Introduction to Deep Learning

Name: **Due Date:** Mar. 22, 2021
Programming Assignment: Number 2

Multilayer Neural Network

A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP). it consist of an input layer, a output layer and at least one hidden layer as shown in Figure 1. Each layer is fully connected and the values for each node is computed via forward propagation and activation using the weights of each layer and the input to such layers.

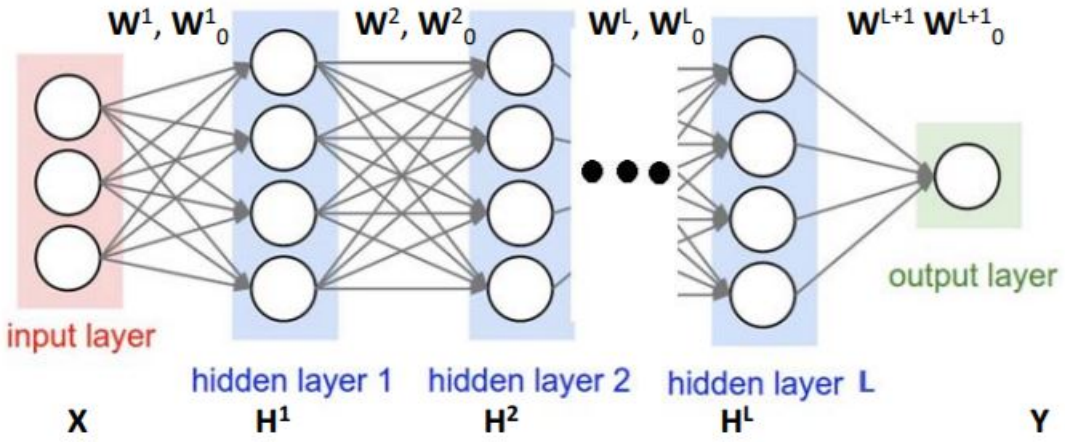

Figure 1: MLP Network

In the forward propagation, a linear discriminate function f and the activation function ReLU are used to obtain the hidden layers vectors. The first hidden layer H_1 matrix is obtained thus,

$$z_1[m] = f\left(X[m], W_1, W_{10}\right) = X[m]W_1 + [W_{1,0}] \tag{1}$$
$$H_1[m] = \sigma\left(z_1[m]\right) \tag{2}$$

$\sigma(z_1[m])$ can be a Sigmoid, or a ReLU function. For the output layer, the activation function can be linear (identity function) for regression, or sigmoid for binary classification or the softmax function for multi-class classification. The weights of the network are trained using the back propagation algorithm.

Model Architecture

The model consist of an input layer, two hidden layers and an output layer. the input layer contains 784 nodes - which is the dimension of the input vector. Each hidden layer contains 100 nodes while

the output layer contains 10 nodes for each digit (digit 0 to 9).
Each weight matrix and their dimensions are given below

$$\mathbf{W_1} \in \mathbb{R}^{784\times100}, \mathbf{W_{10}} \in \mathbb{R}^{1\times100}$$
$$\mathbf{W_2} \in \mathbb{R}^{100\times100}, \mathbf{W_{20}} \in \mathbb{R}^{1\times100}$$
$$\mathbf{W_3} \in \mathbb{R}^{100\times10}, \mathbf{W_{30}} \in \mathbb{R}^{1\times10}$$

ReLU activation function was used for the hidden nodes while a softmax function was used for the output layer.

$$ReLU(z[m]) = max(0, z[m]) \tag{3}$$
$$Softmax = \frac{\exp(z[m][k])}{\sum_{k'=1}^{K}\exp(z[m][k])} \tag{4}$$

A forward propagation is carried out to obtain a prediction vector $\hat{y}[m]$ which contain the probability for each digit 0 to 9.
A cross-entropy loss function is used to calculate the model loss and the gradient of the loss.

$$l(\mathbf{y}[m], \hat{\mathbf{y}}[m]) = -\sum_{k'=1}^{K}\mathbf{y}[m][k] \times \log(\hat{\mathbf{y}}[m][k]) \tag{5}$$

$$\nabla\hat{\mathbf{y}}[m] = \frac{\partial l(\mathbf{y}[m], \hat{\mathbf{y}}[m])}{\partial\hat{\mathbf{y}}} = -\begin{bmatrix} \frac{\mathbf{y}[m][1]}{\hat{\mathbf{y}}[m][1]} \\ \frac{\mathbf{y}[m][2]}{\hat{\mathbf{y}}[m][2]} \\ \vdots \\ \frac{\mathbf{y}[m][K]}{\hat{\mathbf{y}}[m][K]} \end{bmatrix} \tag{6}$$

