CS 535 Deep Learning

CIFAR-10 Image Classification using Fully Connected Neural Network

2020 Winter Term

Oregon State of University

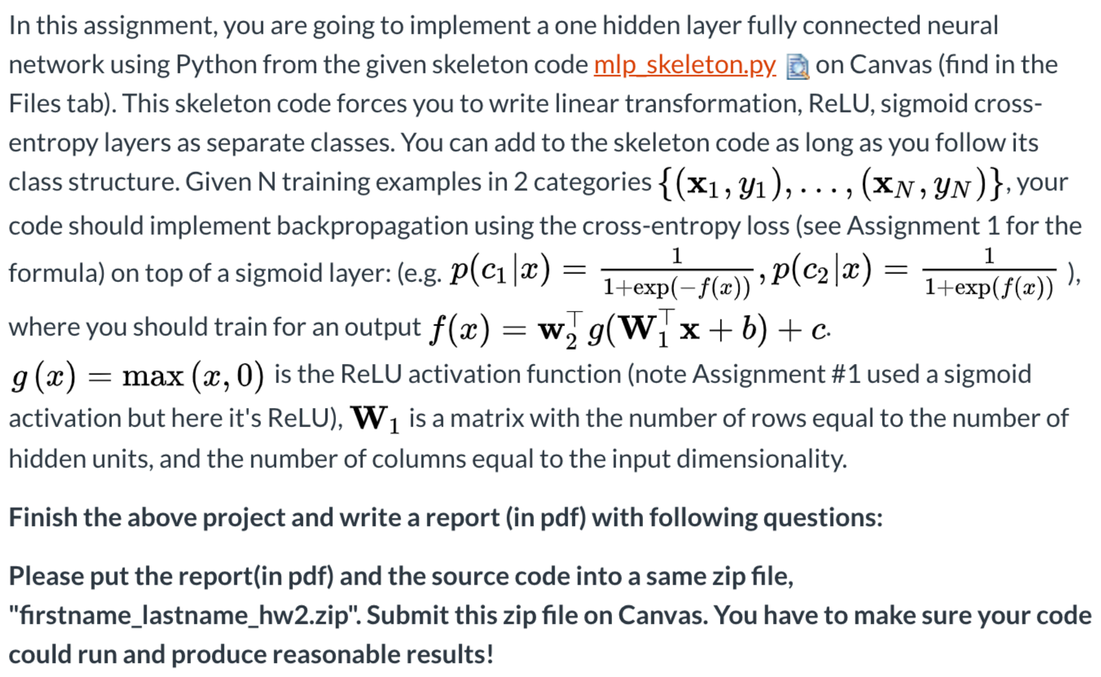
Instructor: Fuxin Li

Name: Yu-Wen Tseng

ONID: 933-652-910

tsengyuw@oregonstate.edu

Description:



In this assignment, you are going to implement a one hidden layer fully connected neural network using Python from the given skeleton code mlp_skeleton.py on Canvas (find in the Files tab). This skeleton code forces you to write linear transformation, ReLU, sigmoid cross-entropy layers as separate classes. You can add to the skeleton code as long as you follow its class structure. Given N training examples in 2 categories $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, your code should implement backpropagation using the cross-entropy loss (see Assignment 1 for the formula) on top of a sigmoid layer: (e.g. $p(c_1|x) = \frac{1}{1+\exp(-f(x))}$, $p(c_2|x) = \frac{1}{1+\exp(f(x))}$), where you should train for an output $f(x) = \mathbf{w}_2^\top g(\mathbf{W}_1^\top \mathbf{x} + b) + c$.
$g(x) = \max(x, 0)$ is the ReLU activation function (note Assignment #1 used a sigmoid activation but here it's ReLU), $\mathbf{W}_1$ is a matrix with the number of rows equal to the number of hidden units, and the number of columns equal to the input dimensionality.

Finish the above project and write a report (in pdf) with following questions:

Please put the report(in pdf) and the source code into a same zip file, "firstname_lastname_hw2.zip". Submit this zip file on Canvas. You have to make sure your code could run and produce reasonable results!

According to the description on the above, we can get the each layer and function as below:

Z1 = $W^T x + b$

Z2 = g(z1) = max(z1 , 0)

Z3 = $W_2^T Z_2 + C$

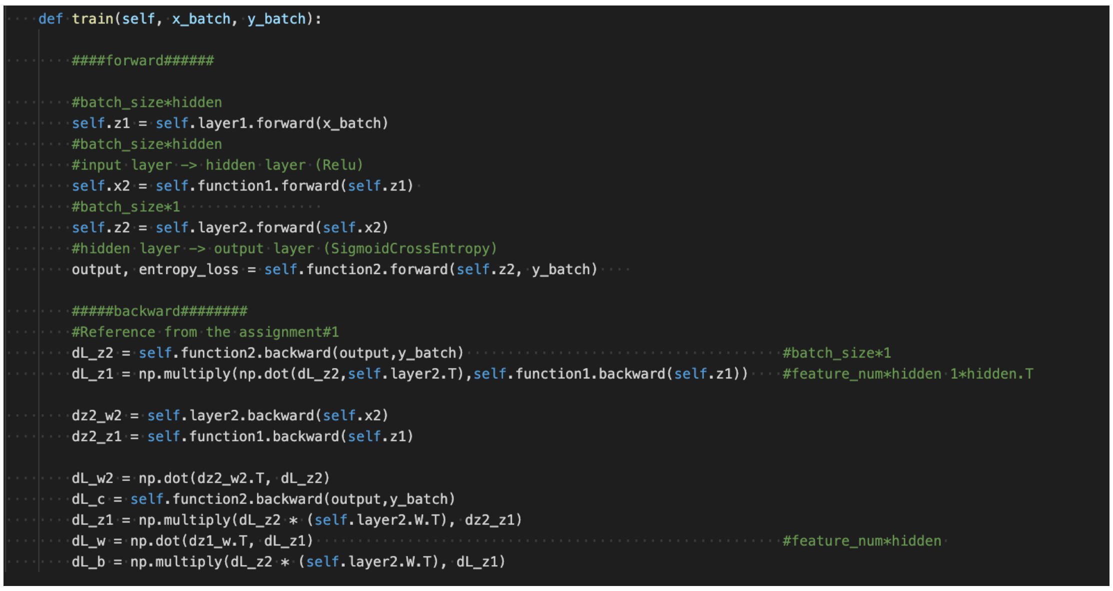
P = $\sigma(Z3) = \frac{1}{1+e^\wedge(-z3)}$

E = (-y)log(p)-(1-y)log(1-p)

