COMP4702/COMP7703 - Machine Learning

Prac 6 –Multilayer Perceptron Neural Networks

Aims:

- To gain some experience in constructing single and multilayer perceptron networks and solving problems with them.
  → To produce some assessable work for this subject.

Procedure:

Matlab has an extensive Neural Network toolbox that we will make use of in this Prac. It has been developed over many years and has lots of different options and functionality. Try not to be distracted by all the features that we *won't* be using! You could potentially use python (scikit-learn?) or another language to do this prac, but it mostly uses Matlab as a tool rather than doing a lot of programming, so Matlab is probably the easiest option.

Introduction to the Matlab NN Toolbox

Several GUI tools are provided. Look under the APPS menu on the Matlab main window and run the pattern recognition helper – or type "nprtool" from the command line.

→ Work through this step-by-step guide to creating and training a neural network to perform classification. Read all of the information at each step (clicking "Next"!) and note what is happening.
→ Some example datasets are provided (which are from the UCI Repository), including the familiar Iris dataset. Open the breast cancer dataset. Use the default training/validation/test split for the data. Choose whatever number of hidden units you like (greater than 2!). Click "Train" to train your network. Spend some time looking at the various outputs available (e.g. click "Performance" in the "Neural Network Training" window that should be open).
→ **Question 1:** Take note of the performance (i.e. Training, Test and Validation errors) that your network achieved. Use the "Train again" and "Adjust network size" buttons a few times (just spend a few minutes on this trial-and-error!) and see how it affects the performance. Record your final training, validation and test errors.
→ Click through to "Save results" and generate a "Simple script" which appears in your Matlab editor window. Read through this code which reproduces what you just did with the GUI tool.

There are several details here that are different from what we have discussed in lectures.

The default training algorithm is "scaled conjugate gradient". Change this to standard backpropagation (traingd). Read the Matlab Help for the traingd function to see what default parameters are being used.

Experiment with x and y. How do your results compare with your results from Q1?

→ **Question 2:** increase and decrease the learning rate value and record the effect it has on the results. Hints:

- You will probably need to turn off some conditions that terminate the training process (maxfail and mingrad in traingd) and might also need to increase the maximum number of epochs (1 epoch = 1 weight update, after passing the entire training set through the network to calculate the error gradient).
- Account for the fact that retraining the network does not always lead to the same result.

MNIST Data

Question 3: Train an MLP on the MNIST data that we have used previously. You will need to make several "design choices" to do this. Make sure you split your dataset into a training, validation and test set. Your answer for this question should describe the design choices you made to create and train your network and a report of your results. You can also include error graphs or any other output you feel is useful.

The MNIST dataset is available at
https://lvdmaaten.github.io/tsne/

The following code may be used in formatting the data. You will have to change it slightly to use the test and train dataset.

```
load('mnist_train.mat');
labels = [];
for i = 1:10
    labels = [labels, train_labels == i];
end
```

Question 4: The weight matrix and bias vector in first hidden layer apply a linear transformation before the activation function is applied. The *i*th row of the weight vector contains the weightings of the 728 inputs to the *i*th neuron. The importance of each input on the *i*th neuron can then be visualised by drawing the *i*th row of the weight vector as a greyscale image. Visualise 9 rows of the weight matrix. Here are some hints:

→ One way to get the weight matrix of the trained network is to click "Matlab Matrix Only Function" in the "Deploy Solution" dialogue of nprtool. You will need to copy and paste IW1_1 and x1_step1_keep into a new script.
→ Each row will have to be normalised so that every element is in [0,1].
→ reshape() and imshow() may be useful.