Undergraduate Research Support Scheme project

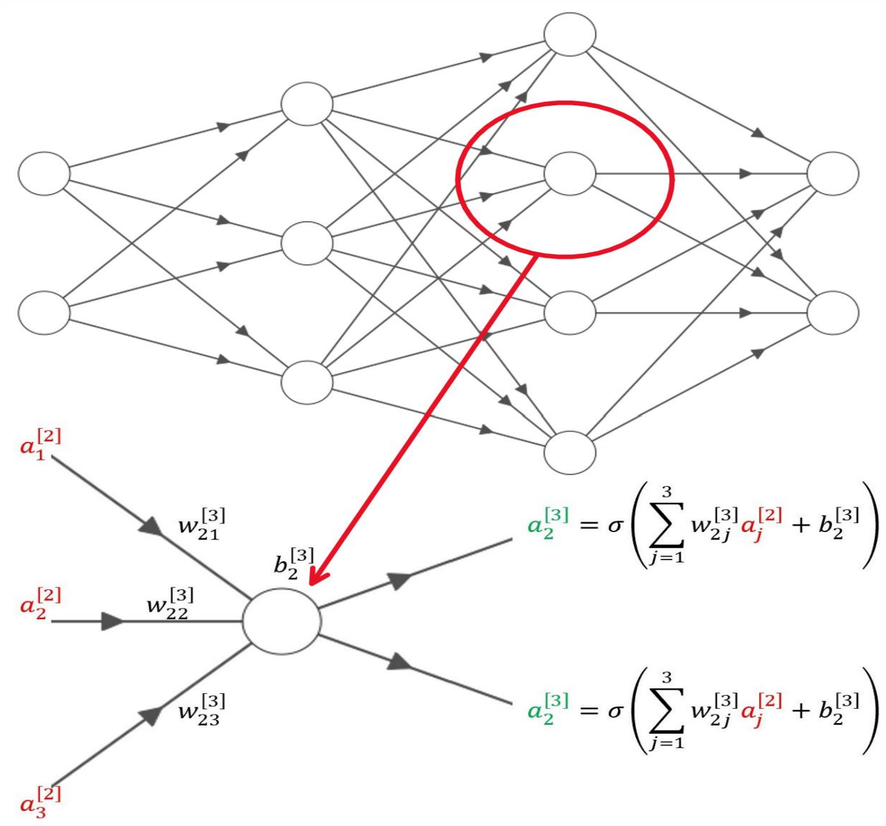
Data Classification Using Neural Networks

Introduction:

A multilayer perceptron (MLP) consists of at least three layers: the first layer is the input layer, the last layer is the output layer, intermediate layers are referred to as hidden layers. Each layer contains neurons which are connected to every other neuron in the preceding layer. The l^{th} layer has an associated matrix of weights matrix $W^{[l]}$ and vector of biases $b^{[l]}$. The ***activation function***, σ, is applied to each layer component-wise. If the neural network has L layers and an input x, the output of the l^{th} layer is given by $a^{[1]}(x) = x$ and $a^{[l]}(x) = \sigma\left(W^{[l]}a^{[l-1]}(x) + b^{[l]}\right)$ for $l \in \{2, 3, \dots, L\}$

We provide the MLP with a training data set $\{(x_i, y_i)\}_{i=1}^{N}$, which is a set of input-output pairs. Learning involves finding the weights and biases that minimise the associated cost function. This is done using gradient descent. The step size used for gradient descent is the ***learning rate*** of the neural network. The derivatives used in gradient descent are calculated using backpropagation[1].

The effect of changing the learning rate and the activation function on the effectiveness of classifying handwritten digits is investigated.



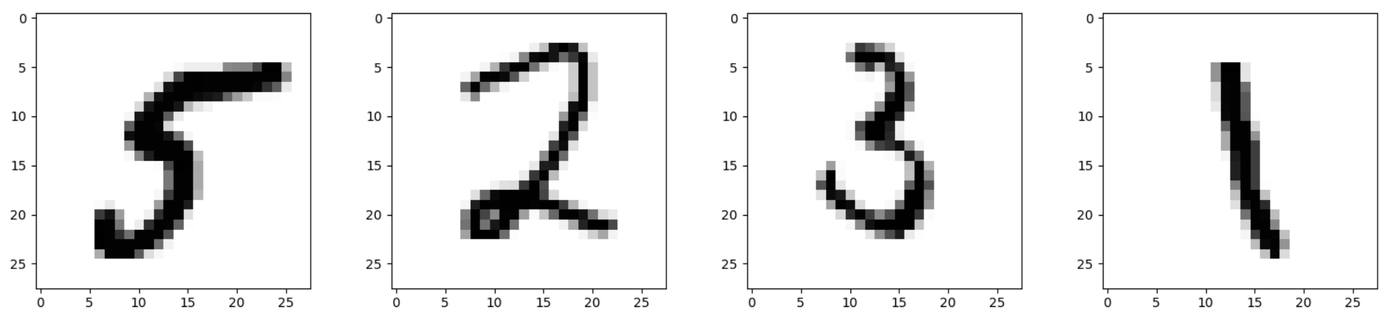
A diagram of an MLP with four layers, and an example calculation taking place in the second neuron in the third layer.

