CS559 HW3:

Name: Yiqun Peng

Problem 1:

------- Iteration 1-------
The distance from (5.9 , 3.2) to (6.2 , 3.2) is 0.3
The distance from (5.9 , 3.2) to (6.6 , 3.7) is 0.86
The distance from (5.9 , 3.2) to (6.5 , 3.0) is 0.63
So the point (5.9 , 3.2) is belongs to red
The distance from (4.6 , 2.9) to (6.2 , 3.2) is 1.63
The distance from (4.6 , 2.9) to (6.6 , 3.7) is 2.15
The distance from (4.6 , 2.9) to (6.5 , 3.0) is 1.9
So the point (4.6 , 2.9) is belongs to red
The distance from (6.2 , 2.8) to (6.2 , 3.2) is 0.4
The distance from (6.2 , 2.8) to (6.6 , 3.7) is 0.98
The distance from (6.2 , 2.8) to (6.5 , 3.0) is 0.36
So the point (6.2 , 2.8) is belongs to blue
The distance from (4.7 , 3.2) to (6.2 , 3.2) is 1.5
The distance from (4.7 , 3.2) to (6.6 , 3.7) is 1.96
The distance from (4.7 , 3.2) to (6.5 , 3.0) is 1.81
So the point (4.7 , 3.2) is belongs to red
The distance from (5.5 , 4.2) to (6.2 , 3.2) is 1.22
The distance from (5.5 , 4.2) to (6.6 , 3.7) is 1.21
The distance from (5.5 , 4.2) to (6.5 , 3.0) is 1.56
So the point (5.5 , 4.2) is belongs to green
The distance from (5.0 , 3.0) to (6.2 , 3.2) is 1.22
The distance from (5.0 , 3.0) to (6.6 , 3.7) is 1.75
The distance from (5.0 , 3.0) to (6.5 , 3.0) is 1.5
So the point (5.0 , 3.0) is belongs to red
The distance from (4.9 , 3.1) to (6.2 , 3.2) is 1.3
The distance from (4.9 , 3.1) to (6.6 , 3.7) is 1.8
The distance from (4.9 , 3.1) to (6.5 , 3.0) is 1.6
So the point (4.9 , 3.1) is belongs to red
The distance from (6.7 , 3.1) to (6.2 , 3.2) is 0.51
The distance from (6.7 , 3.1) to (6.6 , 3.7) is 0.61
The distance from (6.7 , 3.1) to (6.5 , 3.0) is 0.22
So the point (6.7 , 3.1) is belongs to blue
The distance from (5.1 , 3.8) to (6.2 , 3.2) is 1.25
The distance from (5.1 , 3.8) to (6.6 , 3.7) is 1.5

The distance from (5.1 , 3.8) to (6.5 , 3.0) is 1.61
So the point (5.1 , 3.8) is belongs to red
The distance from (6.0 , 3.0) to (6.2 , 3.2) is 0.28
The distance from (6.0 , 3.0) to (6.6 , 3.7) is 0.92
The distance from (6.0 , 3.0) to (6.5 , 3.0) is 0.5
So the point (6.0 , 3.0) is belongs to red
red is 7
green is 1
blue is 2
u1 = [5.171, 3.171]
u2 = [5.500, 4.200]
u3 = [6.450, 2.950]
------- Iteration 2-------
The distance from (5.9 , 3.2) to (5.171 , 3.171) is 0.73
The distance from (5.9 , 3.2) to (5.5 , 4.2) is 1.08
The distance from (5.9 , 3.2) to (6.45 , 2.95) is 0.6
So the point (5.9 , 3.2) is belongs to blue
The distance from (4.6 , 2.9) to (5.171 , 3.171) is 0.63
The distance from (4.6 , 2.9) to (5.5 , 4.2) is 1.58
The distance from (4.6 , 2.9) to (6.45 , 2.95) is 1.85
So the point (4.6 , 2.9) is belongs to red
The distance from (6.2 , 2.8) to (5.171 , 3.171) is 1.09
The distance from (6.2 , 2.8) to (5.5 , 4.2) is 1.57
The distance from (6.2 , 2.8) to (6.45 , 2.95) is 0.29
So the point (6.2 , 2.8) is belongs to blue
The distance from (4.7 , 3.2) to (5.171 , 3.171) is 0.47
The distance from (4.7 , 3.2) to (5.5 , 4.2) is 1.28
The distance from (4.7 , 3.2) to (6.45 , 2.95) is 1.77
So the point (4.7 , 3.2) is belongs to red
The distance from (5.5 , 4.2) to (5.171 , 3.171) is 1.08
The distance from (5.5 , 4.2) to (5.5 , 4.2) is 0.0
The distance from (5.5 , 4.2) to (6.45 , 2.95) is 1.57
So the point (5.5 , 4.2) is belongs to green
The distance from (5.0 , 3.0) to (5.171 , 3.171) is 0.24
The distance from (5.0 , 3.0) to (5.5 , 4.2) is 1.3
The distance from (5.0 , 3.0) to (6.45 , 2.95) is 1.45
So the point (5.0 , 3.0) is belongs to red
The distance from (4.9 , 3.1) to (5.171 , 3.171) is 0.28
The distance from (4.9 , 3.1) to (5.5 , 4.2) is 1.25
The distance from (4.9 , 3.1) to (6.45 , 2.95) is 1.56
So the point (4.9 , 3.1) is belongs to red

