CSE 569: Fundamentals of Stat. Learning Project Part2 Report

Ziming Dong[1]

***Abstract*—This part of the project contains two tasks. Task 1 is to implement a three-layer MLP from scratch and then train and test it with the given data. Task 2 was to experiment with ”deep” learning (compared to ”shallow” 3-layer MLP) on the MNIST dataset.**

I. TASK1

For task1, we will implement a 2-nH-1 MLP from scratch, without using any built-in library. We need to implement MLP with different number of hidden nodes and choose effective activation function, then we will use MSE error as the loss to train the network. We also need to decide on other details such as batch/mini-batch/stochastic learning, learning rate, whether to use momentum term, etc.

A. Data Preprocessing

We have four TXT files as training and testing data. For training data, we have total 4000 samples for two classes, each sample has two features. For each class, we split the samples into training and validation dataset, for each class, we remaining 500 (1000 total for 2 class) as the ”validation set”. After we train the network until the learning loss/error for the validation set no longer decreases, we will use the network to test the classification accuracy based on the test samples. I use numpy library to transfer sample from TXT format to array. For the next step, I calculate mean and standard deviation for training examples, I apply the normalization formula to normalize the whole samples dataset. Then, I create the train, validation, and test labels based on the number of samples. After preprocessing the dataset, the next step will be building the MLP network.

B. Build MLP Network

To build the three layer MLP network. I create a class called ’MLP’ to initial the input nodes, output nodes, weights, and learning rate. The forward and backward function are used to update the weights between layers. I also create the activation class called ”relu” to use ReLu function activate the neuron in the layers. ReLu stands for Rectified Linear Unit, it is usually used in deep learning and convolutional neural networks. As figure shows below, the ReLu is half rectified. The output wil become zero when value less than zero, otherwise, the output is equal to z if value of z is greater or equal than zero.

Furthermore, I write the function calculate the mean squared error as loss. Then I set up the batch size 128 to do the mini training. Finally, I write a function to get mean classification

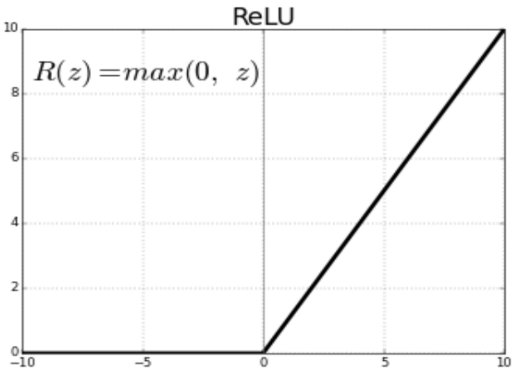


Fig. 1. ReLu Function

accuracy based on the testing dataset. For the last step, I write a loop with 350 epochs to use training data to train the network and test the network on testing data. There are some parameters can be varied, such as number of hidden nodes in the hidden layer and batch size, learning rate etc. Based on the different number of hidden nodes, I report the classification accuracy and I plot the learning curve with corresponding hidden nodes. Below figures shows the learning curve for training, validation, and testing dataset.

C. Plot Learning Curves

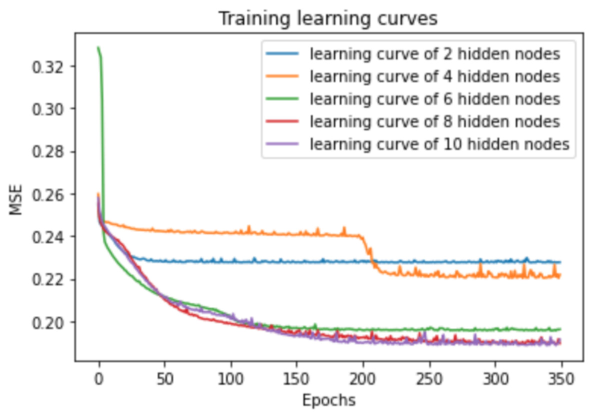


Fig. 2. Training learning curves

Moreover, the below table shows classification result for different hidden nodes.

