

## EE5904/ME5404 Neural Networks: Homework #2

**Important note:** the due date is **01/03/2021**. You should submit your scripts to the folder in LumiNus. Late submission is not allowed unless it is well justified. Please include the MATLAB code or Python Code as attachment if computer experiment is involved.

### Q1. Rosenbrock's Valley Problem (10 Marks)

Consider the Rosenbrock's Valley function:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

which has a global minimum at  $(x, y) = (1, 1)$  where  $f(x, y) = 0$ . Now suppose the starting point is randomly initialized in the open interval  $(0, 1)$  for  $x$  and  $y$ , find the global minimum using:

a). Steepest (Gradient) descent method

$$w(k+1) = w(k) - \eta g(k)$$

with learning rate  $\eta = 0.001$ . Record the number of iterations when  $f(x, y)$  converges to (or very close to) 0 and plot out the trajectory of  $(x, y)$  in the 2-dimensional space. Also plot out the function value as it approaches the global minimum. What would happen if a larger learning rate, say  $\eta = 0.5$ , is used?

(5 Marks)

b). Newton's method (as discussed on page 13 in the slides of lecture Four)

$$\Delta w(n) = -H^{-1}(n)g(n)$$

Record the number of iterations when  $f(x, y)$  converges to (or very close to) 0 and plot out the trajectory of  $(x, y)$  in the 2-dimensional space. Also plot out the function value as it approaches the global minimum.

(5 Marks)

### Q2. Function Approximation (20 Marks)

Consider using MLP to approximate the following function:

$$y = 1.2 \sin(\pi x) - \cos(2.4\pi x) \quad \text{for } x \in [-2, 2].$$

The training set is generated by dividing the domain  $[-2, 2]$  using a uniform step length 0.05, while the test set is constructed by dividing the domain  $[-2, 2]$  using a uniform step length 0.01. You may use the MATLAB neural network toolbox to implement a MLP (see the Appendix for guidance) and do the following experiments:

a). Use the sequential mode with BP algorithm and experiment with the following different structures of the MLP: 1-n-1 (where  $n = 1, 2, \dots, 10, 20, 50, 100$ ). For each architecture plot out the outputs of the MLP for the test samples after training and compare them to the desired outputs. Try to determine whether it is under-fitting, proper fitting or over-fitting. Identify the minimal number of hidden neurons from the

experiments, and check if the result is consistent with the guideline given in the lecture slides. Compute the outputs of the MLP when  $x=-3$  and  $+3$ , and see if the MLP can make reasonable predictions outside of the domain of the input limited by the training set.

(7 Marks)

b). Use the batch mode with `trainlm` algorithm to repeat the above procedure.







(7 Marks)

c). Use the batch mode with `trainbr` algorithm to repeat the above procedure.

(6 Marks)

### Q3. Facial Attribute Recognition (40 Marks)

Multi-layer perceptron (MLP) can be used to solve real-world pattern recognition problems. In this assignment, MLP will be designed to handle a binary classification task. Specifically, students are divided into 3 groups based on matric numbers and each group is assigned with different tasks as illustrated in the following Table.

Group ID	Task	Example
1	Gender Classification	 vs.  Male [1]      Female [0]
2	Smile Face Detection	 vs.  Smile [1]      Non-Smile [0]
3	Glasses Wearing Detection	 vs.  Glass [1]      Non-Glass [0]

You may download the zipped dataset (**Face Database.zip**) from IVLE. After unzipping, you will find two folders: *TrainImages* and *TestImages* that hold the training set and test set respectively. As demonstrated in the above Table, the training/test data are 1,000/250 RGB images of human faces; and the ground-truth labels associated with each image are given in the *.att* file that shares the same filename with corresponding image. It is noted that there are 3 binary labels in each *.att* file as shown below, but you only need one of them according to your assigned group.

1		male[1]/female[0]	for group 1
2		smile[1]/non-smile[0]	for group 2
3		glass[1]/non-glass[0]	for group 3

In order to find your group, you need to calculate '**mod(LTD, 3) + 1**' where *LTD* is the last two digits of your matric number, e.g. A1234567X is assigned to group  $\text{mod}(67, 3) + 1 = 2$  (Smile Face Detection).

