Hw4

1.

```
import numpy as np
import matplotlib.pyplot as plt
import random
log1 = []
log2 = []
def training(V1, V2):
    W1 = np.outer(V1, V1)
    W2 = np.outer(V2, V2)
    W = (W1+W2)/5.0
    np.fill_diagonal(W, 0)
    return W


def evolving1(U, W):
    count = 0
    random_list1 = [3, 1, 5, 2, 4]
    while True:
        for i in range(0,5):
            sum1 = 0
            for j in range(0,5):
                index = random_list1[i]-1
                if j != index:
                    sum1 += W[j,index]*U[j]
            if sum1 >= 0 and U[random_list1[i]-1] != 1:
                U[random_list1[i]-1] = 1
                log1.append(1)
                count = 0

            if sum1 < 0 and U[random_list1[i]-1] != -1:
                U[random_list1[i]-1] = -1
                log1.append(-1)
                count = 0
            else:
                count+=1
        if count >= 5:
            break


def evolving2(U, W):
    count = 0
```

```
    random_list2 = [2,4,3,5,1]
    while True:
        for i in range(0,5):
            sum2 = 0
            for j in range(0,5):
                index = random_list2[i]-1
                if j != index:
                    sum2 += W[j,index]*U[j]
            if sum2 >= 0 and U[random_list2[i]-1] != 1:
                U[random_list1[i]-1] = 1
                log2.append(1)
                count = 0

            if sum2 < 0 and U[random_list2[i]-1] != -1:
                U[random_list2[i]-1] = -1
                log2.append(-1)
                count = 0
            else:
                count+=1
        if count >= 5:
            break



V1 = [-1, 1, 1, -1, 1]
V2 = [1, -1, 1, -1, 1]
V_new = [1, 1, 1, 1, 1]
W = training(V1, V2)
evolving1(V_new, W)
V_new = [1, 1, 1, 1, 1]
evolving2(V_new, W)
```

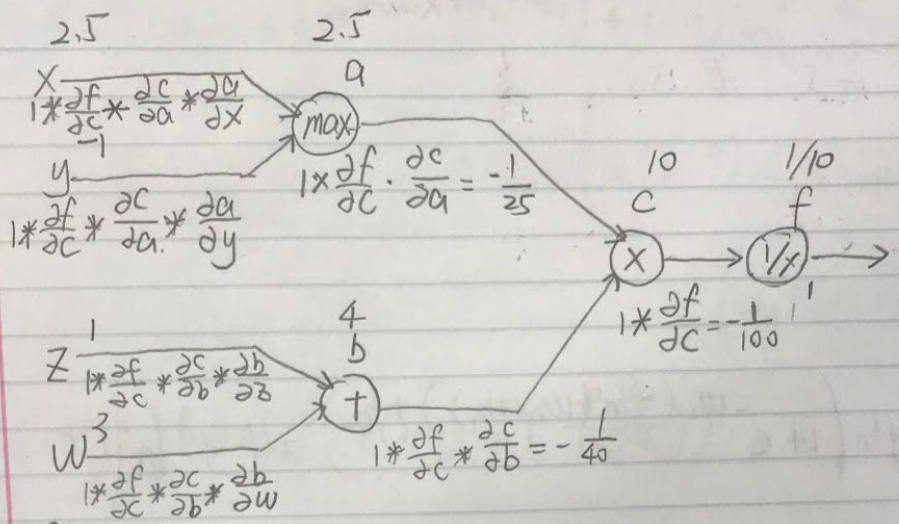
Updating history
pattern (3, 1, 5, 2, 4, 3, 1, 5, 2, 4...)
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[-1, 1, 1, 1, 1]
[-1, 1, 1, 1, 1]
[-1, 1, 1, 1, 1]
[-1, 1, 1, -1, 1]
[-1, 1, 1, -1, 1]
[-1, 1, 1, -1, 1]
[-1, 1, 1, -1, 1]
[-1, 1, 1, -1, 1]