Training Using Backpropagation

The weights of the model are updated using backpropation to calculate the gradients of the weights. Weights are updated until the model loss approached zero. For the outer layer, the gradients are given as

$$\nabla\mathbf{W}_3[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{W}_3}\nabla\hat{\mathbf{y}}[m] = \frac{\partial\mathbf{z}(m)}{\partial\mathbf{W}_3}\frac{\partial g(\mathbf{z}(m))}{\partial\mathbf{z}(m)}\nabla\hat{\mathbf{y}}[m] \tag{7}$$
$$\nabla\mathbf{W}_{30}[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{W}_{30}}\nabla\hat{\mathbf{y}}[m] = \frac{\partial\mathbf{z}(m)}{\partial\mathbf{W}_{30}}\frac{\partial g(\mathbf{z}(m))}{\partial\mathbf{z}(m)}\nabla\hat{\mathbf{y}}[m] \tag{8}$$
$$\nabla\mathbf{H}2[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{H}2[m]}\nabla\hat{\mathbf{y}}[m] = \frac{\partial\mathbf{z}(m)}{\partial\mathbf{H}2}\frac{\partial g(\mathbf{z}(m))}{\partial\mathbf{z}(m)}\nabla\hat{\mathbf{y}}[m] \tag{9}$$
$$\tag{10}$$

Where

$$\frac{\partial g(\mathbf{z}(m))}{\partial \mathbf{z}(m)} = \left(\frac{\partial \sigma_M(\mathbf{z}[m])}{\partial \mathbf{z}[m]}\right)^{K\times K} \tag{11}$$

$$\frac{\partial \sigma_M(\mathbf{z}[m][k])}{\partial \mathbf{z}[m][i]} = \begin{cases} \sigma_M(\mathbf{z}[m][i])\left(1-\sigma_M(\mathbf{z}[m][i])\right) & \text{if k = i} \\ -\sigma_M(\mathbf{z}[m][k])\sigma_M(\mathbf{z}[m][i]) & \text{else} \end{cases}. \tag{12}$$

$$\frac{\partial \mathbf{z}(m)}{\partial \mathbf{W}_3} = \left(\frac{\partial \mathbf{z}[m]}{\partial \mathbf{W}_3}\right)^{N_L \mathbf{x} K\times K} \tag{13}$$

$$\frac{\partial \mathbf{z}[m][k]}{\partial \mathbf{W}_{3,i}} = \begin{cases} \mathbf{H}^2(m) & \text{if k = i} \\ 0 & \text{else} \end{cases} \tag{14}$$

For each of the hidden layer, the gradients are given as

$$\nabla \mathbf{W}_0^l[m] = \frac{\partial \phi(\mathbf{z}[m])}{\partial \mathbf{W}_0^l[m]} \nabla \mathbf{H}^l[m] = \frac{\partial \mathbf{z}[m]}{\partial \mathbf{W}_0^l[m]} \frac{\partial \phi(\mathbf{z}[m])}{\partial \mathbf{z}[m]} \nabla \mathbf{H}^l[m] = \frac{\partial \phi(\mathbf{z}[m])}{\partial \mathbf{z}[m]} \nabla \mathbf{H}^l[m][i] \tag{15}$$

$$\nabla \mathbf{W}^l[m] = \frac{\partial \mathbf{z}[m]}{\partial \mathbf{W}^l} \frac{\partial \phi(\mathbf{z}[m])}{\partial \mathbf{z}[m]} \nabla \mathbf{H}^l[m] \tag{16}$$

Where

$$\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{z}[m][j]} = \begin{cases} 1 \text{ if i = j and } \mathbf{z}[m][i] > 0 \\ 0 \text{ else} \end{cases} \tag{17}$$

$$\frac{\partial \mathbf{z}[m][i]}{\partial \mathbf{W}_j^l} = \begin{cases} H^{l-1}[m] & \text{if i = j} \\ 0 & \text{else} \end{cases} \tag{18}$$

Each weight is updated using the averaged gradients.

$$\mathbf{W}^l[t] = \mathbf{W}^l[t-1] - \eta \left(\frac{1}{M}\sum_{m=1}^{M} \nabla \mathbf{W}^l[m]\right). \tag{19}$$

$$\mathbf{W}_0^l[t] = \mathbf{W}_0^l[t-1] - \eta \left(\frac{1}{M}\sum_{m=1}^{M} \nabla \mathbf{W}_0^l[m]\right). \tag{20}$$

Experimental setting

- Python 3.8 was used for coding.
- Training and testing data were imported and processed using the hints provided. The data were normalized by dividing with 255.0.
- Training and testing labels were used to obtain one of k encoding.
- Tensorflow version 2.4.0 and Numpy were used to build the model class and carry out back-propagation. While the model class was based on Tensorflow,Numpy was used for computing the gradients because of it faster speed compared to Tensorflow.
- Model is initialized by calling the model class and training was carried out by calling model's train model function.