**Please specify the group ID in your report. Take note that if you have selected wrongly, there will be some mark deduction!**

All the images are provided in JPEG format with size 101\*101. You can use **I = imread(filename)** to read these image files, where *filename* specifies the relative path to an image (for code efficiency, you may use function **dir()** to get filenames of all the images within a folder). The returned value **I** is an array (101-by-101-by-3 in this assignment) containing the image data. For example,

```
I = imread('TrainImages/Abba_Eban_0001.jpg');
```

will read image 'Abba\_Eban\_0001.jpg' from the training set into MATLAB workspace. For simplicity, grayscale images are used in this assignment, and you can convert the RGB image into grayscale image by function **rgb2gray()**:

```
G = rgb2gray(I);
```

Then, you could get the grayscale image as a 101-by-101 array which can be displayed using **imshow()**. In order to efficiently process all the image data, you may need to rearrange the matrix form data **G** into a vector by

```
V = G(:);
```

and the resulting **V** is a column vector whose elements are taken column-wisely from **G**. Subsequently, you could group all the training images together using

```
train_images = [V1, V2, ...];
```

and group all the test images together following the same way. In the next, these matrices (101\*101-by-image\_number) are used as input to the networks.

As mentioned before, the ground-truth labels are stored in the *.att* file associated with each image and can be extracted by

```
L = load('TrainImages/Abba_Eban_0001.att');
l = L(i);
```

where **i** is your group ID and **l** is a binary number holding the ground-truth label of image 'Abba\_Eban\_0001.jpg' for your assigned task.

You are required to complete the following tasks:

- a) For your assigned task, plot and analyse the label distribution of both the training set and test set.

**(2 Marks)**

- b) Apply Rosenblatt's perceptron (single layer perceptron) to your assigned task. After the training procedure, calculate the classification accuracy for both the training set and test set, and evaluate the performance of the perceptron.

**(8 Marks)**

- c) The original input dimension is 10,201 ( $101 \times 101$ ), which may be redundant and leave space for reduction. Try to naively downsample the images or apply a more sophisticated technique like PCA to these images. Then, retrain the perceptron in b) with the dimensionally reduced images and compare the results. (you may use `imresize()` and `pca()` in this task)

**(6 Marks)**

- d) Apply MLP to your assigned task using batch mode training. After the training procedure, calculate the classification accuracy for both the training set and test set, and evaluate the performance of the network.

**(10 Marks)**

- e) Apply MLP to your assigned task using sequential mode training. After the training procedure, calculate the classification accuracy for both the training set and test set, and evaluate the performance of the network. Compare the result with that of d) and make your recommendation on these two approaches.

**(10 Marks)**

- f) You may notice that all the images, either for training or test, are already aligned by placing eyes at a certain location. Do you think it is necessary to do so? Justify your answer.

**(4 Marks)**

**Important note:** There are many design and training issues to be considered when you apply neural networks to solve real world problems. We have discussed most of them in the lecture four. Some of them have clear answers, some of them may rely on empirical rules, and some of them have to be determined by trial and error. I believe that you will have more fun playing with these design parameters and making your own judgment rather than solving the problem with a prescribed set of parameters. Hence, there is no standard answer to this problem, and the marking will be based upon the whole procedure rather than the final classification accuracy. (Use “help” and “doc” commands in MATLAB to get familiar with the functions that you don't know and Google everything that confuse you.)

## Appendix

1. Create a feed-forward back propagation network using MATLAB toolbox using:

```
net = patternnet(hiddenSizes, trainFcn, performFcn)
```

where the arguments are specified as follows:

hiddenSizes -- Row vector of one or more hidden layer sizes (default = 10);  
trainFcn -- Training function (default = 'trainscg');  
performFcn -- Performance function (default = 'crossentropy').