The distance from (6.7 , 3.1) to (5.171 , 3.171) is 1.53
The distance from (6.7 , 3.1) to (5.5 , 4.2) is 1.63
The distance from (6.7 , 3.1) to (6.45 , 2.95) is 0.29
So the point (6.7 , 3.1) is belongs to blue
The distance from (5.1 , 3.8) to (5.171 , 3.171) is 0.63
The distance from (5.1 , 3.8) to (5.5 , 4.2) is 0.57
The distance from (5.1 , 3.8) to (6.45 , 2.95) is 1.6
So the point (5.1 , 3.8) is belongs to green
The distance from (6.0 , 3.0) to (5.171 , 3.171) is 0.85
The distance from (6.0 , 3.0) to (5.5 , 4.2) is 1.3
The distance from (6.0 , 3.0) to (6.45 , 2.95) is 0.45
So the point (6.0 , 3.0) is belongs to blue
red is 4
green is 2
blue is 4
u1 = [4.800, 3.050]
u2 = [5.300, 4.000]
u3 = [6.200, 3.025]
------- Iteration 3-------
The distance from (5.9 , 3.2) to (4.8 , 3.05) is 1.11
The distance from (5.9 , 3.2) to (5.3 , 4.0) is 1.0
The distance from (5.9 , 3.2) to (6.2 , 3.025) is 0.35
So the point (5.9 , 3.2) is belongs to blue
The distance from (4.6 , 2.9) to (4.8 , 3.05) is 0.25
The distance from (4.6 , 2.9) to (5.3 , 4.0) is 1.3
The distance from (4.6 , 2.9) to (6.2 , 3.025) is 1.6
So the point (4.6 , 2.9) is belongs to red
The distance from (6.2 , 2.8) to (4.8 , 3.05) is 1.42
The distance from (6.2 , 2.8) to (5.3 , 4.0) is 1.5
The distance from (6.2 , 2.8) to (6.2 , 3.025) is 0.23
So the point (6.2 , 2.8) is belongs to blue
The distance from (4.7 , 3.2) to (4.8 , 3.05) is 0.18
The distance from (4.7 , 3.2) to (5.3 , 4.0) is 1.0
The distance from (4.7 , 3.2) to (6.2 , 3.025) is 1.51
So the point (4.7 , 3.2) is belongs to red
The distance from (5.5 , 4.2) to (4.8 , 3.05) is 1.35
The distance from (5.5 , 4.2) to (5.3 , 4.0) is 0.28
The distance from (5.5 , 4.2) to (6.2 , 3.025) is 1.37
So the point (5.5 , 4.2) is belongs to green
The distance from (5.0 , 3.0) to (4.8 , 3.05) is 0.21
The distance from (5.0 , 3.0) to (5.3 , 4.0) is 1.04

