BT3041 Assignment 2

Yogesh Deokar

EE17B006

1 Network Architecture

The Multilayer Perceptron used in this assignment has 3 layers, input layer, hidden layer and output layer.

Input layer :- 784 neurons.
Hidden Layer :- 200 neurons.
Output Layer :- 10 neurons.

The input layer has 784 neurons as the image size is 28*28. So after flattening the image , the size becomes 784. The hidden layer neurons were selected to be 200. The output layer has 10 neurons because we are predicting the digits in the MNIST image dataset. Since the digits as 0 to 9, we need 10 neurons corresponding to each of the 10 digits. The value of neurons corresponding to each digit represent the probability of the image belonging to that class.

2 Activation Functions

The following activation functions are used in the Multilayer Perceptron using the above architecture :-

Input layer :- No activation function
Hidden Layer :- Sigmoid activation function.
Output Layer :- Softmax activation function.

The Input layer doesn't require any activation function as it just takes the same values as the input 784 size vector. The hidden layer has sigmoid activation activation function. The output layer has softmax activation function because we need the output vector to denote the probabilities of the image corresponding to a particular digit.

3 Loss function

We will use **Cross Entropy** loss function since the last layer is a softmax layer. The cross entropy loss is defined as :-

$$L(\hat{y}, y) = -\sum_{i=o}^{c} y_i log(\hat{y}_i)$$

Here c is the number of classes, $\hat{y}$ is the predicted vector from last layer and y is the corresponding target vector. This corresponding target vector is obtained by one hot encoding the target labels. For example, consider there are 10 classes and for a given image, target label is 2. The corresponding one-hot encoded vector will be $y = \begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$.

Let us calculate loss for a particular datapoint. Consider the following example :-
$\hat{y} = \begin{bmatrix}0.1 & 0.2 & 0.5 & 0.05 & 0.05 & 0 & 0 & 0 & 0 & 0.1\end{bmatrix}$
$y = \begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$.
Then $L(\hat{y}, y) = -log(0.5)$

4 Learning Algorithm

The above Multilayer perceptron is trained using **backpropogation algorithm**. Let W1 and B1 represent weight matrix and bias vector connecting input layer to hidden layer, similarly W2 and B2 represent weight matrix and bias vector connecting hidden layer to output layer.
Our update rule will be :-

$$W1 \leftarrow W1 - \alpha \frac{\partial L}{\partial W1}$$
$$B1 \leftarrow B1 - \alpha \frac{\partial L}{\partial B1}$$
$$W2 \leftarrow W2 - \alpha \frac{\partial L}{\partial W2}$$
$$B2 \leftarrow B2 - \alpha \frac{\partial l}{\partial B2}$$

Here α is the learning rate and L represents the loss functions.

5 Learning Rate

All the learning rates in the list [0.01, 0.1, 1, 10] were tried and the learning rate selected for back-propogation algorithm is **1**.

6 Accuracy and Loss plots

Below are the plots for accuracy and loss for each iteration while training the network.

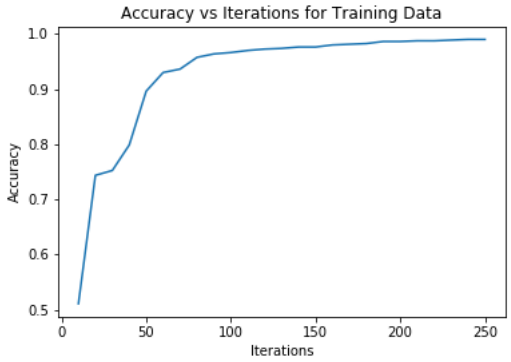

Figure 1: Accuracy for different iterations for training data

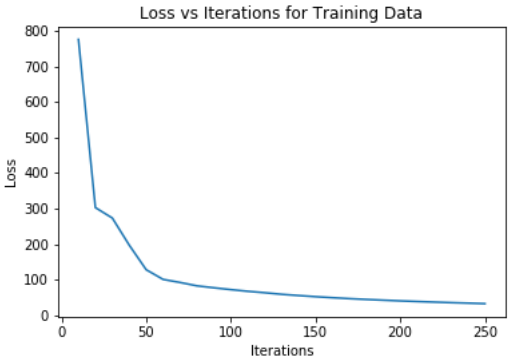

Figure 2: Loss for different iterations for training data

7 Results on Test dataset

The table below shows the test accuracy and loss as compared to that of train data.

	Accuracy	**Loss**
Train data	99%	32.7834
Test data	80%	55.7989

Table 1: Loss and Accuracy numbers