Methodology:

The dataset being used is the MNIST dataset[2], which provides 28 by 28 pixel images of handwritten digits, 0 to 9, in greyscale, four examples are shown below. The value of each pixel is scaled to be between 0.01 and 1.0. The code was implemented into python. The MLPs used had four layers: the input layer had 784 neurons, one for each pixel, the output layer had 10 neurons, one for each digit, and the two hidden layers had 200 neurons each. The training set had 60,000 examples and training ran for three epochs. The activation functions compared were $\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$, $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$, $\text{ReLU}(x) = \max\{0, x\}$ and $\text{LeakyReLU}(x) = \max\{0.01x, x\}$[3].

For each activation function, the accuracy of the MLP was plotted against the learning rate. This was done with three repeats for each learning rate, averages and uncertainties were recorded to account for randomness.

The learning rate which gives the greatest accuracy is selected for each activation function then the accuracy was plotted against the number of iterations to observe how quickly the neural networks learned.



Algorithm 1: Multilayer Perceptron Pseudocode[1]

Result: Training the multilayer perceptron

Randomly initialise weights $W^{[l]}$ and biases $b^{[l]}$ normally with mean zero and variance the reciprocal of the number of neurons in layer l

σ = activation function

η = learning rate

L = total number of layers

N = size of training set

for *epoch* $= 1$ *up-to NumberOfEpochs* **do**

  Randomly shuffle training set $\{(x_i, y_i)\}_{i=1}^{N}$

  for $i = 1$ *up-to* N **do**

    $a^{[1]} = x_i$

    for $l = 2$ *up-to* L **do** — Forward Pass

      $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$

      $a^{[l]} = \sigma(z^{[l]})$

      $D^{[l]} = \text{diag}(\sigma'(z^{[l]}))$

    end

    $\delta^{[L]} = D^{[L]}(a^{[L]} - y_i)$

    for $l = L - 1$ *down-to* 1 **do** — Backpropagation

      $\delta^{[L]} = D^{[l]}(W^{[l+1]})^T\delta^{[l+1]}$

    end

    for $l = L$ *down-to* 1 **do** — Gradient Descent

      $W^{[l]} \to W^{[l]} - \eta\delta^{[l]}a^{[l-1]^T}$

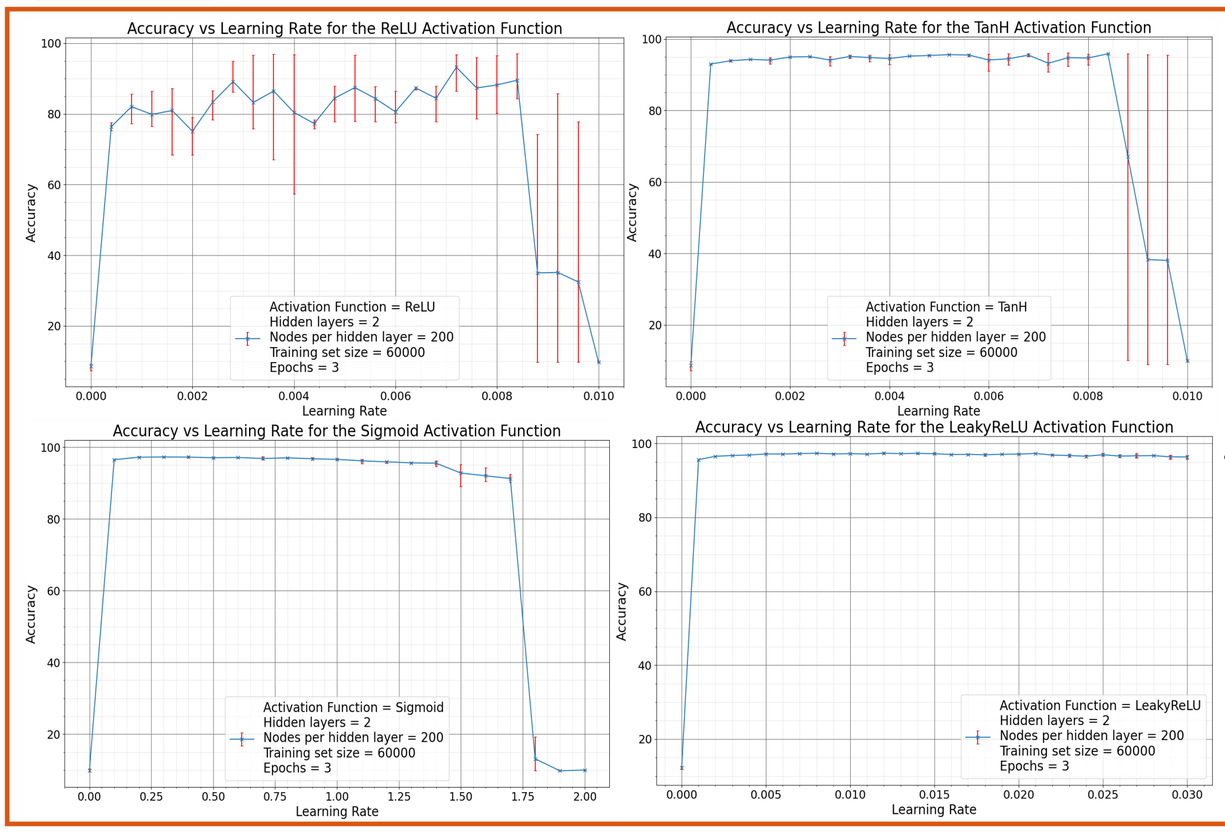
      $b^{[l]} \to b^{[l]} - \eta\delta^{[l]}$