Pattern (2, 4, 3, 5, 1, 2, 4, 3, 5, 1...)
[1, 1, 1, 1, 1]
[1, -1, 1, 1, 1]
[1, -1, 1, -1, 1]
[1, -1, 1, -1, 1]
[1, -1, 1, -1, 1]
[1, -1, 1, -1, 1]
[1, -1, 1, -1, 1]
[1, -1, 1, -1, 1]
[1, -1, 1, -1, 1]
[1, -1, 1, -1, 1]

Retrieved pattern and evolving sequence will eventually be the same. Evolving sequence will converge to retrieved pattern.

2.

1.



2. forward:

$a = \max(x, y) = \max(2.5, -1) = 2.5$

$b = z + w = 1 + 3 = 4$

$c = a \times b = 2.5 \times 4 = 10$

$f = 1/c = 1/10$

backward:

$\frac{\partial f}{\partial f} = 1 \qquad 1 \times \frac{\partial f}{\partial c} = 1 \times (-\frac{1}{c^2}) = 1 \times (-\frac{1}{100}) = -\frac{1}{100}$

$1 \times \frac{\partial f}{\partial c} \times \frac{\partial c}{\partial a} = -\frac{1}{100} \times b = -\frac{4}{100} = -\frac{1}{25}$

$1 \times \frac{\partial f}{\partial c} \times \frac{\partial c}{\partial b} = -\frac{1}{100} \times a = -\frac{2.5}{100} = -\frac{1}{40}$

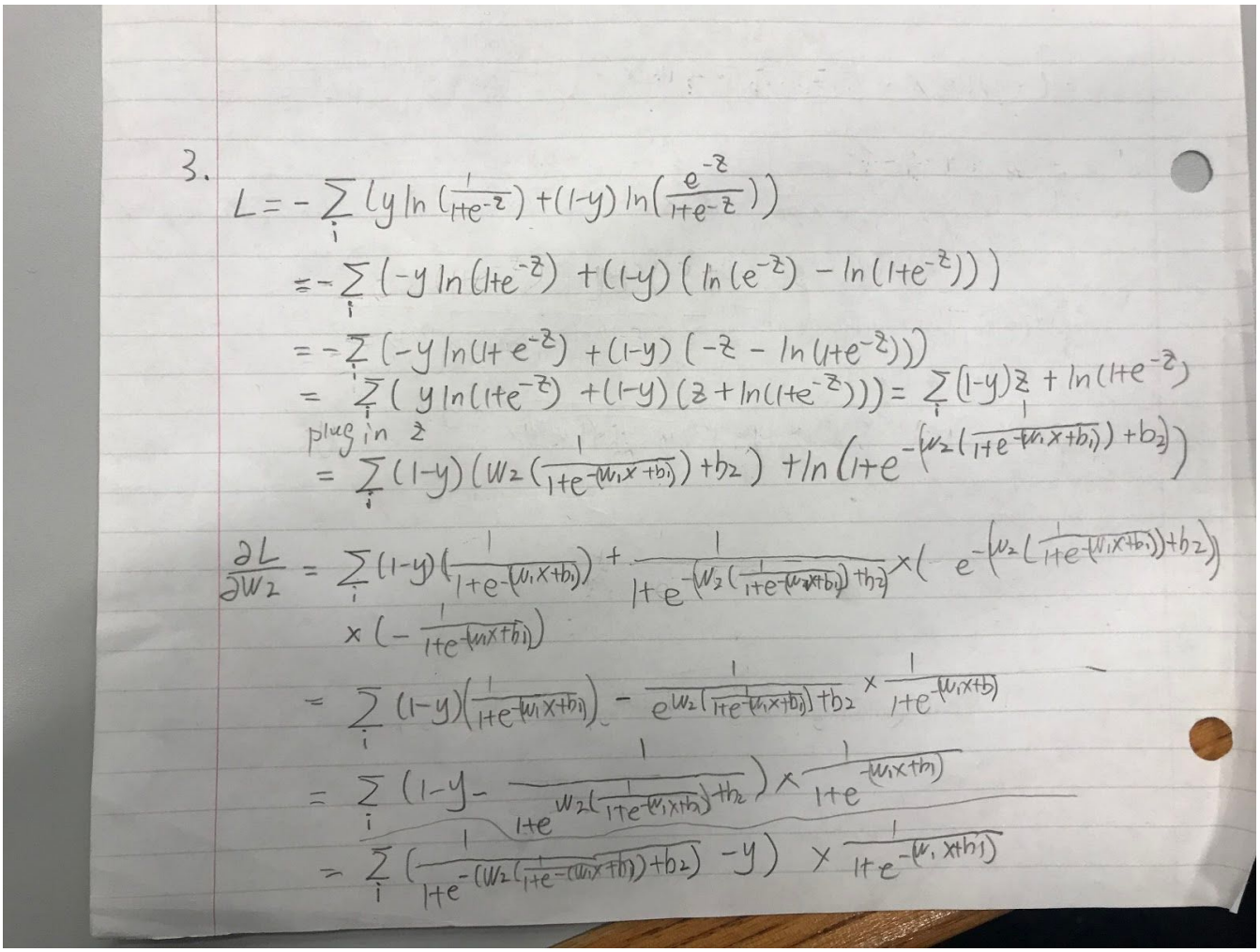
$1 \times \frac{\partial f}{\partial c} \times \frac{\partial c}{\partial a} \times \frac{\partial a}{\partial x} = -\frac{1}{25} \times 1 = -\frac{1}{25}$

