Construction of Two-Layer Neural Network Classifier

Gu Yue 21210980032

Abstract

In this paper, we construct a 2-layer fully-connected neural network to identify the handwritten digit images in MNIST dataset. Combining stochastic gradient descent method and the learning rate decay with back propagation, we train the neural network and find that the model performs best on the test set with a 200 dimensional hidden layer, an exponential decayed learning rate which starts at 0.05 and zero l_2 penalty for the weight matrix. This leads to a model with 98.0% accuracy. According to the visulization of the weight matrices, the weight matrix $\boldsymbol{W}^{(1)}$ in the first layer indicates a rough outline of the digits while $\boldsymbol{W}^{(2)}$ provides some corresponding details for different digits respectively.

Code Link: https://github.com/Yue-Gu/NNDL_hw1

Model Link: 链接:https://pan.baidu.com/s/1XaLjJj9LrcDiZ9xw9rF4Eg 提取码:6666 (the parameters $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are saved in the corresponding text files, "W1.txt" for example. The codes to import the model are already implemented in the python file in the above link.)

Contents

1 Introduction

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. An example of a handwritten zero in the training set is demonstrated in Figure1.1.

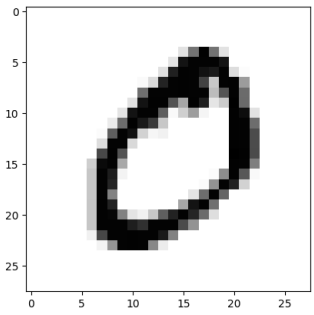

Figure 1.1: A Hand-Written Zero in Training Set

We consider a 2-layer fully-connected neural network to identify the handwritten digit images in MNIST dataset.

2 Network Structure

In this problem, we consider a 2-layer fully-connected neural network. The structure of the neural network is shown in Figure2.1.

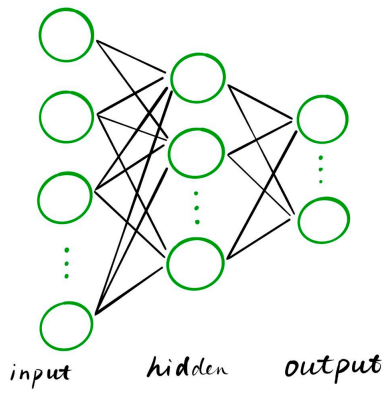

Figure 2.1: the Structure of the Neural Network

Now we explain each part of the neural network respectively.

2.1 Input Layer

The input layer in this case is a vector which represents the picture. The training set includes 60,000 grayscale images that are 28-pixel long and 28-pixel wide, which indicates that each image could be represented by a 784-dimensional vector with each component between 0 and 255. We then normalize the gray values between 0 and 1, which is conducive to convergence of gradient descent algorithm and improvement of training speed.

In conclusion, the input of the neural network is a 784-dimensional vector with each component between 0 and 1.

2.2 Hidden Layer

Given the activation function f, the hidden layer is given by

$$a^{(1)} = f(W^{(1)}a^{(0)} + b^{(1)})$$

where $a^{(0)}$ is the input of the neural network.

The dimension of the hidden layer is a hyper-parameter. Assuming that the dimension is p, $W^{(1)}$ is a weight matrix with 784 columns and p rows and $b^{(1)}$ is a p-dimensional vector.

2.3 Output Layer

The output layer is a 10-dimension vector and the ith (i is s an integer ranging from 0 to 9) component represents the probability that the input image is a handwritten number i.

The output layer is given by

$$a^{(2)} = f(z^2) = f(W^{(2)}a^{(1)} + b^{(2)})$$

where $W^{(2)}$ is a weight matrix with p columns and 10 rows and $b^{(2)}$ is a 10-dimensional vector.

3 Back Propagation

3.1 Loss Function and Gradient Descent

Given the network structure in Section 2, we summarize that the output of the network is

$$\begin{aligned}\hat{\boldsymbol{y}} &= softmax(\boldsymbol{z}^{(2)}) \\ &= softmax(\boldsymbol{W}^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)}) \\ &= softmax(\boldsymbol{W}^{(2)} softmax(\boldsymbol{W}^{(1)}\boldsymbol{a}^{(0)} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)})\end{aligned} \tag{1}$$

and the loss function with L_2 regularization is

$$\begin{aligned}\mathcal{L} &= \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \lambda \sum_{i,j,l} w_{i,j,l} \\ &= \frac{1}{2}(\boldsymbol{y} - \hat{\boldsymbol{y}})^\top (\boldsymbol{y} - \hat{\boldsymbol{y}}) + \lambda \sum_{i,j,l} w_{i,j,l}\end{aligned} \tag{2}$$

