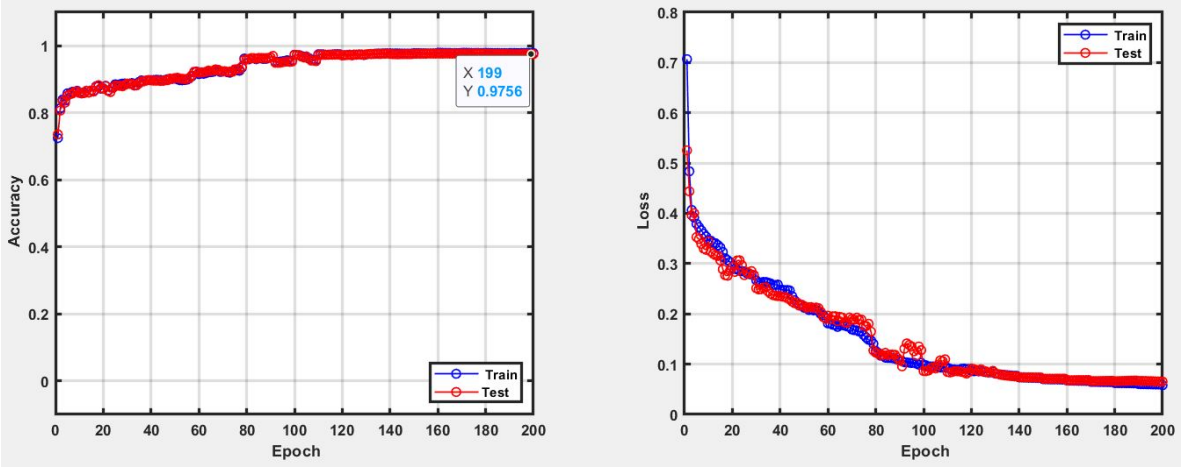
learning rate scheduling: 0.9

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'

The below figure shows the final result of the code:



As you can see, the accuracy of the result is 0.9756 (very good), and loss function experiences minimum state at the end of iteration (epoch).