$1 \times \frac{\partial f}{\partial c} \times \frac{\partial c}{\partial a} \times \frac{\partial a}{\partial y} = -\frac{1}{25} \times 0 = 0$

$$1 \times \frac{\partial f}{\partial c} \times \frac{\partial c}{\partial b} \times \frac{\partial b}{\partial z} = -\frac{1}{40} \times 1 = -\frac{1}{40}$$

$$1 \times \frac{\partial f}{\partial c} \times \frac{\partial c}{\partial b} \times \frac{\partial b}{\partial w} = -\frac{1}{40} \times 1 = -\frac{1}{40}$$

3.


3.
$L = -\sum_i \left(y \ln\left(\frac{1}{1+e^{-z}}\right) + (1-y)\ln\left(\frac{e^{-z}}{1+e^{-z}}\right)\right)$
$= -\sum_i \left(-y\ln(1+e^{-z}) + (1-y)\left(\ln(e^{-z}) - \ln(1+e^{-z})\right)\right)$
$= -\sum_i \left(-y\ln(1+e^{-z}) + (1-y)\left(-z - \ln(1+e^{-z})\right)\right)$
$= \sum_i \left(y\ln(1+e^{-z}) + (1-y)\left(z + \ln(1+e^{-z})\right)\right) = \sum_i (1-y)z + \ln(1+e^{-z})$
plug in z
$= \sum_i (1-y)\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right) + \ln\left(1+e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}\right)$
$\frac{\partial L}{\partial w_2} = \sum_i (1-y)\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right) + \frac{1}{1+e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}} \times \left(e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}\right)$
$\times \left(-\frac{1}{1+e^{-(w_1x+b_1)}}\right)$
$= \sum_i (1-y)\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right) - \frac{1}{e^{w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2}} \times \frac{1}{1+e^{-(w_1x+b_1)}}$
$= \sum_i \left(1-y-\frac{1}{1+e^{w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2}}\right) \times \frac{1}{1+e^{-(w_1x+b_1)}}$
$= \sum_i \left(\frac{1}{1+e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}} - y\right) \times \frac{1}{1+e^{-(w_1x+b_1)}}$

