Deep Learning for the MNIST Data Set

H. K
1923114

Artificial Neural Network (ANN) and its implementation

Artificial neural network (ANN) is a computational modelling technique that extracts complex representative information from the data by establishing empirical relationships between dependent and independent variables. It is a supervised learning scheme that consists of three fundamental layers i.e., input, hidden, and output layers.

Artificial neurons are the core component of ANN. In the proposed network the input layer is composed of 784(28X28) neurons followed by 3 hidden layers, the number of the nodes in those layers are given by the user and an output layer with 5 output neurons. The output nodes predicted 4 desired digit classes along with a None class which represented all the other classes. Weights, deciding factor of input's influence on output, and biases are the two learnable parameters that are being optimised in the training phase of the networks. In this network, the initial weights are assigned as a random value ranging from -0.5 to 0.5 and zeros are allocated as initial biases.

The weights and biases were initialized using the following code:

$$W_n = 0.5 - rand(Number\ of\ nodes\ in\ previous\ layer, number\ of\ node\ in\ self\ layer)$$

$$b_n = zeros(Number\ of\ nodes\ in\ previous\ layer, number\ of\ nodes\ in\ self\ layer)$$

In the training phase, the network tries to optimise the weights and biases with the help of *forward propagation* and *backpropagation.*

Forward propagation

As neurons are interconnected with their neighbouring layer neurons, the inputs are successively transmitted between neurons with the applied weights along with the bias.

The processing of each neuron happens in two steps, i.e.,

1) Pre activation, this is a *weighted sum of inputs with biasing.*

$$n_n = \Sigma(weight^T \cdot input) + bias$$

2) Activation, the sum is then passed through the activation function, a mathematical function that determines whether to activate the neuron or not by introducing non-linearity into the output. In this network, the sigmoid and SoftMax function is used as the activation function in hidden and output layers respectively.

Backpropagation

From the activation values of the final layer, an error value is calculated and this error is then transferred in the opposite direction. The process is known as backpropagation. Two

backpropagation methods have been used in this work, Unscaled Heuristic Backpropagation (UHB) and Calculus Backpropagation (CB).

For performing backpropagation first we make a matrix with the errors in the final layer, in this case, e5. The error is defined as

$$e5 = y(actual) - y(predicted)$$

In UHB, this error flows back to the previous layers with matrix multiplication to the weights. The errors in the hidden layer are:

$$e(4,3,2) = weight * e(5,4,3)$$

To find the final backpropagation coefficient, S_n, the diagonal of the derivatives of the unactivated forward propagated values (n_n) are calculated and put into a matrix call A_n. The activated values are differentiated using the Jacobian function,

$$A_n = diag(\sigma'(n_n))$$

These errors are then multiplied with the activated values of the same layer.

$$S_n = -2 \times A_n * e_n$$

This is the backpropagation coefficient in UHB.

In CB, it was proven using the Jacobian function that,

$$\hat{S} = (\frac{dn}{d\hat{n}})^T * S = A_n * W_{n+1} * S_{n+1}$$

That is, if the backpropagation coefficient of the forward layer is known, the previous layer can be calculated using it; this definition is used in CB.

Learning and learning rate

The backpropagation coefficient was multiplied with the weights and was added with the previous layer to backpropagate the error and update the weights, which is the way the neural network learns.

The theory used here:

$$W_n = W_n - Learning\ rate * A_{n-1} * S_n{}'$$

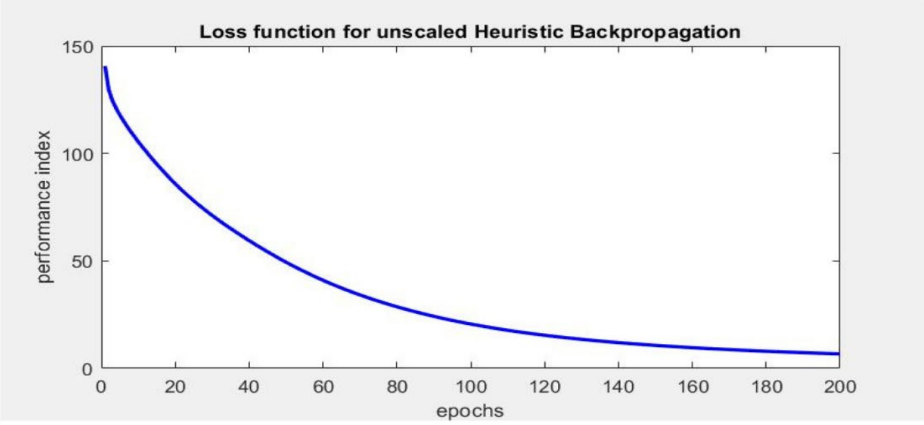
Similarly, the bias is also updated with the following equation.

