Neural Network: Two Layers Classifier via Numpy

22110980014 Xu Shi

April 3, 2023

Abstract

In this project, we build a neural network two layers classifier only using Numpy on MNIST. We first train a model, then, seek the best parameters *i.e.*, learn rate, hidden layer size and regularization strength. Finally, we import the model, test it with the model after parameter search, and output the classification accuracy.

1 Introduction

Neural network classifier is a very powerful tool, the two layers classifier is a special case. Here we use the handwriting dataset in MNIST which is one of the most popular datasets for training the neural network model. It contains 50000, 10000, 10000 figures for training, validating and testing, respectively.

We train our model by two linear matrices, weight and biases matrix, and for every layer, respectively. Meanwhile, the output is connected to the activation function, sigmoid and ReLU,respectively.

The readers can get the detail codes in GitHub[1]. Meanwhile, the trained model can be found in Baidu Netdisk[2] with fetch code is {t5jp} (If you can't download it from Baidu Netdisk, you can find it in GitHub, as well).

2 Algorithms

Firstly, we denote the model loss $\mathcal{L}(x, \delta)$, where x is data, δ is parameter. Meanwhile, the model loss also contains the regular term $\frac{1}{2}\lambda\|\delta\|_2^2$. Then we can propose the important stochastic gradient descent (SGD) algorithm as following, Then, we know that to train a model on MNIST datasets is sampling a minibatch

Algorithm 1 Stochastic Gradient Descent (SGD)

```
1: **Input**: Data $x_0$, model loss function $\mathcal{L}$, parameter $\delta_k$, learn rate $\eta$, regular parameter $\lambda$.
2: for $k = 0 : m$ do
3:     Compute $x_{k+1}$ in some rules.
4: end for
5: for $k = m : 0$ do
6:     Compute $\frac{\partial \mathcal{L}}{\partial \delta_k}$ in chain rule.
7:     Update $\delta_k = \delta_k - \eta \frac{\partial \mathcal{L}}{\partial \delta_k}$.
8: end for
```

from the whole datasets and using the SGD method. So we should first randomly shuffle the datasets for stochastic updates. Then, after 1 epoch or more, we validate the model on the validation data and check the accuracy. If the accuracy changes, we should change the learning rate decay strategy accordingly. We propose our training model strategy as following,

[1] https://github.com/Shixu-max/Numpy-neural-network-two-layers-classifier

[2] https://pan.baidu.com/s/1hzkMCHe8OqT3ZuXBRyiuug

Algorithm 2 Training Model Strategy

```
1:  Input: Data $x$, data label $y$, validation data $\hat{x}$, data label $\hat{y}$, learn rate $\eta$, constant $0 < \rho < 1$, iteration $n$.
2:  for $k = 1 : n$ do
3:      Randomly shuffle $x, y$.
4:      for choose a batch of $x, y$ do
5:          Use SGD method on the choosen batch
6:      end for
7:      Compute the accuracy on validation data.
8:      if Accuracy descent then
9:          $\eta = \rho\eta$.
10:     end if
11: end for
```

3 Parameters Seek

In this section, we will propose the best parameters $i.e.$, learn rate, hidden layer size and regularization strength.

3.1 Loss Function

Above all, we should choose a suitable loss function, here we consider two loss functions, the mean square error (MSE) and the binary cross entropy (BCE) loss. Suppose that y, y' are real label and prediction label, respectively. Then, the MSE vs BCE is as following,

$$\begin{aligned} \mathcal{L}_{MSE} &= \frac{1}{2n}\sum_{i=1}^{n}(y_i - y_i')^2, \\ \mathcal{L}_{BCE} &= -\frac{1}{N}\sum_{i=1}^{n}(y_i \ln(y_i') + (1-y_i)\ln(1-y_i')). \end{aligned} \tag{1}$$

Then, we test which is better under activation function is ReLU, fixed learn rate = 1e-2, hidden size = 500, batch size = 40 and epochs = 10, we have following results, From this Figure 1 we can see that, MSE performs better than BCE, so in following part, we choose MSE as loss function. (If you want to see detailed results, you only need to change MSE or BCE in **Change Loss Function** in the code file **twolayerclassifier**).