$$\frac{\partial L}{\partial w_1} = \sum_i (1-y) w_2 \frac{(-1)}{(1+e^{-(w_1x+b_1)})^2} \times e^{-(w_1x+b_1)} \times (-x)$$

$$+ \frac{1}{1+e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}} \times e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)} \times (-w_2) \times \frac{(-1)}{(1+e^{-(w_1x+b_1)})^2}$$

$$\times e^{-(w_1x+b_1)} \times (-x)$$

$$= \sum_i (1-y) w_2 \frac{x e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2} \frac{e^{-\left(w_2\left(\frac{1}{1+e^{-w_1x+b_1}}\right)+b_2\right)}}{e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}+1} \times \frac{w_2(-x) \cdot e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2}$$

$$= \sum_i (1-y) w_2 x \cdot \frac{e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2} - w_2 x \cdot \frac{e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2} \cdot \frac{e^{-\left(w_2\left(\frac{1}{1+e^{-w_1x+b_1}}\right)+b_2\right)}}{1+e^{+w_2\left(\frac{1}{1+e^{-w_1x+b_1}}\right)+b_2}}$$

$$= \sum_i (1-y) w_2 x \cdot \frac{1}{(1+e^{-(w_1x+b_1)})^2} - w_2 x \cdot \frac{e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2} \cdot \frac{1}{1+e^{w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}+b_2\right)}}$$

$$= \sum_i w_2 x \cdot \frac{e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2} \left(1-y - \frac{1}{1+e^{w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2}}\right)$$

$$= \sum_i w_2 x \cdot \frac{e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2} \left(\frac{1}{1+e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}} - y\right)$$

$$\frac{\partial L}{\partial b_2} = \sum_i (1-y) + \frac{1}{1+e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}} \cdot e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}+b_2\right)\right)} \cdot (-1)$$

$$= \sum_i 1 - y - \frac{e^{-w_2 \cdots}}{1+e^{-w_2 \cdots}}$$

$$= \sum_i \frac{1}{1+e^{-\left(w_2\left(\frac{1}{1+e^{-(w_1x+b_1)}}\right)+b_2\right)}} - y$$

$$\frac{\partial L}{\partial b_1} = \sum_i (1-y) w_2 \frac{(-1)}{(1+e^{-(w_1x+b_1)})^2} \cdot e^{-(w_1x+b_1)} \cdot (-1)$$

$$+ \frac{1}{1+e^{-(w_2 \cdots)}} \times e^{-(w_2 \cdots)} \times (-w_2) \times \frac{(-1)}{(1+e^{-(w_1 \cdots)})^2} \cdot e^{-(w_1x+b_1)} \cdot (-1)$$

$$= \sum_i (1-y) w_2 \frac{e^{-(w_1x+b_1)}}{(1+e^{-(w_1x+b_1)})^2} - \frac{e^{-(w_2 \cdots)}}{1+e^{-(w_2 \cdots)}} \times \frac{e^{-(w_1 \cdots)}}{(1+e^{-(w_1 \cdots)})^2} w_2$$

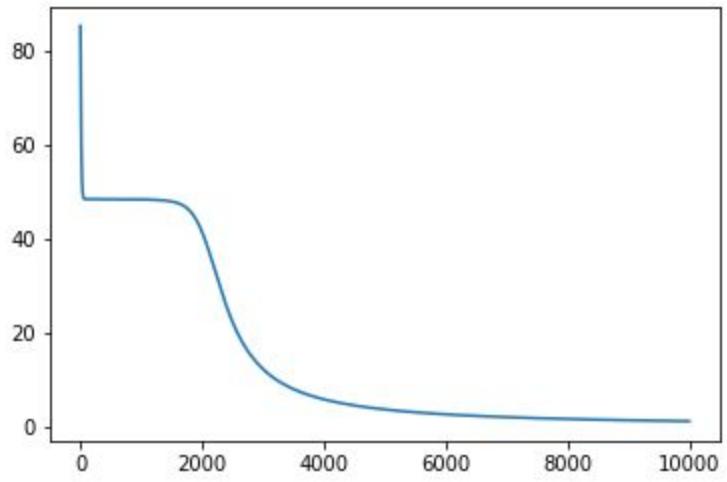
$$= \sum_i w_2 \frac{e^{-(w_1 \cdots)}}{(1+e^{-(w_1 \cdots)})^2} \left(1 - y - \frac{e^{-(w_2 \cdots)}}{1+e^{-(w_2 \cdots)}}\right)$$

