Tutorial 4
(Neural Computing)

Artur S. d'Avila Garcez
Department of Computer Science, City, University of London

EXERCISE 9

Download *tutorial4files* from Moodle, unzip and open Matlab file *nnTrain.m.* This file contains a simple Matlab implementation (outside the neural net toolbox) of the backpropagation learning algorithm for single-hidden layer networks. If you run this file it will perform cross-validation on the promoter dataset as seen in class, mapping DNA sequences as input to promoter/non-promoter classifications in the output. The code splits the data into M folds and it trains M networks, computing the average accuracy.

Check in the code how the average accuracy is calculated. Does it use the mean squared error function seen in class or *hit and miss* counts? Does it refer to the training set or the validation set accuracy?

Find where in the code the number of folds can be specified. Change the number of folds and run the file, inspecting how this changes the accuracy and running time.

Why should the data be *shuffled* when using cross-validation? Why should *shuffle* be turned off for reproducibility?

Change the various *stopping criteria* and inspect the results. Analyse how the use of *number of epochs* alone as stopping criterion compares with the use of various values for *Delta*.

You can also change the *number of hidden neurons*, the *learning rate*, the *learning rate decay*, and the *momentun constant* and inspect results.

Plot the evolution of the training set error and the validation set error.

You can use smote oversampling or not. Should using it make any difference in the case of this dataset?

How would you go about implementing *weight decay*?

EXERCISE 10 Backpropagation through multiple hiddden layers

Open file `nn_lab.m`. The most important function in this script is
`model=train_nn(conf,Ws,bs,trn_dat,trn_lab,vld_dat,vld_lab)`

This will be used to train a neural network with backpropagation. Here, the structure of the network and the training parameters are defined in `conf`, a data structure consisting of the following fields:

- hidNum: An array of number of hidden units for each layer. For example, conf.hidNum = [10 20] defines a network with 2 hidden layers with 10 and 20 units respectively.
- activationFnc: A list of activation functions for each layer (bottom-up). For example: conf.activationFnc = {'tansig','tansig','logsig'} defines activation functions 'tansig','tansig','logsig' for hidden layers 1, 2 and the output layer respectively.
- eNum: Number of training epochs
- bNum: Number of batch in one epoch
- sNum: Number of samples in one batch
- params: An array consisting of learning parameters. params(1) = params(2) are the learning rates [1]. params(3) is the momentum and params(4) is the regularization cost.
- E_STOP is the early stopping criteria. If the validation accuracy does not improve after E_STOP epochs then the training should stop.

Function `get_data_from_file` loads data from a .mat file.

[1] The use of two params is for initial learning rate and final learning rate which will be implemented in next labs

A note on batch learning: Normally, updating the parameters takes into account the whole data set at a time, i.e. averaging the update for all training samples. However, for large data sets it is common to split the data into small batches and perform updates for each batch. In `train_nn.m` we first loop over all epochs (see `for=1:conf.eNum`. For each epoch, we loop over all batches (see `forb=1:conf.bNum`). Inside this loop, we perform the update.

Change the number of hidden layers and train the network.

In `nn_lab.m` set `CASE=1` and test your code with the glass dataset. This dataset can be used to create a neural network that classifies glass either as window or non-window depending on the glass chemistry. In the code, we divide the data into training and validation. The data consists of the following attributes: 1. Refractive index; 2. Sodium (unit measurement: weight percent in corresponding oxide); 3. Magnesium; 4. Aluminum; 5. Silicon; 6. Potassium; 7. Calcium; 8. Barium; 9. Iron. The labels represent two classes: 1. Window glass; and 2. Non-window glass.

After running the first test, set `CASE=2` to test with MNIST Handwritten dataset. The dataset consist of either 5,000 or 20,000 training samples, 10,000 validation and 10,000 test samples. Each sample is an image of a handwritten digit with size 28×28, which we vectorize into a vector with 784 elements.

Inspect the weight vectors at each network layer. Can you spot if your network suffers from the problem of vanishing gradients?

Implement the Early Stopping criteria and compare with your previous results. Improve the code for your own use.