Week 4

Handwriting Recognition

1.1. Objectives

In this laboratory session you will use the general purpose two-layer MLP from laboratory two to perform automatic recognition of human handwriting. Specifically, you will devise and train an MLP to recognise digits from the MNIST database. Your objectives are:

1. Learn about the MNIST database and the format of the images
2. Learn how to read files in Python and display the MNIST images
3. Implement a two-layer MLP to perform classification on the dataset
4. Train the network, and evaluate the performance against a test set
5. Tweak the network parameters to try and improve the performance

At the end of this laboratory session you should have a practical neural network that can solve quite a complex task, this is the starting point for practical pattern recognition. You are free to modify or extend this in any way you want for your future coursework.

1.2. Introduction

We will be the same tools as the previous laboratory (i.e Visual Studio Code & the Anaconda Python distribution). Follow the instructions from Laboratory 1 to create a new project, be sure to select Anaconda as the Python environment to run.

MNIST

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. Recognising human handwriting is an ideal test for neural networks, because the problem is well defined but relatively difficult. The MNIST database contains 60,000 training images and 10,000 testing images. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23%.

> MNIST Files
>
> The MNIST database in CSV format is available to download from Moodle. There are four files, the full test and training sets, and the smaller subsets of 100 training and 10 test sets.

Reading Files

The images are stored in Comma Separated Value (CSV) file format, which is easily readable in Python. You do not need to worry too much about how the data is represented, essentially the first value is the label (the actual number represented) and the subsequent values are a 28 x 28 array of pixels (giving a total of 784). To begin with, you should work on the smaller subset of MNIST, which provides you with 100 training examples and 10 test examples. In code listing 5.1, I have given you an example of how to open the smaller training set, read the first image, and display it using matplotlib. Figure 5.1 shows the resulting plot, in this case the first sample in the training set is the number 5. You should run this example and take a look at the first few images from both of the smaller training and test sets.

Draw Number

```
# Import numpy for arrays and matplotlib for drawing the numbers

import numpy
import matplotlib.pyplot as plt

# Open the 100 training samples in read mode
data_file = open("mnist_train_100.csv", 'r')

# Read all of the lines from the file into memory
data_list = data_file.readlines()

# Close the file (we are done with it)
data_file.close()

# Take the first line (data_list index 0, the first sample), and split it up based on the commas
# all_values now contains a list of [label, pixel 1, pixel 2, pixel 3, ... ,pixel 784]
all_values = data_list[0].split(',')

# Take the long list of pixels (but not the label), and reshape them to a 2D array of pixels
image_array = numpy.asfarray(all_values[1:]).reshape((28, 28))

# Plot this 2D array as an image, use the grey colour map and don't interpolate
plt.imshow(image_array, cmap='Greys', interpolation='None')
plt.show()
```

Code 5.1: drawNumber.py

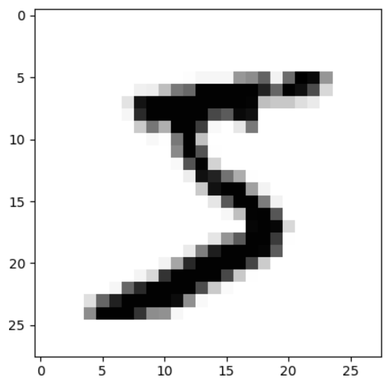

Figure 5.1: Greyscale plot of the first example from the training set of 100 numbers (in this case, the number 5).

Table 5.1: Suggested parameters for two-layer MNIST MLP.

Parameter	**Suggested Value**
Input Nodes	784
Hidden Nodes	100
Output Nodes	10
Learning Rate	0.3

1.3. Getting Started

If you have not already done so, create a new Python project using the instructions given in the lab from week 1. Type in the example code that plots the handwritten numbers, and make sure you can run the code. Read through the comments and make sure you understand the principles at work before you continue with the neural network example.

