

Homework-5

Code for home grown fully connected multi-layer backpropagation network:

```
import numpy as np
from random import *
def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))
def sigmoid_prime(x):
    return sigmoid(x)*(1.0-sigmoid(x))

def compute_cost(A, Y):
    cost = np.multiply(np.log(A), Y) + np.multiply((1 - Y), np.log(1 - A))
    cost = -1 * cost
    cost = np.squeeze(cost)
    return cost

def CrossEntropy(yhat, y):
    # if y == 1:
    #     return -np.log(yHat)+dummy
    # else:
    #     return np.log(1 - yHat)
    res = -y/yhat + (1-y)/(1-yhat)
    if abs(res) < 0.1:
        return 0
    else:
        return -1*res
class NeuralNetwork:

    def __init__(self, layers, activation='sigmoid'):
        if activation == 'sigmoid':
            self.activation = sigmoid
            self.activation_prime = sigmoid_prime
            self.cost = compute_cost
            self.ce = CrossEntropy
        self.weights = []
        # layers = [2,2,1]
        # range of weight values (-1,1)
        # input and hidden layers - random((2+1, 2+1)) : 3 x 3
        for i in range(1, len(layers) - 1):
            r = 2*np.random.random((layers[i-1] + 1, layers[i] + 1)) - 1
            self.weights.append(r)
        # output layer - random((2+1, 1)) : 3 x 1
        r = 2*np.random.random( (layers[i] + 1, layers[i+1])) - 1
        self.weights.append(r)

    def fit(self, X, y, learning_rate=0.2, epochs=100000):
        # Add column of ones to X
```

```

# This is to add the bias unit to the input layer
ones = np.atleast_2d(np.ones(X.shape[0]))
X = np.concatenate((ones.T, X), axis=1)
arr = []

for k in range(epochs):
    if k % 10000 == 0: print('epochs:', k)

    i = np.random.randint(X.shape[0])
    a = [X[i]]
    arr=[X[i]]

    for l in range(len(self.weights)):
        dot_value = np.dot(a[l], self.weights[l])
        arr.append(dot_value)
        activation = self.activation(dot_value)
        a.append(activation)

    # output layer
    error = y[i] - a[-1]
    #
    error = self.cost(a[-1], y[i])
    error = self.ce(a[-1], y[i])
    deltas = [error * self.activation_prime(arr[-1])]

    # we need to begin at the second to last layer
    # (a layer before the output layer)
    for l in range(len(a) - 2, 0, -1):
        deltas.append(deltas[-
1].dot(self.weights[l].T)*self.activation_prime(arr[l]))

    # reverse
    # [level3(output)->level2(hidden)] => [level2(hidden)-
>level3(output)]
    deltas.reverse()

    # backpropagation
    # 1. Multiply its output delta and input activation
    #    to get the gradient of the weight.
    # 2. Subtract a ratio (percentage) of the gradient from the
weight.

    for i in range(len(self.weights)):
        layer = np.atleast_2d(a[i])
        delta = np.atleast_2d(deltas[i])
        self.weights[i] += learning_rate * layer.T.dot(delta)

#
    print(np.max(arr))

def predict(self, x):
    a = np.concatenate((np.ones(1).T, np.array(x)))

```

```

        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a

nn = NeuralNetwork([2,3,3,1])
inp_class1=[[1,0],[-1,0]]
inp_class2=[[0,1],[0,-1]]
out_class1=[0,0]
out_class2=[1,1]
for i in range(0,48):
    x=uniform(1,10)
    y=uniform(-1,0)
    a=uniform(-1,0)
    b=uniform(1,10)
    foo=[x,y]
    foo1=[a,b]
    inp_class1.append(foo)
    out_class1.append(0)
    inp_class2.append(foo1)
    out_class2.append(1)
inp=inp_class1+inp_class2
out=out_class1+out_class2
X = np.array(inp)
y = np.array(out)
nn.fit(X, y)
for e in X:
    print(e,nn.predict(e))
print("The weights are:")
print(nn.weights)

```

1.100% classification was achieved with as much as just 1 hidden layer. Subsequent hidden layers also performed the same. So using a single hidden layer was enough to obtain 100% classification. Three hidden units per layer were used for all layers.