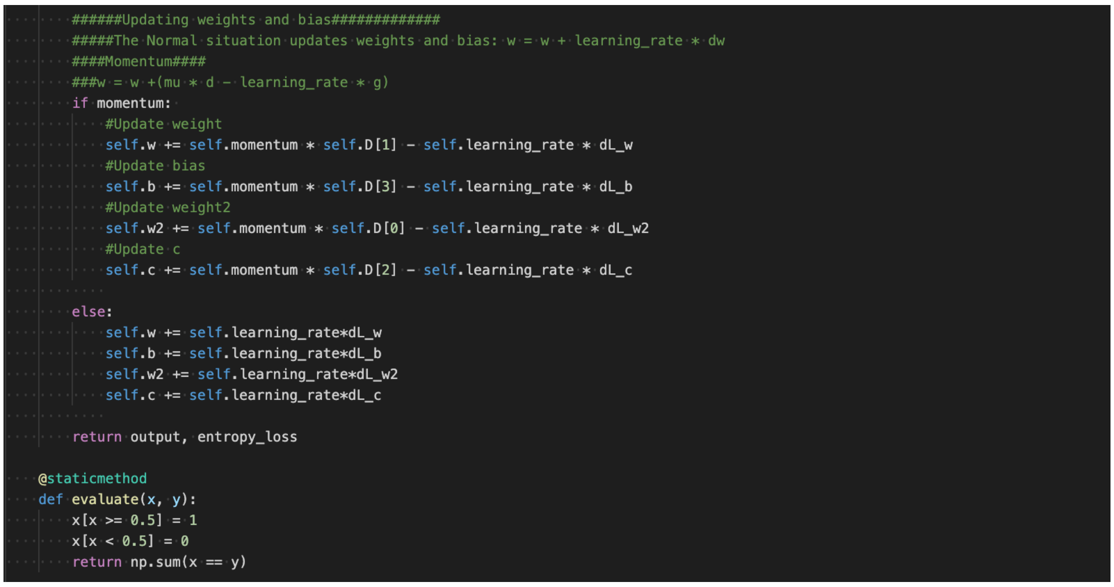
1) Write a function that evaluates the trained network, as well as computes all the subgradients of W 1 and W 2 using backpropagation.

The Code is in the YuWen_Tseng_hw2.zip.

The backwardpropagation is reference from the Assignment#1

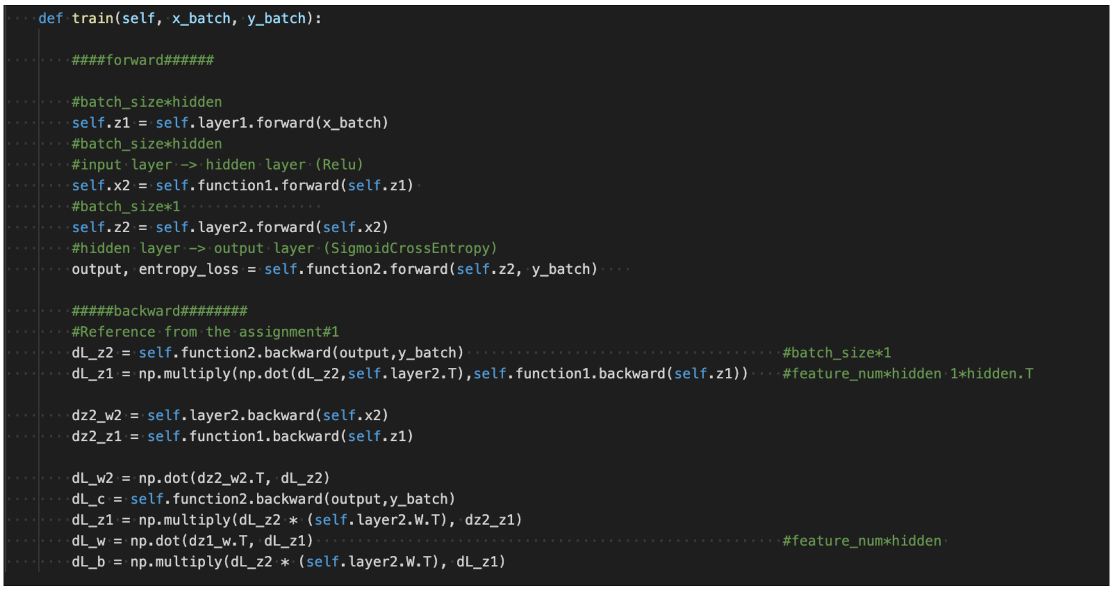


```
    def train(self, x_batch, y_batch):

        ####forward######

        #batch_size*hidden
        self.z1 = self.layer1.forward(x_batch)
        #batch_size*hidden
        #input layer -> hidden layer (Relu)
        self.x2 = self.function1.forward(self.z1)
        #batch_size*1
        self.z2 = self.layer2.forward(self.x2)
        #hidden layer -> output layer (SigmoidCrossEntropy)
        output, entropy_loss = self.function2.forward(self.z2, y_batch)

        #####backward#######
        #Reference from the assignment#1
        dL_z2 = self.function2.backward(output,y_batch)                                      #batch_size*1
        dL_z1 = np.multiply(np.dot(dL_z2,self.layer2.T),self.function1.backward(self.z1))    #feature_num*hidden 1*hidden.T

        dz2_w2 = self.layer2.backward(self.x2)
        dz2_z1 = self.function1.backward(self.z1)

        dL_w2 = np.dot(dz2_w2.T, dL_z2)
        dL_c = self.function2.backward(output,y_batch)
        dL_z1 = np.multiply(dL_z2 * (self.layer2.W.T), dz2_z1)
        dL_w = np.dot(dz1_w.T, dL_z1)                                                        #feature_num*hidden
        dL_b = np.multiply(dL_z2 * (self.layer2.W.T), dL_z1)
```

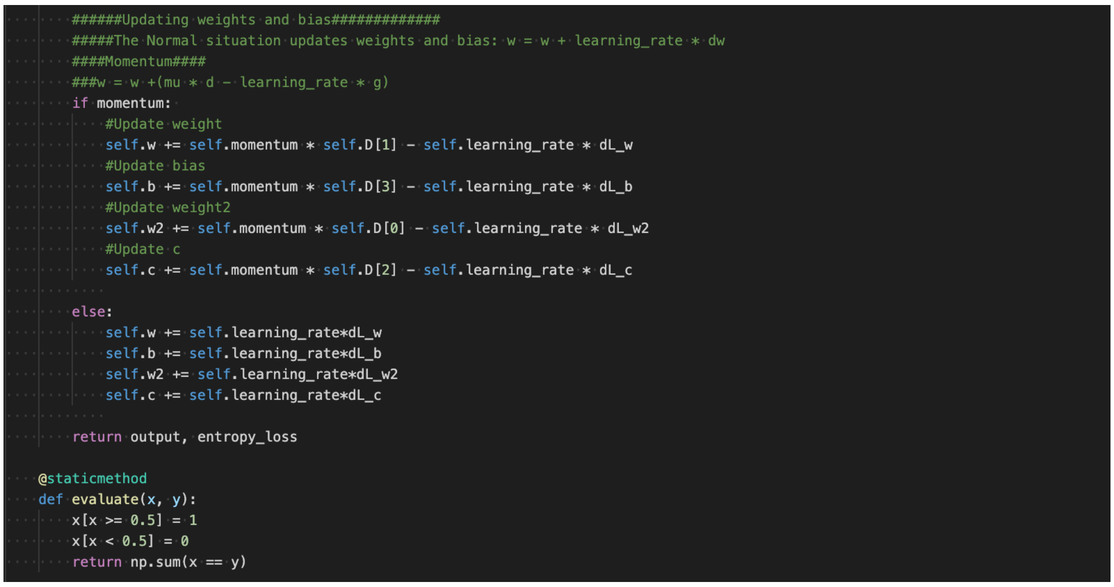


```
        ######Updating weights and bias############
        #####The Normal situation updates weights and bias: w = w + learning_rate * dw
        ####Momentum####
        ###w = w +(mu * d - learning_rate * g)
        if momentum:
            #Update weight
            self.w += self.momentum * self.D[1] - self.learning_rate * dL_w
            #Update bias
            self.b += self.momentum * self.D[3] - self.learning_rate * dL_b
            #Update weight2
            self.w2 += self.momentum * self.D[0] - self.learning_rate * dL_w2
            #Update c
            self.c += self.momentum * self.D[2] - self.learning_rate * dL_c

        else:
            self.w += self.learning_rate*dL_w
            self.b += self.learning_rate*dL_b
            self.w2 += self.learning_rate*dL_w2
            self.c += self.learning_rate*dL_c

        return output, entropy_loss

    @staticmethod
    def evaluate(x, y):
        x[x >= 0.5] = 1
        x[x < 0.5] = 0
        return np.sum(x == y)
```