$$= \sum_i w_2 \frac{e^{-(w_1 \cdots)}}{(1+e^{-(w_1 \cdots)})^2} \left(\frac{1}{1+e^{-(w_2 \cdots)}} - y\right)$$

```
import numpy as np
import matplotlib.pyplot as plt
import random

labels = ['Iris-versicolor', 'Iris-virginica']
data = np.loadtxt('Q4_data.txt',  delimiter=",",converters={ 4: lambda s: (labels.index(s))})
X = data[:,0:4]
Y = data[:,4]

trainx = np.array(X[range(15, 50)+range(65, 100),:])
testx = np.array(X[range(0, 15)+range(50, 65),:])
trainy = np.array(Y[range(15, 50)+range(65, 100)])
testy = np.array(Y[range(0, 15)+range(50, 65)])

#output 2*1
def sigmoid1(x, w1, b1):
    return 1.0/(1+np.exp(-(np.dot(w1.T,x.reshape(4,1))+b1)))

#output scaler
def sigmoid2(x, w1, w2, b1, b2):

    return (1.0/(1+np.exp(-(np.dot(w2.T, sigmoid1(x,w1,b1))+b2))))[0][0]

#output 4*2
def dw1(x, y, w1, w2, b1, b2):
    result = 0
    for i in range(len(y)):
        #2*1
        part1 = w2 * sigmoid1(x[i,:], w1, b1) * (1-sigmoid1(x[i,:], w1, b1))
        #1*4
        part2 = (sigmoid2(x[i,:], w1, w2, b1, b2)-y[i])*x[i,:].reshape(1,4)
        result += np.dot(part1, part2).T
    return result

#output 2*1
def dw2(x, y, w1, w2, b1, b2):

    result = 0
    for i in range(len(y)):
        result += (sigmoid2(x[i,:],w1,w2,b1,b2)-y[i]) *sigmoid1(x[i,:],w1,b1)
    return result

#output 2*1
```

```
def db1(x, y, w1, w2, b1, b2):
    result = 0
    for i in range(len(y)):
        part1 = w2 * sigmoid1(x[i,:], w1, b1) * (1-sigmoid1(x[i,:], w1, b1))
        part2 = sigmoid2(x[i,:], w1, w2, b1, b2)-y[i]
        result += part1*part2
    return result

#output scaler
def db2(x, y, w1, w2, b1, b2):
    result = 0
    for i in range(len(y)):
        result += sigmoid2(x[i,:], w1, w2, b1, b2)-y[i]
    return result

def train_loss(x, y, w1, w2, b1, b2):
    error = 0
    for i in range(len(y)):
        error += -(y[i]*np.log(sigmoid2(x[i,:], w1, w2, b1, b2))+(1-y[i])*np.log((1-sigmoid2(x[i,:], w1,
w2, b1, b2))))
    return error

def find_error(x, y, w1, w2, b1, b2):
    predict = []
    for i in range(0, len(y)):
        val = sigmoid2(x[i,:], w1, w2, b1, b2)
        if(val >= 0.5):
            predict.append(1)
        else:
            predict.append(0)
    error = sum(predict != y)/float(len(y))
    return error


w1 = np.random.rand(4,2)
w2 = np.random.rand(2,1)
b1 = np.random.rand(2,1)
b2 = random.random()
alpha = 0.001
errors = []

for i in range(500):
    tmp_b1 = b1-alpha*db1(trainx, trainy, w1, w2, b1, b2)
```

```
    tmp_b2 = b2-alpha*db2(trainx, trainy, w1, w2, b1, b2)
    tmp_w1 = w1-alpha*dw1(trainx, trainy, w1, w2, b1, b2)
    tmp_w2 = w2-alpha*dw2(trainx, trainy, w1, w2, b1, b2)

    w1 = tmp_w1
    w2 = tmp_w2
    b1 = tmp_b1
    b2 = tmp_b2
    errors.append(train_loss(trainx, trainy, w1, w2, b1, b2))


train_error = find_error(trainx, trainy, w1, w2, b1, b2)
test_error = find_error(testx, testy, w1, w2, b1, b2)
plt.plot([x for x in range(0,len(errors))], errors)
plt.savefig('./HW4_Q3.png')
```

