Homework 3

J. Chen

***Abstract*—This homework asks us to use the neural network method to train a model that identifies the exact number written on a 28x28 grayscale image. 60000 images are used to train and validate the model, and an additional 10000 images are used for testing. Using a default parameter of 2 hidden layers, 0.01 learning rate, and 150 epochs, the model is able to achieve an accuracy of 0.9228 at a cross-entropy loss of 62.635848. This report also contains parametric studies on the three hyper-parameters(number of hidden layers, number of epochs, and learning rate), yielding interesting insights on how to properly set these parameters.**

I. Introduction

Using perceptron-based neural networks to predict the probability distribution of labels has been a hot topic in the past years thanks to advancements in computer science. Compared to traditional prediction models such as logistic regression and SVM, the neural network method often yields higher accuracy and is more versatile in terms of input types. In this homework, we will be exploring the algorithm of neural networks and how to construct one for our specific example of numbers on images. We will also discuss the influence of common hyper-parameters in terms of accuracy and converging behavior.

II. Overall Architecture of Neural Network

The overall architecture of our neural network model is shown in the image below: There are in total 6 steps to convert

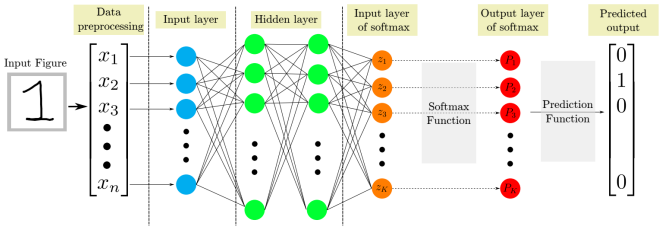

Fig. 1. Architecture of Neural Network Model[1]

raw data to the final prediction. The first step is to convert the raw data to a format that can be easily utilized by the input layer. The input layer will then transfer the data to the hidden layers for further processing. The processed data will arrive at the output layer of the model and be converted to probability distribution through the softmax function. Based on the final probability distribution, a definitive prediction is given.

III. Model Construction

A. Data Preparation

Data preparation is the first step of making any neural network model. Specifically for the homework, the 28x28 grayscale images will be converted to a 728x1 vector, with each element in the vector containing the grayscale value of a pixel. The grayscale value is then normalized to a range [0,1] by dividing 255, which is the range of grayscale before normalization. Then, the label of each image is converted to a 10x1 vector through one hot encoding. Basically, the label is converted to an array with 0 everywhere except where the raw label matches the index of the array. The same step is repeated exactly for the testing portion of the data. It is important to process the training and testing data separately to ensure the soundness of the accuracy test.

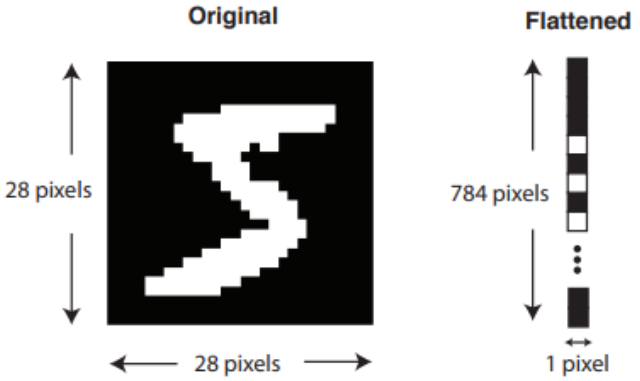

Fig. 2. Transformation from 2D Image to Input Vector[1]

B. Parameters Initialization

Before the data can be processed in the hidden layers, the parameters of each hidden layer need to be initialized. There are two major parameters inside each layer, being the weight and the bias. A cell the same size as the number of layers is first constructed as the mean of storing parameters. Then, within each layer, the initial value of bias is set to 0 while the weight values are set randomly with randn() function.

C. Activation Function

The activation function used for this neural network will be the hyperbolic tangent function(AKA Tanh). Here is the formula for Tanh:

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

The activation of each layer is calculated by applying the corresponding activation function to the processed output of the previous layer(after multiplying by weight and adding bias).

D. Softmax

Softmax function will only appear at the output layer of the neural network. The softmax function is a way of transforming activation to the probability distribution of encoded labels. The softmax function is as follows:

$$f(x) = \frac{e^{x_i}}{\Sigma e^{x_j}} \tag{2}$$

Softmax is calculated for each individual value inside the output of the last hidden layer, and Σe^{x_j} is the sum of every x inside the output.