$$b_n = b_n - Learning\ rate * b_{n-1} * S_n{}'$$

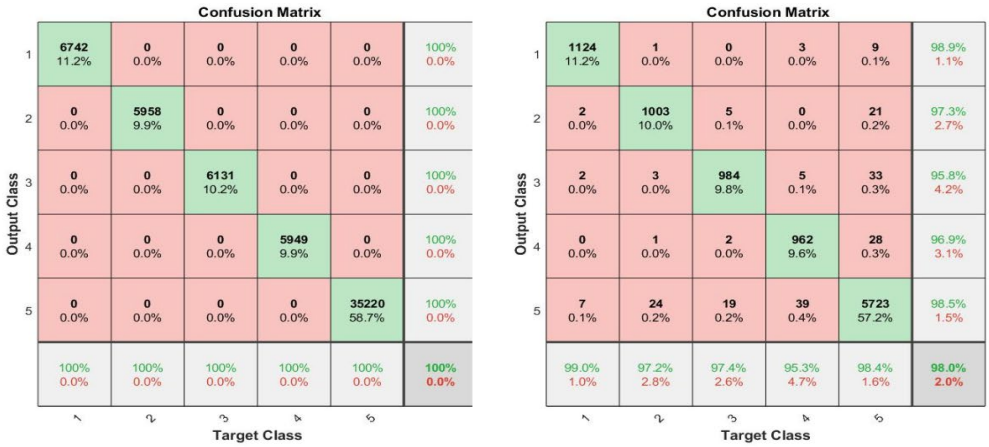
The learning rate is a variable that determines how quickly the gradient descent can find the minimum point. This is a user-controlled variable that has important implications on results.

Performance using UHB

Detailed experiments have been performed on the large MNIST dataset using the Unscaled.



(a)



(b) (c)

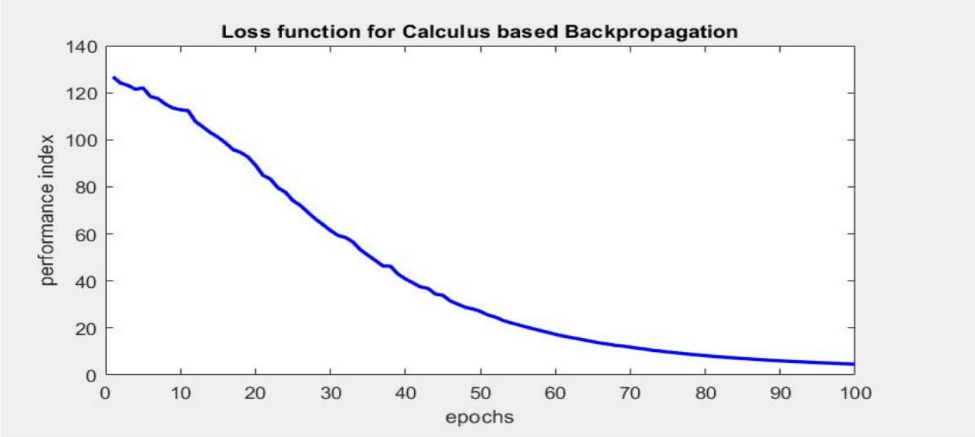
Figure 1. A graph of loss function against epochs (a) and confusion matrices for the large training set (b) and testing set (c) using UHB

For UHB the sample testing on the test set has shown that around 200 epochs of training with a 0.001 learning rate is necessary for getting the optimum results using the UHB method. From the confusion matrix, we can see that the training accuracy reached 100% while the testing accuracy was 97.96%.

