The Chinese University of Hong Kong, Shenzhen

CSC 4020

Machine Learning

Assignment2 Report

Author:
xxxxxx

Student Number:
ID xxxxxxxxx

April 5, 2021

Contents

1 Written Questions

1.1 Question 1

a.

$$\phi(x_1) = [1, \tag{1}$$

$$\phi(x_2) = [1, \tag{2}$$

Suppose ...

Let

$$\vec{u} = \cdots \tag{3}$$

b.

The distance between x_1 and x_2 is:

$$d = \tag{4}$$

c.

Let $w = [0, 2c, 2c]^T$, then we have:

$$||w|| \tag{5}$$

Solve the equation, we can have $c = \frac{1}{}$

d. Plug $\phi(x_1)$, $\phi(x_2)$ and w into the constraint of the optimization question, we can have the below two equations:

(6)

(7)

By solving the above two equations, we can have...

e.

$$f(x) = \tag{8}$$

1.2 Question 2

1.3 Question 3

Let ....

From the graph ...

$$\frac{\partial L}{\partial b_1} = \tag{9}$$

$$\frac{\partial L}{\partial b_2} = \tag{10}$$

$$\tag{11}$$

$$\tag{12}$$

$$\frac{\partial L}{\partial w_1} = \tag{13}$$

$$\frac{\partial L}{\partial w_2} = — \tag{14}$$

$$\frac{\partial L}{\partial w_3} = \tag{15}$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial J} \tag{16}$$

1.4 Question 4

Note: In this question, I denote one computation unit as

1. ..
2. ..
3. ..

Take a simple example,

Conv1

Shape: ..

$$\tag{17}$$

Number of parameters: ..

(18)

Computation cost:

For each filter, it requires..

(19)

It requires ..

(20)

Hence, the total ..

(21)

Maxpool1

Shape: ..

(22)

Number of parameters:

(23)

Computation cost:

(24)

Conv2

Shape: ..

(25)

Number of parameters: ..

(26)

Computation cost:

For each filter, it requires multiply operations:

(27)

It requires sum operations:

(28)

Hence, the total computation cost is:

(29)

Maxpool2

Shape: ..

(30)

Number of parameters: ..

(31)

Computation cost:

(32)

2 Programming Question

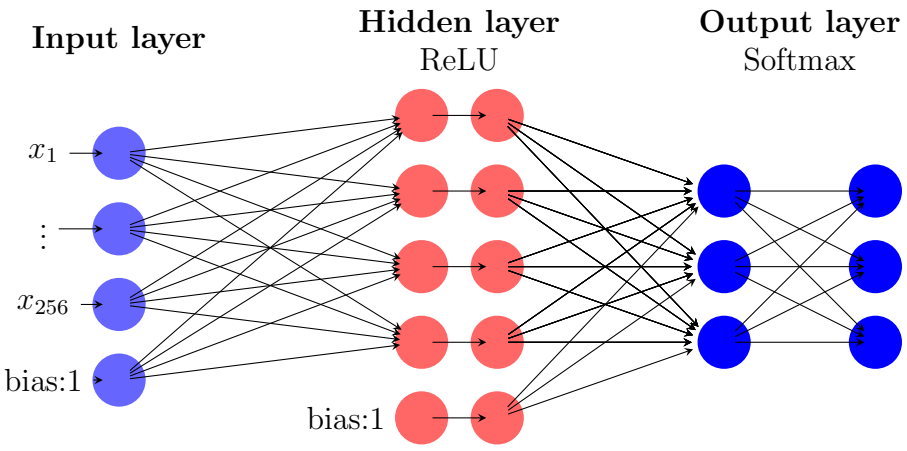
2.1 Network Structure

In this assignment, we implement a simple two layer network. The structure of the neural network is shown as the below figure.

Input layer we have 257 nodes, as for each input x we have 256 features and 1 node for bias parameter.

For hidden layer, we have totally **h** nodes and 1 node for bias parameter. The value of **h** can be set when initializing the neural network. In the below figure we just take 4 as an example. The activation function for this layer is ReLU.

For the output layer, we have three nodes as our classification task is group the hand written characters into three classes. The activation function for this layer is softmax.



2.1.1 Hyperparameter Settings

In this simple neural network, we have three hyperparameters. They are **batch size**, **learning rate** and **hidden layer node number**.

Batch size is set to 9 as default. Batch size is related to the update process of SGD, the effect of different batch size will be discussed in the latter part of the report.