Final Matlab code:

```
% Simple neural network demonstration
% This code shows how to implement a neural network with one hidden layer.
% You can change some basic parameters including
% learning rate, number of neurons, number of epoch, etc. You can also see
% similar demo from here:
% https://github.com/tensorflow/playground
% optimization: Stochastic gradient descent
% https://archive.ics.uci.edu/ml/datasets/Nursery
% https://archive.ics.uci.edu/ml/machine-learning-databases/nursery/

% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Updated by Saeed Hatefi Ardakani
% Last update: 04/23/2020
clear; close all; clc;

addpath('base');

dataset = ReadNurseryData('data/nursery.data');
net = set_nn; % you need to configure your network depending your dataset
test_nn(net, dataset);

function data = ReadNurseryData(file)

n_data = 12960;

fid = fopen(file);
data_cell = textscan(fid,'%s %s %s %s %s %s %s %s %s', n_data, 'Delimiter',',');
fclose(fid);

data = zeros(n_data, 9);
```

```
for ii=1:9
    [c,ia,ic] = unique(data_cell{ii}); % text to scalar
    data(:,ii) = ic;
end

%{
parents: usual, pretentious, great_pret
has_nurs: proper, less_proper, improper, critical, very_crit
form: complete, completed, incomplete, foster
children: 1, 2, 3, more
housing: convenient, less_conv, critical
finance: convenient, inconv
social: non-prob, slightly_prob, problematic
health: recommended, priority, not_recom
class: not_recom, priority, recommend, spec_prior, very_recom
%}
end

function net = set_nn
net.param.numepoch = 200; % number of epoch
net.param.learningrate = 0.1; % learning rate
net.learning_rate_scheduling = 0.9;

net.param.train_test_ratio = [0.7 0.3]; % training and testing division

net.input.node = 8;
net.layer.node = 18;
net.output.node = 5;

net.layer.activation = 'sigmoid';
net.output.activation = 'softmax';
net.output.loss = 'cross_entropy';

% layer: hidden layer (input to hidden)
net.layer.weight = rand(net.input.node, net.layer.node)-1; % set random value
between -1 ~ 1
net.layer.bias = zeros(net.layer.node, 1); % set bias as zero

% output: output layer (hidden to output)
net.output.weight = rand(net.layer.node, net.output.node)-1; % -1 ~ 1
net.output.bias = zeros(net.output.node, 1);

end

function test_nn(net, dataset)

train_test_ratio = net.param.train_test_ratio;
numepoch = net.param.numepoch;  % # of iteration

[x, t] = data2xt_Nursery(dataset); % data transformation

% training and testing split
[trainInd, ~, testInd] = ...
    dividerand(size(x,2),train_test_ratio(1), 0, train_test_ratio(2));

close all; clc;
figure('Name', 'Neural Network Training', 'NumberTitle','off', 'Position', [10
10 1300 450]);
```

```
h1 = subplot(1,2,1);
h2 = subplot(1,2,2);

loss = zeros(2, numepoch);
accuracy = zeros(2, numepoch);

for ii=1:numepoch

    % learning rate scheduling
    if ~mod(ii,10)
        net.param.learningrate = net.param.learningrate *
net.learning_rate_scheduling;
    end

    % training
    [net, train_loss] = nn_train_net(net, x(:,trainInd), t(:, trainInd));

    % testing
    out = nn_test_net(net, x(:,testInd));

    % plotting loss curve
    loss(1, ii) = train_loss;

    test_loss = mean(nn_loss(out,t(:, testInd), net.output.loss, 'forward'));

    loss(2, ii) = test_loss; % testing loss

    show_loss(h2, ii, loss);

    % plotting accuracy curve (check if the data is classified correctly)
    [~,out_clss] = max(out, [], 1); % predicted class
    [~,true_clss] = max(t(:, testInd), [], 1); % true class
    n_test = numel(out_clss);
    n_test_correct = sum(out_clss == true_clss);
    accuracy(2, ii) = n_test_correct/n_test;

    out_train = nn_test_net(net, x(:,trainInd));
    [~,out_train_clss] = max(out_train, [], 1); % predicted class
    [~,true_train_clss] = max(t(:, trainInd), [], 1); % true class
    n_train = numel(out_train_clss);
    n_train_correct = sum(out_train_clss == true_train_clss);
    accuracy(1, ii) = n_train_correct/n_train;

    show_acc(h1, ii, accuracy);
end
end

function show_acc(cur_ax, epoch, accuracy)

axes(cur_ax);
plot(1:epoch, accuracy(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, accuracy(2,1:epoch), '-or', 'linewidth', 1);
legend('\bf Train', 'Test', 'Location','southeast')
xlabel('\bf Epoch');
ylabel('\bf Accuracy');ylim([-0.1 1.1]);
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;
```

```
end

function show_loss(cur_ax, epoch, loss)

axes(cur_ax);
plot(1:epoch, loss(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, loss(2,1:epoch), '-or', 'linewidth', 1);

legend('\bf Train','\bf Test', 'Location','northeast');
xlabel('\bf Epoch');
ylabel('\bf Loss');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function [x, t] = data2xt_Nursery(dataset)

% 5-class
n_data = size(dataset,1);

out = dataset(:,end);
x = dataset(:,1:end-1);

% normalize each column
t = zeros(5, n_data);

for ii=1:n_data
    t(out(ii),ii) = 1;
end

% shuffling
rind = randperm(n_data);
x = x(rind,:)';
t = t(:,rind);
end
```

4: Glass Identification Data Set (classification problem)

This is a classification problem for classifying 7 types of glass.

The input dimension is equal to 9 and the number of class is equal to 7.