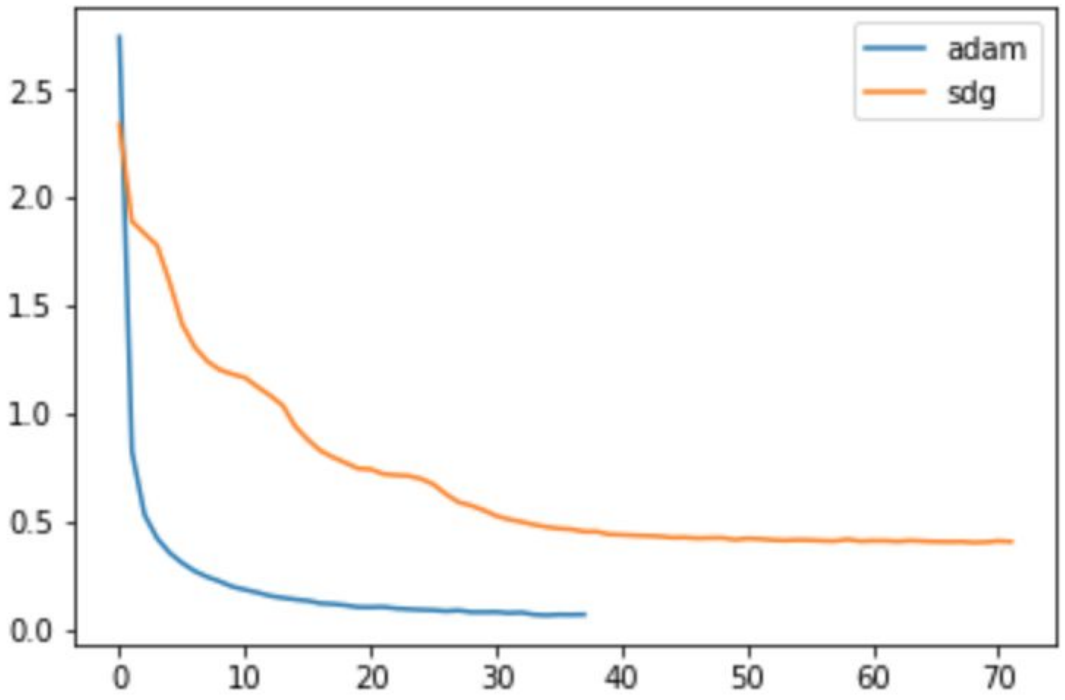


Trainning error: 0.00274
Testing error: 0.00333

4.
(1)

```
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
mnist = fetch_mldata('MNIST original',data_home='./data')
```

```
trainx, trainy = shuffle(mnist.data[:60000], mnist.target[:60000], random_state = 1)
testx, testy = shuffle(mnist.data[60000:], mnist.target[60000:])
adam = MLPClassifier(hidden_layer_sizes=(60,), solver='adam', max_iter=300)
sgd = MLPClassifier(hidden_layer_sizes=(60,), solver='sgd', max_iter=300)

adam.fit(trainx, trainy)
sgd.fit(trainx, trainy)
plt.plot(adam.loss_curve_)
plt.plot(sgd.loss_curve_)
plt.legend(['adam', 'sdg'])
plt.show()
```

