CS303B: Lab7 – Multilayer Perceptron

Aim: To understand how to train a neural network using backpropagation and conduct classification using multilayer perceptron.

The dataset

In this lab, we will use the iris dataset, which comes with MATLAB as a built-in dataset called 'fisheriris'. (If you are already familiar with the iris dataset, please skip this section)

The description of this dataset could also be found in

http://archive.ics.uci.edu/ml/datasets/Iris

The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Each sample is described by a feature vector containing the following attributes: (features or measurements)
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

Name of the three classes:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

Load in the dataset and data scaling

Load the dataset 'fisheriris' using the *load* command in MATLAB.

1. Load fisheriris. This will create two variables. *meas* is a matrix with each row containing the measures of the four attributes for each sample, and *species* is a column vector containing the ground truth of each sample, i.e., the type of iris plant. Out of the 150 items, 1-50 are setosa, 51-100 are versicolor and 101-150 are virginica.
2. Scale the data in each column of *meas* (i.e., each measurement) into the range of [0, 1], by the formula (x-min)/(max-min), with x being the measurements.
   You may try the following code:

```
for i=1:size(meas, 2)
   t = meas(:, i);
   t = (t – min(t(:))) ./ (max(t(:)) – min(t(:)));
   meas(:, i) = t;
end
```

Please play the NN with GUI before completing the following experiments. You can type *nnstart* in the command window to start.
https://ww2.mathworks.cn/help/deeplearning/ref/nnstart.html?requestedDomain=cn

Classification – Single Layer Perceptron

- **Simple classification: XOR**

1. Create the data.

Begin by creating the input (X) and target (T) data that should be used in the training.

```
% Create the data.
X = [[0; 0] [0; 1] [1; 0] [1;1]];
T = [0 1 1 0];
```

2. Plot the data.

The following code can be used to plot the data.

```
% Plot the data.
figure(1)
plot(X(1, 1), X(2, 1), 'ro')
hold on
grid on
plot(X(1, 4), X(2, 4), 'ro')
plot(X(1, 2:3), X(2, 2:3), 'bd')
xlim([-.25, 1.25])
ylim([-.25, 1.25])
```

3. Create a neural network.

We now want to create an MLP and train it on this data. From the problem definition it is clear that the MLP should have two input nodes and one output node. We will also create a hidden layer containing two nodes. We can create an MLP in MATLAB with the *feedforwardnet* command. To create such a network, called net, run the following.

```
net = feedforwardnet(2, 'traingd');
```

The arguments to *feedforwardnet* are, in order

i. hiddenSizes: row vector of one or more hidden layer sizes (default = 10). Since we want only one hidden layer (with size 2), it contains only a 2.
ii. trainFcn: a training algorithm to use when training the network (default = 'trainlm'). We use gradient descent training (*traingd*), which is MATLAB's name for batch backpropagation.

For more details, run the following command in the command window.

```
help feedforwardnet
```

4. Configure the neural network.

We use the logistic function (*logsig*) as the transfer function for the hidden layer, and the logistic function (*logsig*) as the transfer function for the output layer. We have to set *net.divideParam.trainRatio = 1; net.divideParam.valRatio = 0;* and *net.divideParam.testRatio = 0.* Then we will use the 4 samples to train the network. There is no validation set and test set. The data will be processed by *feedforwardnet* automatically but we do not want to do this. Therefore, we also need to modify the process function. Here is the code.

```
% Configure the net
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio   = 0; % validation set [%]
net.divideParam.testRatio  = 0; % test set [%]
net.inputs{1}.processFcns  = {}; % modify the process function for inputs
net.outputs{2}.processFcns = {}; % modify the process function for outputs
net.layers{1}.transferFcn = 'logsig'; % the transfer function for the first layer
net.layers{2}.transferFcn = 'logsig'; % the transfer function for the second layer
net.trainParam.lr = 1; % learning rate. You may need to adjust it in the experiment.
```

5. Train.

Now we have a neural network which has been configured according to the defined problem. It is time to train it. Please try the following code.

```
net = train(net, X, T);
```

This will open up a new training window, where information about the training is displayed as it is being performed. The information includes the number of epochs trained and the performance. You can click on the **Performance** button to open up a graph of the error as a function of the epoch number (logarithmic scale). Note that having such a graph open during training will slow MATLAB down, so set the plot interval to the maximum value (100 epochs) to mitigate the slowdown if you want the performance plot displayed during training.