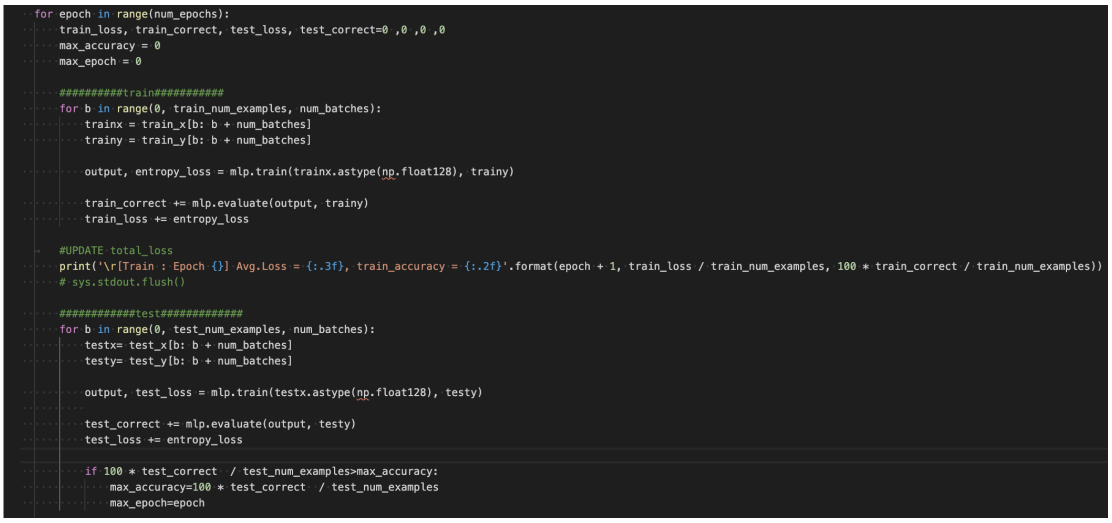
2) Write a function that performs stochastic mini-batch gradient descent training. You may use the deterministic approach of permuting the sequence of the data. Use the momentum approach described in the course slides.



```python
    def train(self, x_batch, y_batch):

        ####forward######

        #batch_size*hidden
        self.z1 = self.layer1.forward(x_batch)
        #batch_size*hidden
        #input layer -> hidden layer (Relu)
        self.x2 = self.function1.forward(self.z1)
        #batch_size*1
        self.z2 = self.layer2.forward(self.x2)
        #hidden layer -> output layer (SigmoidCrossEntropy)
        output, entropy_loss = self.function2.forward(self.z2, y_batch)

        #####backward#######
        #Reference from the assignment#1
        dL_z2 = self.function2.backward(output,y_batch)                                     #batch_size*1
        dL_z1 = np.multiply(np.dot(dL_z2,self.layer2.T),self.function1.backward(self.z1))   #feature_num*hidden 1*hidden.T

        dz2_w2 = self.layer2.backward(self.x2)
        dz2_z1 = self.function1.backward(self.z1)

        dL_w2 = np.dot(dz2_w2.T, dL_z2)
        dL_c = self.function2.backward(output,y_batch)
        dL_z1 = np.multiply(dL_z2 * (self.layer2.W.T), dz2_z1)
        dL_w = np.dot(dz1_w.T, dL_z1)                                                       #feature_num*hidden
        dL_b = np.multiply(dL_z2 * (self.layer2.W.T), dL_z1)
```



```python
        ######Updating weights and bias############
        #####The Normal situation updates weights and bias: w = w + learning_rate * dw
        ####Momentum####
        ###w = w +(mu * d - learning_rate * g)
        if momentum:
            #Update weight
            self.w += self.momentum * self.D[1] - self.learning_rate * dL_w
            #Update bias
            self.b += self.momentum * self.D[3] - self.learning_rate * dL_b
            #Update weight2
            self.w2 += self.momentum * self.D[0] - self.learning_rate * dL_w2
            #Update c
            self.c += self.momentum * self.D[2] - self.learning_rate * dL_c

        else:
            self.w += self.learning_rate*dL_w
            self.b += self.learning_rate*dL_b
            self.w2 += self.learning_rate*dL_w2
            self.c += self.learning_rate*dL_c

        return output, entropy_loss

    @staticmethod
    def evaluate(x, y):
        x[x >= 0.5] = 1
        x[x < 0.5] = 0
        return np.sum(x == y)
```

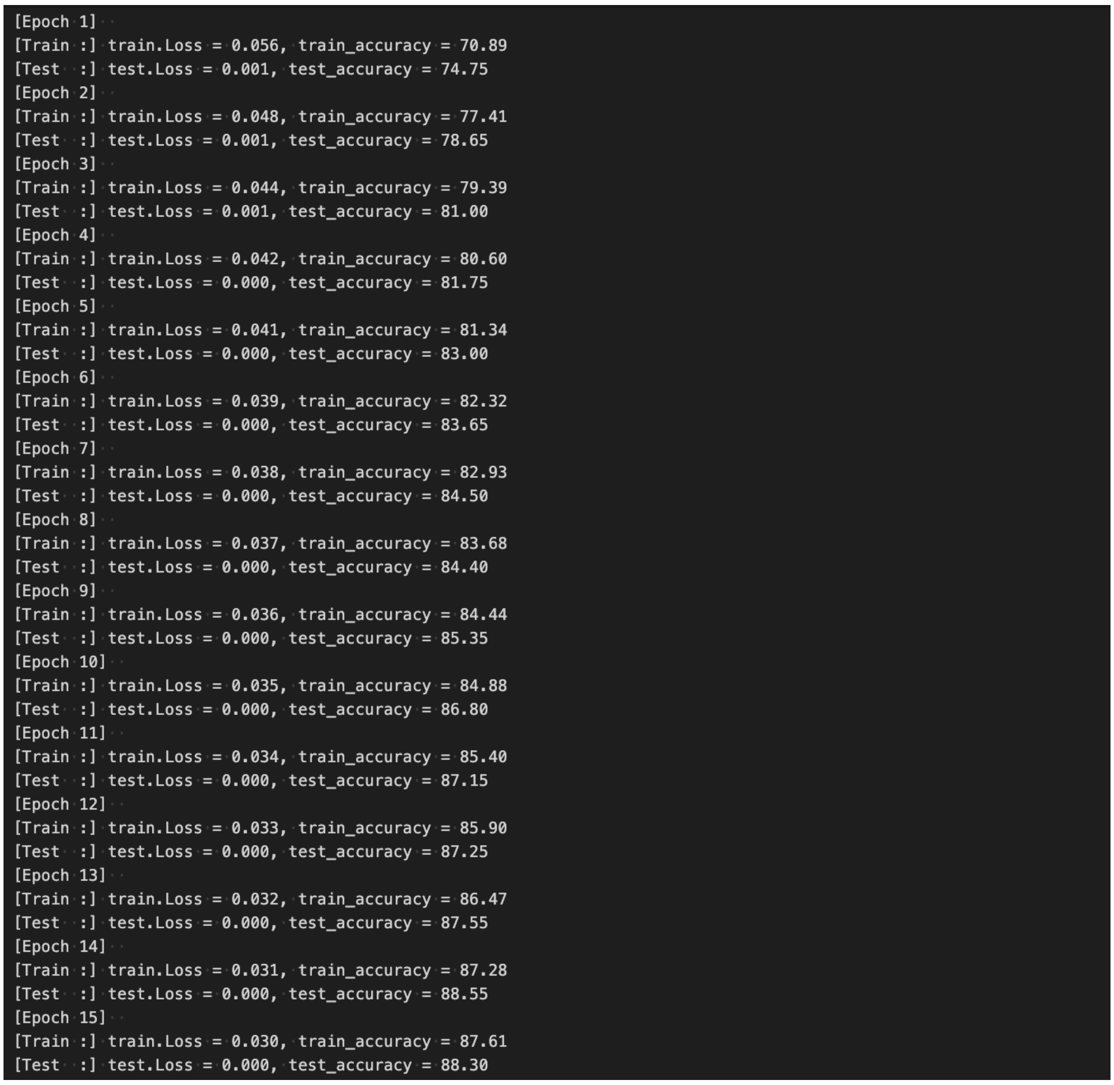


```python
  for epoch in range(num_epochs):
      train_loss, train_correct, test_loss, test_correct=0 ,0 ,0 ,0
      max_accuracy = 0
      max_epoch = 0

      ##########train###########
      for b in range(0, train_num_examples, num_batches):
          trainx = train_x[b: b + num_batches]
          trainy = train_y[b: b + num_batches]

          output, entropy_loss = mlp.train(trainx.astype(np.float128), trainy)

          train_correct += mlp.evaluate(output, trainy)
          train_loss += entropy_loss

      #UPDATE total_loss
      print('\r[Train : Epoch {}] Avg.Loss = {:.3f}, train_accuracy = {:.2f}'.format(epoch + 1, train_loss / train_num_examples, 100 * train_correct / train_num_examples))
      # sys.stdout.flush()

      ############test#############
      for b in range(0, test_num_examples, num_batches):
          testx= test_x[b: b + num_batches]
          testy= test_y[b: b + num_batches]

          output, test_loss = mlp.train(testx.astype(np.float128), testy)

          test_correct += mlp.evaluate(output, testy)
          test_loss += entropy_loss

          if 100 * test_correct  / test_num_examples>max_accuracy:
              max_accuracy=100 * test_correct  / test_num_examples
              max_epoch=epoch
```