[1] Z. Dong with the School of Computer Sciences, Big Data System, Arizona State University `zdong27@asu.edu`

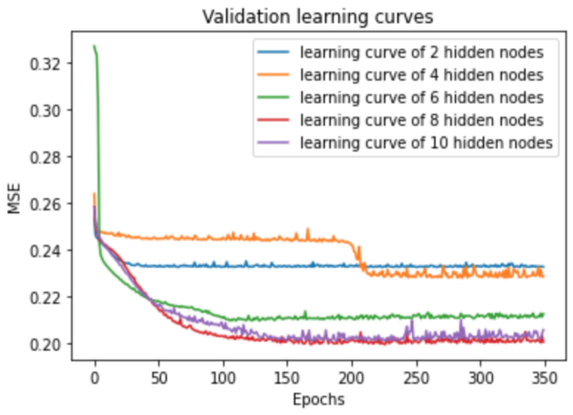


Fig. 3. Validation learning curves

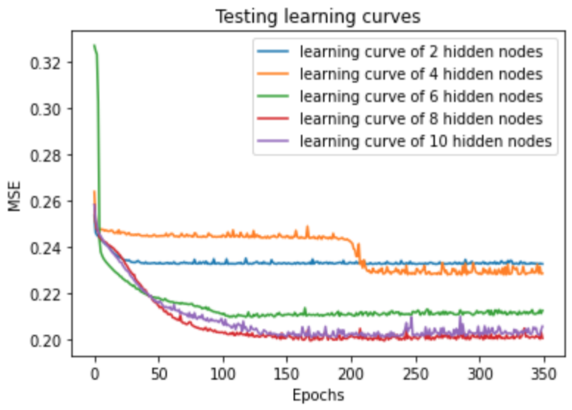


Fig. 4. Testing learning curves

TABLE I
CLASSIFICATION ACCURACY WITH VARIED NUMBER OF HIDDEN NODES

	2 hidden nodes	4 hidden nodes	6 hidden nodes	8 hidden nodes	10 hidden nodes
Accuracy	0.616	0.6405	0.7065	0.713	0.721

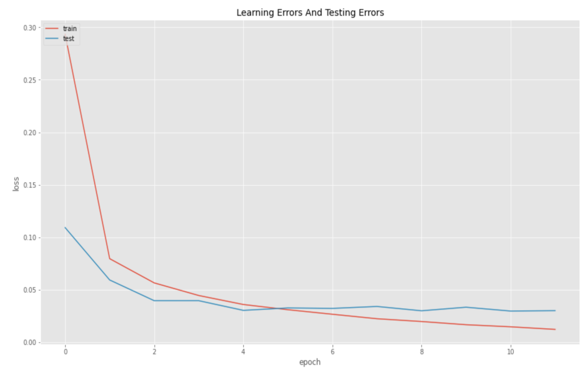
D. *Conclusion*

Based on the result which I got, I find the classification accuracy will increase if more hidden nodes in the hidden layers, I try to start learning from different random initializations, I have different results. So I report average classification accuracy. As we see the table, 10 hidden nodes gives the best classification accuracy for the testing set, which is 0.721. Based on the learning curve plots, we can find the MSE loss still decrease when reaching the end of epochs. However, there is a sharp decreasing event happened around the100th Epochs, then the loss decrease slowly. I consider parameters is the factor to effect the moment when learning loss/error no longer decrease. When I tried increase the learning rate and batch size, I get different stop point for learning curves, but I think it is fine as long as the loss keep decreasing. For this task, I learned the effect of different number of hidden nodes to the classification accuracy and loss. If I have more time in the future, I will choose different activation functions to test the dataset to see if I can get higher accuracy.

