EEE-543 - Fall 2023 Mini Project 1: Two Layer Neural Network

Salih Deniz Uzel (deniz.uzel@bilkent.edu.tr)
22201382

I. Introduction

A neural network with 1 hidden layer is implemented in python programming language. Model is trained with given hyper-parameters in the assignment for the MNIST dataset [1]. Mnist dataset is used for training number recognition neural network model and it has 10 different classes from number 0 to 9 included.

In the report, options 1 to 9 in the assignment are explained together with their implementation details in the Section II. The section related to Evaluation is detailed under the Section II-A, and the results are included and explained in the section Results.

II. Implementation

A. Option 1, Data Preparation

Accessing the MNIST dataset at yannlecun.com/mnist prompted for providing a username and password. Therefore, I downloaded the MNIST data through TensorFlow. If requested by the TA, I can demonstrate the download process on my device. There are 60000 Training samples and 10000 Test samples in MNIST data. For the experiments test set is divided into 5000 Validation and 5000 Test sets with the same class distribution, stratified. All sample sets are normalized by dividing by 255. Stratified distribution of the labels in the training, validation, and test sets are shown in the Table I.

TABLE I: Stratified Distribution of the Labels in the Training, Validation, and Test Sets

Label	Training	Validation	Test
0	5923	490	490
1	6742	568	568
2	5958	516	516
3	6131	505	505
4	5842	491	491
5	5421	446	446
6	5918	479	479
7	6265	514	514
8	5851	487	487
9	5949	504	504

B. Option 2, Implementing the Neural Network

The network mainly consist of forward and backward propagation. Terms of partial differential equations is implemented in the *backward_propagation* method of class NN. In order to make weight and bias updates, the change of the weight and bias values in the 1st layer and 2nd layer must be calculated for the change made in the Loss function equation. The Δ (delta) notation will be used to present amount of change in the corresponding differential equation variable. To denote the change of the a layer's weight following notation is used $\Delta w^{(L)} = dw^{(L)} = \frac{\partial L}{\partial w^{(L)}}$. This equation can be solved by using the chain rule. The general partial differential equation for obtaining the amount of change with respect to (w.r.t.) Mean Square Error(MSE) Loss function of Weights and Biases are given in the equations (1), (2), (3)(4). Solution of these chain equations for MSE Loss function are given in the generic from in the equations (5), (6), (7), (8). The general formulas used for weight and bias updates are given in the (9), and (10). Calculations are implemented in vectorized way. L2 regularization extension for the weight update will be given in Option 9.

$$\frac{\partial L}{\partial a^{(L)}} = \frac{d}{da} MSE \tag{1}$$

$$\frac{\partial L}{\partial z^{(L)}} = \frac{\partial \mathcal{L}}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \tag{2}$$

$$\frac{\partial L}{\partial w^{(L)}} = \Delta w^{(L)} = \frac{\partial L}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} \tag{3}$$

$$\frac{\partial L}{\partial b^{(L)}} = \Delta b^{(L)} = \frac{\partial L}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial b^{(L)}} \tag{4}$$

$$\frac{dL}{a^{(L)}} = \frac{1}{n} \sum_{i=1}^{n} (a_i - y_i) \tag{5}$$

$$\frac{da^{(L)}}{dz^{(L)}} = \frac{d}{dz}(\text{Activation Function}) \tag{6}$$

$$\frac{dz^{(L)}}{dw^{(L)}} = \frac{d}{dw^{(L)}} \left(w^{(L)} a^{(0)} + b^{(L)} \right) = a^{(L-1)} \tag{7}$$

$$\frac{dz^{(L)}}{db^{(L)}} = \frac{d}{db^{(L)}} \left(w^{(L)} a^{(0)} + b^{(L)} \right) = 1 \tag{8}$$

$$W^{(L)} := W^{(L)} - \text{LearningRate} \cdot \Delta w^{(L)} \tag{9}$$

$$b^{(L)} := b^{(L)} - \text{LearningRate} \cdot \Delta b^{(L)} \tag{10}$$

C. Option 3, Activation Functions

The python implementation of activation function for both forward and backward propagation is given below. The backward propagation of activation functions corresponds to (6).

TABLE II: Model training parameters for the given 18 different "Run No".