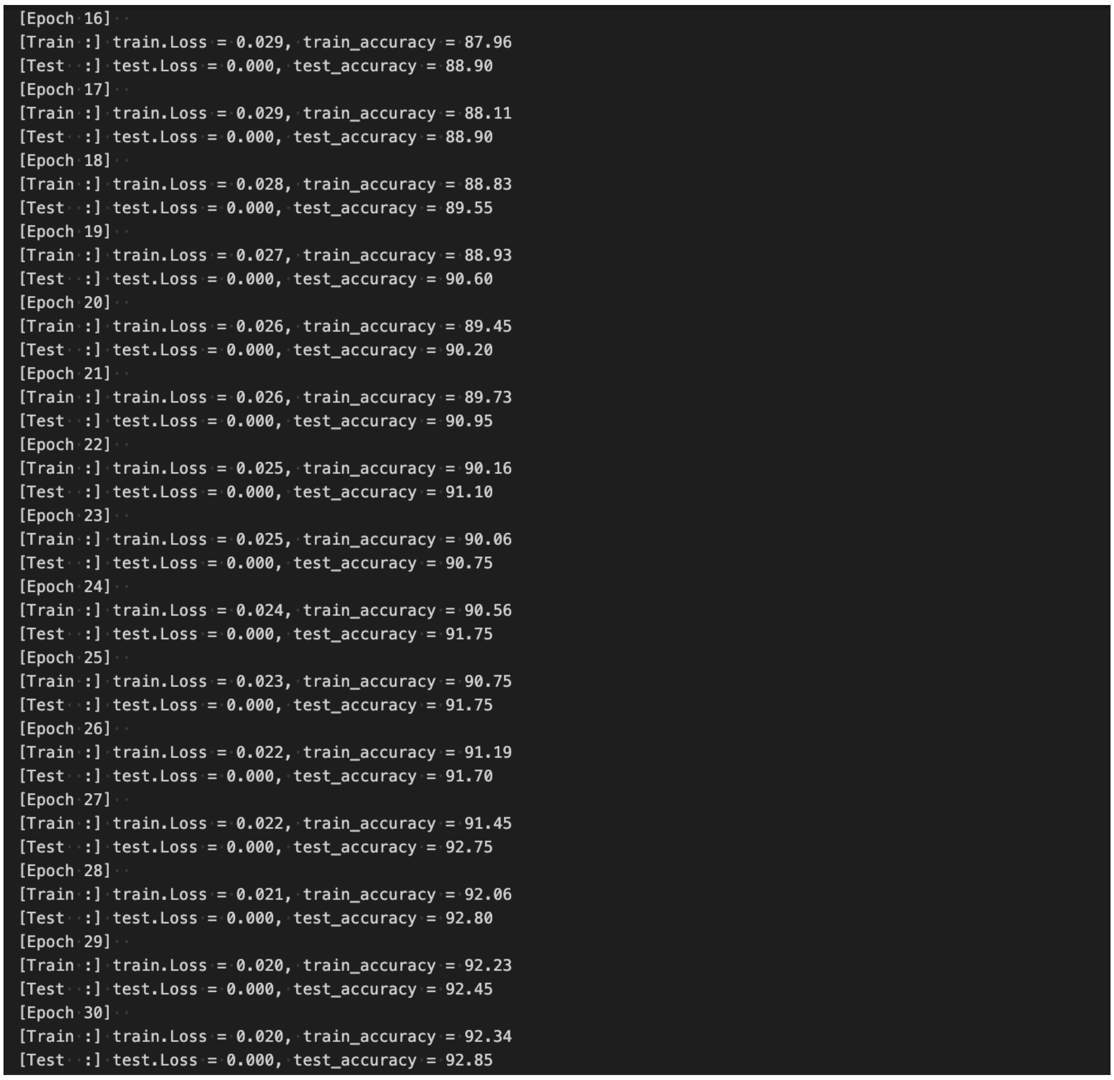
3) Train the network on the attached 2-class dataset extracted from CIFAR-10: (data can be found in the cifar-2class-py2.zip file on Canvas.). The data has

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?

