Deep Learning

MNIST Digits Classification with MLP

Author: Yantian Luo

Institute: Electronic Engineering

ID: 2018310742

Update: March 21, 2019

Contents

Chapter 1 Introduction

MNIST digits dataset is a widely used database for image classification in machine learning field. It contains 60,000 training samples and 10,000 testing samples. Each sample is a 784×1 column vector, which is transformed from an original 28×28 pixels grayscale image.

In this homework, we use multilayer perceptron (MLP) to perform digits classification. We construct a neural network with one hidden layer and two hidden layers to compare the results. We also use Relu and Sigmoid as activation functions to compare different results. And we also compare the differenc ebetween EuclideanLoss and SoftmaxCrossEntropyLoss for all the above experiments.

Chapter 2 Algorithm Design

For multilayer perceptron, forward and backward is most important. Forward represents the data processing performed by the layer and backward performs backpropagation operations. According to the chain rule, the most import thing for backward is compute the derivation of output with respect to input. In this chapter, we compute and derive the data processing formula of forward and backward.

2.1 Layers

In this section, we compute and derive the formula of three types of layers, which are Relu, Sigmoid and Linear layers.

2.1.1 Relu layer

Figure 2.1 show the Relu functions.

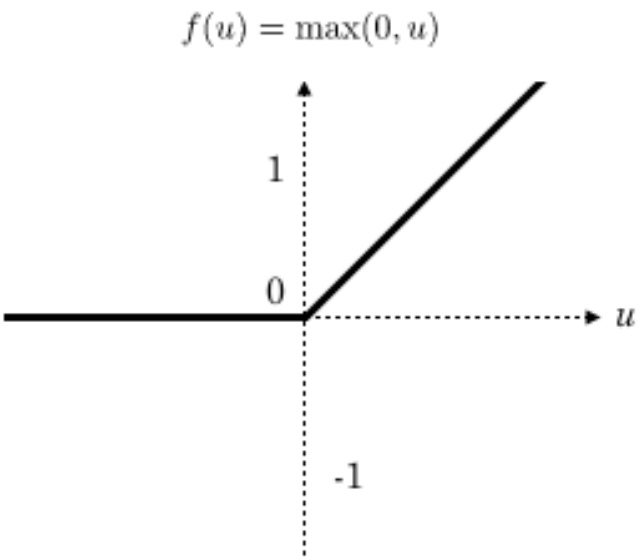

Figure 2.1: Relu activation functions

Suppose the input of Relu layer is X, and the output is Y. According to relu function,

we can get forward operation as formula (2.1).

$$\begin{aligned} Y &= \begin{cases} X & X > 0 \\ 0 & X \leq 0 \end{cases} \\ &= \max\{0, X\} \end{aligned} \tag{2.1}$$

And we can easily get the derivation of it as follow:

$$\frac{\partial Y}{\partial X} = \begin{cases} 1 & X > 0 \\ 0 & X \leq 0 \end{cases} \tag{2.2}$$

2.1.2 Sigmoid layer

Figure 2.2 show the Sigmoid functions.

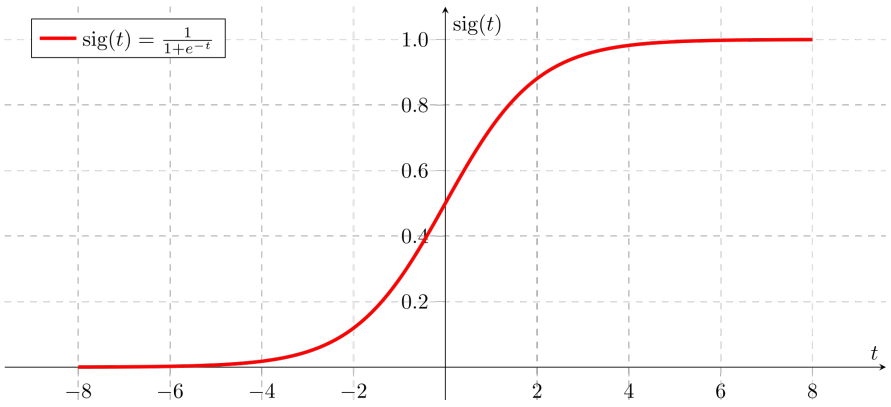

Figure 2.2: Sigmoid activation functions

Suppose the input of Relu layer is X, and the output is Y. According to sigmoid function, we can get forward operation as formula (2.3).

$$Y = \frac{1}{1 + e^{-X}} \tag{2.3}$$

Then we can compute the derivation of Y with respect to X as follow:

$$\begin{aligned} \frac{\partial Y}{\partial X} &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\ &= Y(1 - Y) \end{aligned} \tag{2.4}$$

2.1.3 Linear layer

Figure 2.3 show the linear layer.