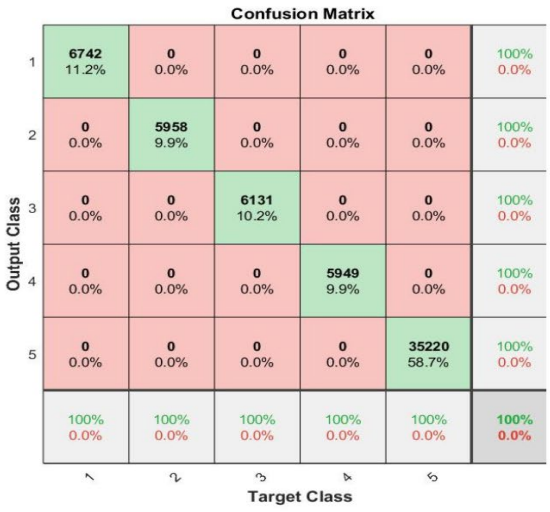
Performance using CB

Detailed experiments were run on the dataset using CB. It was found that low learning rates perform pretty badly using CB. This can be due to the slow progress in gradient descent in CB. A learning rate of 0.01 was used with 100 epochs to find the optimum loss graph. The confusion matrix shows 100% training accuracy with a test accuracy of 98.11%. The accuracy difference

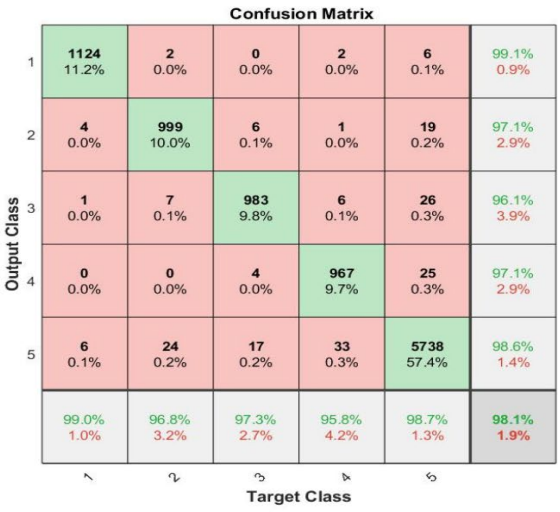
between the two different types of backpropagation was pretty small and can be attributed to the random test set selection.



(a)



(b)



(c)

Figure 2. A graph of loss function against epochs (a) and confusion matrices for the large training set (b) and testing set (c) using CB

Moreover, because the selected 4 classes and the None class had a high number disparity, the class imbalance was prevalent in the dataset, and that is where the errors were mostly found.

Hyperparameters

The hyperparameters used for two back propagations are:

Epochs = 100, 200

Learning rate = 0.01, 0.001

The weights and biases were initialized using the following formulae:

$$W_n = 0.5 - rand(Number\ of\ nodes\ in\ previous\ layer, number\ of\ node\ in\ self\ layer)$$

$$b_n = zeros(Number\ of\ nodes\ in\ previous\ layer, number\ of\ nodes\ in\ self\ layer)$$

Here, rand gives us a random value between 0 and 1. The weight value is initialized to be a random value between -0.5 to 0.5. This system of random value initiation is followed because of the sigmoid activation function. The sigmoid activation function converts all values which are >0.5 into 1. The usage of values less than 0.5 ensures that the nodes are not activated by default.

The activation function used in the code was the Sigmoid activation function. The formula for sigmoid activation function is:

$$sigmoid(x) = \frac{x}{1 + e^{-x}}$$

The derivative of this sigmoid function is defined as:

$$sigmoid'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

Sigmoid activation has a range between 0 to 1 and is used in the classification tasks because it gives the probability for each class (as the probability value is between 0 to 1). The sigmoid function is more stabilized than the linear activation function in the sense that its value is limited within 1 and is more sensitive than the tanh activation. ReLu is a better choice for non-classification tasks. The sigmoid function is also differentiable which is necessary for the calculation of cost function.

In the final layer, the softmax activation function is used as this is a multiclass classification problem. The definition of softmax classifier is

$$softmax(x) = \frac{e^x}{\sum_1^n \quad e^x}$$

Where n is the number of classes.

For the loss function, cross-entropy has been used. The equation of cross-entropy is defined as

$$cross - entropy = -\sum_1^n \quad y_{train} * log\ (y)$$

For classification tasks, cross-entropy avoids the problem of oversaturation on the output layers. Moreover, softmax is used as an activation function as required for using cross-entropy.

In conclusion, the model showed quite a high accuracy on both the train and test set, ensuring the efficacy of the model and choice of the hyperparameters and the activation function.