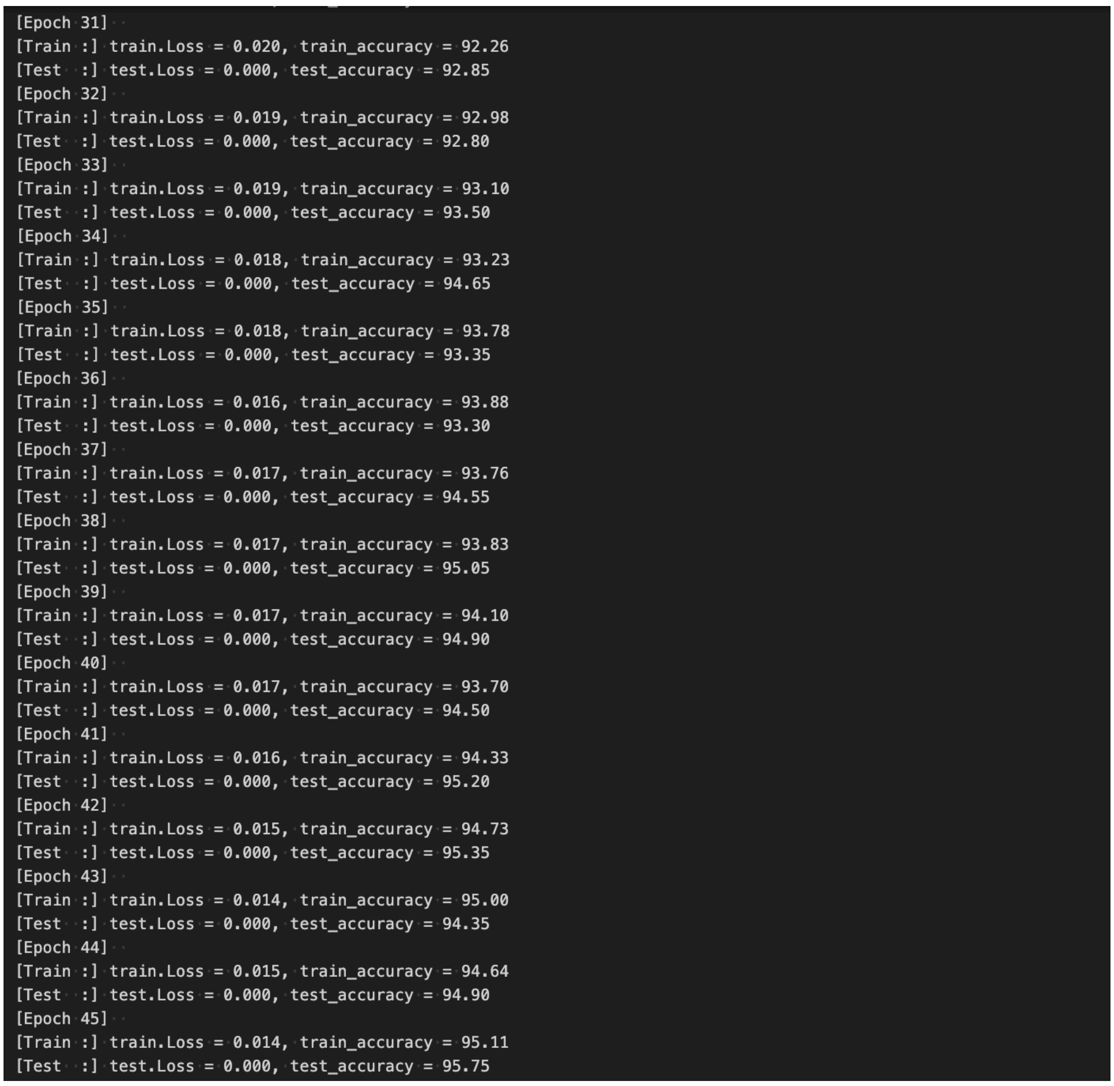
Epoch: 14, hidden units: 50, learning rate: 1e-3, batch size: 10, momentum: 0.5



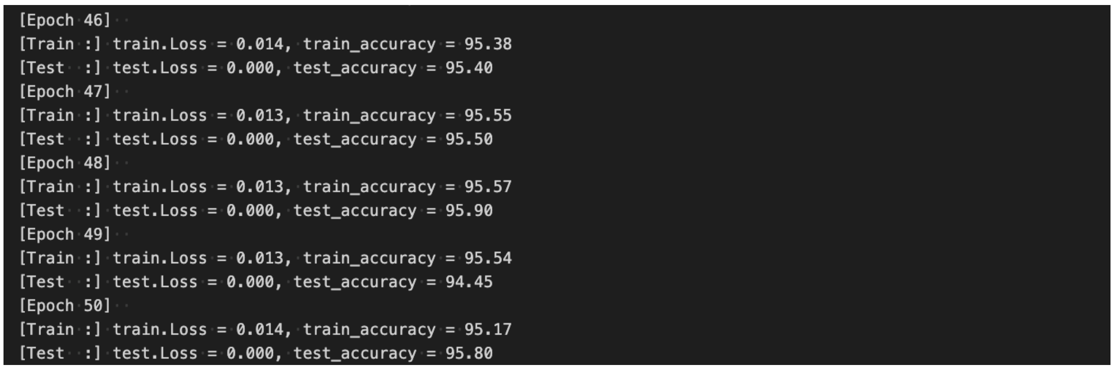
```
[Epoch 1]
[Train :] train.Loss = 0.056, train_accuracy = 70.89
[Test  :] test.Loss = 0.001, test_accuracy = 74.75
[Epoch 2]
[Train :] train.Loss = 0.048, train_accuracy = 77.41
[Test  :] test.Loss = 0.001, test_accuracy = 78.65
[Epoch 3]
[Train :] train.Loss = 0.044, train_accuracy = 79.39
[Test  :] test.Loss = 0.001, test_accuracy = 81.00
[Epoch 4]
[Train :] train.Loss = 0.042, train_accuracy = 80.60
[Test  :] test.Loss = 0.000, test_accuracy = 81.75
[Epoch 5]
[Train :] train.Loss = 0.041, train_accuracy = 81.34
[Test  :] test.Loss = 0.000, test_accuracy = 83.00
[Epoch 6]
[Train :] train.Loss = 0.039, train_accuracy = 82.32
[Test  :] test.Loss = 0.000, test_accuracy = 83.65
[Epoch 7]
[Train :] train.Loss = 0.038, train_accuracy = 82.93
[Test  :] test.Loss = 0.000, test_accuracy = 84.50
[Epoch 8]
[Train :] train.Loss = 0.037, train_accuracy = 83.68
[Test  :] test.Loss = 0.000, test_accuracy = 84.40
[Epoch 9]
[Train :] train.Loss = 0.036, train_accuracy = 84.44
[Test  :] test.Loss = 0.000, test_accuracy = 85.35
[Epoch 10]
[Train :] train.Loss = 0.035, train_accuracy = 84.88
[Test  :] test.Loss = 0.000, test_accuracy = 86.80
[Epoch 11]
[Train :] train.Loss = 0.034, train_accuracy = 85.40
[Test  :] test.Loss = 0.000, test_accuracy = 87.15
[Epoch 12]
[Train :] train.Loss = 0.033, train_accuracy = 85.90
[Test  :] test.Loss = 0.000, test_accuracy = 87.25
[Epoch 13]
[Train :] train.Loss = 0.032, train_accuracy = 86.47
[Test  :] test.Loss = 0.000, test_accuracy = 87.55
[Epoch 14]
[Train :] train.Loss = 0.031, train_accuracy = 87.28
[Test  :] test.Loss = 0.000, test_accuracy = 88.55
[Epoch 15]
[Train :] train.Loss = 0.030, train_accuracy = 87.61
[Test  :] test.Loss = 0.000, test_accuracy = 88.30
```



```
[Epoch 16]
[Train :] train.Loss = 0.029, train_accuracy = 87.96
[Test  :] test.Loss = 0.000, test_accuracy = 88.90
[Epoch 17]
[Train :] train.Loss = 0.029, train_accuracy = 88.11
[Test  :] test.Loss = 0.000, test_accuracy = 88.90
[Epoch 18]
[Train :] train.Loss = 0.028, train_accuracy = 88.83
[Test  :] test.Loss = 0.000, test_accuracy = 89.55
[Epoch 19]
[Train :] train.Loss = 0.027, train_accuracy = 88.93
[Test  :] test.Loss = 0.000, test_accuracy = 90.60
[Epoch 20]
[Train :] train.Loss = 0.026, train_accuracy = 89.45
[Test  :] test.Loss = 0.000, test_accuracy = 90.20
[Epoch 21]
[Train :] train.Loss = 0.026, train_accuracy = 89.73
[Test  :] test.Loss = 0.000, test_accuracy = 90.95
[Epoch 22]
[Train :] train.Loss = 0.025, train_accuracy = 90.16
[Test  :] test.Loss = 0.000, test_accuracy = 91.10
[Epoch 23]
[Train :] train.Loss = 0.025, train_accuracy = 90.06
[Test  :] test.Loss = 0.000, test_accuracy = 90.75
[Epoch 24]
[Train :] train.Loss = 0.024, train_accuracy = 90.56
[Test  :] test.Loss = 0.000, test_accuracy = 91.75
[Epoch 25]
[Train :] train.Loss = 0.023, train_accuracy = 90.75
[Test  :] test.Loss = 0.000, test_accuracy = 91.75
[Epoch 26]
[Train :] train.Loss = 0.022, train_accuracy = 91.19
[Test  :] test.Loss = 0.000, test_accuracy = 91.70
[Epoch 27]
[Train :] train.Loss = 0.022, train_accuracy = 91.45
[Test  :] test.Loss = 0.000, test_accuracy = 92.75
[Epoch 28]
[Train :] train.Loss = 0.021, train_accuracy = 92.06
[Test  :] test.Loss = 0.000, test_accuracy = 92.80
[Epoch 29]
[Train :] train.Loss = 0.020, train_accuracy = 92.23
[Test  :] test.Loss = 0.000, test_accuracy = 92.45
[Epoch 30]
[Train :] train.Loss = 0.020, train_accuracy = 92.34
[Test  :] test.Loss = 0.000, test_accuracy = 92.85
```