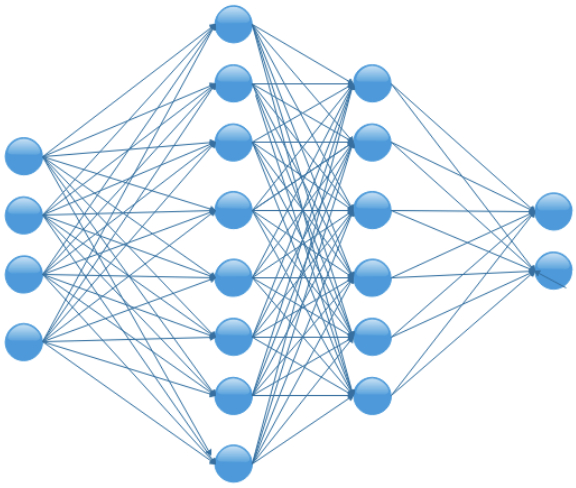

Figure 2.3: Linear layer (also called fully connected layer)

Suppose the input of linear layer is X, and the output is Y. Assuming the weight matrix is W and the bias vector is b, then we have formula (2.3).

$$Y = W \cdot X + b \tag{2.5}$$

Then we can compute the derivation of Y with respect to X, W, b as follow:

$$\begin{cases} \dfrac{\partial Y}{\partial X} = W \\ \dfrac{\partial Y}{\partial W} = X \\ \dfrac{\partial Y}{\partial b} = 1 \end{cases} \tag{2.6}$$

2.2 Losses

In this section, we compute the forward and backward formulas of two types of losses: EuclideanLoss and SoftmaxCrossEntropyLoss.

2.2.1 EuclideanLoss

Suppose the predict label is y and the true label is T. And assuming the T is in one-hot encoding form.

Forward

Assuming the batchsize is N, we can derivate the EuclideanLoss as follow:

$$E = \frac{1}{2N}\sum_{n=1}^{N}||T(n) - y(n)||^2 \tag{2.7}$$

Backward

We can compute the derivation of E with respect to y as follow:

$$\begin{aligned}\frac{\partial E}{\partial y(n)} =& \frac{1}{2N}\cdot 2(y(n) - T(n)) \\ =& \frac{1}{N}\left[y(n) - T(n)\right] \\ & \forall n = 1, 2, \cdots, N\end{aligned} \tag{2.8}$$

2.2.2 SoftmaxCrossEntropyLoss

Suppose the predict label is x and the true label is t. And assuming the t is in one-hot encoding form.

Forward

To compute the SoftmaxCrossEntropyLoss, we need three steps:

- Compute softmax of each label;

$$h_k^{(n)} = P(t_k^{(n)} = 1|x^{(n)}) = \frac{\exp(x_k^{(n)})}{\sum_{j=1}^{K}\exp(x_j^{(n)})} \tag{2.9}$$

- Compute crossentropy;

$$E^{(n)} = -\sum_{k=1}^{K} t_k^{(n)} \ln h_k^{(n)} \tag{2.10}$$

- Compute Loss;

$$E = \frac{1}{N}\sum_{n=1}^{N} E^{(n)} \tag{2.11}$$

2.2.3 Backward

We use the chain rule to compute the derivation of E with respect to x as follow:

$$\frac{\partial E}{\partial x^{(n)}} = \frac{\partial E}{\partial E^{(n)}} \frac{\partial E^{(n)}}{\partial x^{(n)}} \tag{2.12}$$

First, we compute $\frac{\partial E^{(n)}}{\partial x^{(n)}}$ as follow:

$$\begin{aligned} \frac{\partial E^{(n)}}{\partial x_k^{(n)}} &= \sum_{i=1}^{K} \frac{\partial E^{(n)}}{\partial h_i^{(n)}} \frac{\partial h_i^{(n)}}{\partial x_k^{(n)}} \\ &= \sum_{i=1}^{K} \left(-t_i^{(n)} \frac{1}{h_i^{(n)}} \right) \frac{\partial h_i^{(n)}}{\partial x_k^{(n)}} \end{aligned} \tag{2.13}$$

Then we compute $\frac{\partial h_i^{(n)}}{\partial x_k^{(n)}}$ to get $\frac{\partial E^{(n)}}{\partial x^{(n)}}$:

$$\begin{aligned} \frac{\partial h_i^{(n)}}{\partial x_k^{(n)}} &= \begin{cases} h_i^{(n)}(1 - h_k^{(n)}) & i = k \\ -h_i^{(n)} h_k^{(n)} & i \neq k \end{cases} \\ &= h_i^{(n)}(\triangle_{i,k} - h_k^{(n)}) \\ where \quad \triangle_{i,k} &= \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases} \end{aligned} \tag{2.14}$$