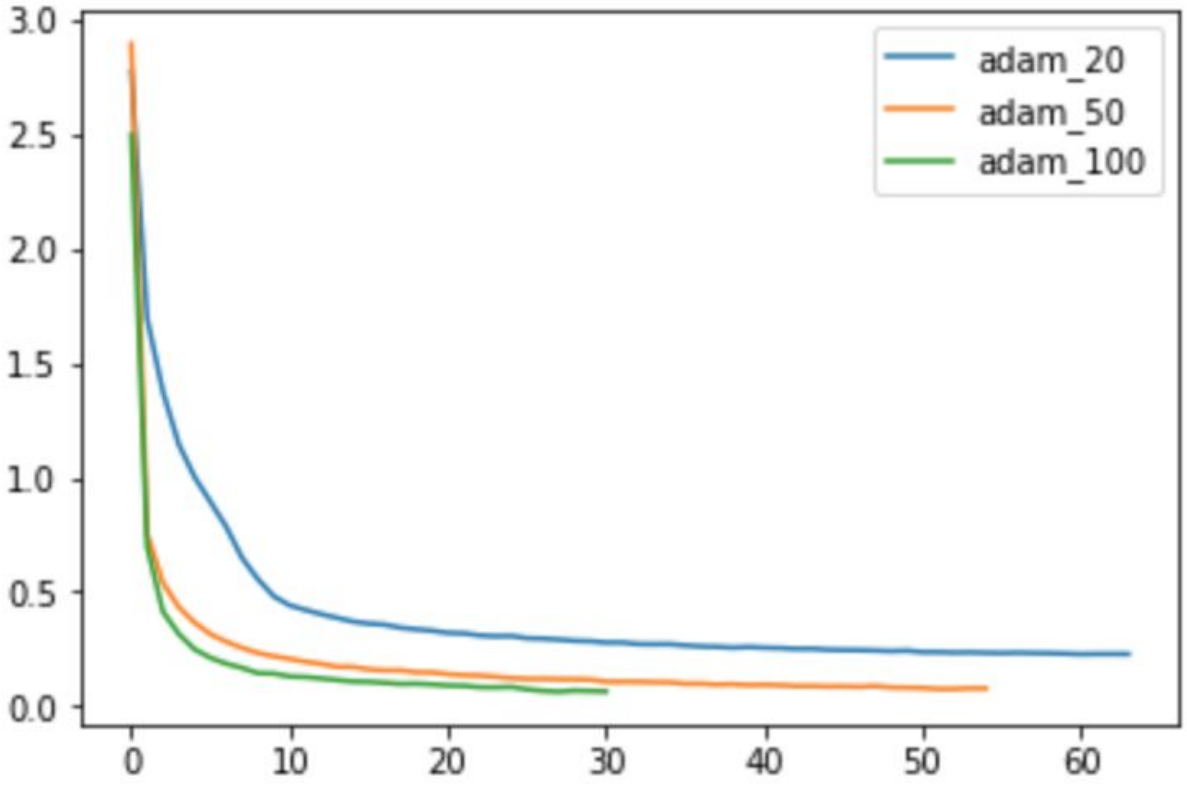


Adam solver converges faster and has lower training loss compared to sdg solver.