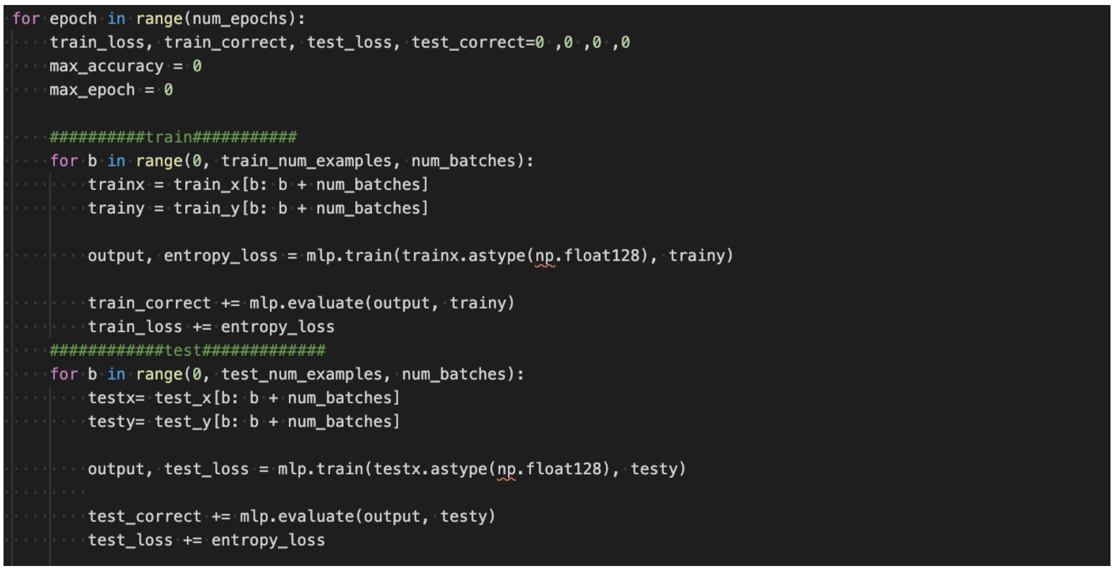


```
[Epoch 31]
[Train :] train.Loss = 0.020, train_accuracy = 92.26
[Test  :] test.Loss = 0.000, test_accuracy = 92.85
[Epoch 32]
[Train :] train.Loss = 0.019, train_accuracy = 92.98
[Test  :] test.Loss = 0.000, test_accuracy = 92.80
[Epoch 33]
[Train :] train.Loss = 0.019, train_accuracy = 93.10
[Test  :] test.Loss = 0.000, test_accuracy = 93.50
[Epoch 34]
[Train :] train.Loss = 0.018, train_accuracy = 93.23
[Test  :] test.Loss = 0.000, test_accuracy = 94.65
[Epoch 35]
[Train :] train.Loss = 0.018, train_accuracy = 93.78
[Test  :] test.Loss = 0.000, test_accuracy = 93.35
[Epoch 36]
[Train :] train.Loss = 0.016, train_accuracy = 93.88
[Test  :] test.Loss = 0.000, test_accuracy = 93.30
[Epoch 37]
[Train :] train.Loss = 0.017, train_accuracy = 93.76
[Test  :] test.Loss = 0.000, test_accuracy = 94.55
[Epoch 38]
[Train :] train.Loss = 0.017, train_accuracy = 93.83
[Test  :] test.Loss = 0.000, test_accuracy = 95.05
[Epoch 39]
[Train :] train.Loss = 0.017, train_accuracy = 94.10
[Test  :] test.Loss = 0.000, test_accuracy = 94.90
[Epoch 40]
[Train :] train.Loss = 0.017, train_accuracy = 93.70
[Test  :] test.Loss = 0.000, test_accuracy = 94.50
[Epoch 41]
[Train :] train.Loss = 0.016, train_accuracy = 94.33
[Test  :] test.Loss = 0.000, test_accuracy = 95.20
[Epoch 42]
[Train :] train.Loss = 0.015, train_accuracy = 94.73
[Test  :] test.Loss = 0.000, test_accuracy = 95.35
[Epoch 43]
[Train :] train.Loss = 0.014, train_accuracy = 95.00
[Test  :] test.Loss = 0.000, test_accuracy = 94.35
[Epoch 44]
[Train :] train.Loss = 0.015, train_accuracy = 94.64
[Test  :] test.Loss = 0.000, test_accuracy = 94.90
[Epoch 45]
[Train :] train.Loss = 0.014, train_accuracy = 95.11
[Test  :] test.Loss = 0.000, test_accuracy = 95.75
```

