Implement a Two-Layer Neural Network Classifier Using Numpy

Miaopeng Yu[*][†]
School of Data Science
Fudan University
19307130203@fudan.edu.cn

Abstract

In this project, we constructed and trained a two-layer neural network using Numpy. We compared the model accuracy in the validation set with different activation functions, step sizes, regularization intensities, and hidden sizes. And we finally achieved considerable accuracy in the validation set. After that, we store the parameters and hyperparameters that performed the best accuracy in the validation set, and test the generalization of this model by calculating the accuracy of prediction of the final model in the test set. Finally, we show the results and visualize them.

1 Introduction

1.1 Background

The neural network is not a newborn subject within just these few years. In contrast, it has quite a long history. However, it has become a highly popular area in just this decade, and many talented scientists are devoting themselves to this area.

With the devotion of so many former scientists and engineers, there are already many python packages for us to use, for example, PyTorch, TensorFlow, Caffe, etc. So it is pretty easy to implement a neural network using these packages these days - we do not need to do the dirty works like writing the back propagation ourselves, which makes the lives of the researchers and engineers easier.

But still, to understand the neural network deeper and better, it is a good exercise to implement a shallow neural network without using any other deep learning packages. Thus, in this project, we are going to implement a two-layer neural network using Numpy only, except when we load the dataset and the parameters and store the parameters and hyperparameters.

1.2 Basic Idea

In this project, we will implement and train a two-layer neural network. At the same time, we will use some tricks like l_2 regularization, weight decay, etc., to improve the accuracy of the model. So after implementing the model, we will also find the "best" model of different hyperparameters by finding the model with the highest accuracy in the validation set. After all of that, we will test the "best" model in the test set and give the accuracy of this final model.

As for the details of implementing the model, the "best" model, and any other functions can be founded on my GitHub Repository and Google Drive provided in the footnote.

[*] GitHub Repository: https://github.com/Bbbstin/CV_Assignment1
[†] Google Drive: https://drive.google.com/drive/folders/1ZDj4w2yfVuSh9TEgspvYqWmGX1onxCgs?usp=sharing

1.3 Organization of this Report

The outline of this report is as follows. In section 2, we will introduce the dataset and activation functions we are going to use in this project. In section 3, we will show the model structure and visualize it, show some details about how to implement this model. In section 4, we will show the hyperparameters we choose to test and the result of these different models. We will also visualize some results and parameters of this model. In section 5, we will discuss what can we do to improve the model accuracy further.

2 Dataset and Methods to be Used in this project

In this section, we will introduce the dataset and activation functions to be used in this project. Since l_2 regularization and Stochastic Gradient Descent method are widely used in the deep learning field, although we will use them in the project, we will not introduce these two methods in this report.

As for the calculation of gradient and loss, learning rate decay strategy, since they are case-oriented, we will show them in the next section.

2.1 Dataset: MNIST

The MNIST (Modified National Institute of Standards and Technology) dataset is a large collection of handwritten digits. It contains 60,000 training samples and 10,000 test samples. Half of the handwritten digits are written by the employees of the United States Census Bureau, and half of them are written by high school students. [1][2]

The MNIST dataset is divided into two parts, one is the training set, and another one is the test set. The number of samples in the training set is 60,000, in the test set is 10,000. The size of all the images in the MNIST dataset is 28×28, and the value of each pixel is 0 to 255. [1][2] Since there is no validation set, we will make a little modification to the training set.

2.2 Activation Functions

In this project, we will try two activation functions: sigmoid and ReLU. At the same time, we will calculate the derivation of them, to help the calculation in next section easier.

2.2.1 Sigmoid Function

The formula of sigmoid function:

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

The range of the function is $(0, 1)$. We can find that when the input of the sigmoid function is in $[-1, 1]$, the slope of this function is not small. However, when the input is pretty large, the function will be saturated, and the slope will close to 0, which might cause the gradient vanishing problem during the training process.

And the derivative of sigmoid function is:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)) \tag{2}$$

2.2.2 ReLU Function

The formula of Rectified Linear Unit (ReLU) function:

$$ReLU(x) = max(0, x) \tag{3}$$

The range of the function is $(0, \infty)$. The ReLU function does not have the problem of saturating when the input is larger than 0, the slope of this function is always 1. However, if the input is smaller than 0, the slope will be 0, and the value of the output is also 0, which is quite similar to the real neural in the creatures.

Although it can somewhat avoid the problem of gradient vanishing and exploding, when one neural is deactivated, it can hardly be activated again.