Learning rate is set to 0.01 as default. Learning rate is the step size we set for updating the weights in the neural network.

Hidden layer node number is set to 20 as default. Too much or too few hidden layer nodes may affect the performance of the model. The effect of it will be discussed in the latter part of the report.

2.1.2 Data loading

Use **numpy.loadtxt()** function to read in train data and test data. Notice that the loading of the data is using relative path instead of absolute path. Hence, when executing the code of the model, one should place the train data txt file and test data test file under the same directory of the model python file.

After read in, split the dataset into two parts. The first part is the first **256** columns, which acts as the input feature values. Denote this part as input data x. The second part is the last **3** columns, which serve as the classification label. Denote this part as label y.

Finally, append one column to the whose entries all equals to one to x, which acts as the bias parameter.

2.1.3 Training

Forward Pass

As this neural network is relatively simple, we can initialize this network by random small numbers. Denote W_i as the weights for input layer to hidden layer. Denote W_o as the weights for hidden layer to the output layer. Besides, we use cross entropy to calculate loss function in this model. The overall equation for loss function is shown as below equation:

$$L(W_i, W_o) = CE(y, softmax(W_o ReLU(W_i x + b_1) + b_2)) \tag{33}$$

The equation for ReLU, softmax and CE(cross entropy) is shown as below:

$$ReLU = \begin{cases} 0, & if \quad x < 0 \\ x, & if \quad x \geq 0 \end{cases} \tag{34}$$

$$Softmax_i = \frac{e_i}{\sum_j e_j} \tag{35}$$

$$CE = -\sum t_i ln y_i \tag{36}$$

Backward

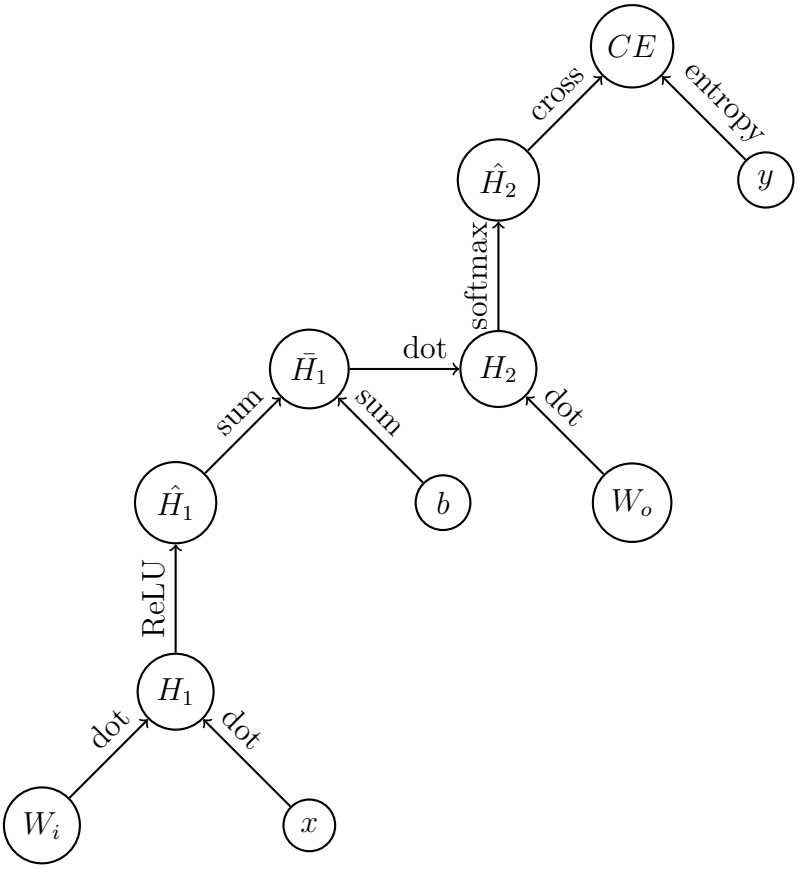
To better illustrate the backward process, we first draw the computational graph of the forward pass process.