**trainFcn** specifies the optimization algorithm based on which the network is updated during training, and there are many choices:

- Backpropagation training functions that use Jacobian derivatives (these algorithms can be faster but require more memory than gradient backpropagation):

trainlm -- Levenberg-Marquardt backpropagation.  
trainbr -- Bayesian Regulation backpropagation.

- Backpropagation training functions that use gradient derivatives (these algorithms may not be as fast as Jacobian backpropagation):

trainbfg -- BFGS quasi-Newton backpropagation.  
traincgb -- Conjugate gradient backpropagation with Powell-Beale restarts.  
traincgf -- Conjugate gradient backpropagation with Fletcher-Reeves updates.  
traincgp -- Conjugate gradient backpropagation with Polak-Ribiere updates.  
traingd -- Gradient descent backpropagation.  
traingda -- Gradient descent with adaptive lr backpropagation.  
traingdm -- Gradient descent with momentum.  
traingdx -- Gradient descent w/momentum & adaptive lr backpropagation.  
trainoss -- One step secant backpropagation.  
trainrp -- RPROP backpropagation.  
trainscg -- Scaled conjugate gradient backpropagation.

**performFcn** specifies the cost/objective function that measures the performance of network during training, and there are many choices:

mae -- Mean absolute error performance function.  
mse -- Mean squared error performance function.  
sae -- Sum absolute error performance function.  
sse -- Sum squared error performance function.  
crossentropy -- Cross-entropy performance.  
msespars -- Mean squared error performance function with L2 weight and sparsity regularizers.

It is very difficult to know which training function and performance function guarantee the best performance for a given problem. It depends on many factors, including the complexity of the problem, the number of samples in training set, the number of weights and biases in the network, and whether the network is being used for pattern recognition or function approximation (regression), etc. You are **encouraged** to try different training functions and compare their performance.

The following example shows how to design a pattern recognition network to classify iris flowers.

```
[x,t] = iris_dataset;
net = patternnet(10);
net = train(net, x, t);
view(net)
y = net(x);
perf = perform(net, t, y);
classes = vec2ind(y);
```

More details about patternnet() can be found by typing 'help patternnet' and 'doc patternnet' in MATLAB command line.

2. Different training functions have different parameters which are stored in 'net.trainParam'. For example, the function parameters for 'traincgf' are

Show Training Window Feedback --	showWindow: true
Show Command Line Feedback --	showCommandLine: false
Command Line Frequency --	show: 25
Maximum Epochs --	epochs: 1000
Maximum Training Time --	time: Inf
Performance Goal --	goal: 0
Minimum Gradient --	min_grad: 1e-06
Maximum Validation Checks --	max_fail: 6
Sigma --	sigma: 5e-05
Lambda --	lambda: 5e-07

Similarly, the parameter of performance functions are stored in 'net.performParam'. For example, the function parameters for 'crossentropy' are

Regularization Ratio --	regularization: 0
Normalization --	normalization: 'none'

The choosing of these parameters are task-dependent. You can keep the default values since they could guarantee a moderate performance; however, in order to achieve a better performance, you are **encouraged** modify these parameters based on your tasks.

3. You can train the network using MATLAB toolbox:

```
[net, tr] = train(net, X, T)
```

where the input arguments are

net -- Network

X -- Network inputs

T -- Network targets (default = zeros)

and returns

net -- Newly trained network

tr -- Training record (epoch and perf)

More details about train() can be found by typing 'help train' and 'doc train' in MATLAB command line.

4. After training, the weights in the hidden neurons are stored in the 'net' object. For example, for the same problem mentioned above, after training, type 'net' in the command line of MATLAB, you may obtain the following message:

net =

Neural Network

name: 'Pattern Recognition Neural Network'

userdata: (your custom info)

dimensions:

numInputs: 1

numLayers: 2

numOutputs: 1

numInputDelays: 0

numLayerDelays: 0

numFeedbackDelays: 0

numWeightElements: 10

sampleTime: 1

connections:

biasConnect: [1; 1]

inputConnect: [1; 0]

layerConnect: [0 0; 1 0]

outputConnect: [0 1]

subobjects:

input: Equivalent to inputs{1}  
output: Equivalent to outputs{2}

inputs: {1x1 cell array of 1 input}  
layers: {2x1 cell array of 2 layers}  
outputs: {1x2 cell array of 1 output}  
biases: {2x1 cell array of 2 biases}  
inputWeights: {2x1 cell array of 1 weight}  
layerWeights: {2x2 cell array of 1 weight}

functions:

adaptFcn: 'adaptwb'  
adaptParam: (none)  
derivFcn: 'defaultderiv'  
divideFcn: 'dividerand'  
divideParam: .trainRatio, .valRatio, .testRatio  
divideMode: 'sample'  
initFcn: 'initlay'  
performFcn: 'crossentropy'  
performParam: .regularization, .normalization  
plotFcns: {'plotperform', plottrainstate, ploterrhist,  
plotconfusion, plotroc}  
plotParams: {1x5 cell array of 5 params}  
trainFcn: 'trainscg'  
trainParam: .showWindow, .showCommandLine, .show, .epochs,  
.time, .goal, .min\_grad, .max\_fail, .sigma,  
.lambda

weight and bias values:

IW: {2x1 cell} containing 1 input weight matrix  
LW: {2x2 cell} containing 1 layer weight matrix  
b: {2x1 cell} containing 2 bias vectors

methods:

adapt: Learn while in continuous use  
configure: Configure inputs & outputs  
gensim: Generate Simulink model  
init: Initialize weights & biases  
perform: Calculate performance  
sim: Evaluate network outputs given inputs  
train: Train network with examples



view: View diagram  
unconfigure: Unconfigure inputs & outputs

evaluate:            outputs = net(inputs)

You may use 'net.LW' and 'net.b' to check the detailed values of weights and biases. Besides, all the information of the trained network is stored in the object 'net'. You may type 'doc' command to open the help manual and search for 'net' (network properties) to find more details.

5. The 'train()' function mentioned above provides batch learning mode only. In order to enable the sequential/incremental learning mode, please refer to <http://www.mathworks.com/help/nnet/ug/neural-network-training-concepts.html>

The most important step is to make sure that the inputs are presented as a cell array of sequential vectors.

A sample MATLAB code for sequential training is as follows:

```
function [ net, accu_train, accu_val ] = train_seq( n, images, labels,
train_num, val_num, epochs )
% Construct a 1-n-1 MLP and conduct sequential training.
%
% Args:
%   n: int, number of neurons in the hidden layer of MLP.
%   images: matrix of (image_dim, image_num), containing possibly
%           preprocessed image data as input.
%   labels: vector of (1, image_num), containing corresponding label of
%           each image.
%   train_num: int, number of training images.
%   val_num: int, number of validation images.
%   epochs: int, number of training epochs.
%
% Returns:
%   net: object, containing trained network.
%   accu_train: vector of (epochs, 1), containing the accuracy on training
%               set of each epoch during training.
%   accu_val: vector of (epochs, 1), containing the accuracy on validation
%             set of each epoch during training.

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

% 2. Construct and configure the MLP
```

```

net = patternnet(n);

net.divideFcn = 'dividetrain'; % input for training only
net.performParam.regularization = 0.25; % regularization strength
net.trainFcn = 'traingdx'; % 'trainrp' 'traingdx'
net.trainParam.epochs = epochs;

accu_train = zeros(epochs,1); % record accuracy on training set of
each epoch
accu_val = zeros(epochs,1); % record accuracy on validation set of
each epoch

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

    display(['Epoch: ', num2str(i)])
    idx = randperm(train_num); % shuffle the input
    net = adapt(net, images_c(:,idx), labels_c(:,idx));
    pred_train = round(net(images(:,1:train_num))); % predictions on
training set
    accu_train(i) = 1 - mean(abs(pred_train-labels(1:train_num)));
    pred_val = round(net(images(:,train_num+1:end))); % predictions
on validation set
    accu_val(i) = 1 - mean(abs(pred_val-labels(train_num+1:end)));

end
end

```

You can copy this .m file into your folder and modify it according to your task.