And the derivation of the ReLU function is:

$$\frac{dReLU(x)}{dx} = 1\{x \geq 0\} \tag{4}$$

3 Implementation of the Model

3.1 Model Structure and its Visualization

The formula of the forward pass of this model is shown in (5)-(7).

$$A_0 = X \tag{5}$$

$$A_1 = activation(A_0 W_1) \tag{6}$$

$$output = A_2 = softmax(A_1 W_2) \tag{7}$$

, where W_1 and W_2 are the parameters of the model, the $activation$ is the activation function, we can choose either ReLU or Sigmoid activation function here.

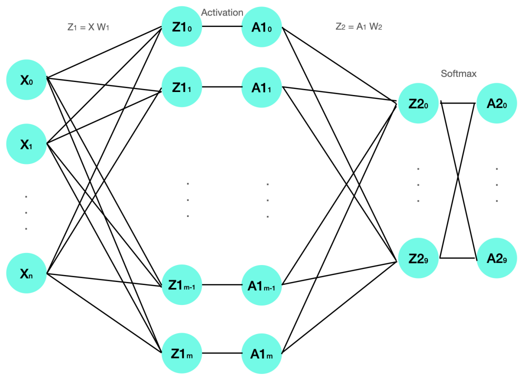


Figure 1: Model Structure

3.2 MNIST Loader

Before training the model, we should first load the MNIST dataset. There is no validation set in the MNIST dataset. However, the validation set is crucial for choosing the model. Since before we decide which model to use, we cannot touch the test set. If we use the test set to choose the final model, it can be regarded as a process of trying to train the model by fitting the test set, which is a kind of cheating.

Therefore, we randomly split 5,000 samples from the training set as the validation set. And normalize it to $N(0, 1)$. After that, we flatten all images from 28×28 to 784, and convert the labels to one-hot version.

3.3 Learning Rate Decay

Learning rate decay is a pretty helpful strategy. It can avoid the oscillation around the global optimal point caused by the large learning rate. And also, it helps us to choose a larger learning rate at the beginning for faster convergence.

The learning rate decay strategy we choose in this model is exponential learning rate decay [3]. The formula is:

$$learning\_rate = initial\_learning\_rate * decay\_rate^{step/decay\_steps} \tag{8}$$

In this model, we set $decay\_rate = 0.9$, and $decay\_step = 10000$.

3.4 Calculation of Loss and Back Propagation

Since we use the softmax function in the last layer, we should the use cross-entropy loss.

$$\mathcal{L} = -\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}\log(p(\hat{y}_{ij})) \tag{9}$$

And also, since we apply an l_2 regularization to this model, the final loss of this model is:

$$\mathcal{L}_w = -\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}\log(p(\hat{y}_{ij})) + \lambda\|W\|_F^2 \tag{10}$$

, where W is all parameters in this model.

After we get the loss of this model, we can use it to calculate the gradient of each parameter using the back propagation.

The equations to calculate forward pass of this model have already shown in section 3.1. So we will show the way to calculate backward pass here.

$$\frac{d\mathcal{L}}{dw_i} = \sum_{k=1}^{K}\frac{d\mathcal{L}_w}{dp_k}\frac{dp_k}{dw_i} = \sum_{k\neq i}^{K}\frac{d\mathcal{L}_w}{dp_k}\frac{dp_k}{dw_i} + \frac{d\mathcal{L}_w}{dp_i}\frac{dp_i}{dw_i} = \sum_{k\neq i}^{K} xy_k p_i - xy_i(1-p_i) = x(p_i - y_i) \tag{11}$$

, where $p_j = softmax(w^T x) = \frac{e^{w_j^T x}}{\sum_{k=1}^{K} e^{w_k^T x}}$, and y is the real label (one-hot).

Therefore,

$$\frac{d\mathcal{L}_w}{dW_2} = A_1(A_2 - y_{onehot}) + 2\lambda\sum_i W_{2i} \tag{12}$$

And

$$\frac{d\mathcal{L}_w}{dA_1} = \frac{d\mathcal{L}_w}{dW_2}\frac{dW_2}{dA_2}\frac{dA_2}{dA_1} = W_2(A_2 - y_{onehot}) \tag{13}$$

Therefore,

$$\frac{d\mathcal{L}_w}{dW_1} = \frac{d\mathcal{L}_w}{dA_1}\frac{dA_1}{dW_1} = A_0 \cdot \frac{d\mathcal{L}_w}{dA_1} \otimes activation\_backprop(A_0 W_1) + 2\lambda\sum_i W_{1i} \tag{14}$$

According to equation (12) and (14) with (2) or (4), we can use the gradient of W_1 and W_2 to optimize the model.