II. TASK2

For this task, we need to adjust the parameters in the convolutional neural network to see the change of loss and accuracy. I changed batch size, number of feature maps, kernel size, learning rate, and optimizer with the baseline code, then I print the testing loss and accuracy and plot the learning curve with corresponding changed parameter. The following figures show the result of five different combinations of parameter setting.

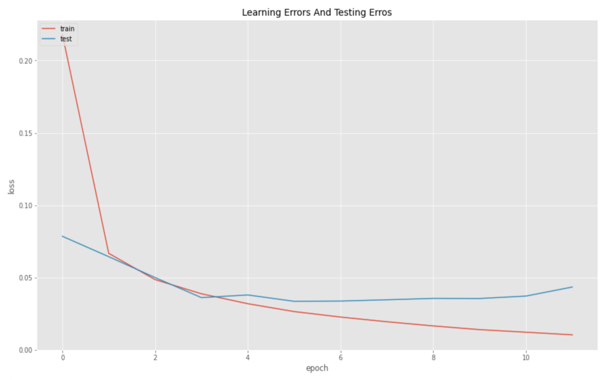
```
Plot the learning/testing errors with original Base Line
```



```
Test loss: 0.029992980882525444
Test accuracy: 0.9912999868392944
```

Fig. 5. Result of Baseline code

```
Plot the learning/testing errors with 64 Batch Size
```

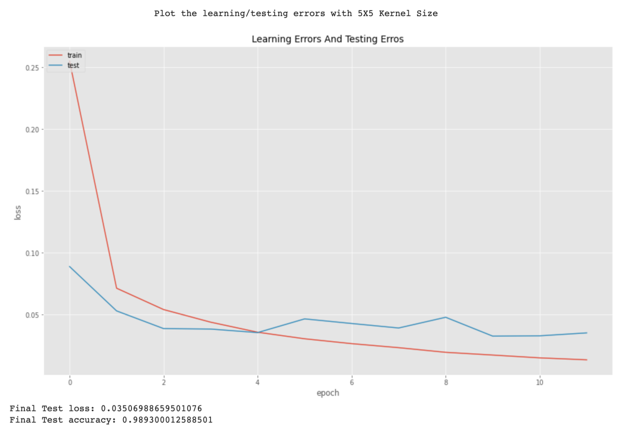


```
Final Test loss: 0.04341362789273262
Final Test accuracy: 0.9886000156402588
```

Fig. 6. Reduce batch size to 64

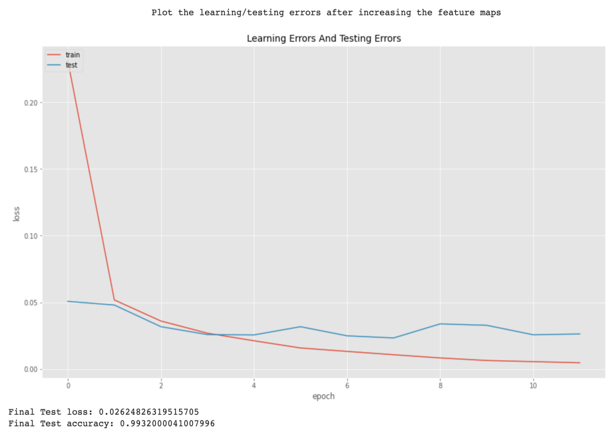
A. *Findings*

Changing the parameter helps me understanding the behavior/performance of the network. In the baseline code, the input size of the image is 28x28 and we have two convolutional layers, first layer has 6 feature maps with 3x3 kernel size. The second convolutional layer has 16 feature maps with 3x3 kernel size. All layers follow a 2x2 pooling with stride 1. After max pooling, the layer is fully connected to the next hidden layer with 120 nodes and relus as the active function. The fully connected layer is followed by another fully connected layer