Input:

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron

It should be mentioned that I didn't consider input #1 as an input because it is only identification number. So, the new modified inputs are:

1. RI: refractive index
2. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
3. Mg: Magnesium
4. Al: Aluminum
5. Si: Silicon
6. K: Potassium
7. Ca: Calcium
8. Ba: Barium
9. Fe: Iron

Output: Type of glass: (class attribute)

1. building_windows_float_processed
2. building_windows_non_float_processed
3. vehicle_windows_float_processed
4. vehicle_windows_non_float_processed (none in this database)
5. containers
6. tableware
7. headlamps

In this example, normalizing the input data and learning rate scheduling are used.

We use the following configuration for solving the problem:

learning rate: 0.01

number of neurons in hidden layer: 20

number of epoch: 150

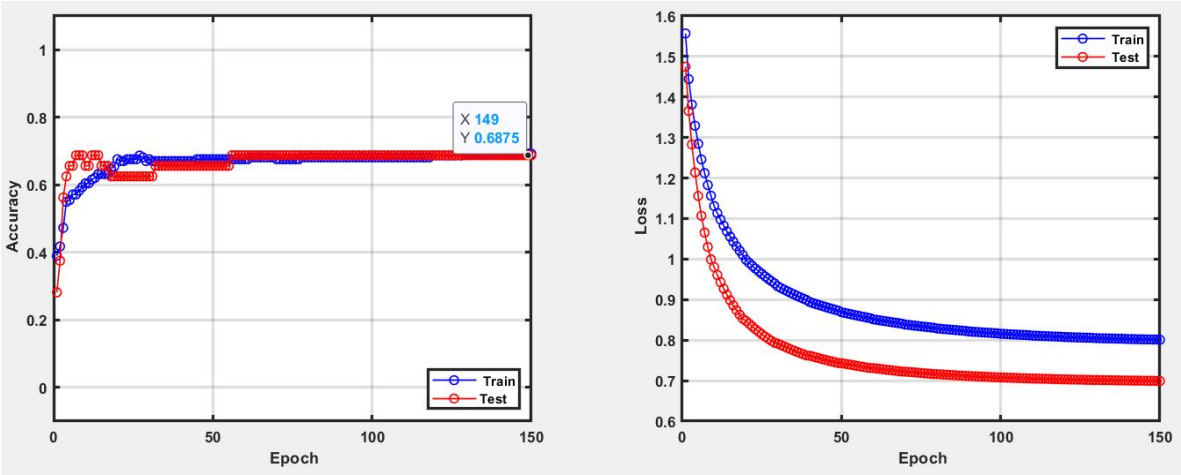
learning rate scheduling: 0.8

type of activation function for hidden layer: 'sigmoid'

type of activation function for output layer: 'softmax'

type of loss function: 'cross_entropy'

The below figure shows the final result of the code:



As you can see, the accuracy of the results is 0.69 (not excellent).