4 Training and Results

4.1 Training

We try to obtain different models using different hyperparameters during the training process. The hyperparameters we choose are shown in the Table 1. Since the models we need to train will go cubically as the hyperparameters increase, we just select a number of hyperparameters from reasonable ranges.

For the step size, since we apply the learning rate decay strategy to train this model, we can use a bigger step size than without using this strategy, so its range will be large. For the regularization intensity, we have tested different lambda before selecting the range, and found that if the lambda is large, the accuracy in the validation set is quite oscillating, therefore, we choose it to be a small number. And we do the same thing to the number of hidden layer units, and we choose the range shown in the Table 1 at last.

The number of models we train in this project is 336.

Table 1: Hyperparameters Tested in Training Process

Hyperparameters	
Step Size	lambda
$10^{-4}, 5 \times 10^{-4}, 10^{-3}, 10^{-2}, 0.5, 0.1$	$10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}$
Hidden Layer Units	Activation Function
$150, 180, 200, 225, 250, 275, 300$	ReLU, Sigmoid

Table 2: Top-5 Models

Hyperparameters				
Step Size	lambda	Hidden Layer Units	Activation Function	Accuracy (Val Set)
0.05	10^{-5}	275	Sigmoid	98.24
0.05	10^{-5}	300	Sigmoid	98.16
0.05	10^{-8}	300	Sigmoid	98.12
0.05	10^{-7}	300	Sigmoid	98.10
0.01	10^{-6}	250	ReLU	98.08

For each model, we use Stochastic Gradient Descent (SGD) to update the parameters for 240,000 iterations, so that we can almost guarantee that the model has achieved its best performance in the training set. After finishing the training process of each model, we evaluate the model using the validation set, and pick up the model that performs the best in the validation set.

4.2 Result

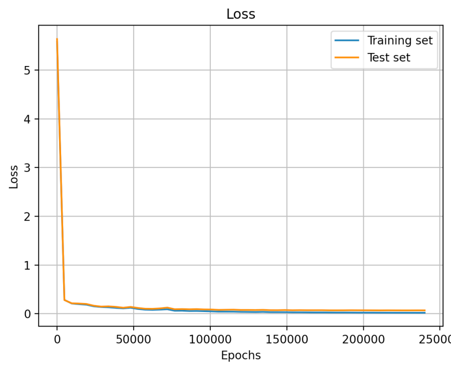
After training for totally 336 models, and compare their accuracy in the validation set, the top-5 models achieve the best accuracy in the validation set is shown in Table 2.

Since we will choose the model with the best accuracy in the validation set. The final model we choose is the top-1 model shown in Table 2.

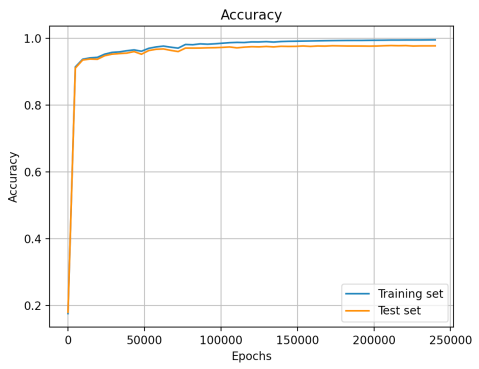
After getting the model, we also need to test its generalization in the test set. So we load the parameters of this model, and test it in the test set. The final accuracy of this model is 97.87%.

4.3 Visualization of the Loss and Accuracy

In this subsection, we will show the loss and accuracy of the model we chose above, the plots are shown below.



(a) Loss of the training set and test set



(b) Accuracy of the training set and test set

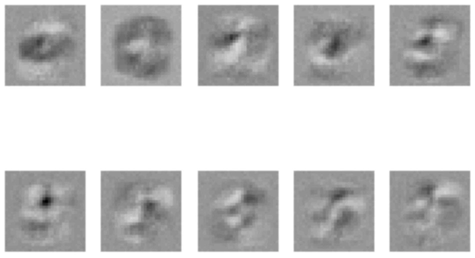
Figure 2: Visualization of loss and accuracy during the training process

We can find that there is a gap between the line of the training set and the validation set in the accuracy plot. So we know that there is still an overfitting problem. But since the gap is small, it is still acceptable.

4.4 Visualization of Parameters

In this subsection, we will show the parameters of this model. If everything goes well, we should be able to visualize the parameters W_1 and W_2 and find some patterns. Since the parameters are tracking the patterns of each image, therefore, the brighter the visualization, the higher the response signal on interaction with the input.

