# 159.372 Intelligent Machines
# Lab 4: The Perceptron

Stephen Marsland

## 1 Introduction

We are going to do a fairly simple classification problem to get used to using the Perceptron. Download the Perceptron code and the demonstrations of it for both the logic functions and Pima Indian datasets (you will also need to download the Pima dataset from the UCI Machine Learning repository at `http://www.ics.uci.edu/~mlearn/`) and check that you can make them work. Read the code to see what it is doing; the main points were covered in the lecture.

### 1.1 The Confusion Matrix

One thing that wasn't covered in the lecture is how to show how well the classifier is doing. If you look at the output from the Pima Indian code it tells you the percentage accuracy on the test data, and also prints out a matrix of results. This is a confusion matrix. It is a nice simple idea, which is to make a square matrix that contains all the possible classes in both the horizontal and vertical directions. We list the classes along the top of a table as the outputs, and then down the left-hand side as the targets. So for example, the element of the matrix at $(i, j)$ tells us how many input patterns were put into class $i$ in the targets, but class $j$ by the network. Anything on the leading diagonal is a correct answer. Suppose that we have three classes: $C_1, C_2$, and $C_3$. Now we count the number of times that the output was class $C_1$ when the target was $C_1$, then when the target was $C_2$, and so on until we've filled in the table:

| | Outputs | | |
|---|---|---|---|
| | $C_1$ | $C_2$ | $C_3$ |
| $C_1$ | 5 | 1 | 0 |
| $C_2$ | 1 | 4 | 1 |
| $C_3$ | 2 | 0 | 4 |

This table tells us that, for the three classes, most examples were classified correctly, but two examples of class $C_3$ were misclassified as $C_1$, and so on. Writing the code to compute this is not too difficult, and for a small number of classes, it is a nice way to look at the outputs. If you just want one number, then it is possible to divide the sum of the elements on the leading diagonal by the sum of all of the elements in the matrix, which gives the fraction of correct responses.

## 2 Your Task

Once you are happy with what the code does, download the file `mnist.pkl.gz` from Stream. If asked if you want to uncompress it, don't – the code below assumes that it is compressed. This is some examples from the MNIST numbers dataset, which is a commonly used dataset for testing machine learning algorithms. Have a look at `http://yann.lecun.com/exdb/mnist/` to find out more. In order to read the dataset in, you should use the following lines of code, which will produce three different datasets – training, testing, and validation.

```
import cPickle, gzip, numpy

# Load the dataset
f = gzip.open('mnist.pkl.gz', 'rb')
```

```
train_set, valid_set, test_set = cPickle.load(f)
f.close()
```

The data itself consists of a set of $28 \times 28$ pixel images of numerical digits taken from US zip codes on envelopes. The idea is to learn to recognise the different numbers 0-9. To see what the images look like, and to get to grips with the data, plot a couple of them using something like:

```
pl.imshow(np.reshape(train_set[0][0,:],[28,28]))
```

where you will have needed to import numpy as `np` and pylab as `pl`. Each element of `train_set[0]` is a 784 row vector that is the values of the pixels, while `train_set[1]` is the correct digit (so `train_set[1][2]` is 4, which is the correct digit for the third image).

Set up a Perceptron that learns the training set and reports results on the validation and test sets. Compare the accuracy on the three sets. Are there particular pairs of numbers that get confused? What difference does changing the learning rate make? Also, have a look at a random collection of the digits and see how well you do by eye – some of them are very difficult!