E. Forward Propagation

Here is a pseudo-code of the forward propagation algorithm: There are three major steps in the forward propagation algorithm. The first step is to take the transformed and normalized vector as input, which is then fed to the hidden layers. Next, inside each hidden layer, the input is multiplied by weight and added bias before being applied to the activation function of choice. Lastly, when all hidden layers are passed, the output layer transforms the final activation to the probability distribution of labels through the softmax function.

Algorithm 1 Forward Propagation Function

```
function FORWARD_PROPAGATION(X, parameters)
    L ← length(parameters)
    A ← X
    activations ← cell(1, L + 1)
    activations[1] ← X
    for l = 1 to L do
        Z ← parameters[l].W · A + parameters[l].b
        if l == L then
            A ← softmax(Z)
        else
            A ← tanh2(Z)
        end if
        activations[l + 1] ← A
    end for
    return activations
end function
```

Fig. 3. Pseudo Code for Forward Propagation[1]

F. Loss Function

Before talking about back-propagation, one must have a loss function to examine the performance of each epoch. The loss function used in the homework is the cross-entropy loss, which can be calculated by using:

$$L(x) = -\Sigma y * log(\hat{y}) \tag{3}$$

where y is the true label, and $\hat{y}$ is the prediction based on neural network. The total cost of one epoch is the sum of errors for every data point fed in that epoch.

G. Backpropagation

Backpropagation inside a neural network is necessary to calculate the gradient of parameters within one epoch. Here is a pseudo-code for backpropagation: The calculated gradients will be stored in a cell array that is the same size as a number of layers.

Algorithm 2 Backward Propagation Function

```
function BACKWARD_PROPAGATION(X, Y, parameters, activations)
    L ← length(parameters)
    m ← size(X, 2)
    gradients ← cell(1, L)
    dZ ← activations[end] − Y
    gradients[L].dW ← dZ · activations[L]′/m
    gradients[L].db ← sum(dZ, 2)/m
    for l ← (L − 1) downto 1 do
        dA ← parameters[l + 1].W′ · dZ
        dZ ← dA · (1 − tanh2(activations[l + 1])²)
        if l == 1 then
            A_prev ← X
        else
            A_prev ← activations[l]
        end if
        gradients[l].dW ← dZ · A′_prev/m
        gradients[l].db ← sum(dZ, 2)/m
    end for
    return gradients
end function
```

Fig. 4. Pseudo Code for Backpropagation[1]

H. Gradient descent

Now, with gradients calculated, the weight and bias can be updated using the gradient descent method. For the next epoch, weights and biases at each layer are calculated by using their respective original value minus their gradients times the global learning rate. The gradient descent method will ensure a continuous decrease in total loss across epochs, and the speed of convergence is controlled by the learning rate. A varying learning rate is possible to implement, but for this homework, a fixed learning rate is set before the learning begins.

I. Inference

The inference function will be fed the softmax output of the testing data. For each image/data point in the testing set, the inference function will go through the probability distribution for that image and give a definitive prediction aligning with the label that has the highest probability.

J. Accuracy Test

After passing the softmax output through the inference function, the final prediction is then compared to the true label of the testing set. The accuracy of the model is calculated by dividing the total number of correct predictions by the total number of predictions.

K. Plotting

The value of total training loss and test accuracy are recorded at the end of each epoch. After the set epoch number is reached, individual points of each epoch are plotted on one single plot to illustrate the change of training loss and test accuracy across time.

L. Main

Inside the main each of the above modulus are called in the listed sequence. Here is a pseudo-code illustrating the sequence:

```
[X_train, Y_train, X_test, Y_test] ← load_train_test_data()
input_size ← size(X_train,1)
output_size ← size(Y_train,1)
neurons ← 64
numLayer ← userinput
learning_rate ← userinput
epochs ← userinput
for i ← 1 to numLayer
    if i == 1 Then
        layer_dim ← input_size
    else if i == last Then
        layer_dim ← output_size
    else
        layer_dim ← neurons
    Endif
Endfor
parameters ← initialize_params(layer_dim)
m ← size(X_train,2)
batch_size ← 64
for i ← 1 to m/batch_size
    for j ← 1 to batch_size
        X_batch ← X_train(:,current batch index)
        Y_batch ← Y_train(:,current batch index)
        forward_output ← forward_propagation(X_batch, parameters)
        cost ← compute_cost(forward_output, Y_batch)
        gradients ← back_propagation(X_batch, Y_batch, parameters, forward_output)
        parameters ← update_parameters(parameters, gradients, learning_rate)
    Endfor
    Y_predict ← predict(X_test, parameters)
    acc ← accuracy(Y_predict, Y_test)
    train_loss(i) ← norm(cost)
    test_acc(i) ← acc
Endfor
Plot(epochs, train_loss)
Plot(epochs, test_acc)
```

