EE5904/ME5404 Neural Networks: Homework #2

Important note: the due date is <mark>27/02/2022</mark>. 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 (15 Marks)**
Consider the Rosenbrock's Valley function:
$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$
a) Show that it has a global minimum at (x, y) = (1,1) where f(x, y) = 0.

(5 Marks)

Now suppose the starting point is randomly initialized in the open interval (-1, 1) for x and y, find the global minimum using:

b). Steepest (Gradient) descent method
$$w(k+1) = w(k) - \eta g(k)$$

with learning rate η = 0.001. Here the weight vector refers to the two-dimensional vector [x, y]. 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 η = 1.0, is used?

(5 Marks)

c). 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. <mark>**Function Approximation**</mark> **(20 Marks)**
Consider using MLP to approximate the following function:
$$y = 1.2\sin(\pi x) - \cos(2.4\pi x) \qquad \text{for } x \in [-1.6, 1.6].$$

The training set is generated by dividing the domain [-1.6, 1.6] using a uniform step length 0.05, while the test set is constructed by dividing the domain [-1.6, 1.6] 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 <mark>BP algorithm</mark> and experiment with the following different structures of the MLP: 1-n-1 (where n = 1, 2, ..., 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. Image Classification (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, i.e. animals vs. man-made objects. Specifically, students are divided into 3 groups based on matric numbers and each group is assigned with different dataset as illustrated in the following Table.

Group ID	**Animals** with label **1**	**Man-Made objects** with label **0**
0	cat	airplane
1	dog	automobile
2	deer	ship

You may download the zipped dataset (e.g. group_0.zip) from IVLE. After unzipping, you will find two folders respectively, one of which is named by an animal and the other of which is named by a man-made object. You should use "0" as the label of man-made objects during training and "1" for animals. In each folder, there are 500 images indexing from 000 to 499. The training set you used should consists of images in each folder indexing from 000 to 449, so there are totally 900 images for training. The remaining images in each folder indexing from 450 to 499 are used as the validation set, so there should be totally 100 images.

All the images are provided in RGB format with size 32*32*3. To simplify the problem, I recommend you transforming colorful images into grayscale, so that each image is in size of 32*32. Besides, you can also reshape the matrix into a one-dimension vector with size 1024. (Hints: Two built-in functions "rgb2gray" and "reshape" would help; Please make full use of the official documentation, you can search all built-in functions from Matlab official website.)

You are required to complete the following tasks:

a) Apply Rosenblatt's perceptron (single layer perceptron) to the dataset of your assigned group. After the training procedure, calculate the *classification accuracy* for *both* the training set and validation set, and evaluate the performance of the network.

(6 Marks)

b) The global mean and variance of a dataset may influence the stability of training and the final performance of the model obtained. You can try to calculate the global mean and variance of the whole dataset, then *subtract* the mean value from each image and *divide* each image by the variance. *Compare* the result with that in a) and *explain* it.

(8 Marks)

c) Apply MLP to the dataset of your assigned group using *batch mode* training. After the training procedure, calculate the classification accuracy for both the training set and validation set, and evaluate the performance of the network.

(8 Marks)

d) Please determine whether your trained MLP in c) is overfitting. If so, please specify when (i.e. after which training epoch) it becomes overfitting. Try weights regularization and observe if it helps. (you may set the regularization strength by 'performParam.regularization')

(6 Marks)

e) Apply MLP to the dataset of your assigned group using *sequential mode training.* After the training procedure, calculate the classification accuracy for both training set and validation set, and evaluate the performance of the network. Compare the result to part c), and make your recommendation on the two approaches.

(8 Marks)

f) Try to propose a scheme that you believe could help to improve the performance of your MLP and please explain the reason briefly.

(4 Marks)

Important note: There are many <mark>design and training issues</mark> 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 confuses 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 <mark>use Jacobian derivatives (these algorithms can be faster but require more memory than gradient backpropation</mark>):

    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.
msesparse -- 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
```

<div align="center">

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

</div>

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 <mark>sequential training</mark> is as follows:

```matlab
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 eopch during trainig.
```

```matlab
%  accu_val: vector of (epochs, 1), containing the accuracy on
validation
%           set of each eopch during trainig.

    % 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.