Now we consider the back propogation of the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ which needs to be updated in each cycle. To implement the gradient descent of the parameters, it is necessary to use the chain rule. On top of the list, we consider the following two partial derivatives:

$$\delta^{(2)} \triangleq \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial z^{(2)}} = \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \cdot \frac{\partial \hat{\boldsymbol{y}}}{\partial z^{(2)}} = (\boldsymbol{y} - \hat{\boldsymbol{y}}) \odot (softmax^{'}(z^{(2)}))$$

$$\delta^{(1)} \triangleq \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial z^{(1)}} = \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial \boldsymbol{a}^{(1)}} \cdot \frac{\partial \boldsymbol{a}^{(1)}}{\partial z^{(1)}} = softmax^{'}(z^{(2)}) \odot ((\boldsymbol{W}^{(2)})^\top \delta^2)$$

where $\odot$ indicates the dot product of vectors and $softmax^{'}(x) = softmax(x)(1-softmax(x))$. softmax(x)(1-softmax(x))

Thus we have

$$\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{(2)}} = \delta^{(2)} \cdot \frac{\partial z^{(2)}}{\partial \boldsymbol{W}^{(2)}} = \delta^{(2)} (\boldsymbol{a}^{(1)})^{\top}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{(1)}} = \delta^{(1)} \cdot \frac{\partial z^{(1)}}{\partial \boldsymbol{W}^{(1)}} = \delta^{(1)} (\boldsymbol{a}^{(0)})^{\top}$$

and

$$\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(2)}} = \delta^{(2)}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(1)}} = \delta^{(1)}$$

Then we update the parameters with learning rate α for $l = 1,\ 2$:

$$\boldsymbol{W}^{l} -= \alpha (\delta^{(l)} (\boldsymbol{a}^{(l-1)})^{\top} + \lambda \boldsymbol{W}^{l})$$

$$\boldsymbol{b}^{l} -= \alpha \delta^{(l)}$$

3.2 Learning Rate Decay

If the learning rate is too large, learning will be accelerated in the early stage of algorithm optimization, making the model easier to approach the local or global optimal solution. However, in the later period, there will be large fluctuations, and even the value of the loss function will hover around the minimum value, with great fluctuations, and it is always difficult to reach the optimal value.

As a straightfoward result, we consider the exponential decay of the learning rate:

$$\alpha_{decayed} = \alpha \times (decay\ rate)^{epoch}$$

3.3 Stochastic Gradient Descent

While updating the parameters, we consider the approximation with a sample of all the samples to adjust the parameters. This may not lead to an a global optimal direction in every iteration, but the overal direction still goes towards the global optimal solution when it comes to the convex optimization ptoblems. And the final result is usually near the global optimal solution. Compared with batch gradient, SGD converges faster.

Algorithm 1 Back Propagation With Stochastic Gradient Descent

Input: The set of training images and labels $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{n=1}^{N}$, learning rate α, regularization coefficient λ, number of epoch k, number of neurons M_l, $1 \leq l \leq 2$;
Output: $\boldsymbol{W}$, $\boldsymbol{b}$.

```
Randomly initialize W, b;
for epo = 1 to k do
    Reorder the training set samples randomly;
    for n = 1 to N do
        Select sample (x_n, y_n);
        Calculate the input z^(l) and the actication value a^(l) in each layer till the output layer
        with forward propagation;
        Calculate the error δ^(l) with back propagation;
        ∀l,   W^l ← W^l − α(δ^(l) a^(l−1) + λ W^(l));
        ∀l,   b^l ← b^l − α δ^(l);
    end for
end for
```

4 Results and Discussions

4.1 Parameter Tuning

There are three parameters to be tuned in this case: the dimension of hidden layer p, the undecayed learning rate α and the regularization parameter λ.

We consider the grid search of the parameters:

$$p : [100,\ 200,\ 300]$$

$$\alpha : [0.05,\ 0.1.\ 0.2]$$

$$\lambda : [0,\ 0.005,\ 0.01]$$

The combination of $p = 200$, $\alpha = 0.05$ and $\lambda = 0$ leads to a neural network model that performs best at the test set.

Now we consider the training process and the results of the best model. The accuracy on the testing set is 98.0%.

The loss curve on the training set and testing set is shown in Figure4.1. The accuracy in each epoch is in Figure4.2.