Fig. 5. Pseudo Code for Main

IV. Hyper-parameters Studies

A. Number of Epochs

Based on the figures, the number of epochs does play a big role on the testing accuracy of the model. When the number of epochs is low(i.e. = 50), the overall accuracy of the model is about 2% lower than the model with 150 epochs. Meanwhile, a much higher epoch number may not give much better result. In the case of HW3, the model with 300 epochs settles on a test accuracy that is not higher than the model with 150 epochs while spending twice as much computing time.

B. Number of Hidden Layers

The number of hidden layers also play a role on the converging behavior of neural network. When the number of hidden layer increase by one(from 2), the model does converge at a much faster way than before. The model with 3 hidden layers is able to reach the accuracy plateau much before 50 epochs compares to around 100 epochs for 2 hidden layers model, with some sacrifices on maximum test accuracy. However, when the number of epochs goes to 65, the model crashes and yield NAN loss for future epochs. When the number of hidden layer is increased further to 5, the model fails to propagate and is unable to yield proper prediction.

C. Learning Rate

Learning rate is the last hyper-parameters that controls the converging behavior of the model. When learning rate is high(at 0.1), the model is able to converge to 0.9 accuracy within a few epochs. So a high learning rate at the beginning will save computation time. Nevertheless, when the model keeps propagating, the model crashes with NAN loss output. Next, a low learning rate(0.001) is tested with the rest of hyper-parameters unchanged. It is found that a low learning rate increases the number of epochs required to improve accuracy.