Configuring the Network

Before we can jump in and start using our neural network to classify the images, we need to think about the topology of network that we want to use. We will also need to think about the format of the input and output data. We know that the input will be a vector of 784 pixels, so it is easy to set the number of input nodes. The output needs to tell us what number the network thinks was in the image, the easiest way to do this is to have 10 output nodes, each one corresponding to the probability that the image contained that number. So, for example a value of 0.99 at output node 0 would mean that the network was certain that the image contained the number 0. The number of hidden nodes and the learning rate are, at this stage, best guess. I have put some starting values in Table 5.1.

Training and Testing the Network

In Code Listing 5.2, is the code to train the network using the 100 sample training set (note that in the code the neural network is called n). There are two important aspects to note. Firstly, the pixels in each image are in the range 0...255, I have rescaled them to be in the range 0.01..1. It is often helpful to have non-zero inputs, as a zero input can cause the training process to become stuck (as changing to associated weight has no effect on the output). The second key aspect is that I create a target vector with 0.01 indicating "not this digit" and 0.99 indicating "this digit". In the previous labs you should have observed that the networks output never reaches 1 or 0 exactly, this is because the logistic sigmoid function is *asymptotic* - if we set targets of 1 or 0 then the training process becomes saturated at the asymptote. Therefore, you may have noticed that your MLP performed worse at the simple logic functions than you hand crafted Perceptron, which used the hard threshold function. In Code Listing 5.3, I have provided example code that tests the network using the 10-sample test set. A scorecard is created, with a 1 appended for each correct classification, and a 0 for an incorrect classification. Finally, the performance score is calculated as the percentage of correct classifications.

Training

```
# Load the MNIST 100 training samples CSV file into a list
training_data_file = open(”mnist_train_100.csv”, ’r’)
training_data_list = training_data_file.readlines()
training_data_file.close()

# Train the neural network on each trainingsample
for record in training_data_list:
    # Split the record by the commas
    all_values = record.split(’,’)
    # Scale and shift the inputs from 0..255 to 0.01..1
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
    # Create the target output values (all 0.01, except the desired label which is 0.99)
    targets = numpy.zeros(output_nodes) + 0.01
    # All_values[0] is the target label for this record
    targets[int(all_values[0])] = 0.99
    # Train the network
    n.train(inputs, targets)
pass
```

Code 5.2: mnistTrain.py

Testing

```
# Load the MNIST test samples CSV file into a list
test_data_file = open(”mnist_test_10.csv”, ’r’)
test_data_list = test_data_file.readlines()
test_data_file.close()

# Scorecard list for how well the network performs, initially empty
scorecard = []

# Loop through all of the records in the test data set
for record in test_data_list:
    # Split the record by the commas
    all_values = record.split(’,’)
    # The correct label is the first value
    correct_label = int(all_values[0])
    print(correct_label, ”Correct label”)
    # Scale and shift the inputs
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
    # Query the network
    outputs = n.query(inputs)
    # The index of the highest value output corresponds to the label
    label = numpy.argmax(outputs)
    print(label, ”Network label”)
    # Append either a 1 or a 0 to the scorecard list
    if (label == correct_label):
        scorecard.append(1)
        else:
        scorecard.append(0)
        pass
    pass

# Calculate the performance score, the fraction of correct answers
scorecard_array = numpy.asarray(scorecard)
print(”Performance = ”, (scorecard_array.sum() / scorecard_array.size)*100, ’%’)
```

Code 5.3: mnistTest.py

1.4. Exercises

Spend some time familiarising yourself with the code for training and testing the network with the MNIST dataset. You do not need to fully understand the internal workings, but you should understand the basic flow of the code. Here are some exercises to try:

1. Using the template network class from week 3, create a neural network with the parameters given in Table 5.1 Note: you will need to create variables for the network parameters such as output nodes.
2. Train this network using the MNIST100 training set, and then test it using the MNIST10 testing set. What is your networks performance? Mine was 60%, but there will be small variances due to the random initial weights.
3. Display the images that were incorrectly classified, do you think that you (as a human), could have got those classifications correct?
4. Modify the network topology, the learning rate and any other parameters you desire – what is the highest network performance you can get?
5. Re-run the process using the full size MNIST training and testing sets. This may take some time. What is the performance now? Why is it different than before?