Homework 3 : Problem 4.2

The batch mode backpropagation algorithm of Section 4.6 to train a two-layer perceptron with two hidden neurons and one neuron in the output.

Python Code:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 13 08:18:11 2018
@author: Nazneen Kotwal
"""
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import numpy as np
import os

path = "C:\\Users\\nazne\\OneDrive\\Documents\\1_ECE 759 Pattern Recognition\\Homework\\Homework3"
print ("The current working directory is", os.getcwd())
os.chdir(path)

################################################################################

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

def s_prime(z):
    return np.multiply((z), (1.0-z))

def init_weights(layers, epsilon):
    weights = []
    for i in range(len(layers)-1):
        w = np.random.rand(layers[i+1], layers[i]+1)
        w = w * 2*epsilon - epsilon
        weights.append(np.mat(w))
    return weights

def cost(X,w,y):
    step_1 = sigmoid(np.dot(X, w[0]))
    step_2 = np.dot(step_1,np.transpose(w[1]))
    step_3 = sigmoid(step_2)
    step_4 = np.zeros([400,1])
    step_4 = (np.reshape(y,(-1,1)) - np.reshape(step_3,(-1,1)))
    J = (1/2)*(np.sum(x**2 for x in step_4))
    return(J)

def fit(X, Y, w, predict=False, x=None):
    w_grad = ([np.mat(np.zeros(np.shape(w[i]))) for i in range(len(w))])
    for i in range(len(X)):
        x = x if predict else X[i]
        y = Y[0,i]
        # forward propagate
        a = x
        a_s = []
        for j in range(len(w)):
            a = np.mat(np.append(1, a)).T
            a_s.append(a)
            z = w[j] * a
            a = sigmoid(z)
        if predict: return a
        # backpropagate
        delta = a - y.T
        w_grad[-1] += delta * a_s[-1].T
        for j in reversed(range(1, len(w))):
            delta = np.multiply(w[j].T*delta, s_prime(a_s[j]))
            w_grad[j-1] += (delta[1:] * a_s[j-1].T)
    return [w_grad[i]/len(X) for i in range(len(w))]

def predict(x):
    return fit(X, Y, w, True, x)

###############################################################################
# Creating a Data Set for Training and Testing
mean1 = (0,0)
mean2 = (1,1)
mean3 = (1,0)
mean4 = (0,1)
cov = ((0.001, 0),(0, 0.001))
sample1 = np.random.multivariate_normal(mean1, cov, 100)
sample2 = np.random.multivariate_normal(mean2, cov, 100)
sample3 = np.random.multivariate_normal(mean3, cov, 100)
sample4 = np.random.multivariate_normal(mean4, cov, 100)
test1 = np.random.multivariate_normal(mean1, cov, 50)
test2 = np.random.multivariate_normal(mean2, cov, 50)
test3 = np.random.multivariate_normal(mean3, cov, 50)
test4 = np.random.multivariate_normal(mean4, cov, 50)
z1 = np.zeros([100,1])
z2 = np.ones([100,1])

# Assigning Classes to the Data Set Created
a = np.hstack((z2[:100,],sample1,z1[:100,]))
b = np.hstack((z2[:100,],sample2,z1[:100,]))
c = np.hstack((z2[:100,],sample3,z2[:100,]))
d = np.hstack((z2[:100,],sample4,z2[:100,]))

e = np.hstack((test1,z1[:50,]))
f = np.hstack((test2,z1[:50,]))
g = np.hstack((test3,z2[:50,]))
h = np.hstack((test4,z2[:50,]))

X = np.concatenate((a,c,d,b))

cmap_bold = ListedColormap(['#FF0000', '#00FF00'])
plt.scatter(X[:,1],X[:,2], c=X[:,3], cmap=cmap_bold,
             edgecolor='k', s=20)
plt.title("Visualization of the Data Points")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()

Jlist_store = []
Y = np.reshape(X[:,3],(1,-1))
X1 = X[:,1:3]
layers = [2,2,1]
epochs = 6000
alpha = 0.5
w = init_weights(layers, 1)

for i in range(epochs):
    w_grad = fit(X1, Y, w)
    Jlist= cost(X1,w_grad,Y)
    Jlist_store.append(Jlist)
#    print (w_grad)
    for j in range(len(w)):
        w[j] -= alpha * w_grad[j]

test_data = np.concatenate((e,f,g,h))
```

```
Y_test = test_data[:,2]
test_data = test_data[:,0:2]

pred = []
count = 0
for i in range(len(test_data)):
    x = test_data[i]
    guess = predict(x)
    print( x, ":", guess)
    if guess >= 0.49:
        pred.append(1)
    else:
        pred.append(0)


for i in range(200):
    if pred[i] != Y_test[i]:
        count += 1

accuracy = (200-count)/200
percerr=(1-accuracy)*100
print("Classification Error is "+str(percerr)+" %")
#
t = np.arange(0, len(Jlist_store), 1)
plt.plot(np.reshape(t,(-1,1)),np.reshape(np.array(Jlist_store),(-1,1)))
plt.title("Error Curve as function of iterations")
plt.xlabel("Iterations")
plt.ylabel("Error")
plt.show()
```

Results:

Note:

Theta1 Values Obtained [w0 w1 w2]:

- Node 1: -8.18597e-05 0.000169448 0.000555113
- Node 2: -9.01732e-05 0.000179444 -0.000580298

Theta2 Values Obtained [w0 w1 w2]:

- Node1: -0.000427147 0.000429502 0.00042732

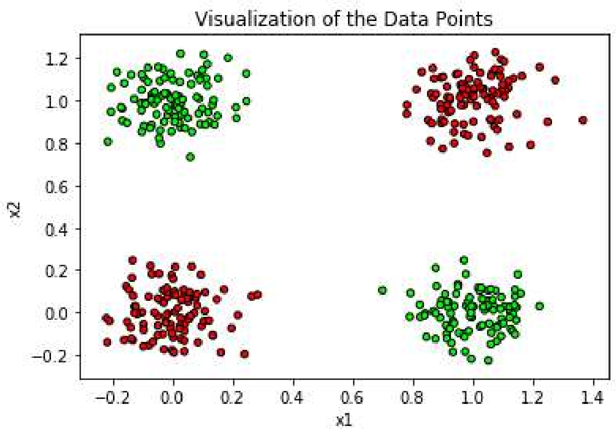
Note:

Classification Error is: 0%

Plots:

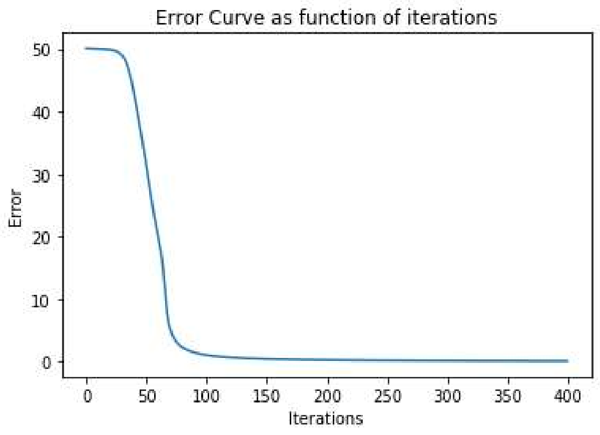
Note:

Data Points Visulaization:



Note:

Error Vs Iteration Plot:


Error Curve as function of iterations
Error
Iterations