**UCI**[**Optical Recognition of Handwritten Digits**](http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of_Handwritten+Digits)

CST3170 Artificial Intelligence module 2nd coursework

Geraldas Kastauna (M00651656)

Middlesex University

London, United Kingdom

GK468@live.mdx.ac.uk

**Description:**

For this coursework students are required to build a machine learning system to categorise one of the UCI digit tasks. Student should develop the system on your own from scratch in JAVA programming language, run a two-fold test, and report the results.

The data is from the University of California at Irvine's Machine Learning Repository. It's the [Optical Recognition of Handwritten Digits Data Set](http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of_Handwritten+Digits). This gives two data sets, training set and a test set. Module leader converted them to two data sets - [data set 1](http://www.cwa.mdx.ac.uk/cst3170/coursework/cw2DataSet1.csv), and [data set 2](http://www.cwa.mdx.ac.uk/cst3170/coursework/cw2DataSet2.csv)that should be used by students system.

More info: [http://www.cwa.mdx.ac.uk/cst3170/coursework/CourseWork2.html](http://www.cwa.mdx.ac.uk/cst3170/coursework/CourseWork2.htm)

**Algorithm:**

For this task I decided to build a multilayer perceptron that uses backpropagation for learning. Multilayer perceptron is a feedforward artificial neural network that usually uses supervised learning techniques and carries out learning using backpropagation. What it essentially does is adjusts the weights between connected neurons. In my system:

Input layer has 64 neurons

Hidden Layer has 20 neurons

Output Layer has 10 neurons

I chose this algorithm because I am interested in artificial neural networks

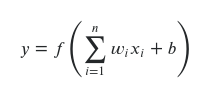
Process is as follows:

1. Have an input layer, hidden layer (could be more than one) and output layer.

2. Set all weights random (optimal values are between -1 and 1).

3. Input layer neuron is connected one-to-all using weights to hidden layer neurons, then hidden layer neurons are connected one-to-all using weights to output layer (or another hidden layer if there is more than one hidden layer).

4. We need to find dot product for each neuron in hidden layer using dot product



where *w* is weight from input to hidden neuron, *x* is input values and *b* is bias (usually equals to 1).

5. Then we need to apply activation function. I used sigmoid activation function in my system. Which is as follows:

*x* is neuron value after dot product is found.

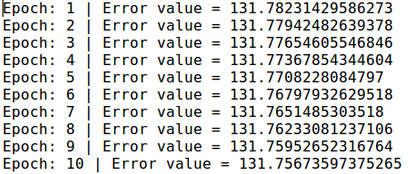
6. Feed forward this with random weights to hidden layer and repeat the dot product and apply sigmoid activation function through each layer until we get to the output layer. Using random weights results will most likely be very low depending on the amount of inputs the perceptron had. For this categorisation, the system usually got between 8% and 11% accuracy using random weights.

7. Then it calculates the error using formula:



Where *d* is the target value and *y* is the value produced by the perceptron.

How error value going down in my system:

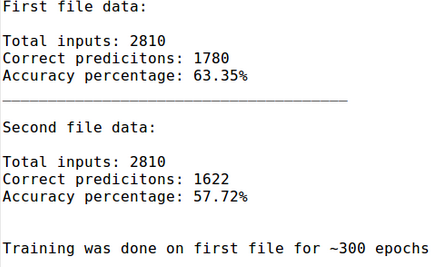
**It takes around 30 seconds to go through 1 epoch.**

8. Then it uses gradient descent for change in each weights:

where y is the output of the previous neuron and n is the [*learning rate.*](https://en.wikipedia.org/wiki/Learning_rate)

*9. It backpropagates back from output to hidden to input and changes the weights during training process. It goes back and fourth changing weights and calculating the error until we reach the maximum amount of epochs that we set in the system or until the error is small enough.*

*ALL THIS PROCESS IS WELL COMMENTED INSIDE THE SYSTEM*

**Results:**