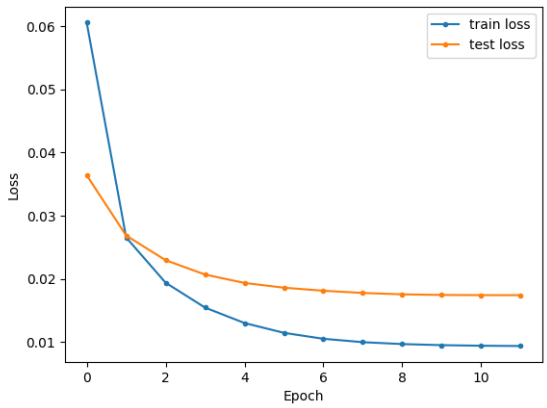

Figure 4.1: Loss on Training Set and Testing Set

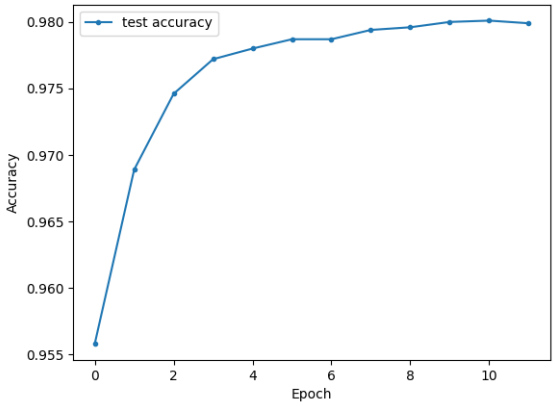

Figure 4.2: Accuracy on Testing Set

4.2 Parameters Visualization

There are two weight matrix in the two-layer neural network. $\boldsymbol{W}^{(1)}$ is a matrix with 789 columns and 200 rows. We conduct PCA on $\boldsymbol{W}^{(1)}$ and get the first principle component which explains % variance of $\boldsymbol{W}^{(1)}$, then we rescale the values between 0 and 255. Thus we get a grey picture with 28-pixel width and 28-pixel length. We can tell from Figure4.3 that $\boldsymbol{W}^{(1)}$ mainly represents the rough outline of the pictures.

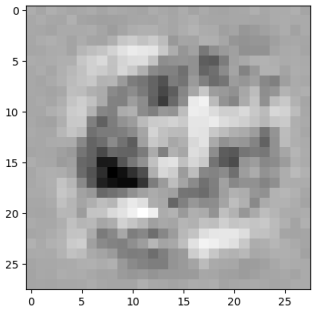

Figure 4.3: Visualization of the First Principal Component of $\boldsymbol{W}^{(1)}$

We also consider the weight matrix $\boldsymbol{W}^{(2)}$ with 200 columns and 10 rows. Because a 200-dimension vector is not convenient to be transformed into a explainable grey picture, we multiply $\boldsymbol{W}^{(1)}$ by $\boldsymbol{W}^{(2)}$ and thus get a matrix with 784 rows and 10 columns which could be transformed into ten grey pictures that represent number 0 to 9 respectively.

We can conclude from Figure4.4 that the weight matrix of the second layer contains more details for each class. For example, the visualization of weight matrix for class 0 and class 3 roughly shows the accurate outline of the corresponding written number.

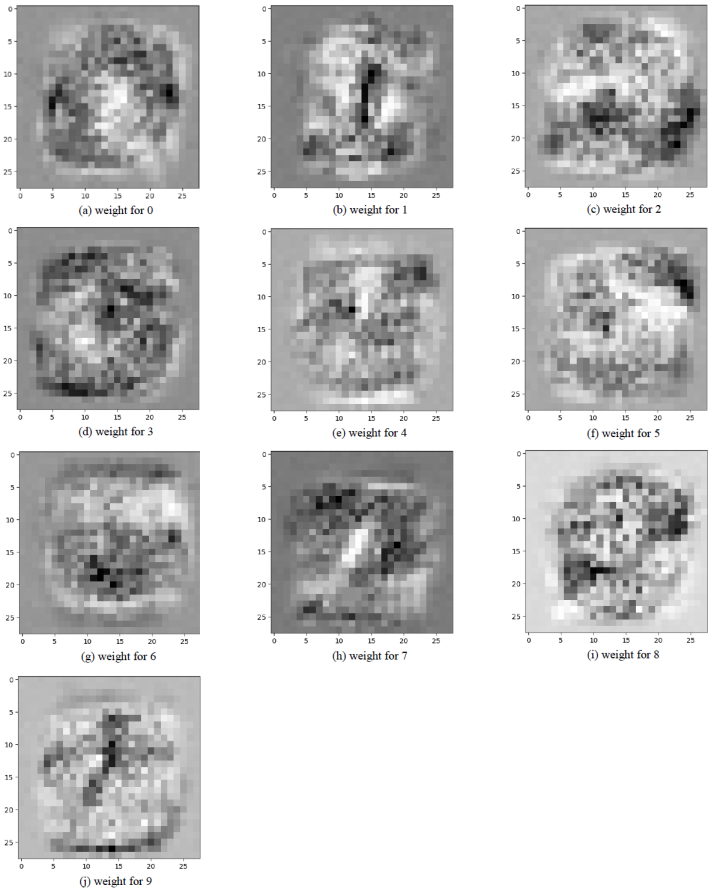

Figure 4.4: Visualization of $\boldsymbol{W}^{(1)\top}\boldsymbol{W}^{(2)\top}$