From the graph, we can denote that the gradient of W_o and W_i can be calculated via the below two equations based on chain rule:

$$\frac{\partial CE}{\partial W_o} = \frac{\partial CE}{\partial \hat{H}_2} \cdot \frac{\partial \hat{H}_2}{\partial H_2} \cdot \frac{\partial H_2}{\partial W_o} \tag{37}$$

$$\frac{\partial CE}{\partial W_i} = \frac{\partial CE}{\partial \hat{H}_2} \cdot \frac{\partial \hat{H}_2}{\partial H_2} \cdot \frac{\partial H_2}{\partial \hat{H}_1} \cdot \frac{\partial \hat{H}_1}{\partial H_1} \cdot \frac{\partial H_1}{\partial W_i} \tag{38}$$

$$\frac{\partial CE}{\partial b} = \frac{\partial CE}{\partial \hat{H}_2} \cdot \frac{\partial \hat{H}_2}{\partial H_2} \cdot \frac{\partial H_2}{\partial \bar{H}_1} \cdot \frac{\partial \bar{H}_1}{\partial b} \tag{39}$$



The x used in the above figure already append one column with value 1, which acts as the bias node. Therefore, no need for considering another bias parameter b.

SGD

Using the back propagation method illustrated as above, we can calculate the gradient of wanted parameters with respect to the loss function we set.

Instead of using gradient descent, here we use SGD (stochastic gradient descent). We use the gradient of a mini batch to approximate the gradient of the original data. The update procedure can be proceeded via below equation:

$$\theta_{i+1} = \theta_i - \frac{lr}{b} \times \sum_{i=1}^{b} \frac{\partial CE(\theta_i, x_i)}{\partial \theta_i} \tag{40}$$

lr: learning rate

b: batch size

2.1.4 Testing

After each iteration/epoch, one can use current state neural network to evaluate the performance of the model. In testing stage, one just simply plug in testing data and go through forward pass process, during which accuracy and loss are calculated. Notice that in testing stage, there is no need for back propagation, i.e. modification of model's parameters.

2.2 Analysis

2.2.1 batch size

time

In the following table we evaluate the time for different batch size to reach similar test accuracy. To be more specific, the accuracy we set for comparison is **97%**

Table 1: Time Comparison

Batch size	Epoch	Time	Time per Epoch
1	7	0.546	0.078
10	35	2.041	0.058
100	472	20.722	0.043
369	994	55.084	0.055

Note. $lr = 0.01$ *and hidden_node_number* $= 20$ *for all the above test cases.*

From the above data, we can notice that changing batch size does not change time per epoch much. As the computation complexity should be the same for different batch size, only differs in coefficient. However, decreasing batch size can greatly decrease the time to reach a similar performance. Because in each epoch, smaller batch size provides more chances updating parameter. Sometimes, too small batch size might not be suitable as it failed to sample data characteristic well. In our cases, due to the simplicity of training data and classification task, batch size equals to 1 still works well.

Table 2: Highest Test Accuracy

Batch size	Epoch	Accuracy (%)
1	16	99.04
10	57	99.04
100	472	97.14
369	994	97.14

Besides, by comparing the highest accuracy the model can reach among different batch size, one can find out that it is easier for bigger batch size to fall into local minimum of loss function.

loss

From the below four figures, we can see that for same number of iterations, as the batch size gets larger, the curve becomes smoother. This is because larger batch size can better sample the characteristic of input data.

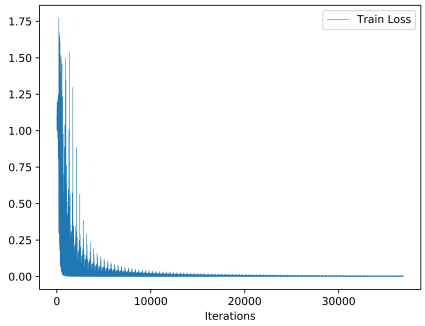


Figure 1: Train loss (batch size = 1)

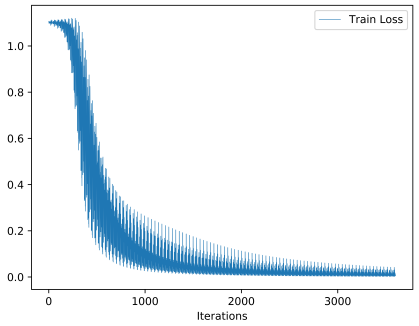


Figure 2: Train loss (batch size = 10)

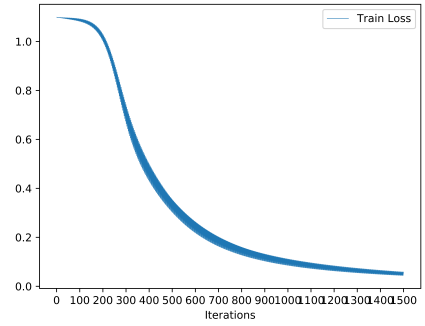


Figure 3: Train loss (batch size = 100)

