Assignment 2

Deqing Qu
qud@oregonstate.edu
933-269-707

1. Write a function that evaluates the trained network (5 points), as well as computes all the subgradients of W1 and W2 using backpropagation (5 points).

   Solution:

   (1). In the MLP class, the function called 'evaluate' evaluates the trained network.

```
        self.l2.backward(self.l3.back_x)
        def evaluate(self)  .l2.back,learning_rate, momentum,l2_penalty=0.0)
    def evaluate(self):
        predic = deepcopy(self.l4.layer_output)
        predic[predic >= 0.5] = 1
        predic[predic < 0.5]  = 0
        return np.sum(np.absolute(predic-self.l4.label))
```

   From the trained network, I will get a prediction for each input (the output of the whole network) which value will between 0 and 1. If the value of z2 (output of the whole network), we will regard the prediction on this input as 1, if it is not, we will regard this prediction as 0. Then we will compare the prediction with the true label for this input for evaluation this trained network.

   (2). I wrote three backward functions in the different layers (linear, ReLu and sigmoid cross entropy) separately.
   For the W1 and W2, it is implemented in the LinearTransform class:

```
):
# DEFINE backward function
    self.back_w = np.zeros((self.d, self.m))
    for j in xrange(self.batch):
        self.back_w += np.dot(self.layer_input[:,j].reshape(self.d,1),grad_output[:,j].reshape(1,self.m))
    self.back_w = self.back_w / self.batch

    self.back_b = np.mean(grad_output,axis = 1).reshape(-1,1)
    self.back_x = np.dot(self.W, grad_output)
    self.dw = momentum * self.dw - learning_rate * self.back_w
    self.db = momentum * self.db - learning_rate * self.back_b
    self.W += self.dw
    self.b += self.db
```

2. Write a function that performs stochastic mini-batch gradient descent training (5 points). You may use the deterministic approach of permuting the sequence of the data. Use the momentum approach described in the course slides.

Solution:

The stochastic mini-batch gradient descent training is implemented through the main function just as following figure. We have a batch size to choose the number of samples in one batch group. That means the dimension of inputs will add one (X is [m, batch]).

```
for b in xrange(int(num_examples/size_batch)):
    # INSERT YOUR CODE FOR EACH MINI_BATCH HERE
    # MAKE SURE TO UPDATE total_loss
    total_loss = 0.0
    batch_x = train_x[start_line:(start_line+size_batch),:]
    batch_y = train_y[start_line:(start_line+size_batch)]
    start_line += size_batch
    n.forward(batch_x.T,batch_y,size_batch)
    n.backward(lr,mu)
    total_loss = np.sum(n.l4.loss)
    train_loss += total_loss
```

```
):
# DEFINE backward function
    self.back_w = np.zeros((self.d, self.m))
    for j in xrange(self.batch):
        self.back_w += np.dot(self.layer_input[:,j].reshape(self.d,1),grad_output[:,j].reshape(1,self.m))
    self.back_w = self.back_w / self.batch

    self.back_b = np.mean(grad_output,axis = 1).reshape(-1,1)
    self.back_x = np.dot(self.W, grad_output)
    self.dw = momentum * self.dw - learning_rate * self.back_w
    self.db = momentum * self.db - learning_rate * self.back_b
    self.W += self.dw
    self.b += self.db
```

In my case, I found the 0.8 of momentum would get the best performance, so I chose 0.8 for my experiments.

3. Train the network on the attached 2-class dataset extracted from CIFAR-10: (data can be found in the cifar-2class-py2.zip file on Canvas.). The data has 10,000 training examples in 3072 dimensions and 2,000 testing examples. For this assignment, just treat each dimension as uncorrelated to each other. Train on all the training examples, tune your parameters (number of hidden units, learning rate, mini-batch size, momentum) until you reach a good performance on the testing set. What accuracy can you achieve? (20 points based on the report).