The distance from (5.0 , 3.0) to (6.2 , 3.025) is 1.2
So the point (5.0 , 3.0) is belongs to red
The distance from (4.9 , 3.1) to (4.8 , 3.05) is 0.11
The distance from (4.9 , 3.1) to (5.3 , 4.0) is 0.98
The distance from (4.9 , 3.1) to (6.2 , 3.025) is 1.3
So the point (4.9 , 3.1) is belongs to red
The distance from (6.7 , 3.1) to (4.8 , 3.05) is 1.9
The distance from (6.7 , 3.1) to (5.3 , 4.0) is 1.66
The distance from (6.7 , 3.1) to (6.2 , 3.025) is 0.51
So the point (6.7 , 3.1) is belongs to blue
The distance from (5.1 , 3.8) to (4.8 , 3.05) is 0.81
The distance from (5.1 , 3.8) to (5.3 , 4.0) is 0.28
The distance from (5.1 , 3.8) to (6.2 , 3.025) is 1.35
So the point (5.1 , 3.8) is belongs to green
The distance from (6.0 , 3.0) to (4.8 , 3.05) is 1.2
The distance from (6.0 , 3.0) to (5.3 , 4.0) is 1.22
The distance from (6.0 , 3.0) to (6.2 , 3.025) is 0.2
So the point (6.0 , 3.0) is belongs to blue
red is 4
green is 2
blue is 4
u1 = [4.800, 3.050]
u2 = [5.300, 4.000]
u3 = [6.200, 3.025]

(1) The center of the first cluster (red) after one iteration is [5.171, 3.171]
(2) The center of the second cluster (green) after two iteration is [5.300, 4.000]
(3) The center of the third cluster (blue) when the clustering converges is [6.200, 3.025]
(4) There should be TWO iterations for the clusters to converge.

Problem 2:

Problem 2:

(1) $p(z) = \prod_{k=1}^{K} \pi_k^{z_k}$

$p(x|z) = \prod_{k=1}^{K} N(x|\mu_k, \Sigma_k)^{z_k}$

(2) ∵ marginal distribution over z is $p(z_k=1) = \pi_k$

∴ $p(z) = \prod_{k=1}^{K} \pi_k^{z_k}$

∴ when given a particular value of z, $p(x|z_k=1) = N(x|\mu_k, \Sigma_k)$

∵ Joint probability distribution is $p(z)p(x|z)$

∴ $p(x) = \sum_{z} p(z)p(x|z) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)$

(3) EM (expectation-maximization) algorithm.

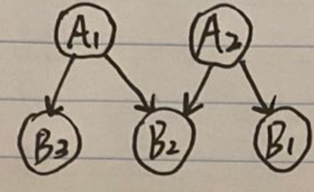
Difference: K-mean: hard assignment, each data point is associated uniquely with one cluster.

EM: soft assignment, based on the posterior probability.

Problem 3:

Problem 3:

(1)

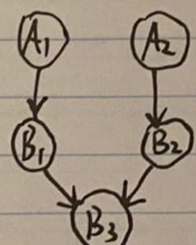


(2) $p(A_1, A_2, B_1, B_2, B_3)$

$= p(A_1) \cdot p(A_2) \cdot p(B_1 | A_2) \cdot p(B_2 | A_1, A_2) \cdot p(B_3 | A_1)$

(3) parameters $= 1 + 1 + 2 + 4 + 2 = 10$

(4) supposed : A_1 and A_2 are marginally indepent and B_1 depends on A_1, B_2 depends on A_2, B_3 depends on B_1 and B_2.