Then we have:

$$\begin{aligned} \frac{\partial E^{(n)}}{\partial x_k^{(n)}} &= \sum_{i=1}^{K} \left(-t_i^{(n)} \frac{1}{h_i^{(n)}} \right) h_i^{(n)}(\triangle_{i,k} - h_k^{(n)}) \\ &= \sum_{i=1}^{K} \left(-t_i^{(n)} \triangle_{i,k} \right) + \sum_{i=1}^{K} t_i^{(n)} h_k^{(n)} \\ &= -(t_k^{(n)} - h_k^{(n)}) \end{aligned} \tag{2.15}$$

Therefore, we can compute the backward formula:

$$\begin{aligned} \frac{\partial E}{\partial x_k^{(n)}} &= \frac{\partial E}{\partial E^{(n)}} \frac{\partial E^{(n)}}{\partial x^{(n)}} \\ &= -\frac{1}{N}(t_k^{(n)} - h_k^{(n)}) \end{aligned} \tag{2.16}$$

Chapter 3 Experients and Results

In this chapter, we compare the difference of Relu vs. Sigmoid active functions, EuclideanLoss vs. SoftmaxCrossEntropyLoss in one hidden layer network and two hidden layers network.

3.1 One hidden layer experiments

The network structure in this section is designed as Figure 3.1 and training arguments is as follow:

```
config = {
  'learning_rate': 0.1,
  'weight_decay': 0.0001,
  'momentum': 0.001,
  'batch_size': 100,
  'max_epoch': 100,
  'disp_freq': 50,
}
```

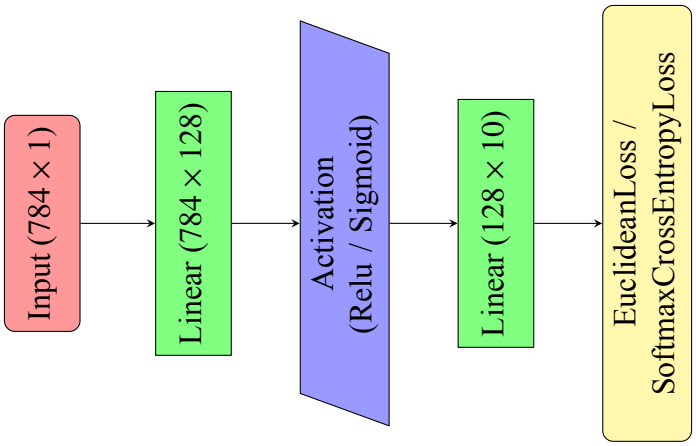

Figure 3.1: Network Structure with one hidden layer

3.1.1 Using Relu and EuclideanLoss

First we use Relu as activation function and EuclideanLoss as loss function to take the experiment.

Then we draw the train accuracy curve and train loss curve as shown in 3.2 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.3

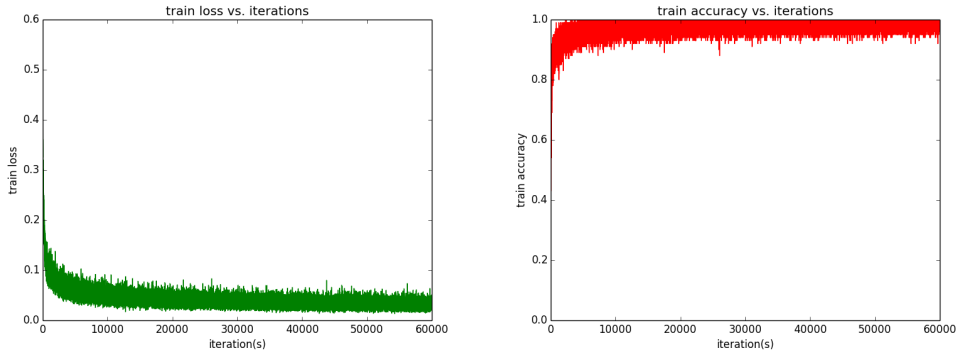

Figure 3.2: train loss curve and train accuracy curve with one hidden layer, Relu activation function and EuclideanLoss

