Handwritten Digit Classification using Neural Networks

Thomas A. E. Smith

Electronics and Computer Science
University of Southampton

May 17, 2012

Abstract

This report assesses the use of a multi-layer perceptron or neural network to classify handwritten digits. An overview of the system is given, followed by sample input and a discussion of the network architecture. A method of analysing the performance is given, and the performances of a number of different architectures are compared.

1 Overview

In order to classify the handwritten digits, a neural network was implemented in python, and trained using Scipy's fmin_cg solver for the conjugate gradient method. This required implementing a neural network structure using Numpy's Matrix classes for fast C-based matrix multiplication, and constructing valid cost and gradient functions for the neural network problem. fmin_cg was then used to minimise the cost J given Θ_1, Θ_2 and the gradient function, over several iterations.

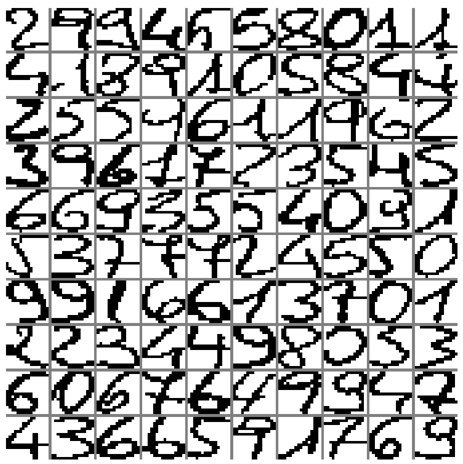

Figure 1: Sample handwritten digits

1.1 Architecture

The neural network architecture is largely governed by the format of the input and output data. As the images are 16x16 pixels, and belong to one of ten classes, there are 256 input nodes, and 10 output nodes. A single layer of hidden nodes has been used, as this should be sufficient to capture the variation within the data without requiring the additional processing burden of surplus layers. For the results given below, 16 hidden nodes were used as this gave better accuracy on the test data.

2 Results

The best way to analyse the performance of the neural network seems to be the Matthews Correlation Coefficient (1), which compares the True and False Positives and Negatives for each class i. It is well suited to highly imbalanced classes, and so the coefficient is calculated for each of the classes, and then the average taken to give a measure of the performance of the neural network as a whole.

$$MCC_i = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{1}$$

Using the MCC, a 16-hidden-node neural network has an accuracy of 0.897. For comparison, a 50-hidden-node network achives 0.878, but takes longer to train, and a 10-hidden-node network is only able to obtain an accuracy of 0.798.

		Actual class									
		0	1	2	3	4	5	6	7	8	9
Predicted class	0	12	-	-	-	1	-	-	-	-	-
	1	-	13	1	-	-	-	-	-	-	-
	2	-	-	11	-	-	-	-	-	-	1
	3	-	-	-	13	-	-	-	-	-	2
	4	-	-	-	-	19	-	-	2	-	-
	5	1	1	-	1	-	11	-	-	-	-
	6	-	-	-	-	1	-	17	-	-	-
	7	-	-	-	-	-	-	-	17	-	-
	8	-	-	-	-	-	1	-	-	13	-
	9	-	-	-	2	-	1	-	-	-	19

Table 1: Confusion matrix after 50 iterations for a 9:1 test:train split and 16 hidden nodes

An advantage of using Python to implement the neural network is the availability of other libraries, such as the PIL image library used to generate 1, and 2 below, which 'visualises' the hidden nodes, or rather the inputs which cause them to activate highest. It can be seen that the hidden nodes act as 'stroke detectors' to a certain degree: most easily in the top right where upper- and lower-horizontal-line identifiers can be seen.

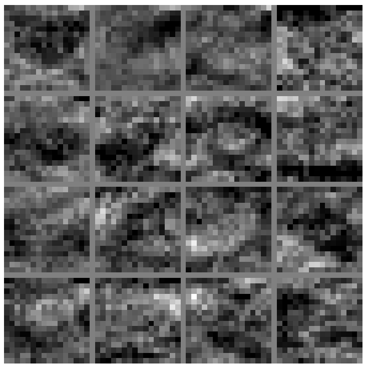

Figure 2: Hidden nodes - 'stroke detectors'