3.2 Activation Function

Firstly, we know that our neural network have two layers, so we have two activation functions, the second activation function is fixed by sigmoid, since it should map the output to $[0, 1]$. So we test ReLU and sigmoid functions only for the first activation function under loss function is MSE, fixed learn rate = 1e-2, hidden size = 500, batch size = 40 and epochs = 10, we have following results, From this Figure 2 we can see that, ReLU performs better than sigmoid, so in following part, we choose ReLU for the first activation function (If you want to see detailed results, you only need to change ReLU or sigmoid (let first activation function = ReLU or sigmoid) in **Change Activation Function** in the code file **twolayerclassifier**).

3.3 Learn Rate

In Algorithm 2, the learn rate descent strategy is important. If the validation accuracy decreases, we should reduce the learn rate. We train our model with learn rate is one of $[1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128]$ under loss function is MSE, activation function is ReLU, hidden size = 500, batch size = 40 and epochs = 10, the result is as following, Then, from Table 1 we know that **the best learn rate is 0.03125 with corresponding accuracy is 0.9815**.

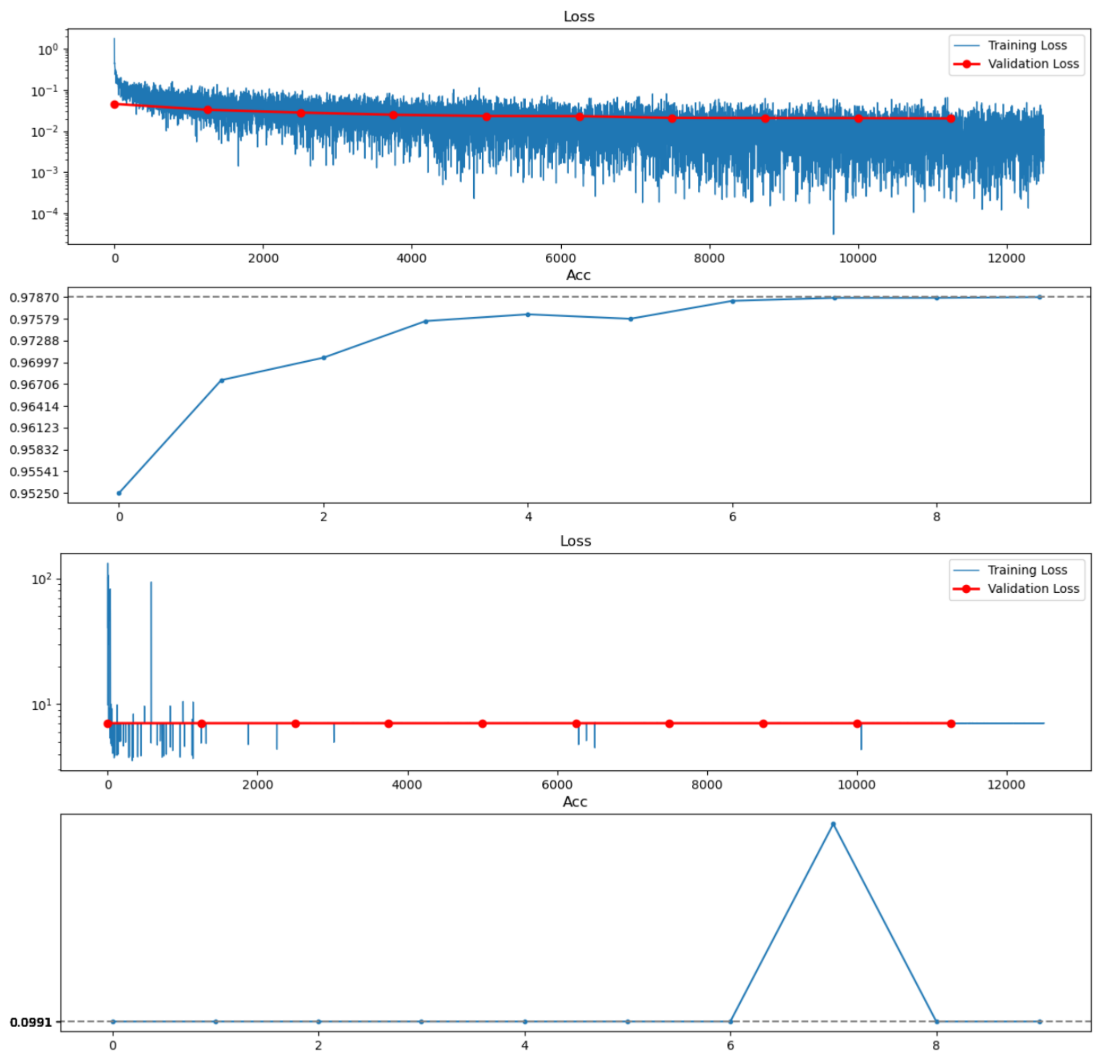

Figure 1: MSE vs BCE

3.4 Hidden Layer Size

The hidden layer size is also very important. We train our model with hidden layer size is one of $[10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]$ under loss function is MSE, activation function is ReLU, learn rate $= 0.03125$, batch size $= 40$ and epochs $= 10$, the result is as following, Then, from Table 2 we know that **the best hidden layer size is 900 with corresponding accuracy is 0.9834**.

3.5 Regularization Strength

For the L2 regularization strength, we train our model with hidden layer size is one of $[1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 0]$ under loss function is MSE, activation function is ReLU, learn rate $= 0.03125$, hidden layer size $= 900$, batch size $= 40$ and epochs $= 10$, the result is as following, Then, from Table 3 we know that **the best regularization strength is 0 with corresponding accuracy is**

Table 1: Learn Rate

Learn rate	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128
Accuracy	0.0991	0.0991	0.0991	0.9729	0.9813	0.9815	0.9815	0.9787