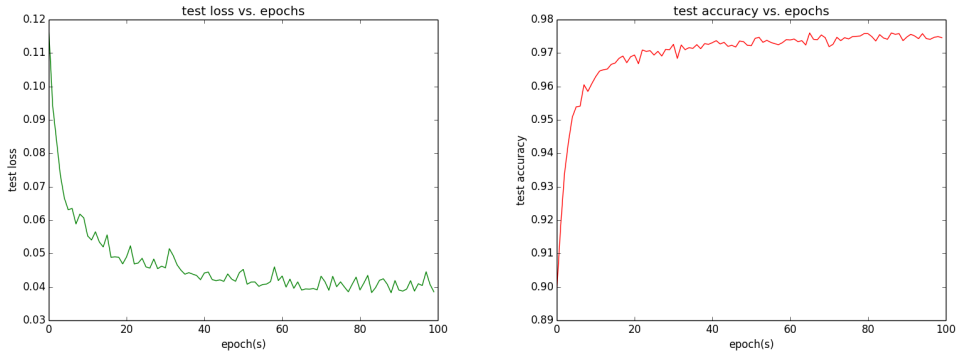

Figure 3.3: test loss curve and test accuracy curve with one hidden layer, Relu activation function and EuclideanLoss

3.1.2 Using Sigmoid and EuclideanLoss

Then we use Sigmoid as activation function and EuclideanLoss as loss function to take the experiment.

Next we draw the train accuracy curve and train loss curve as shown in 3.4 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.5

3.1.3 Using Relu and SoftmaxCrossEntropyLoss

Then we use Relu as activation function and SoftmaxCrossEntropyLoss as loss function to take the experiment.

Next we draw the train accuracy curve and train loss curve as shown in 3.6 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.7

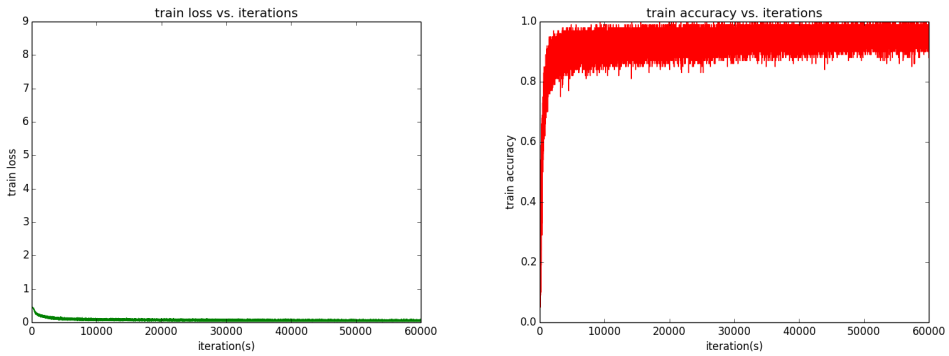

Figure 3.4: train loss curve and train accuracy curve with one hidden layer, Sigmoid activation function and EuclideanLoss

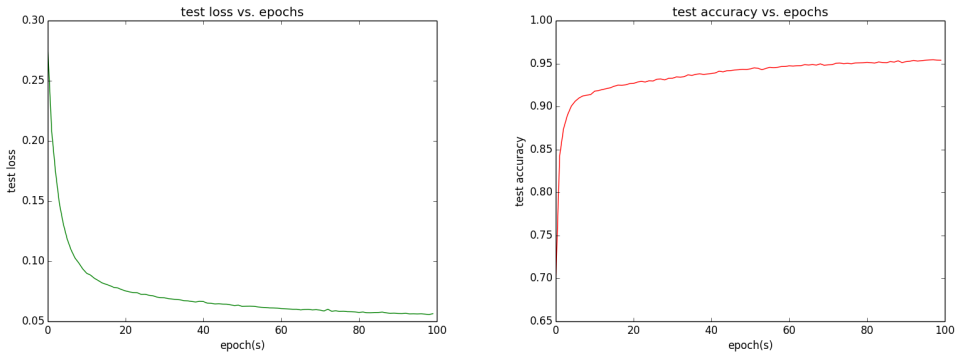

Figure 3.5: test loss curve and test accuracy curve with one hidden layer, Sigmoid activation function and EuclideanLoss

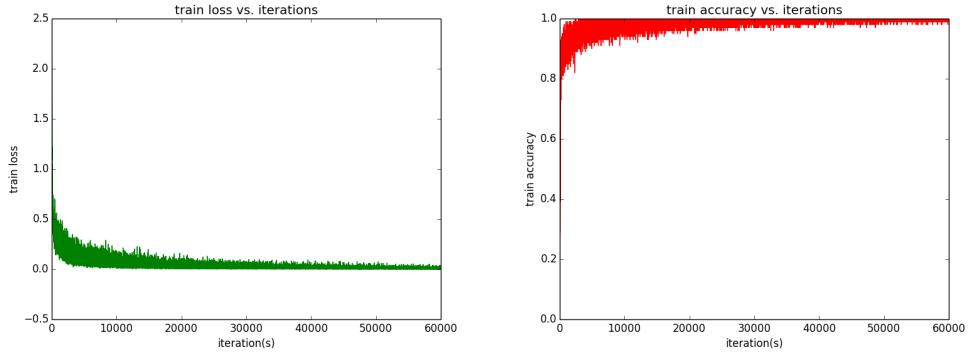

Figure 3.6: train loss curve and train accuracy curve with one hidden layer, Relu activation function and SoftmaxCrossEntropyLoss

3.1.4 Using Sigmoid and SoftmaxCrossEntropyLoss

Then we use Sigmoid as activation function and SoftmaxCrossEntropyLoss as loss function to take the experiment.