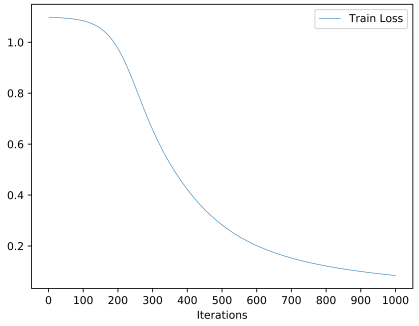


Figure 4: Train loss (batch size = 369)

2.2.2 Hidden Layer Node Number

In the below table, we compare the highest test accuracy for different hidden layer node number within 100 epochs, together with learning rate set to 0.01 and batch size set to 10.

Table 3: Hidden Layer Node Number

Hidden Layer Node Number	1	10	20	200
Test Accuracy (%)	94.29	98.09	99.04	99.04
Train Accuracy (%)	97.56	97.56	97.56	97.56

The train accuracy for the above four cases is shown as below figures.

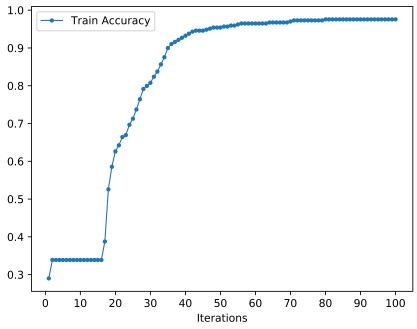


Figure 5: Train Accuracy (h = 1)

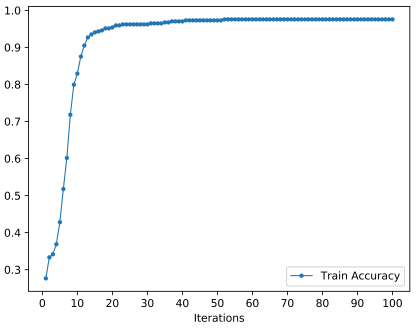


Figure 6: Train Accuracy (h = 10)

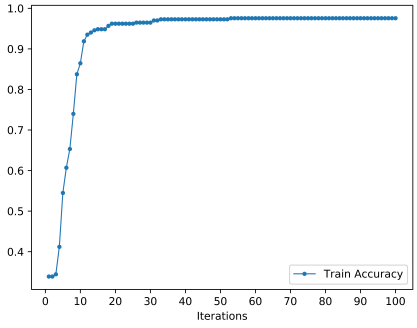


Figure 7: Train Accuracy (h = 20)

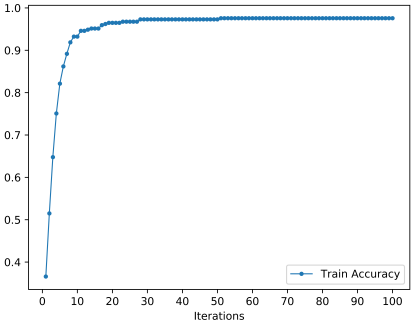


Figure 8: Train Accuracy (h = 200)

The comparison between train accuracy and test accuracy is shown as below figures. Since the model is rather simple, even though some large value (as large as 20000) for hidden layer node number has been tried, the overfitting problem does not appear.

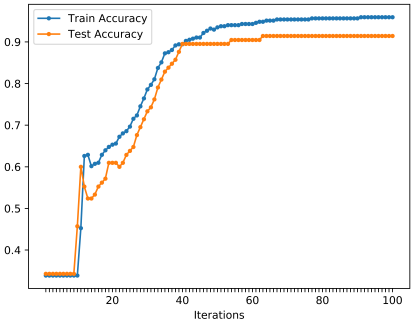


Figure 9: Train and Test Accuracy (h = 1)

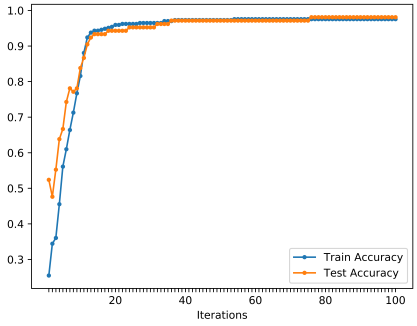


Figure 10: Train and Test Accuracy (h = 10)