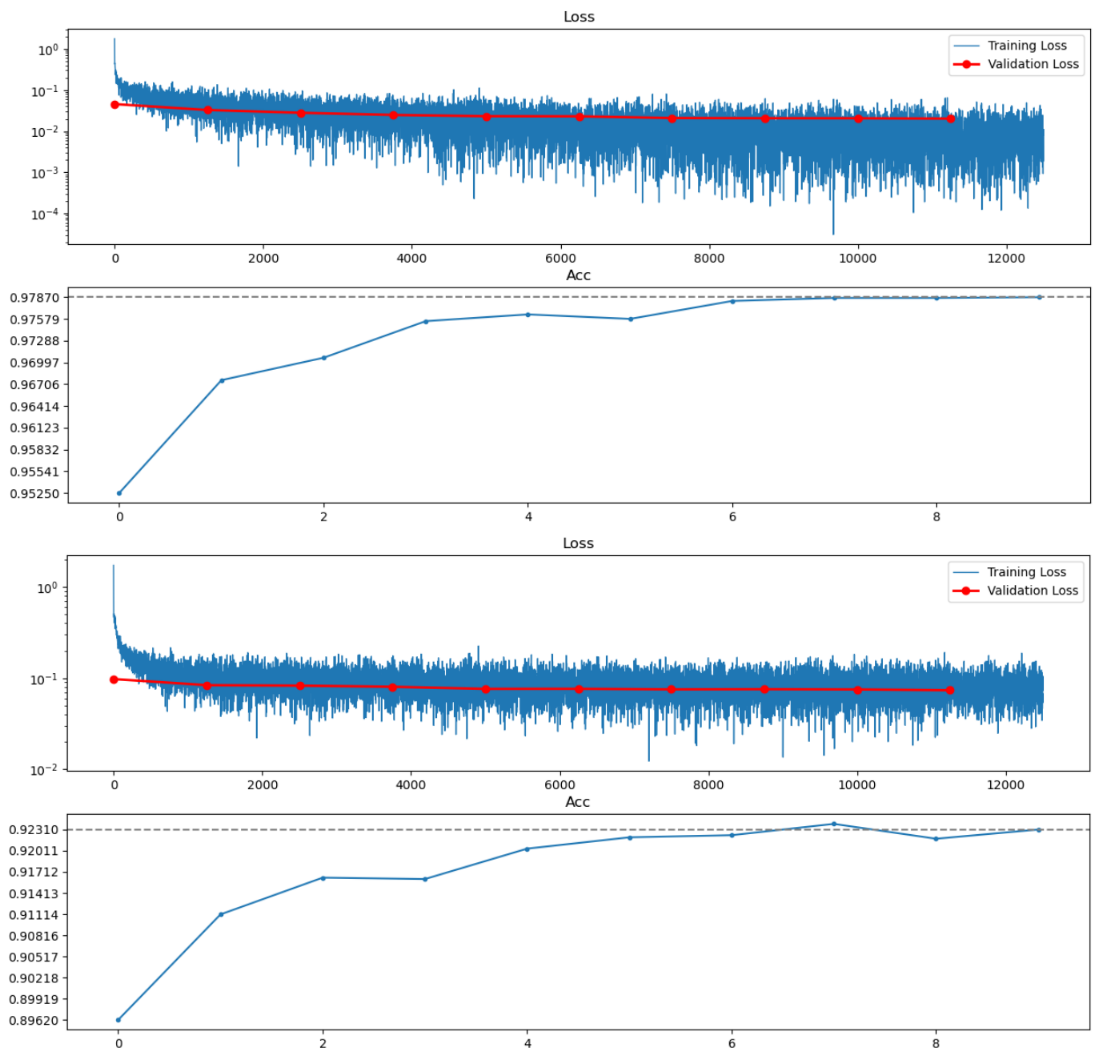

Figure 2: ReLU vs sigmoid

0.9834.

Finally, we conclude that the best parameter of our model is **learn rate = 0.03125, hidden layer size = 900, regularization strength = 0**. Then we will use these parameters to make the visualization.

4 Visualization

In this section, we will use the best parameters: learn rate = 0.03125, hidden layer size = 900, regularization strength = 0, batch size = 40 and epochs = 10. Moreover, we use MSE as the loss function, ReLU as the first activation function.

Firstly, we propose the figure of the training loss, validation loss and accuracy as following (Figure

Table 2: Hidden Layer Size

Hidden Size	50	100	200	300	400	500	600	700	800	900	1000
Accuracy	0.9391	0.9744	0.9793	0.9802	0.9825	0.9829	0.9815	0.9826	0.9821	0.9834	0.9833

Table 3: Regularization Strength