Solution:

Parameters: **Different hidden units: [10, 100, 200]**

Different learning rate: [0.05, 0.01, 0.005]

Different mini-batch size: [50, 200, 1000]
Momentum:0.8

In my own experiment, I used the following parameters (number of hidden units:100, learning rate:0.0500, mini-batch size:50, momentum:0.8) to achieve the best performance after 40 epochs. For that epoch, the loss on the training set is 0.046, the accuracy on the training set is 98.53%, the error rate on the training set is 1.47%, the loss on the test set is 0.748, the accuracy on the test set is 82.15%, the error rate on the test set is 17.85%

The result just as the following figure showing.

```
[Epoch 37, mb 200]    Avg.Loss = 0.042
    Train Loss: 0.070    Train Acc: 97.58%     Error Rate: 2.42%
    Test Loss:  0.699    Test Acc:  82.05%     Error Rate: 17.95%
[Epoch 38, mb 200]    Avg.Loss = 0.045
    Train Loss: 0.064    Train Acc: 97.88%     Error Rate: 2.12%
    Test Loss:  0.742    Test Acc:  82.15%     Error Rate: 17.85%
[Epoch 39, mb 200]    Avg.Loss = 0.019
    Train Loss: 0.063    Train Acc: 97.97%     Error Rate: 2.03%
    Test Loss:  0.744    Test Acc:  82.10%     Error Rate: 17.90%
[Epoch 40, mb 200]    Avg.Loss = 0.006
    Train Loss: 0.046    Train Acc: 98.53%     Error Rate: 1.47%
    Test Loss:  0.748    Test Acc:  82.15%     Error Rate: 17.85%
hidden units: 100, size batch: 50, learning rate: 0.0500, momentum:
0.8
Running time is 1222.8330 s.
```

4. Training Monitoring: For each epoch in training, your function should evaluate the training objective, testing objective, training misclassification error rate (error is 1 for each example if misclassifies, 0 if correct), testing misclassification error rate (5 points).

Solution:

For each epoch in the training process, my function evaluates the training objective (training loss function), testing objective (testing loss function), training misclassification (Error Rate) and testing misclassification (Error Rate).

The following figure is a sample for the results:

```
[Epoch 25, mb 200]    Avg.Loss = 0.297
    Train Loss: 0.372    Train Acc: 83.51%     Error Rate: 16.49%
    Test Loss:  0.460    Test Acc:  78.25%     Error Rate: 21.75%
[Epoch 26, mb 200]    Avg.Loss = 0.292
    Train Loss: 0.368    Train Acc: 83.79%     Error Rate: 16.21%
    Test Loss:  0.460    Test Acc:  78.20%     Error Rate: 21.80%
[Epoch 27, mb 200]    Avg.Loss = 0.287
    Train Loss: 0.364    Train Acc: 83.95%     Error Rate: 16.05%
    Test Loss:  0.460    Test Acc:  78.35%     Error Rate: 21.65%
[Epoch 28, mb 200]    Avg.Loss = 0.282
    Train Loss: 0.360    Train Acc: 84.19%     Error Rate: 15.81%
    Test Loss:  0.460    Test Acc:  78.65%     Error Rate: 21.35%
[Epoch 29, mb 200]    Avg.Loss = 0.277
    Train Loss: 0.357    Train Acc: 84.48%     Error Rate: 15.52%
    Test Loss:  0.460    Test Acc:  78.80%     Error Rate: 21.20%
[Epoch 30, mb 200]    Avg.Loss = 0.272
    Train Loss: 0.353    Train Acc: 84.65%     Error Rate: 15.35%
    Test Loss:  0.460    Test Acc:  78.95%     Error Rate: 21.05%
[Epoch 31, mb 200]    Avg.Loss = 0.267
    Train Loss: 0.349    Train Acc: 84.82%     Error Rate: 15.18%
    Test Loss:  0.461    Test Acc:  79.00%     Error Rate: 21.00%
[Epoch 32, mb 200]    Avg.Loss = 0.262
    Train Loss: 0.345    Train Acc: 84.99%     Error Rate: 15.01%
    Test Loss:  0.461    Test Acc:  79.05%     Error Rate: 20.95%
[Epoch 33, mb 200]    Avg.Loss = 0.258
    Train Loss: 0.342    Train Acc: 85.08%     Error Rate: 14.92%
    Test Loss:  0.462    Test Acc:  79.20%     Error Rate: 20.80%
[Epoch 34, mb 200]    Avg.Loss = 0.253
    Train Loss: 0.338    Train Acc: 85.26%     Error Rate: 14.74%
    Test Loss:  0.462    Test Acc:  79.20%     Error Rate: 20.80%
```