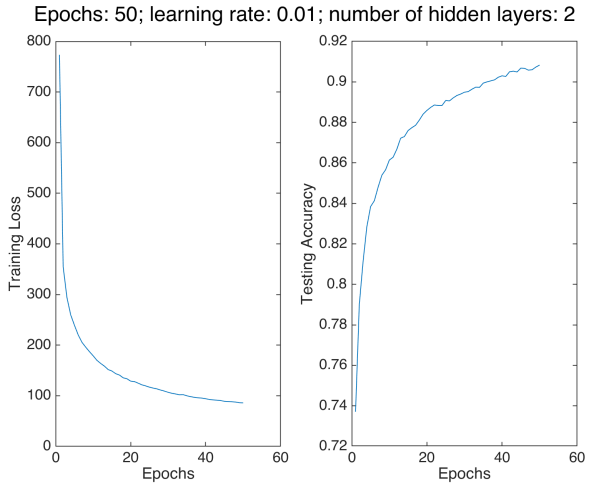

Fig. 6. Output Plot for Epoch=50, Learning Rate = 0.01, Number of Layers = 2

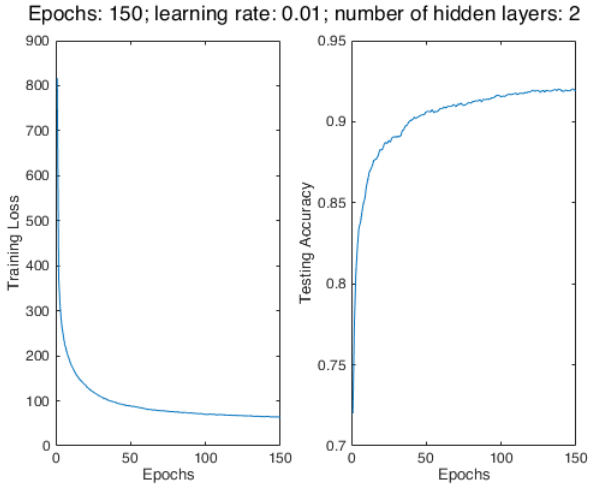

Fig. 7. Output Plot for Epoch=150, Learning Rate = 0.01, Number of Layers = 2

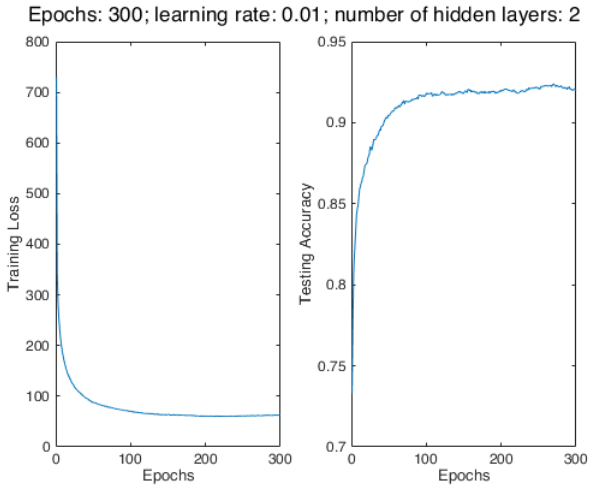

Fig. 8. Output Plot for Epoch=300, Learning Rate = 0.01, Number of Layers = 2

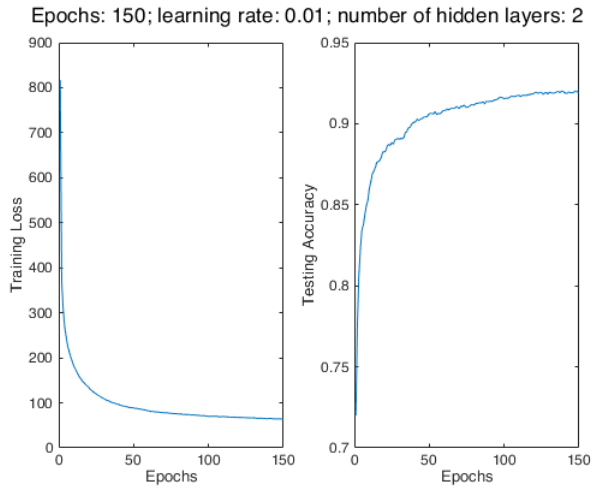

Fig. 9. Output Plot for Epoch=150, Learning Rate = 0.01, Number of Layers = 2

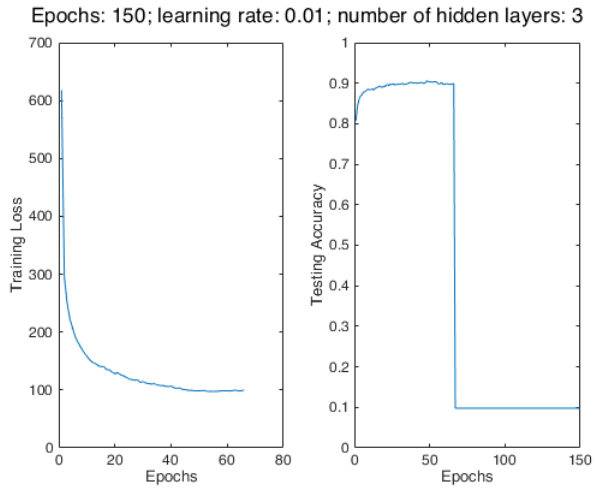

Fig. 10. Output Plot for Epoch=150, Learning Rate = 0.01, Number of Layers = 3

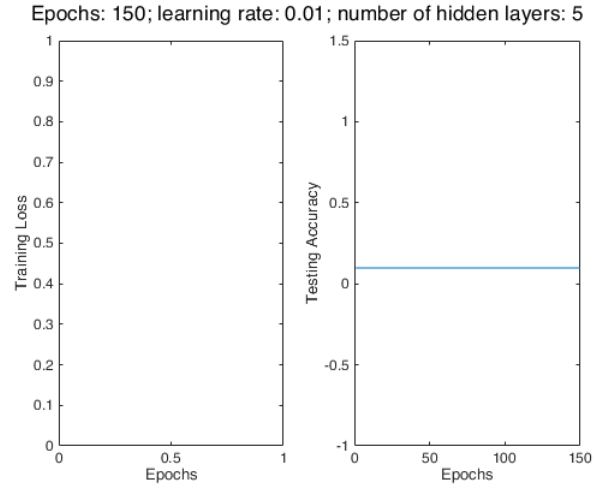

Fig. 11. Output Plot for Epoch=150, Learning Rate = 0.01, Number of Layers = 5

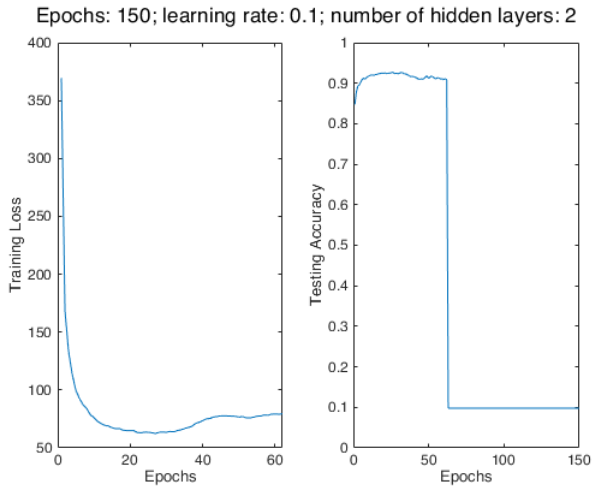

Fig. 12. Output Plot for Epoch=150, Learning Rate = 0.1, Number of Layers = 2

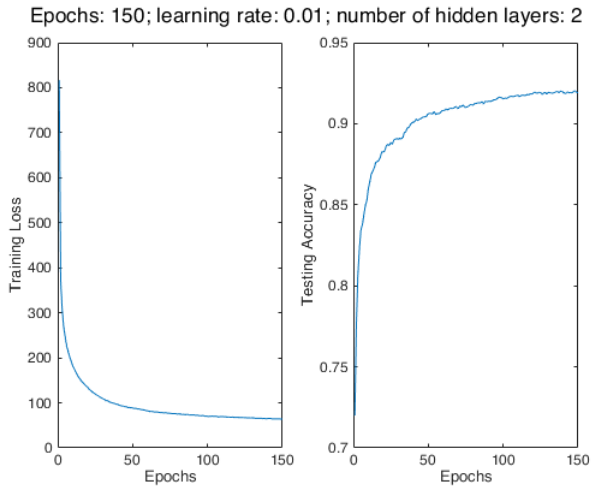

Fig. 13. Output Plot for Epoch=150, Learning Rate = 0.01, Number of Layers = 2

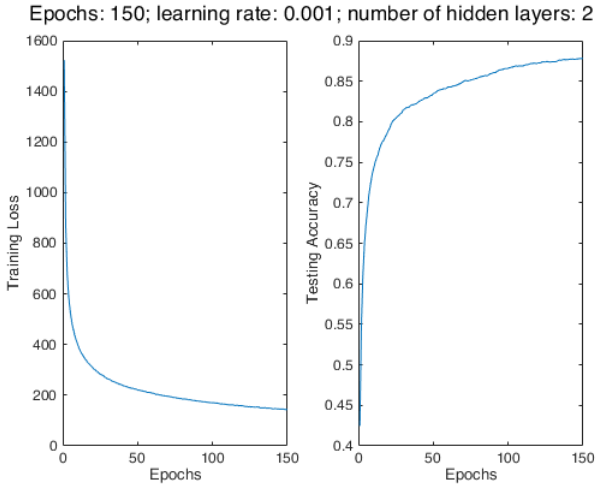

Fig. 14. Output Plot for Epoch=150, Learning Rate = 0.001, Number of Layers = 2

Within the 150 epochs given to the training process, the model with a 0.001 learning rate fails to obtain a higher accuracy than the model with learning rate=0.01.

V. Conclusion

In all, neural network method is able to train a excellent model of predicting the number on image given the proper hyper-parameters and correct supporting functions. According to the hyper-parameters studies, each hyper-parameters can not be increased or decreased arbitrarily without considering the types of data and other hyper-parameters. In addition, the mere increase or decrease of any hyper-parameters will not guarantee a higher performance in terms of prediction accuracy. Hence, for this homework, the most cost-effective hyper-parameters are epoch=150, learning rate = 0.01, hidden layers=2. This combination of parameters is able to provide a test accuracy of 0.9228 while avoiding wasting time on excessive epochs.

References

1 M. Khalid Jawed, Class Homework, Topic: "Homework 3.", University of California, Los Angeles. Nov, 2023.