Hyper-parameters used

- Number of iterations: $N = 10000$
- learning rate $\eta = 0.1$
- batch size $= 50$

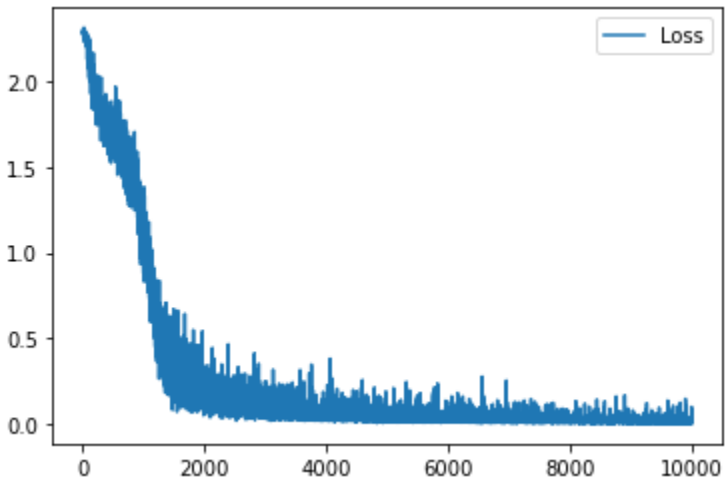

Figure 2: Cross-entropy Loss

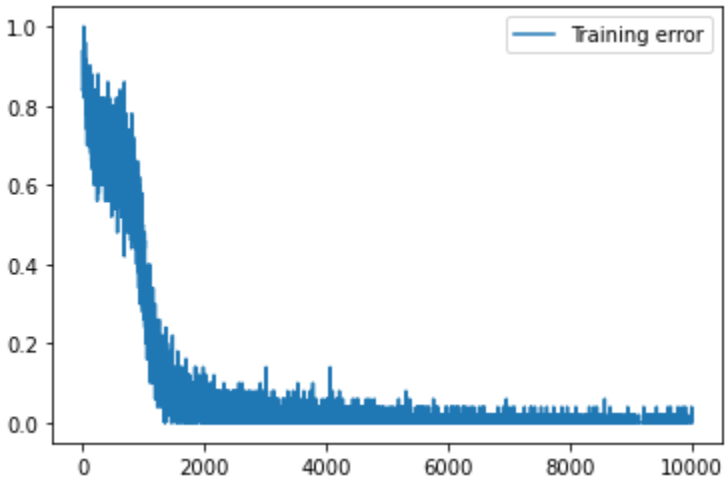

Figure 3: Training Error

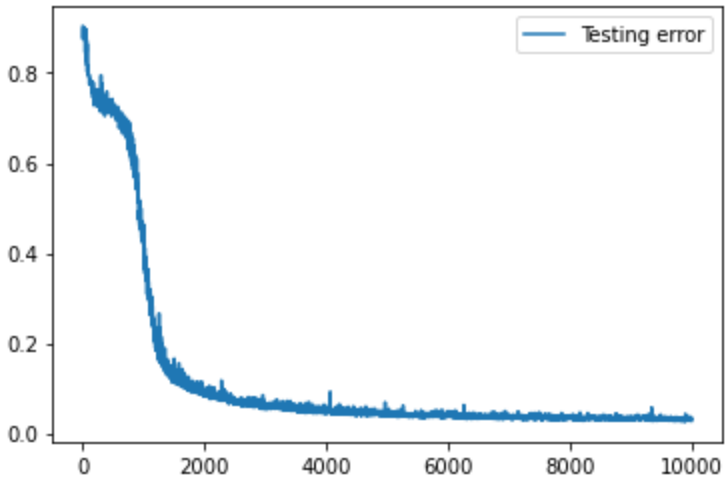

Figure 4: Testing Error

Classification Errors	
Digit data	Error
0	0.00796
1	0.00749
2	0.02213
3	0.02879
4	0.01860
5	0.07002
6	0.01237
7	0.03605
8	0.03301
9	0.04393
Overall	0.02760