(2)

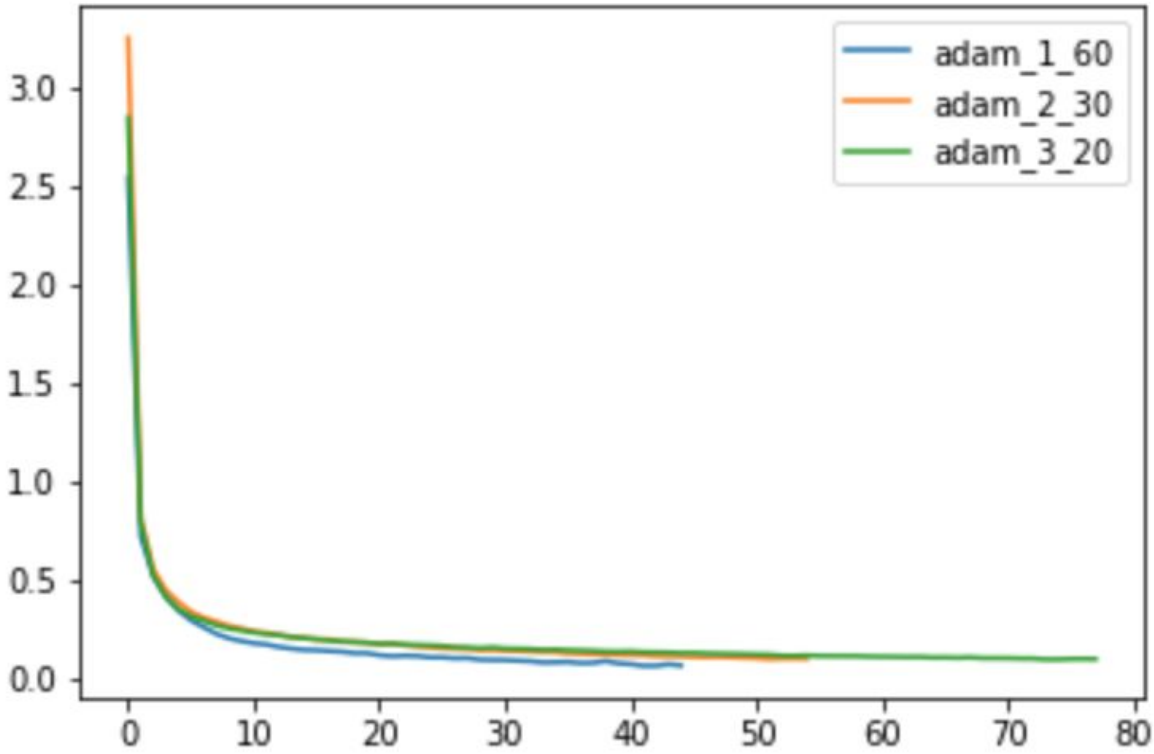
```
adam_20 = MLPClassifier(hidden_layer_sizes=(20,), solver='adam', max_iter=300)
adam_50 = MLPClassifier(hidden_layer_sizes=(50,), solver='adam', max_iter=300)
adam_100 = MLPClassifier(hidden_layer_sizes=(100,), solver='adam', max_iter=300)

adam_20.fit(trainx, trainy)
adam_50.fit(trainx, trainy)
```

```
adam_100.fit(trainx, trainy)

plt.plot(adam_20.loss_curve_)
plt.plot(adam_50.loss_curve_)
plt.plot(adam_100.loss_curve_)
plt.legend(['adam_20', 'adam_50', 'adam_100'])
plt.show()
```



More hidden unit solver has lower training loss and converge faster than solver with less hidden unit. Therefore more hidden layer solver is more accurate.

(3)



When number of hidden units are the same, solver with less hidden layer has lower training loss and converge faster.

(4)
print adam.score(testx, testy)
print sgd.score(testx, testy)
print adam_20.score(testx, testy)
print adam_50.score(testx, testy)
print adam_100.score(testx, testy)
print adam_1_60.score(testx, testy)
print adam_2_30.score(testx, testy)
print adam_3_20.score(testx, testy)

0.9566
0.877
0.9238
0.9519
0.9634
0.9589

0.948
0.9469

Adam solver with 1 layer and 100 hidden units has highest accuracy and performance(0.9589). Therefore more complex model like 2 or 3 hidden layers model may overfit training data, and they performance not very good on test data.