Regularization	1e-1	1e-2	1e-3	1e-4	1e-5	1e-6	1e-7	0
Accuracy	0.0967	0.0989	0.7724	0.9803	0.9832	0.982	0.9829	0.9834

3), Moreover, this model have accuracy 99.996% **on train data,** 98.34% **on validation data,** 98.38%

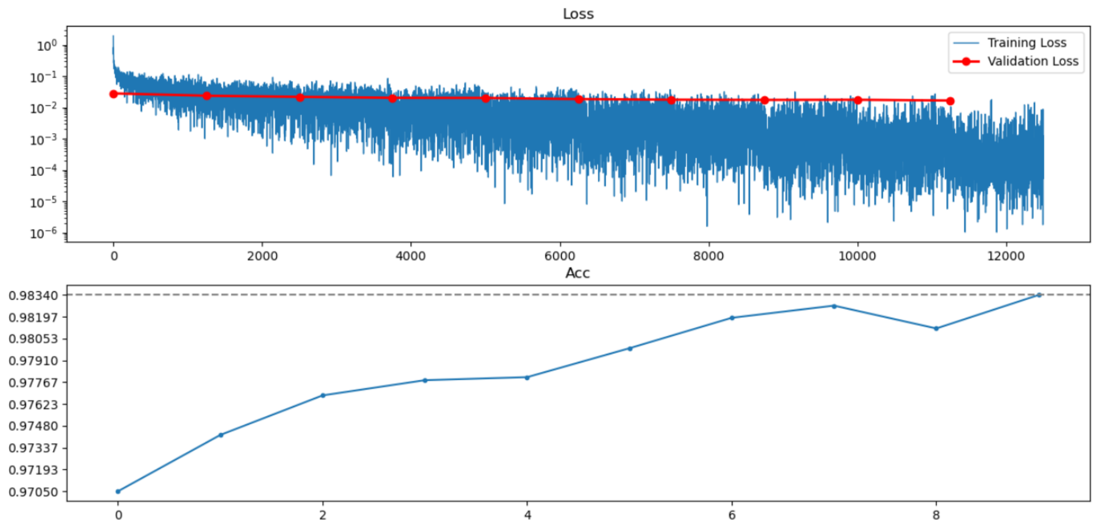

Figure 3: Model of Optimal Parameters

on test data.

Although our model have a high accuracy, we may fail in some datasets, there are still shortages of our model.

5 Conclusion

In this paper, we build a neural network two layers classifier only using Numpy. We use different parameters to seek the optimal parameters: learn rate, hidden layer size and regularization strength. The trained model have an accuracy 98.38% on MNIST datasets. Although we have high accuracy, there are also some shortages can be improved in our model, such as, more intensive parameters search and another adjustment technique.