First, we will show the visualization of W_1, since the W_1 of the best model has a shape of $(784, 275)$, we need to apply a Principal Component Analysis (PCA) to convert it to shape $(784, 10)$, which corresponds to ten different labels. We will visualize it according to each column, and reshape each column to shape of $(28, 28)$. The visualization is shown in Figure 3(a).


(a) W_1 visualization

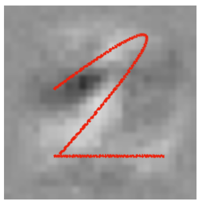

(b) The third image in Figure 3(a)

Figure 3: Visualization of $W1$

In Figure 3, we can only find a little pattern inside each column of W_1, which is shown in Figure 3(b).

Then, we will show the visualization of W_2, since W_2 has a shape of $(275, 10)$, we do not need to apply any PCA to this matrix. Then, we will discard the last three elements of each column, and reshape it to $(16, 17)$. The visualization of W_2 is shown in Figure 4.

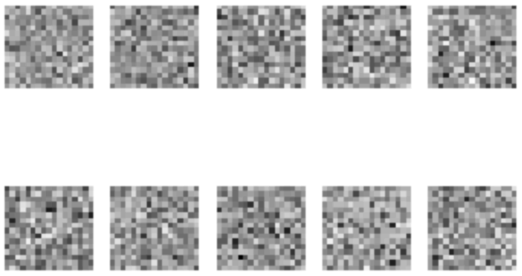


Figure 4: W_2 visualization

We can hardly find any pattern of W_2. Since this is a two-layer neural network, we think we can also check the pattern of $W = W_1 \cdot W_2$. Although in this model, there is an activation function between W_1 and W_2, to make it simpler, we just ignore it here. W has a shape of $(784, 10)$, for each column, it corresponds one specific label. The visualization of $W = W1 \cdot W2$ is shown in Figure 5.

In Figure 5, we can clearly see the number inside each image. And it proves that the model is learning the pattern of the training samples.

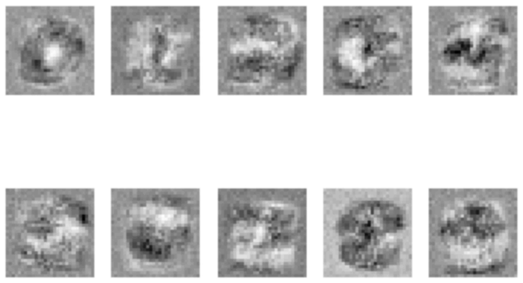


Figure 5: W visualization

5 Further Improvements

Since the range of these hyperparameters we choose is quite large, the number of points we chose in these ranges is small. Therefore, after getting the result shown above, we can use it to truncate the hyperparameters' range, select the points within these ranges, and train again.

And also, since the top-2 models we get both chose lambda to be 10^{-5}, which is the smallest number of the range of lambdas. Therefore, we guess maybe the range we have chosen is too large. We can test it by retraining many models with larger lambda.

At last, adding bias to each layer maybe also a way to improve the accuracy of the model. And we have done this, but time limited, we just tested this method on the model we select (Step size: 0.05, lambda: 10^{-5}, Hidden layer units: 275, Activation function: Sigmoid). And it finally achieves an accuracy of 97.76% in the validation set. It has not shown any huge improvement, maybe because there will be more parameters to train, therefore, it will take longer to train. But it is hard to say whether it will generate an even better model. So, we should try it if time permitting.

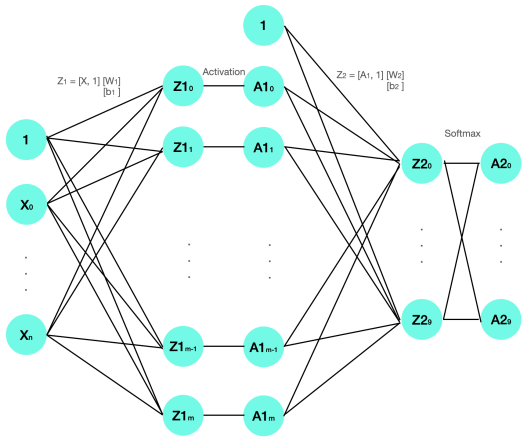


Figure 6: Model Structure with Bias

References

[1] MNIST dataset. http://yann.lecun.com/exdb/mnist/

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[3] Zhiyuan Li, Sanjeev Arora. An Exponential Learning Rate Schedule for Deep Learning. arXiv preprint arXiv:1901.07454, 2019.