Computer Exercise 2

王永浩 自 62 2016011420

Part I. MLP classifier design

The MLP classifier I wrote in Python consists of 3 parts: Loss function, Activation function and linear layer. Every part has its forward and backward methods. Forward method is used during forward computation and backward method is used during backward computation, which is aimed to update the weight of linear layer.

1. Loss function:

   I designed two kinds of loss function here: cross entropy and square loss. For cross entropy, its forward method shows in the following formula:

   $$Loss_{cross\ entropy} = -\frac{1}{n}\sum_{i=1}^{n}(\widehat{y_i}\log(y_i) + (1-\widehat{y_i})\log(1-y_i))$$

   n is number of data, $\widehat{y_i}$ is label and y_i is the actual output.

   During backward computation, we use its backward method showed below:

   $$\frac{\partial Loss_{cross\ entropy}}{\partial y_i} = -\frac{1}{n}\left(\frac{\widehat{y_i}}{y_i} + \frac{1-\widehat{y_i}}{y_i - 1}\right)$$

   For square loss, its forward method shows below:

   $$Loss_{square\ loss} = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$$

   Its backward method shows below:

   $$\frac{\partial Loss}{\partial y_i} = \frac{1}{n}(y_i - \widehat{y_i})$$

2. Activation function

   There are two kinds of activation functions here: relu and sigmoid. For relu we have its forward method below:

   $$\mathrm{g(x) = x > 0 ? x : 0}$$

   Its backward method is:

$$\frac{\partial \mathrm{g(x)}}{\partial \mathrm{x}} = x > 0 ? 1 : 0$$

For sigmoid its forward method is:

$$\mathrm{g}(x) = \frac{1}{1 + \exp(-\mathrm{x})}$$

Then its backward method is:

$$\frac{\partial \mathrm{g(x)}}{\partial \mathrm{x}} = \mathrm{g(x)}\big(1 - \mathrm{g(x)}\big)$$

3. Linear layer

   For linear layer it has weight, we need to update its weight through backward computation. Its forward method shows below:

$$\mathrm{f(x)} = \mathrm{W}x$$

   Its backward method needs a loss to update weight, which was obtained by back propagation. The pseudo code of its backward method shows below:

```
def backward(loss):
    residual = loss*activation_function.backward()
    grad_w = self.w * residual
    self.w -= self.learning_rate * grad_w
    residual_x = np.dot(residual,self.w.T)
    return residual_x
```

Combine these three parts, we can get an MLP classifier easily. For this experiment, the structure of MLP I designed shows below:

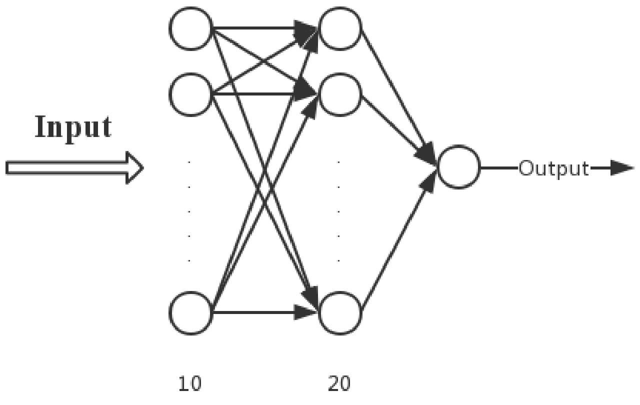

Figure 1 the structure of MLP

Each layer consists of a linear layer and a sigmoid activation function, the loss function of the MLP I used is cross entropy.

Part II. Experiment

1. Use TrainingSet-1 to train the MLP, set the learning rate = 0.01, batch size = 64, the learning curve is as follows (after 1000 iterations of training):

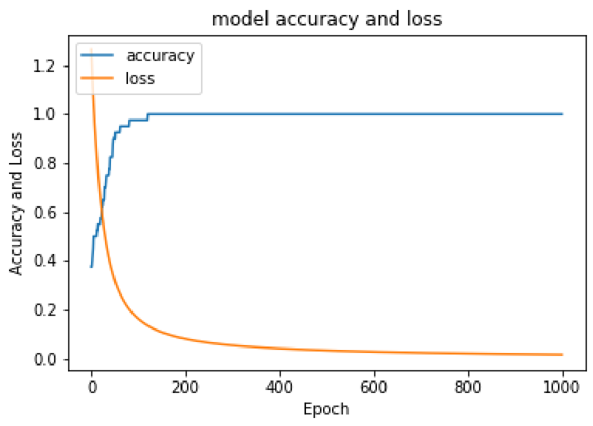

Figure 2 Learning curve on TrainingSet-1

The training error is 0 after 1000 iterations of training.

Apply the trained MLP on TestSet and TestSet-2, the results are shown in the table below:

	Test Accuracy
TestSet	0.93
TestSet-2	0.9125

Table 1 Results on each test set

Using 10 fold cross validation on the training set, so sample size of each training is 36 and that of each validation is 4. The cross-validation error is 0.05. The details show below:

```
accuracy 1.0 loss: 0.1397443321959342
accuracy 0.75 loss: 0.9503006965577853
accuracy 1.0 loss: 0.006251469371555l945
accuracy 1.0 loss: 0.019021727962923594
accuracy 1.0 loss: 0.007979836652368191
accuracy 1.0 loss: 0.15122446568291753
accuracy 1.0 loss: 0.11908017548618552
accuracy 1.0 loss: 0.019117408533353905
accuracy 1.0 loss: 0.04101039041399212
accuracy 1.0 loss: 0.22995587548703525
mean acc 0.975
```

	Errors
Training error	0
Cross validation error	0.025
TestSet	0.07
TestSet-2	0.0875

From above we can see that even training error is 0, the test errors on the two test sets exist. Maybe over-fitting appeared in the course of training.

2. Use TrainingSet-2 to train the MLP, set the learning rate = 0.1, batch size = 64, the learning curve is as follows (after 1000 iterations of training):

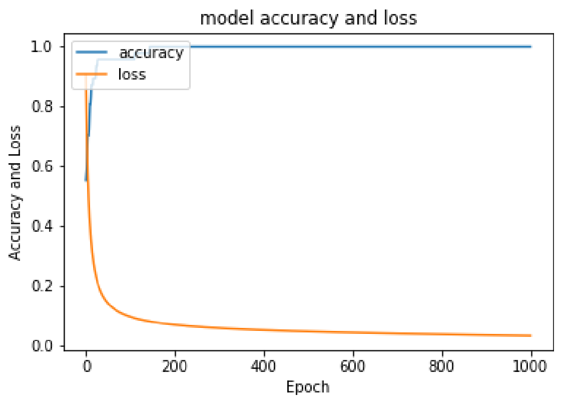

Figure 3 Learning curve on TrainingSet-2

The results are shown in the table below:

	Errors
Training error	0
Cross validation error	0.053
TestSet	0.05

Because TrainingSet-2 has more samples than TrainingSet-1, that means it has more information than the latter. So it will perform better on TestSet.