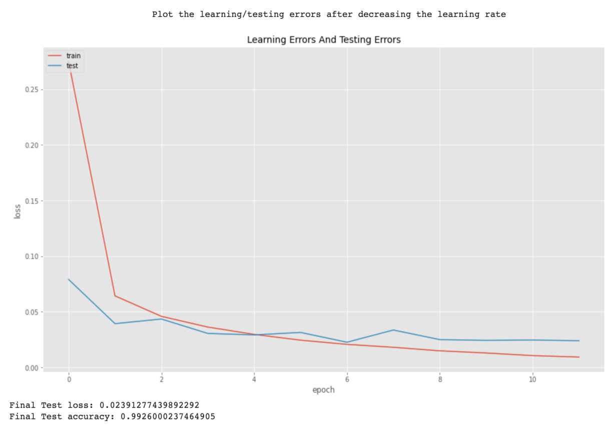
```
Final Test loss: 0.03506988659501076
Final Test accuracy: 0.989300012588501
```

Fig. 7. Change the kernel size to 5x5



```
Final Test loss: 0.02624826319515705
Final Test accuracy: 0.9932000041007996
```

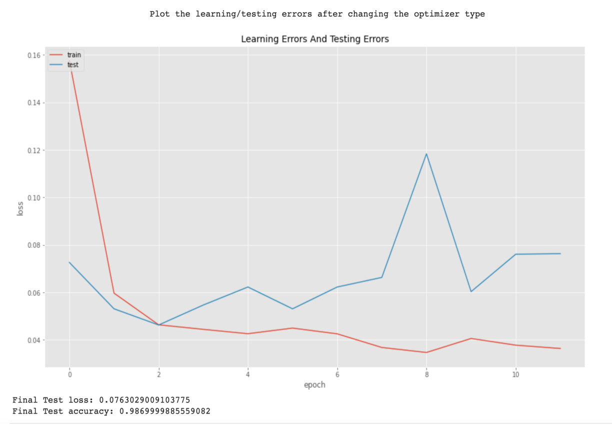
Fig. 8. Double the feature maps



```
Final Test loss: 0.02391277439892292
Final Test accuracy: 0.9926000237464905
```

Fig. 9. Decreasing the learning rate

with 84 nodes and relus as the active function. Then there will be 10 classes connected with 10 output nodes in the soft max layer. As you see, the accuracy on the test dataset is very high based on the baseline code, which is 0.9913. After I change the batch size to 64, the test accuracy is a little lower than 128 batch size, which is 0.988, so I assume the suitable batch size



```
Final Test loss: 0.0763029009103775
Final Test accuracy: 0.9869999885559082
```

Fig. 10. Change Optimizer to Adam

for training the dataset is 128 rather than 64. After I change the kernel size to 5x5 for all layers, the accuracy of testing is also lower than the 3x3 kernel size, but the learning/testing errors and accuracy do not change too much, but the testing loss is higher than the baseline code, which layer's kernel size is 3x3.Since this experiment is composed of 2 convolutions block 2 dense layers. The convolutional kernel size need for convolutional layer is c * (k2 * 1 2), to reduce the number of weights, we need c * (k2 * 1 2) + 1 2 * c2, let‘s consider the 5*5 kernel size convolutional layer, k square is smaller than c ¿128, so we have a ratio of rough kernel surface: 9 or 25. At this point, I think both 3*3 and 5*5 kernel size convolutional layers will have good performance in this experiment. After I increase the feature maps,the testing accuracy is higher than the baseline code, which means increase number of the feature maps can improve the experiment for the testing data. After we decrease the learning rate, we found the accuracy do not change very much than the baseline, which means the learning rate between 0.5-1.0 is fine for the network. The last change is for the optimizer of network, I change the optimizer from Adadelta to Adam, but the accuracy is lower than baseline. I believe the adam is not the best the optimizer for some tasks as others, there are some flaws of adam optimizer which might be the reason to get the lower accuracy, for exapmple, the network may not converge very well and it might miss the the optimal solution. There is a solution to solve the problems of adam optimize, we can use adam in the early stage and benefit from the advantage of adam's fast convergence, in the later stage, we can switch to SGD and search for the optimal solution.