$p(A_1, A_2, B_1, B_2, B_3)$

$= p(A_1) p(A_2) p(B_1 | A_1) p(B_2 | A_2) p(B_3 | B_1, B_2)$

parameters $= 1 + 1 + 2 + 2 + 4 = 10$

Problem 3:

Question (1):

Apply one-hot encoding:

```
if data.iloc[i, 4] == 'Iris-setosa':
    labelMat.append(1)
elif data.iloc[i, 4] == 'Iris-versicolor':
    labelMat.append(2)
else:
    labelMat.append(3)
```

```
Y = pd.get_dummies(Y).values
Y = Y.astype(np.float64)
```

Split the data with 70% training data and 30% test data:

```
X_training, X_test, Y_training, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
```

Use sigmoid activation function:

```
def sigmoid(self, z):
    return 1.0 / (1 + np.exp(-z))

def sigmoid_derivative(self, z):
    return np.multiply(z, (1-z))
```

Build a neural network with one hidden layer:

```
def forward(self, X):

    self.z1 = np.dot(X, self.w1)
    self.z2 = self.sigmoid(self.z1)
    self.z3 = np.dot(self.z2, self.w2)
    o = self.sigmoid(self.z3)
```

```
def backward(self, X, Y, o):
    self.o_error = o - Y
    self.o_delta = np.multiply(self.o_error,self.sigmoid_derivative(o))
    self.z2_error = self.o_delta.dot(self.w2.T)
    self.z2_delta = np.multiply(self.z2_error, self.sigmoid_derivative(self.z2))
    self.w1 -= X.T.dot(self.z2_delta)*0.01
    self.w2 -= self.z2.T.dot(self.o_delta)*0.01
    cost = self.crossEntropy(Y, o)
    self.costs.append(cost)
```

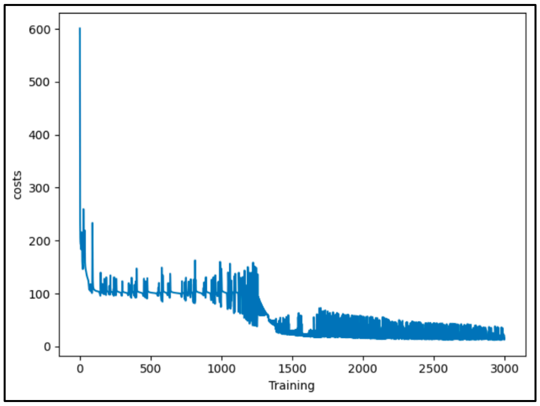
Construct loss function using cross-entropy:

```
def crossEntropy(self, Y, o):
    cost= -(np.multiply(Y, np.log(o)) + np.multiply((1 - Y), np.log(1 - o)))
    # print("cost:",cost)
    return np.sum(cost)
```

Result:

When using 6 hidden units, 0.05 learning rate and 3000 times iteration:

Loss function:

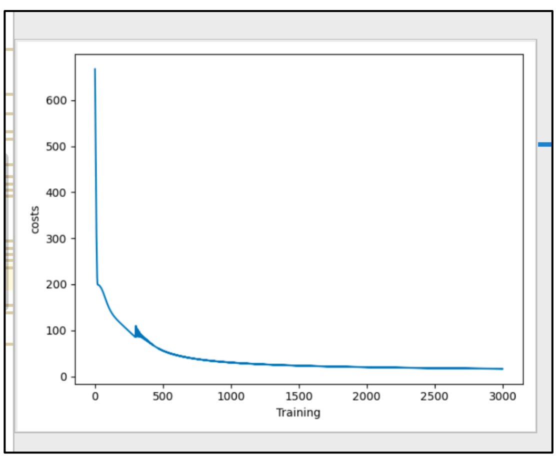


Error rate:

```
Error:  4
Error rate:  0.09
```

When using 6 hidden units, 0.01 learning rate and 3000 times iteration:

Loss function:

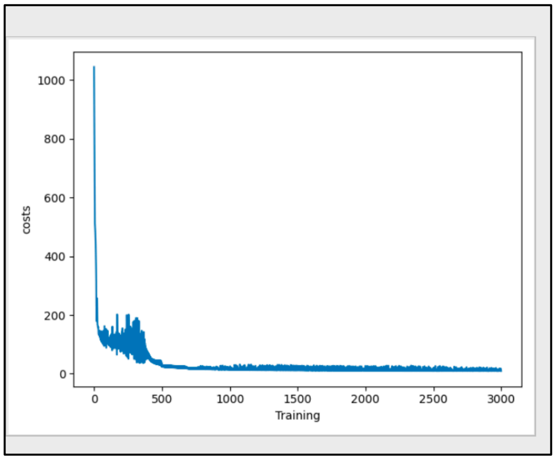


Error rate:

```
Error:  2
Error rate:  0.04
```

When using 10 hidden units, 0.05 learning rate and 3000 times iteration:

Loss function:

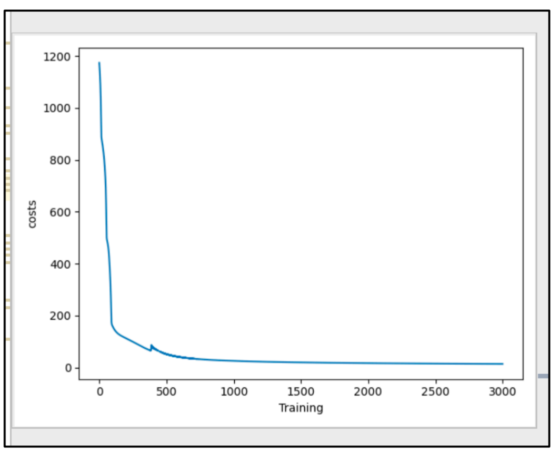


Error rate:

```
Error:  3
Error rate:  0.07
```

When using 10 hidden units, 0.01 learning rate and 3000 times iteration:

Loss function:



Error rate:

```
Error:  1
Error rate:  0.02
```

Question (2):

Use the MLPRegressor method in the sklear.neural_network package.

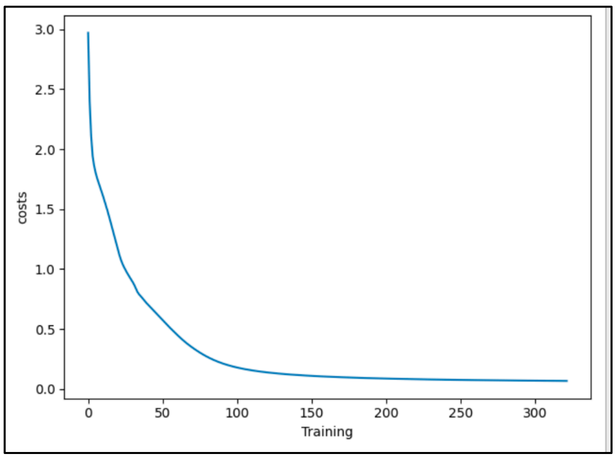
Increase the number of hidden layers to two using the ReLU activation function.

```
# print(Y_training)
mlp = MLPClassifier(solver='sgd', activation='relu', alpha=1e-4, hidden_layer_sizes=(10, 8), random_state=1
                    max_iter=3000, learning_rate_init=.01)
mlp.fit(X_training, Y_training)
Y_pred = mlp.predict(X_test)
```

Result:

When using 10 hidden units in the first hidden layer, using 8 hidden units in the second hidden layer, 0.01 learning rate and 3000 times iteration：

Loss function：



Error rate：

```
(mlp.score(X_test, Y_test))
```

```
Error rate 0.022222222222222254
```

The prediction accuracy is not much different with part (1). However, it can be found from the image that the curve is smoother, which means the training process is more accurate, saving more training time and improving performance.

The source code:

Question (1):

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

def dataProcessing(data):
    dataMat = []
    labelMat = []
    for i in range(data.iloc[:, 0].size):
        dataMat.append([1.0, float(data.iloc[i, 0]), float(data.iloc[i, 1]),
float(data.iloc[i, 2]), float(data.iloc[i, 3])])
        if data.iloc[i, 4] == 'Iris-setosa':
            labelMat.append(1)
        elif data.iloc[i, 4] == 'Iris-versicolor':
            labelMat.append(2)
        else:
            labelMat.append(3)
    return dataMat, labelMat

class Neural_Network(object):
    def __init__(self):
        self.w1 = np.random.random((5, 10))
        self.w2 = np.random.random((10, 3))
        self.costs = []
        # print(self.w1, self.w2)

    def sigmoid(self, z):
        return 1.0 / (1 + np.exp(-z))

    def sigmoid_derivative(self, z):
        return np.multiply(z, (1-z))