Final Matlab code:

```
% Simple neural network demonstration
% This code shows how to implement a neural network with one hidden layer.
% You can change some basic parameters including
% learning rate, number of neurons, number of epoch, etc. You can also see
% similar demo from here:
% https://github.com/tensorflow/playground
% optimization: Stochastic gradient descent

% Dataset: Glass Identification dataset
% https://archive.ics.uci.edu/ml/datasets/Glass+Identification
% The dataset can be downloaded from the following link (data/glass.data)
% https://archive.ics.uci.edu/ml/machine-learning-databases/glass/

% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Updated by Saeed Hatefi Ardakani
% Last update: 04/22/2020
clear; close all; clc;

addpath('base');

dataset = ReadGlassData('data/glass.data');
net = set_nn; % you need to configure your network depending your dataset
test_nn(net, dataset);

function data = ReadGlassData(file)
fid = fopen(file);
data_cell = textscan(fid,'%f %f %f %f %f %f %f %f %f %f %f', 214,
'Delimiter',',');
data = cell2mat(data_cell);
fclose(fid);
end

function net = set_nn
net.param.numepoch = 150; % number of epoch
net.param.learningrate = 0.01; % learning rate
net.learning_rate_scheduling = 0.8;

net.param.train_test_ratio = [0.85 0.15]; % training and testing division

net.input.node = 9;
net.layer.node = 20;
net.output.node = 7;

net.layer.activation = 'sigmoid';
net.output.activation = 'softmax';
net.output.loss = 'cross_entropy';

% layer: hidden layer (input to hidden)
net.layer.weight = rand(net.input.node, net.layer.node)-1; % set random value
between -1 ~ 1
net.layer.bias = zeros(net.layer.node, 1); % set bias as zero

% output: output layer (hidden to output)
net.output.weight = rand(net.layer.node, net.output.node)-1; % -1 ~ 1
net.output.bias = zeros(net.output.node, 1);
```

```
end

function test_nn(net, dataset)

train_test_ratio = net.param.train_test_ratio;
numepoch = net.param.numepoch;  % # of iteration

[x, t] = data2xt_Glass(dataset); % data transformation

% training and testing split
[trainInd, ~, testInd] = ...
    dividerand(size(x,2),train_test_ratio(1), 0, train_test_ratio(2));

close all; clc;
figure('Name', 'Neural Network Training', 'NumberTitle','off', 'Position', [10
10 1300 450]);
h1 = subplot(1,2,1);
h2 = subplot(1,2,2);

loss = zeros(2, numepoch);
accuracy = zeros(2, numepoch);

for ii=1:numepoch

    % learning rate scheduling
    if ~mod(ii,10)
        net.param.learningrate = net.param.learningrate *
net.learning_rate_scheduling;
    end

    % training
    [net, train_loss] = nn_train_net(net, x(:,trainInd), t(:, trainInd));

    % testing
    out = nn_test_net(net, x(:,testInd));

    % plotting loss curve
    loss(1, ii) = train_loss;

    test_loss = mean(nn_loss(out,t(:, testInd), net.output.loss, 'forward'));

    loss(2, ii) = test_loss; % testing loss

    show_loss(h2, ii, loss);

    % plotting accuracy curve (check if the data is classified correctly)
    [~,out_clss] = max(out, [], 1); % predicted class
    [~,true_clss] = max(t(:, testInd), [], 1); % true class
    n_test = numel(out_clss);
    n_test_correct = sum(out_clss == true_clss);
    accuracy(2,ii) = n_test_correct/n_test;

    out_train = nn_test_net(net, x(:,trainInd));
    [~,out_train_clss] = max(out_train, [], 1); % predicted class
    [~,true_train_clss] = max(t(:, trainInd), [], 1); % true class
    n_train = numel(out_train_clss);
    n_train_correct = sum(out_train_clss == true_train_clss);
```

```
    accuracy(1, ii) = n_train_correct/n_train;

    show_acc(h1, ii, accuracy);
end
end

function show_acc(cur_ax, epoch, accuracy)

axes(cur_ax);
plot(1:epoch, accuracy(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, accuracy(2,1:epoch), '-or', 'linewidth', 1);
legend('\bf Train', 'Test', 'Location','southeast')
xlabel('\bf Epoch');
ylabel('\bf Accuracy');ylim([-0.1 1.1]);
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function show_loss(cur_ax, epoch, loss)

axes(cur_ax);
plot(1:epoch, loss(1,1:epoch), '-ob', 'linewidth', 1); hold on;
plot(1:epoch, loss(2,1:epoch), '-or', 'linewidth', 1);

legend('\bf Train','\bf Test', 'Location','northeast');
xlabel('\bf Epoch');
ylabel('\bf Loss');
set(cur_ax,'fontsize',10,'linewidth',2,'fontweight','bold')
grid on; hold off;

end

function [x, t] = data2xt_Glass(dataset)

% 7-class
n_data = size(dataset,1);

out = dataset(:,end);
in = dataset(:,2:end-1);

% normalize each column
x = normalize(in, 1);
t = zeros(7, n_data);

for ii=1:n_data
    t(out(ii),ii) = 1;
end

% shuffling
rind = randperm(n_data);
x = x(rind,:)';
t = t(:,rind);
end
```