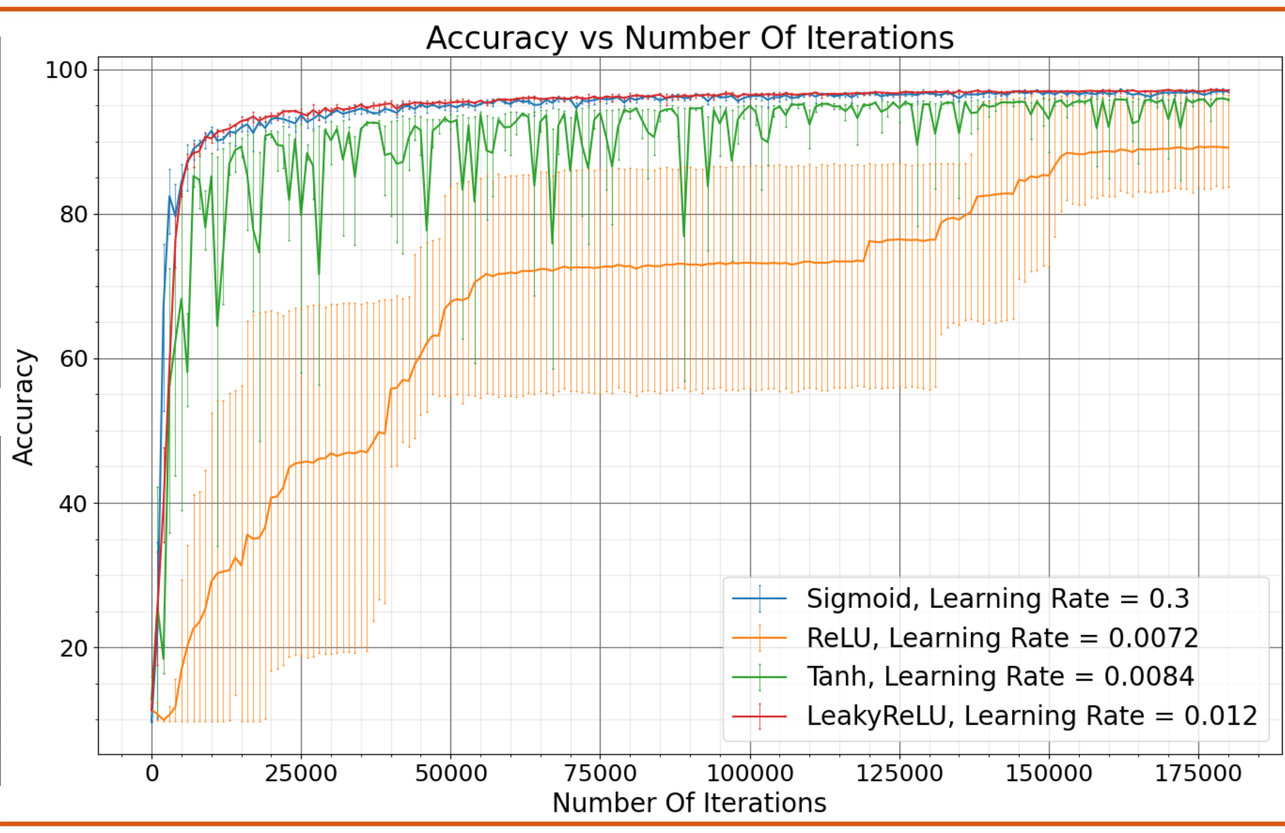
    end

  end — One Iteration

end — One Epoch

Results:





Conclusion:

From what was observed, the optimal learning rates were 0.3, 0.0072, 0.0084 and 0.0012 for Sigmoid, ReLU, tanh, LeakyReLU respectively. Sigmoid and LeakyReLU learned very quickly and steadily. The accuracy of ReLU jumped every few thousand iterations, whilst tanh's oscillated as it was learning. The four MLPs examined all seemed to be approaching an accuracy of approximately 97%.

1. Higham, Catherine F., and Desmond J. Higham. "Deep learning: An introduction for applied mathematicians." SIAM Review 61.4 (2019): 860-891.
2. Redmon, Joseph. MNIST in CSV, Dec. 2013, pjreddie.com/projects/mnist-in-csv/.
3. Sharma, Sagar. "Activation Functions in Neural Networks." Medium, Towards Data Science, 14 Feb. 2019, towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Created by Donnell Obovu, D.Obovu@warwick.ac.uk

Supervised by Professor Charlie Elliott, Warwick Mathematics Institute