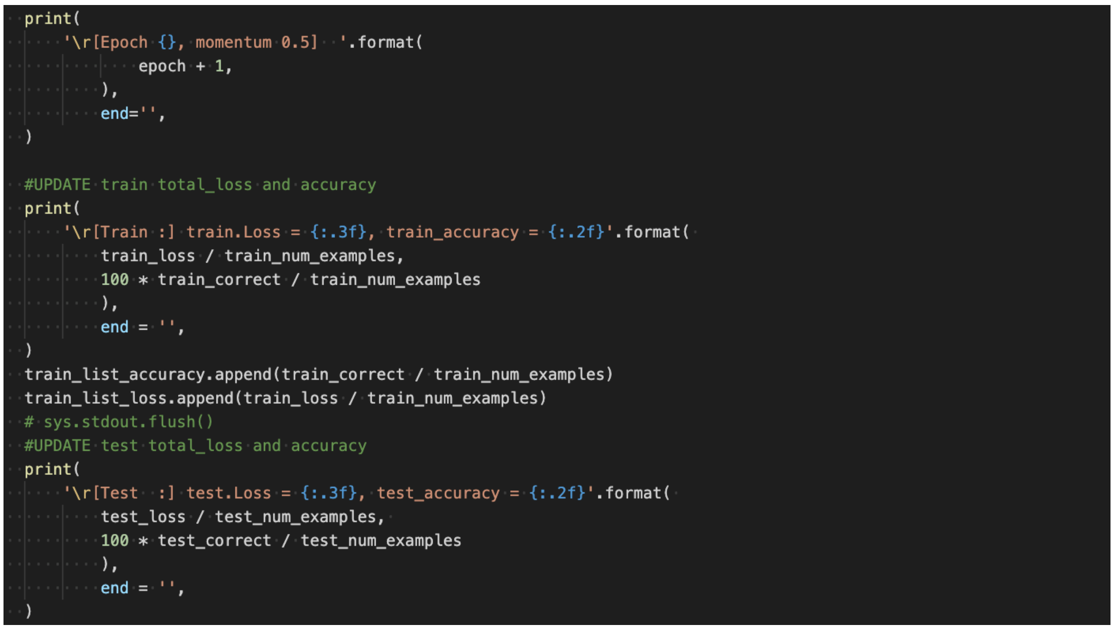


```
[Epoch 46]
[Train :] train.Loss = 0.014, train_accuracy = 95.38
[Test  :] test.Loss = 0.000, test_accuracy = 95.40
[Epoch 47]
[Train :] train.Loss = 0.013, train_accuracy = 95.55
[Test  :] test.Loss = 0.000, test_accuracy = 95.50
[Epoch 48]
[Train :] train.Loss = 0.013, train_accuracy = 95.57
[Test  :] test.Loss = 0.000, test_accuracy = 95.90
[Epoch 49]
[Train :] train.Loss = 0.013, train_accuracy = 95.54
[Test  :] test.Loss = 0.000, test_accuracy = 94.45
[Epoch 50]
[Train :] train.Loss = 0.014, train_accuracy = 95.17
[Test  :] test.Loss = 0.000, test_accuracy = 95.80
```

(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.

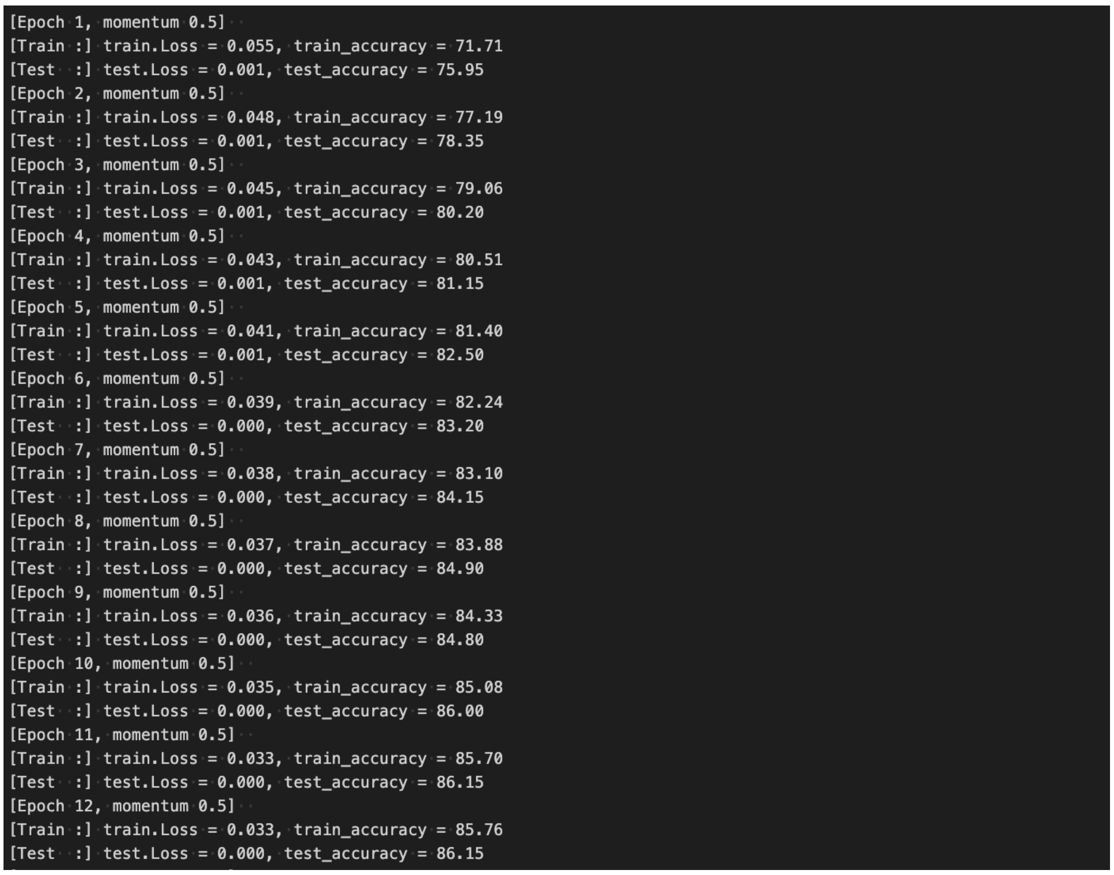


```
for epoch in range(num_epochs):
    train_loss, train_correct, test_loss, test_correct=0 ,0 ,0 ,0
    max_accuracy = 0
    max_epoch = 0

    #########train##########
    for b in range(0, train_num_examples, num_batches):
        trainx = train_x[b: b + num_batches]
        trainy = train_y[b: b + num_batches]

        output, entropy_loss = mlp.train(trainx.astype(np.float128), trainy)

        train_correct += mlp.evaluate(output, trainy)
        train_loss += entropy_loss
    ###########test############
    for b in range(0, test_num_examples, num_batches):
        testx= test_x[b: b + num_batches]
        testy= test_y[b: b + num_batches]

        output, test_loss = mlp.train(testx.astype(np.float128), testy)

        test_correct += mlp.evaluate(output, testy)
        test_loss += entropy_loss
```



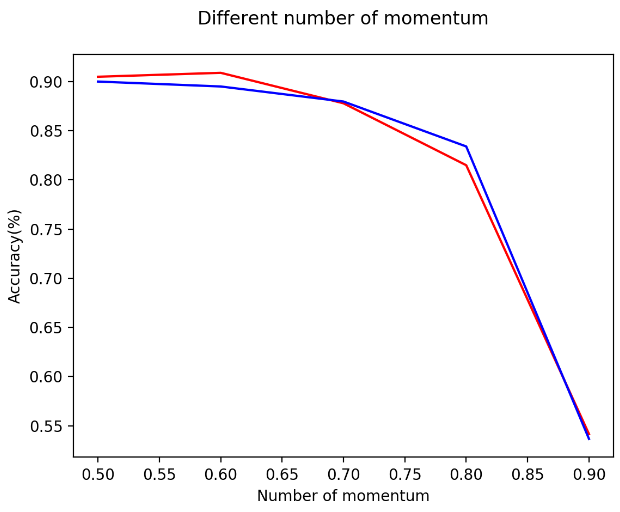
```
  print(
      '\r[Epoch {}, momentum 0.5]  '.format(
              epoch + 1,
          ),
          end='',
  )

  #UPDATE train total_loss and accuracy
  print(
      '\r[Train :] train.Loss = {:.3f}, train_accuracy = {:.2f}'.format(
          train_loss / train_num_examples,
          100 * train_correct / train_num_examples
          ),
          end = '',
  )
  train_list_accuracy.append(train_correct / train_num_examples)
  train_list_loss.append(train_loss / train_num_examples)
  # sys.stdout.flush()
  #UPDATE test total_loss and accuracy
  print(
      '\r[Test  :] test.Loss = {:.3f}, test_accuracy = {:.2f}'.format(
          test_loss / test_num_examples,
          100 * test_correct / test_num_examples
          ),
          end = '',
  )
```