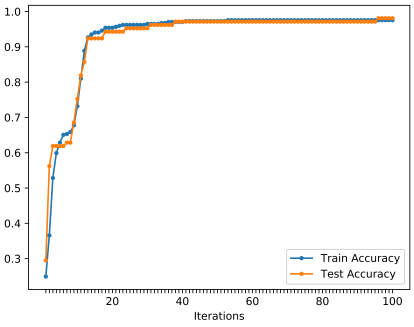


Figure 11: Train and Test Accuracy (h = 20)

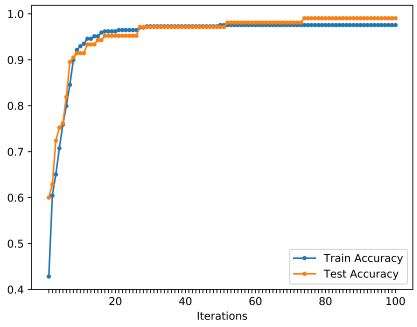


Figure 12: Train and Test Accuracy (h = 200)

2.2.3 Final Model Class Accuracy

The below table and figure shows the class accuracy for training model with $batchsize = 10$, $learningrate = 0.01$, $hiddenlayer = 200$ and $epoch = 100$.

Table 4: Hidden Layer Node Number

Class	1	2	3
Test Accuracy (%)	100	100	92.91
Train Accuracy (%)	97.22	100	100

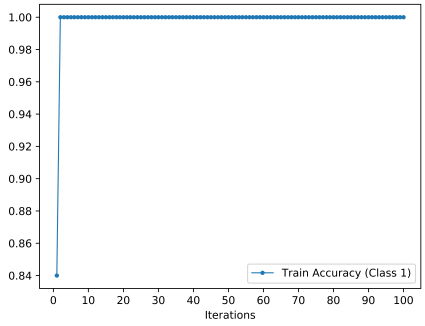


Figure 13: Train Accuracy (Class = 1)

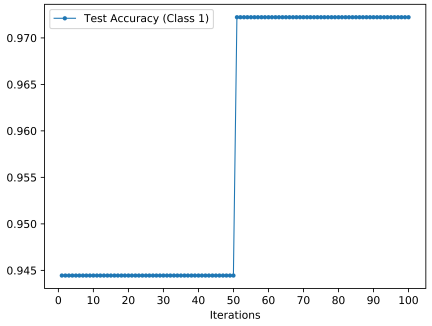


Figure 14: Test Accuracy (Class = 1)

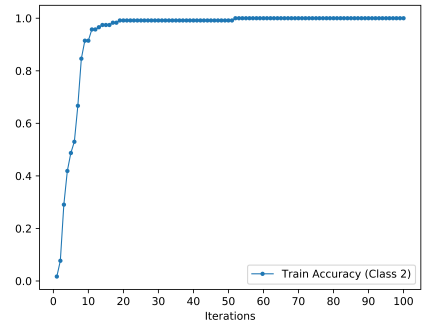


Figure 15: Train Accuracy (Class = 2)

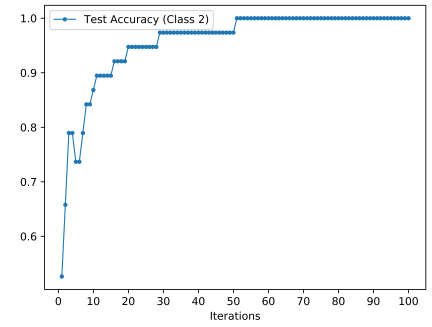


Figure 16: Test Accuracy (Class = 2)

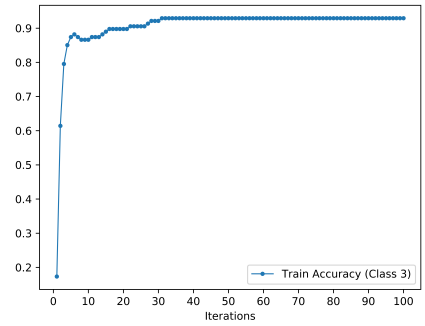


Figure 17: Train Accuracy (Class = 3)

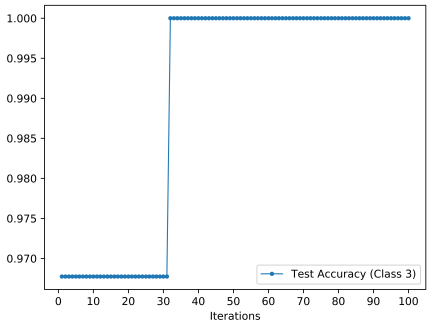


Figure 18: Test Accuracy (Class = 3)