Run No	case	N	lr	lambda	epochs	batch_size	act1	act2	time (seconds)
1	case1	300	0.01	0	100	60000	Tanh	Tanh	394.432828
2	case1	300	0.05	0	100	60000	Tanh	Tanh	410.961058
3	case1	300	0.09	0	100	60000	Tanh	Tanh	420.870059
4	case1	500	0.01	0	100	60000	Tanh	Tanh	619.016482
5	case1	500	0.05	0	100	60000	Tanh	Tanh	641.056748
6	case1	500	0.09	0	100	60000	Tanh	Tanh	652.793084
7	case1	1000	0.01	0	100	60000	Tanh	Tanh	1195.384507
8	case1	1000	0.05	0	100	60000	Tanh	Tanh	1228.655823
9	case1	1000	0.09	0	100	60000	Tanh	Tanh	1240.340535
10	case2	300	0.01	0	100	60000	ReLu	Sigmoid	254.276848
11	case2	300	0.05	0	100	60000	ReLu	Sigmoid	245.894848
12	case2	300	0.09	0	100	60000	ReLu	Sigmoid	244.450534
13	case2	500	0.01	0	100	60000	ReLu	Sigmoid	373.389450
14	case2	500	0.05	0	100	60000	ReLu	Sigmoid	367.175328
15	case2	500	0.09	0	100	60000	ReLu	Sigmoid	366.020398
16	case2	1000	0.01	0	100	60000	ReLu	Sigmoid	725.415430
17	case2	1000	0.05	0	100	60000	ReLu	Sigmoid	712.224111
18	case2	1000	0.09	0	100	60000	ReLu	Sigmoid	715.124548

* *N* is the number of neurons in the hidden layer (layer 1).
* λ is the L2 regularization variable.
* *act1* is the activation function used for hidden layer (layer 1).
* *act2* is the activation function used output layer (layer 2).
* *batch_size* = 60000 corresponds to full-batch training.

1) Sigmoid

$$\text{Forward:} \quad f(x) = \frac{1}{1-e^{-x}}$$
$$\text{Backward:} \quad \frac{df(x)}{dx} = f(x)\cdot(1-f(x))$$

2) ReLU

$$\text{Forward:} \quad f(x) = \max(0,x)$$
$$\text{Backward:} \quad \frac{df(x)}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \le 0 \end{cases}$$

3) Tanh

$$\text{Forward:} \quad f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$
$$\text{Backward:} \quad \frac{df(x)}{dx} = (1-f^2(x))$$

D. Option 4, One Hot Encoding of the output

One Hot Encoding is a method that converts categorical data with labels, such as 0,1,2,3 to vector representations as given for case1 equation (11) and case2 equation (12). One hot encoding method for case 1 and case 2 is implemented as $create\_onehot\_minus1$ and $create\_onehot$ functions.

$$0 = \begin{bmatrix}1\\-1\\-1\\-1\end{bmatrix}, \quad 1= \begin{bmatrix}-1\\1\\-1\\-1\end{bmatrix}, \quad 2 = \begin{bmatrix}-1\\-1\\1\\-1\end{bmatrix}, \quad 3= \begin{bmatrix}-1\\-1\\-1\\1\end{bmatrix} \tag{11}$$

$$0 = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \quad 1= \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \quad 2 = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \quad 3= \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{12}$$

E. Option 5, Error Function

Mean Squared Error Loss function is used for the error function. The MSE Loss implemented in *MSE_cost_onehot_encode* function. Loss calculation are implemented in vectorized way in equation (13). L2 regularization extension for the total loss calculation will be given in Option 9.

$$\text{Average MSE Loss} = \frac{1}{m}\left(\sum_{i=1}^{m}(a_i-y_i)^2\right)_{vectorized} \tag{13}$$

$$m : \text{number of samples}$$

F. Option 6, Learning Coefficient

Learning coefficient [0.01, 0.05, 0.09] implemented at weight and bias update w.r.t equation (9) and equation (10) in the *backward_propagation* method of NN class.

G. Option 7, Weigh Initialization and Running 18 Different Experiments

Model training for each run number was started with 3 different weights. Weights are initialized with a Gaussian distribution where mean=0, std=0.01. [42, 268, 975] seeds were used for these 3 different initialization.Sample set accuracy calculated by averaging the accuracy results of these 3 different runs. Experiment parameters for each run along with their total training time in seconds, are given in the Table II. Experiment results are given in the Table III. Table III consist of 3 run mean and std values of the each run no for the training, validation, and test sets.

TABLE III: Training, Validation, Test Set 3 Run Average Accuracy After 100 Epochs Training

Run	Train.		Val.		Test	
No.	μ	σ	μ	σ	μ	σ
1	0.480	0.034	0.490	0.033	0.477	0.033
2	0.854	0.001	0.865	0.001	0.863	0.001
3	0.879	0.001	0.885	0.001	0.886	0.002
4	0.551	0.008	0.563	0.011	0.550	0.011
5	0.864	0.001	0.875	0.001	0.873	0.000
6	**0.879**	**0.001**	**0.888**	**0.002**	**0.885**	**0.001**
7	0.674	0.009	0.687	0.011	0.669	0.009
8	0.869	0.003	0.879	0.003	0.876	0.003
9	0.623	0.367	0.629	0.372	0.627	0.371
10	0.155	0.056	0.159	0.058	0.158	0.059
11	0.243	0.009	0.255	0.008	0.247	0.011
12	0.313	0.004	0.321	0.006	0.315	0.003
13	0.145	0.019	0.148	0.020	0.145	0.020
14	0.302	0.002	0.316	0.003	0.305	0.003
15	0.381	0.006	0.384	0.006	0.382	0.006
16	0.167	0.016	0.171	0.014	0.167	0.014
17	0.406	0.009	0.418	0.005	0.410	0.008
18	0.580	0.002	0.591	0.006	0.580	0.004

* Run No: 6, has the highest accuracy with a lower std.