Next we draw the train accuracy curve and train loss curve as shown in 3.6 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.7

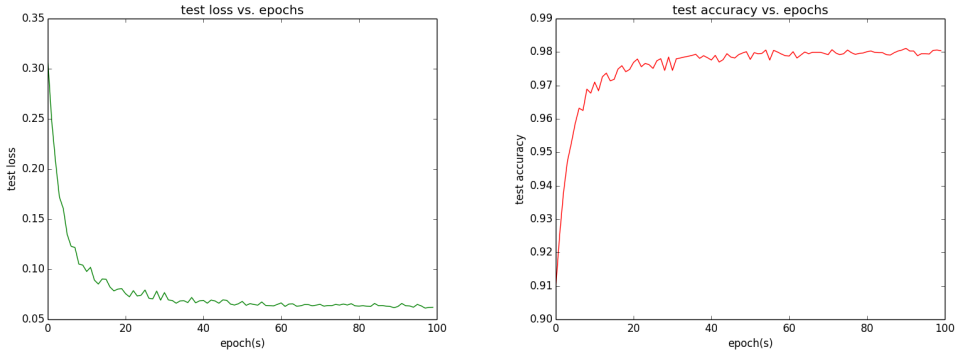

Figure 3.7: test loss curve and test accuracy curve with one hidden layer, Relu activation function and SoftmaxCrossEntropyLoss

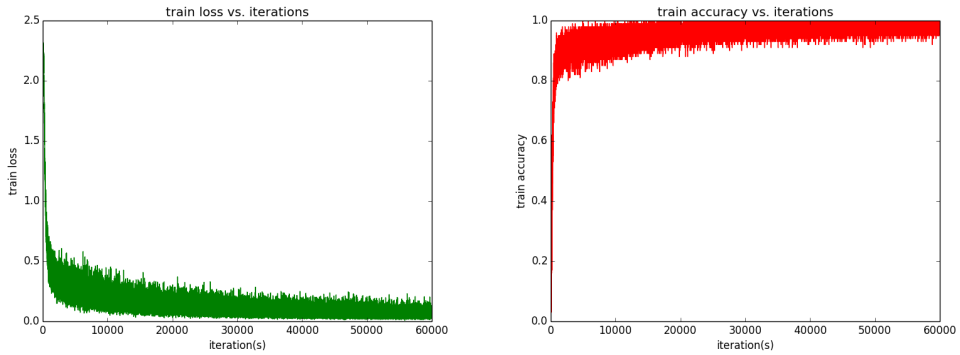

Figure 3.8: train loss curve and train accuracy curve with one hidden layer, Sigmoid activation function and SoftmaxCrossEntropyLoss

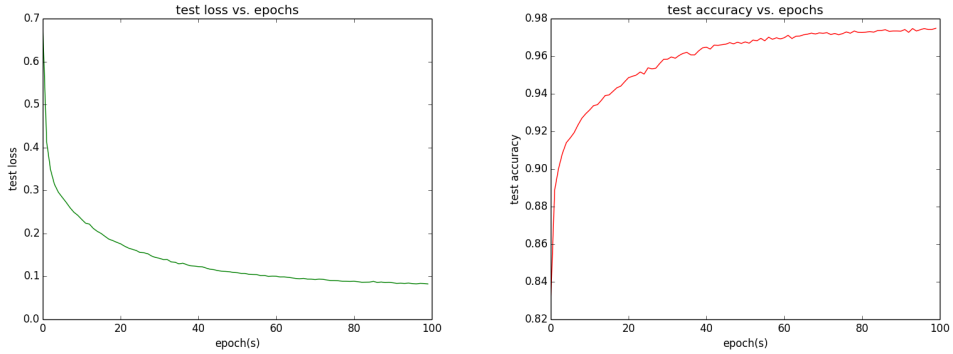

Figure 3.9: test loss curve and test accuracy curve with one hidden layer, Sigmoid activation function and SoftmaxCrossEntropyLoss

3.2 Two hidden layers experiments

The network structure in this section is designed as Table**??** and training arguments is as follow:

```
config = {
    'learning_rate': 0.1,
    'weight_decay': 0.0001,
```

```
    'momentum': 0.001,
    'batch_size': 100,
    'max_epoch': 100,
    'disp_freq': 50,
}
```