    def crossEntropy(self, Y, o):
        cost= -(np.multiply(Y, np.log(o)) + np.multiply((1 - Y), np.log(1 - o)))
        # print("cost:",cost)
        return np.sum(cost)

    def forward(self, X):
```

```
        self.z1 = np.dot(X, self.w1)
        self.z2 = self.sigmoid(self.z1)
        self.z3 = np.dot(self.z2, self.w2)
        o = self.sigmoid(self.z3)
        # print("----------z1------------:", self.z1)
        # print("----------z2------------:", self.z2)
        # print("----------z3------------:", self.z3)
        # print("---------- o ------------:", o)
        return o

    def backward(self, X, Y, o):
        self.o_error = o - Y
        self.o_delta = np.multiply(self.o_error,self.sigmoid_derivative(o))
        self.z2_error = self.o_delta.dot(self.w2.T)
        self.z2_delta = np.multiply(self.z2_error,
self.sigmoid_derivative(self.z2))
        self.w1 -= X.T.dot(self.z2_delta)*0.01
        self.w2 -= self.z2.T.dot(self.o_delta)*0.01
        cost = self.crossEntropy(Y, o)
        self.costs.append(cost)

    def predict(self, X):
        return  self.forward(X)


if __name__ == "__main__":
    iris = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data', header=None)
    #print(iris)
    X, Y = dataProcessing(iris)
    X = np.mat(X)
    Y = pd.get_dummies(Y).values
    Y = Y.astype(np.float64)
    # print(X)
    # print(Y)
    X_training, X_test, Y_training, Y_test = train_test_split(X, Y,
test_size=0.3, random_state=0)
    # print(X_training)
    # print(Y_training)
    NN = Neural_Network()
```

```python
for i in range(3000):
    o = NN.forward(X_training)
    NN.backward(X_training, Y_training, o)
print(NN.costs)
plt.plot(NN.costs)
plt.xlabel('Training')
plt.ylabel('costs')
plt.show()

pre = NN.predict(X_test)
pre = np.argmax(pre, axis=1)
error = 0
for i in range(Y_test[:, 0].size):
    index = pre[i, 0]
    if Y_test[i, index] == 0:
        error += 1
accuracy =  (Y_test[:, 0].size - error)/Y_test[:, 0].size
print(pre)
print(Y_test)
print("Error: ",error)
print("Error rate: ",np.round(1-accuracy, 2))
```

Question (2):

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor,MLPClassifier
from sklearn.preprocessing import StandardScaler

def dataProcessing(data):
    dataMat = []
    labelMat = []
    for i in range(data.iloc[:, 0].size):
        dataMat.append([1.0, float(data.iloc[i, 0]), float(data.iloc[i, 1]),
float(data.iloc[i, 2]), float(data.iloc[i, 3])])
        if data.iloc[i, 4] == 'Iris-setosa':
            labelMat.append(1)
        elif data.iloc[i, 4] == 'Iris-versicolor':
            labelMat.append(2)
        else:
            labelMat.append(3)
    return dataMat, labelMat


if __name__ == "__main__":
    iris = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data', header=None)
    #print(iris)
    X, Y = dataProcessing(iris)
    X = np.mat(X)
    Y = pd.get_dummies(Y).values
    Y = Y.astype(np.float64)
    # print(X)
    # print(Y)
    X_training, X_test, Y_training, Y_test = train_test_split(X, Y,
test_size=0.3, random_state=0)
    # print(X_training)
    # print(Y_training)
    mlp = MLPClassifier(solver='sgd', activation='relu', alpha=1e-4,
hidden_layer_sizes=(10, 8), random_state=1,
                        max_iter=3000, learning_rate_init=.01)
    mlp.fit(X_training, Y_training)
```

```
Y_pred = mlp.predict(X_test)
# print(Y_pred)
# print("---------")
# print(Y_test)

print("Error rate", 1-mlp.score(X_test, Y_test))
print(mlp.n_layers_)
print(mlp.n_iter_)
print(mlp.loss_curve_)
plt.plot(mlp.loss_curve_)
plt.xlabel('Training')
plt.ylabel('costs')
plt.show()
```