The default setting when using *traingd* function is to train for at most 1000 epochs. You may need to change some settings to achieve satisfactory training results. The training parameters for the network's training algorithm are stored

in the structure *net.trainParam* (net is the name of your network). You can type *net.trainParam* in the command window to see the parameters and their current values. You can change all of the parameters by writing to the variables in this structure. Note that *net.trainParam.min_grad* gives the minimum gradient of the error function after which MATLAB will stop the training (if it is reached before the maximum number of epochs have passed).
To produce the output of the network, the *sim* function can be used. Feeding an input vector (or a vector of input vectors) to *sim* will return the network's output values for those inputs.

```
y = sim(net, [0; 0])  % return one output
Y = sim(net, X)       % return a vector of outputs
```

Another method to produce the output of the network is:

```
y = net([0; 0])  % return one output
Y = net(X)       % return a vector of outputs
```

Please try different activation function (Tanh and ReLU) for the hidden layer. You can use the following code to change the transfer function for the hidden layer.

```
% change the transfer function for the first layer to Tanh
net.layers{1}.transferFcn = 'tansig';

% change the transfer function for the first layer to ReLU
net.layers{1}.transferFcn = 'poslin';
```

- **Iris dataset classification**

1. Load the Iris dataset.

```
load fisheriris
```

2. Please use the method provided in section **load in the dataset and data scaling** to normalize the dataset.
3. Note that the input of the network created in the XOR classification task is a column vector. However, the size of *meas* is 150 by 4, which means that the feature vector is a row vector. We need to transpose the matrix *meas*. In MATLAB, we can do this by typing *meas = meas'*. Besides, we need to encode the label with one-hot vector. Please try the following code:

```
str = unique(species);
[found, idx] = ismember(species, str);
species_vec = full(ind2vec(idx'));
```

4. Create a neural network following the procedures provided in the XOR classification task. The configurations for this neural network are as follows:

```
% Configure the net
net.inputs{1}.processFcns  = {}; % modify the process function for inputs
net.outputs{2}.processFcns = {}; % modify the process function for outputs
net.layers{1}.transferFcn = 'logsig'; % the transfer function for the first layer
net.layers{2}.transferFcn = 'softmax'; % the transfer function for the second layer
net.performFcn = 'crossentropy'; the loss function
net.trainParam.lr = 0.1; % learning rate. You may need to adjust it in the experiment.
```

When the network is trained, the dataset will be divided into training set, validation set and test set automatically.

5. Now please divide the dataset manually. We can use the following code:

```
cvo = cvpartition(species, 'k', 2);
tridx = cvo.training(1);
teidx = cvo.test(1);
training_x = meas(tridx, :);
training_y = species(tridx);
test_x = meas(teidx, :);
test_y = species(teidx);
```

The code will divide the dataset into training set and test set. You can use the same method to divide the test set into validation set and new test set. Then please train the neural network on the training set and adjust the hyperparameters on the validation set. Finally, test your network on the new test set and report classification accuracy, AUC (area under ROC) of the network.

Hint: if the dataset is divided manually, we need to set *net.divideParam.trainRatio = 1; net.divideParam.valRatio = 0;* and *net.divideParam.testRatio = 0.*

Please try different activation function (Tanh and ReLU) for the hidden layer. Based on the XOR classification and Iris dataset classification, please discuss the effect of activation functions. Please try different hidden layer sizes and discuss the effect of the hidden layer sizes.

<u>Classification—two hidden layers perceptron</u>

Please repeat the same experiments (XOR classification and Iris dataset classification). However, you need to create a neural network with two hidden layers. **After you complete the experiments, please discuss the effect of the number of layers.**

Finally

Please try *patternnet* and deeper neural network. You can type *help patternnet* in the command window to get detailed information of *patternnet.*