H. Option 8, Running the best performing model with Mini Batches

1) Best Performing Case

When comparing the test set accuracy values among runs, Run No: 6 has the highest accuracy with a lower standard deviation. This is shown on the Table III with bold font. The parameters belongs to the 6'th run explicitly given in the Table IV.

TABLE IV: Parameters of Run No:6.

Parameter	Value
case	case1
N	500
learning rate	0.09
L2 Regularization, λ	0
epochs	100
batch size	60000
activation1	Tanh
activation2	Tanh
training time	10.88 min

2) Running the best performing case with mini-batch

The 3 run average results for different batch sizes are given in Table V. The batch size 10 is too small and cause almost stochastic learning process. Therefore test accuracy performance stuck at %10 with 100 epochs. The model with batch size 10 requires more training time. Accuracy of the batch size 50 and 100 is very close. However, batch size 100 is $1e-3$ higher than the batch size 50. Therefore batch size is picked 100.Its mean test set accuracy is 98.1%.

I. Option 9, L2 Weight Regularization

L2 Regularization is used to penalize the weights by the square of their values. This penalization helps prevent the weights from becoming too large. As a result, L2 regularization reduces the risk of over-fitting because a few selected weights represent a larger portion of the samples. L2 regularization implemented on Loss Function equation 14. Therefore the formula of loss function and its derivative need to be modified. As a result this modification takes place in the chain rule formula and causes a modification in the *backward_propagation* method where Δw is calculated, shown in equation 15.

TABLE V: Training, Validation, Test Set 3 Run Average Accuracy After 100 Epochs Training for Given Batch Sizes

Run	Train.		Val.		Test	
Batch Size.	μ	σ	μ	σ	μ	σ
10	0.101	0.009	0.101	0.010	0.101	0.010
50	0.996	0.000	0.982	0.001	0.979	0.001
100	0.997	0.000	0.982	0.002	0.981	0.001

* Batch size 100, has the highest accuracy with a lower std.

$$\text{Average MSE Loss} = \frac{1}{m}\left(\sum_{i=1}^{N_j}(a_i - y_i)^2\right)_{\text{vectorized}} + \frac{\lambda}{2m}\left(\sum_{l=1}^{L}(W^{(L)})^2\right)_{\text{vectorized}} \tag{14}$$

L2 Regularization is implemented in the NN.fit method between the *forward_pass* and *backward_propagation* method calls. The calculated value passed to *MSE_cost_onehot_encode* function and accumulated with the MSE loss.

$$\frac{\partial L}{\partial w^{(L)}} = \Delta w^{(L)} = \frac{\partial L}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial w^{(L)}} + \frac{\lambda}{m}W^{(L)} \tag{15}$$

The 3 run average results for different L2 regularization parameters are given in Table VI. For regularization value 0.01 and 0.001, difference between average test set accuracy is less then $1e-3$. The mean test accuracy value of 3 runs is 98.0%. The performance metrics for different experiments are found to be very close to each other which is not to enough to make distinction between two parameters.

TABLE VI: Training, Validation, Test Set 3 Run Average Accuracy After 100 Epochs Training for Given Regularization Coefficient λ

Run	Train.		Val.		Test	
L2 Reg.	μ	σ	μ	σ	μ	σ
0.01	0.997	0.000	0.983	0.001	0.980	0.001
0.001	0.997	0.000	0.981	0.002	0.980	0.000

* The $\lambda = 0.001$, has the same accuracy with a lower std.

J. Option 10, Interpretation of the Results

Case1 and case2 were able to learn the given data set. However activation functions effected the speed of the learning process. Relu activation function were able to learn faster than Tanh function used in case1 with greater learning rates.

Greater learning rate caused vanishing gradients for case1. But weight initialization with Gaussian Distribution and smaller learning rates prevented this problem. On the contrary Relu with small learning rates, given in the assignment, caused slow learning. Therefore Tanh function performed better. Lastly, small batch sizes 10/60000, 50/60000, and 100/60000 caused almost stochastic gradient descent with mini-batches.

References

[1] LeCun, Y., Cortes, C., & Burges, C. J. (2010). MNIST handwritten digit database. ATT Labs [Online]. Available: Http://Yann. Lecun. Com/Exdb/Mnist, 2.