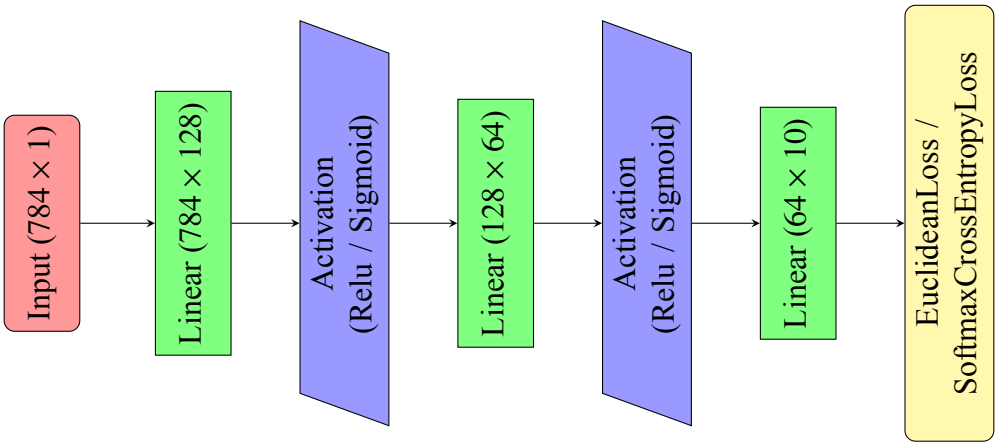

Figure 3.10: Network Structure with two hidden layers

3.2.1 Using Relu and EuclideanLoss

First we use Relu as activation function and EuclideanLoss as loss function to take the experiment.

Then we draw the train accuracy curve and train loss curve as shown in 3.11 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.12

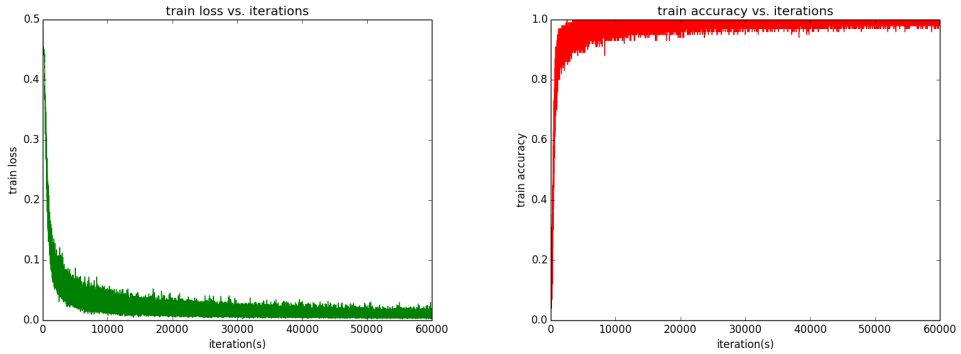

Figure 3.11: train loss curve and train accuracy curve with two hidden layers, Relu activation function and EuclideanLoss

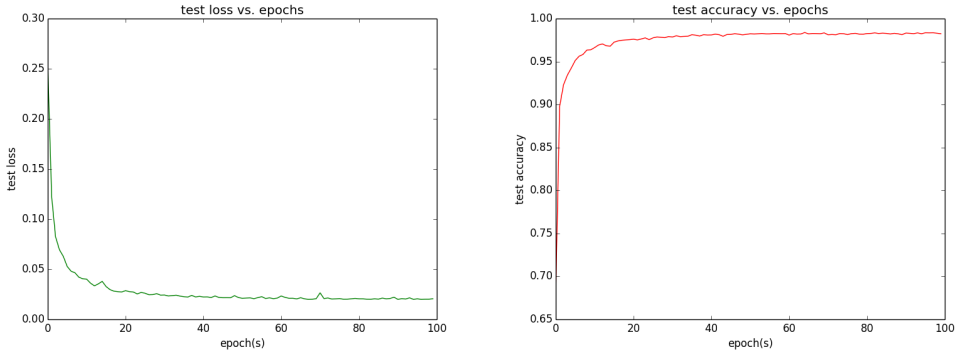

Figure 3.12: test loss curve and test accuracy curve with two hidden layers, Relu activation function and EuclideanLoss

3.2.2 Using Sigmoid and EuclideanLoss

First we use Sigmoid as activation function and EuclideanLoss as loss function to take the experiment.