I also upload a word document for the whole running data result in the zip file.

5. Tuning Parameters: please create three figures with following requirements. Save them into jpg format:
   i) test accuracy with different number of batch size
   ii) test accuracy with different learning rate
   iii) test accuracy with different number of hidden unit

   Solution:

   I print three figures for the different parameters for training the test set as following. And I also save them as the JPG format in the zip file.

   (lr means learning rate)

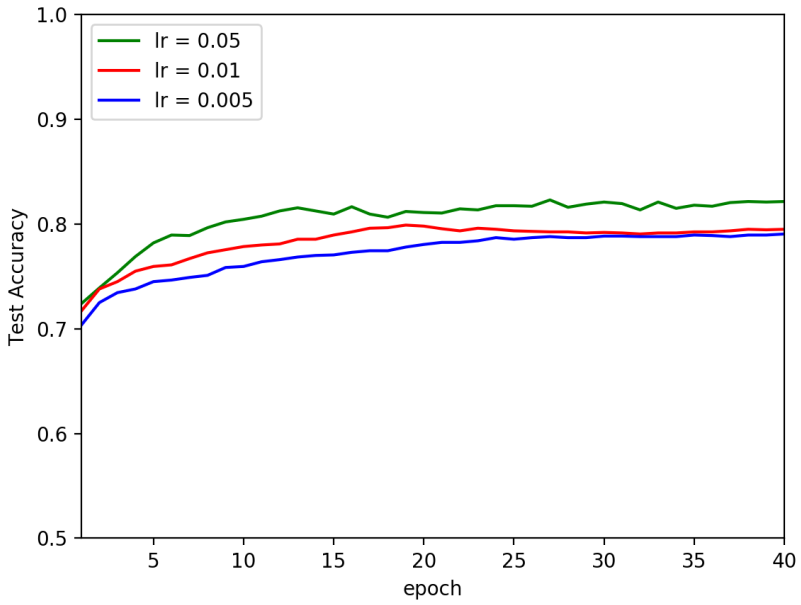

Figure 1. Compare the test accuracy in different learning rate

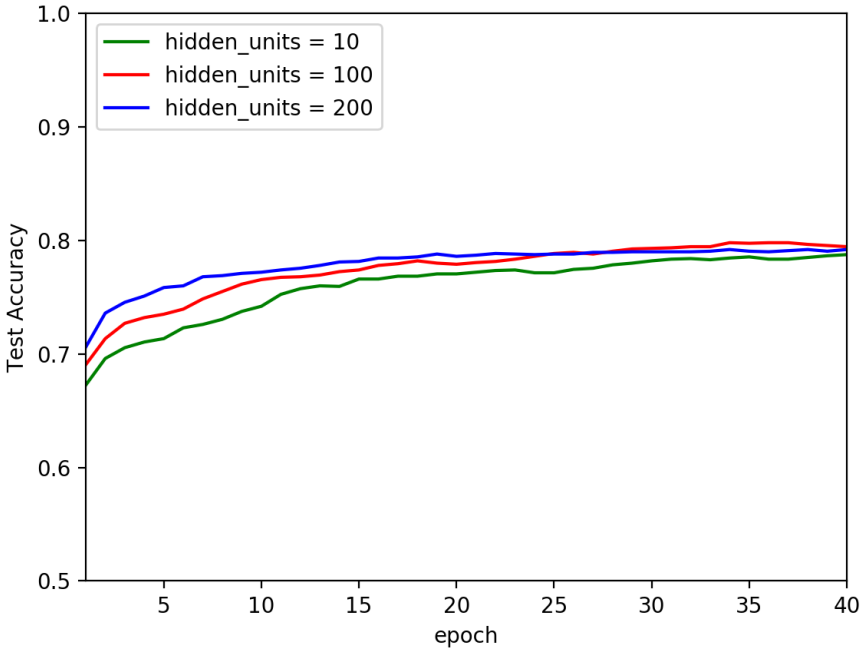

Figure 2. Comparing the test accuracy with different numbers of hidden units

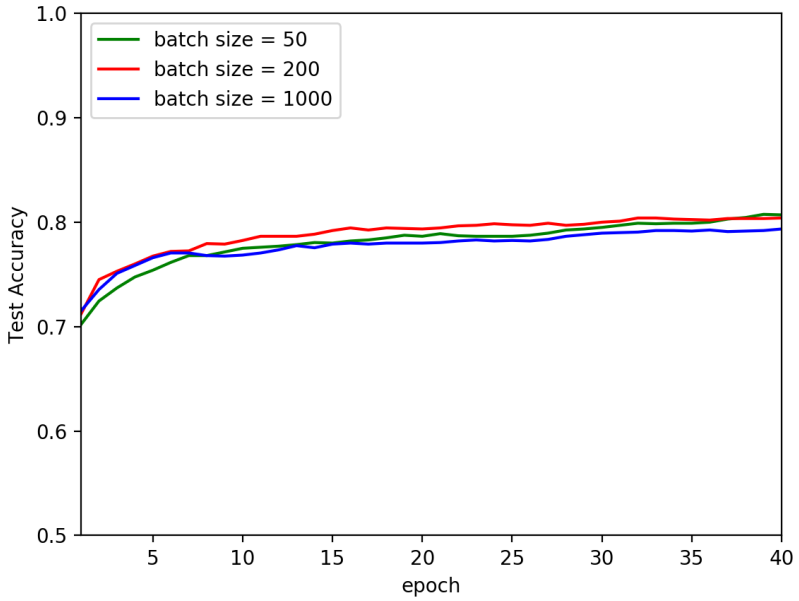

Figure 3. Comparing the test accuracy with different batch sizes.

After the comparing, I found that the larger learning rate and the more hidden unit number would improve the test accuracy effectively but changing the batch size would not change too much on accuracy.

6. Discussion about the performance of your neural network.

Solution:

In my tests, I found that when the hidden unit was 200, mini-batch size was 50, learning rate was 0.05 and momentum was 0.8, it would get the best performance with 40 epochs in my trained network. (the accuracy on the test set is 82.15 and the Error Rate is 17.85%). I satisfied with this trained network, but I think the accuracy can be higher if I add more hidden units for the same epochs.

Hidden Units	Time
50	152.1622s (faster)
100	1154.2792s
200	1365.3317s

Table 1. Comparing the running time with different hidden units.

Learning rate	Time
0.05	1222.8330s
0.01	1261.4984s
0.005	1345.2932s

Table 2. Comparing the running time with different learning rates

Batch Size	Time
50	572.1912s
200	570.1902s
1000	570.8614s

Table 3. Comparing the running time with different batch sizes

So, I just report the best number of hidden units is 200, but the tendency is that the more hidden units will make the result more accurate. As for the mini batch size, I think the different value of batch size doesn't change too much to the accuracy. The learning rate is interesting, at first, I just set the small value as the learning rate (0.001,0.005,0.0001), they are really slowly and cannot get a good accuracy after a long time. Finally, I try some other numbers, I found enlarge the learning rate will help to get a higher accuracy with less epochs, but it doesn't mean that the larger is the better. When I set the learning rate 0.005, it will achieve the best performance in my network.