Example:



```
[Epoch 1, momentum 0.5]
[Train :] train.Loss = 0.055, train_accuracy = 71.71
[Test  :] test.Loss = 0.001, test_accuracy = 75.95
[Epoch 2, momentum 0.5]
[Train :] train.Loss = 0.048, train_accuracy = 77.19
[Test  :] test.Loss = 0.001, test_accuracy = 78.35
[Epoch 3, momentum 0.5]
[Train :] train.Loss = 0.045, train_accuracy = 79.06
[Test  :] test.Loss = 0.001, test_accuracy = 80.20
[Epoch 4, momentum 0.5]
[Train :] train.Loss = 0.043, train_accuracy = 80.51
[Test  :] test.Loss = 0.001, test_accuracy = 81.15
[Epoch 5, momentum 0.5]
[Train :] train.Loss = 0.041, train_accuracy = 81.40
[Test  :] test.Loss = 0.001, test_accuracy = 82.50
[Epoch 6, momentum 0.5]
[Train :] train.Loss = 0.039, train_accuracy = 82.24
[Test  :] test.Loss = 0.000, test_accuracy = 83.20
[Epoch 7, momentum 0.5]
[Train :] train.Loss = 0.038, train_accuracy = 83.10
[Test  :] test.Loss = 0.000, test_accuracy = 84.15
[Epoch 8, momentum 0.5]
[Train :] train.Loss = 0.037, train_accuracy = 83.88
[Test  :] test.Loss = 0.000, test_accuracy = 84.90
[Epoch 9, momentum 0.5]
[Train :] train.Loss = 0.036, train_accuracy = 84.33
[Test  :] test.Loss = 0.000, test_accuracy = 84.80
[Epoch 10, momentum 0.5]
[Train :] train.Loss = 0.035, train_accuracy = 85.08
[Test  :] test.Loss = 0.000, test_accuracy = 86.00
[Epoch 11, momentum 0.5]
[Train :] train.Loss = 0.033, train_accuracy = 85.70
[Test  :] test.Loss = 0.000, test_accuracy = 86.15
[Epoch 12, momentum 0.5]
[Train :] train.Loss = 0.033, train_accuracy = 85.76
[Test  :] test.Loss = 0.000, test_accuracy = 86.15
```

Tune the number of Momentum = [0.5, 0.6, 0.7, 0.8, 0.9]

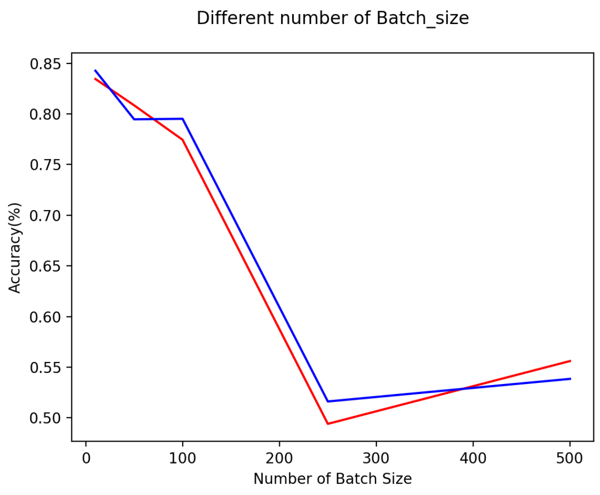


Red: Test Accuracy

Blue: Train Accuracy

(5) Tuning Parameters: please create three figures with following requirements. Save them into jpg format:

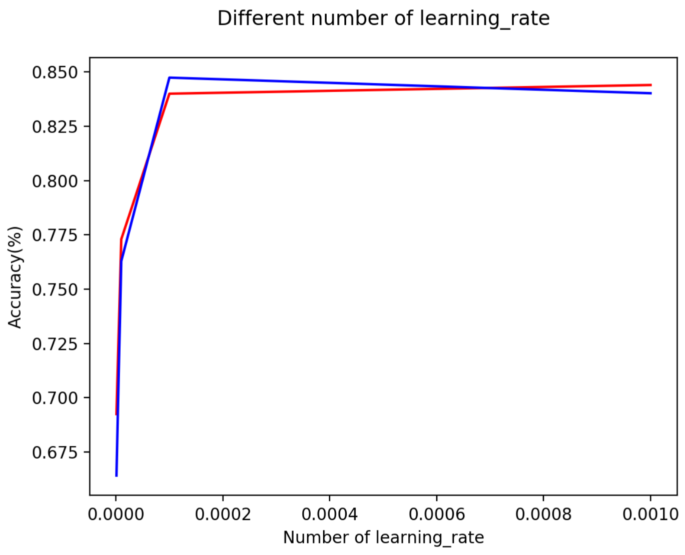
→ Accuracy with different number of "Batch size":
Tune Batch size = [10, 50, 100, 250, 500]



Red: Test Accuracy

Blue: Train Accuracy

→ Accuracy with different "Learning rate":
Tune Learning rate = [0.000001, 0.00001, 0.0001, 0.001]

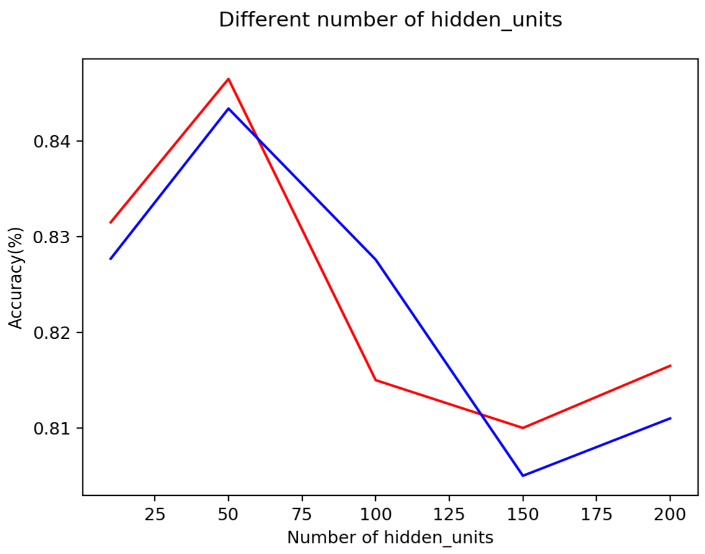


Red: Test Accuracy

Blue: Train Accuracy

→ Accuracy with different number of "hidden units":

Tune Hidden units = [10, 50, 100, 150, 200]



Red: Test Accuracy

Blue: Train Accuracy

(6) Discussion about the performance of your neural network

In my code, I found the best performance is 88.55%. Even if I tuned the epoch number to the 50 and got the performance 95.80%(shown in question3), this was not the best performance since the value was overfitting. Based on the data, we can also know that the accuracy of the epoch 15 is smaller than epoch 14. According to the compared data I ran, when tuning the batch size, the result would be the best when I set the value between 10 and 100. When the batch size is larger than 100, it will extremely decrease. The second compared data is the hidden unit, the picture shows that the accuracy will be the best accuracy when hidden unit occurring to the 50. Otherwise, it will decrease and oscillate a lot. About the learning rate, I made a list, which includes 0.000001, 0.00001, 0.0001 and 0.001 to test. I found the best performance would show when I set to 0.001. And the momentum I set to the 0.5 because of the smaller momentum I used the more accuracy I got. Therefore, in my neural network, the best performance is happened with number of epochs = 14, batch size = 10, hidden units = 50, learning rate = 0.001, and momentum parameter= 0.5 which are also shown in question 3.