Then we draw the train accuracy curve and train loss curve as shown in 3.13 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.14

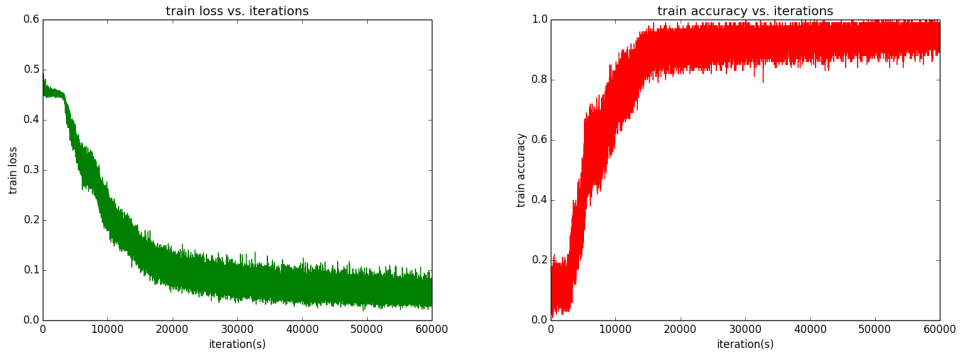

Figure 3.13: train loss curve and train accuracy curve with two hidden layers, Sigmoid activation function and EuclideanLoss

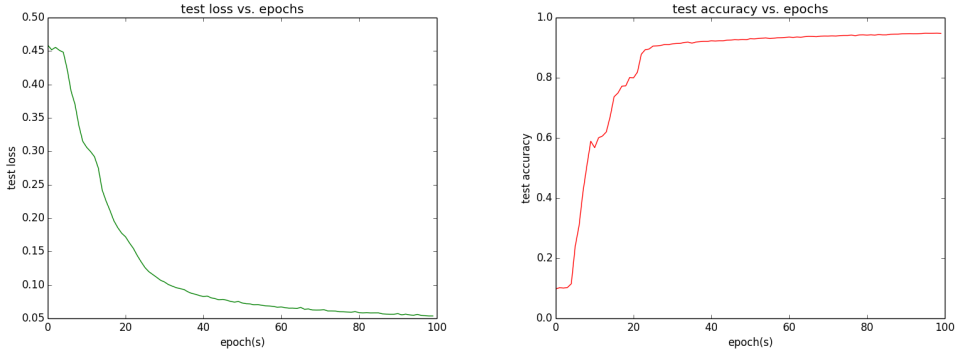

Figure 3.14: test loss curve and test accuracy curve with two hidden layers, Sigmoid activation function and EuclideanLoss

3.2.3 Using Relu and SoftmaxCrossEntropyLoss

First we use Relu as activation function and SoftmaxCrossEntropyLoss as loss function to take the experiment.

Then we draw the train accuracy curve and train loss curve as shown in 3.15 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.16

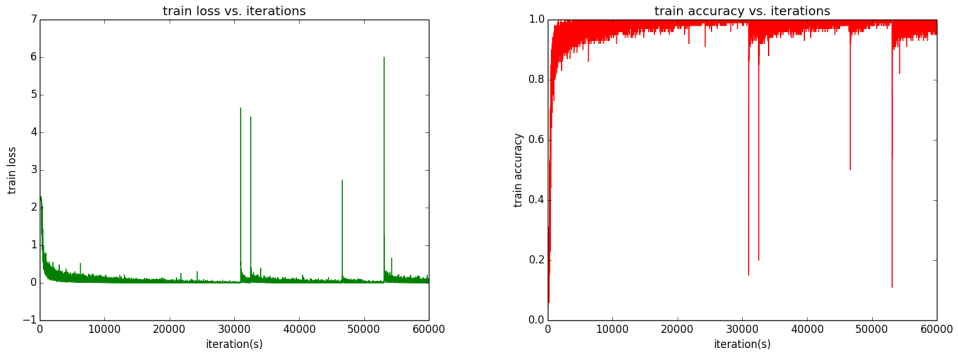

Figure 3.15: train loss curve and train accuracy curve with two hidden layers, Relu activation function and SoftmaxCrossEntropyLoss

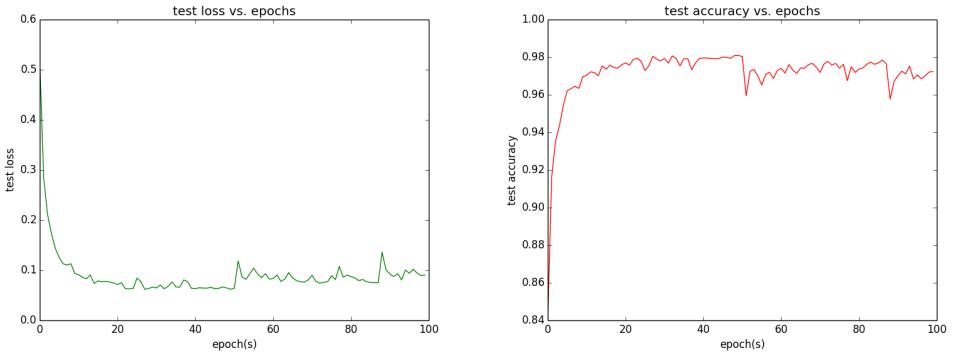

Figure 3.16: test loss curve and test accuracy curve with two hidden layers, Relu activation function and SoftmaxCrossEntropyLoss

3.2.4 Using Sigmoid and SoftmaxCrossEntropyLoss

First we use Sigmoid as activation function and SoftmaxCrossEntropyLoss as loss function to take the experiment.

Then we draw the train accuracy curve and train loss curve as shown in 3.17 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in 3.18

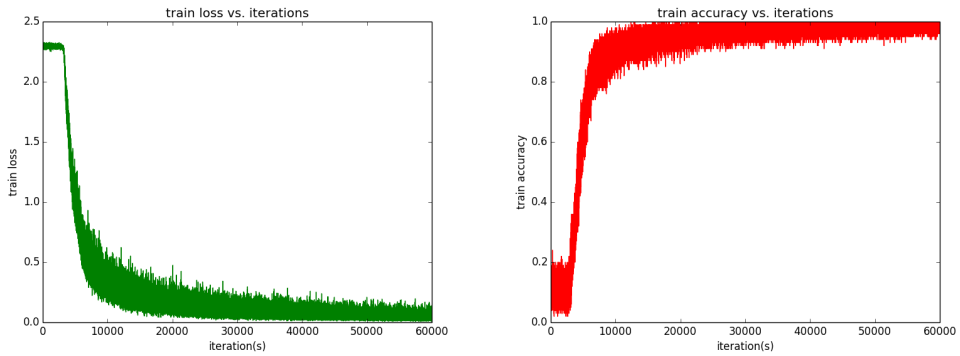

Figure 3.17: train loss curve and train accuracy curve with two hidden layers, Sigmoid activation function and SoftmaxCrossEntropyLoss

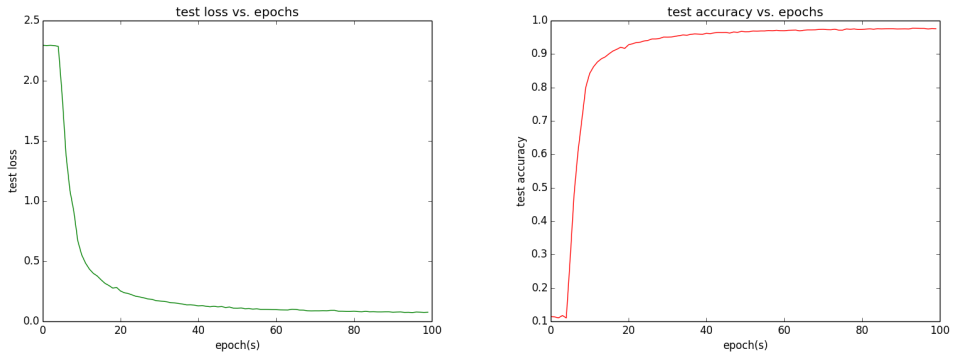

Figure 3.18: test loss curve and test accuracy curve with two hidden layers, Sigmoid activation function and SoftmaxCrossEntropyLoss

3.3 Results

We summarize all test accuracys and test losses of every experiments as shown in Table 3.1(In this table, RE represents the experiments using Relu activation function and EuclideanLoss, SE represents the experiments using Sigmoid activation function and EuclideanLoss, RS represents the experiments using Relu activation function and SoftmaxCrossEntropyLoss, SS represents the experiments using Sigmoid activation function and SoftmaxCrossEntropyLoss). From the results, we can find the best chance is the experiments that use Relu activation function and EuclideanLoss with two hidden layers, the test accuracy is 0.983.

Table 3.1: Results Summary

Cases	RE	SE	RS	SS
Test Acc with one hidden layer	0.975	0.951	0.982	0.975
Test Loss with one hidden layer	0.039	0.062	0.062	0.083
Test Acc with two hidden layers	0.983	0.947	0.981	0.976
Test Loss with two hidden layers	0.021	0.053	0.065	0.075

3.3.1 Comparation of Relu and Sigmoid

From the results we can find that if we use Relu as activation function, we have bigger test accuracy and faster convergence speed. And Sigmoid may result in Gradient disappear, thus for the mnist digits classification, Relu is much better.

3.3.2 Comparation of One hidden layer and Two hidden layers

From the results we can find that two hidden layers have bigger test accuracy than one hidden layer. But Two hidden layers have more arguments than one hidden layer, thus the training time is much longer and may lead to overfitting.

3.3.3 Comparation of EuclideanLoss and SoftmaxCrossEntropyLoss

From the results we can find SoftmaxCrossEntropyLoss is more smooth than EuclideanLoss and the test accuracy of the two loss functions is close.