2.The weights that were learned after the final epoch to achieve 100% classification using 1 hidden layer was:

```

The weights are:
[array([[ 3.4567762, -3.76820817,  3.25401411,  4.73080111],
       [ 0.45141204,  0.34519883, -0.39284026, -4.9277709 ],
       [ 6.40775739,  5.61454257, -4.21068074, -0.13569049]]), array([[ -8.98417619],
       [ 8.08000711],
       [-5.40810615],
       [ 9.76937696]])]

```

Keeping learning rate=0.2 and epochs=100000, the sigmoidal slope parameter(beta) was varied.

Beta	Accuracy
0.1	98%
0.3	98%
0.5	98%
1	100%

Keeping Beta = 1 and epochs =100000, learning rate(alpha) was varied.

Alpha	Accuracy
0.01	98%
0.1	98%
0.15	98%
0.2	100%

Keeping Beta=1 and learning rate(alpha)=0.2, epochs were varied.

Epochs	Accuracy
10000	98%
30000	98%
50000	98%
100000	100%

Therefore the optimal solution to obtain 100% classification was

Beta	Alpha	Epochs
1	0.2	100000

Output for the above code(only a part of it is shown):

```
[8] [ 7.84672949 -0.2218682 ] [4.63997329e-06]
[ 5.31465077 -0.84364805] [1.96764765e-05]
[ 1.85591579 -0.54026318] [1.03071307e-05]
[ 6.58052889 -0.74334816] [2.71780168e-06]
[ 7.80073537 -0.84482825] [3.08290052e-06]
[ 4.35377483 -0.28948391] [1.16515362e-06]
[ 1.45206267 -0.50223365] [1.88999869e-05]
[ 9.53905478 -0.40099428] [3.40615343e-06]
[ 2.10334999 -0.58693881] [1.35470832e-05]
[ 8.37996484 -0.48879266] [1.51089786e-06]
[ 2.09764527 -0.69462823] [5.99832457e-05]
[ 5.52647269 -0.66497192] [2.61479888e-06]
[ 8.17183886 -0.00996736] [6.58089875e-05]
[ 3.45512512 -0.52689797] [2.72170209e-06]
[ 7.28530035 -0.93254827] [1.04859894e-05]
[ 4.85177646 -0.21365049] [1.42231223e-06]
[0. 1.] [0.99790404]
[ 0. -1.] [0.97904027]
[-0.72626066  4.63959754] [0.99985816]
[-0.58546859  4.82261615] [0.99985749]
[-0.4042974   9.98880891] [0.99985225]
[-0.53336203  7.28109903] [0.99985643]
[-0.05008048  6.54634843] [0.99983415]
[-0.82306721  7.8216226 ] [0.99985819]
[-0.85490131  9.96288139] [0.9998581 ]
[-0.49226767  7.82965446] [0.99985566]
[-0.68305105  3.39424357] [0.99985812]
[-0.46823578  6.73582352] [0.99985576]
[-0.13462379  1.76852149] [0.99983452]
[-0.02009948  1.27750274] [0.99964062]
[-0.68202342  6.31822764] [0.9998578 ]
[-0.8093414   9.31713   ] [0.999858 ]
[-0.94250713  2.7284743 ] [0.99985832]
[-0.30556158  8.39185414] [0.99985017]
[-0.65646939  7.77703923] [0.99985742]
[-0.67134482  5.84616758] [0.99985782]
[-0.31923197  8.85118528] [0.99985022]
[-0.7182241   5.71036301] [0.99985803]
[-0.38216996  3.3069493 ] [0.99985586]
[-0.82916203  8.46642871] [0.99985816]
[-0.86063110  4.46603600] [0.99985810]
```

I/P patterns

Predicted class

Class 1 ~ 0

Class 2 ~ 1

Input Patterns are on the left side with their predicted class labels. Here we attribute values approximately equal to 0 as belonging to Class 1 and values approximately equal to 1 as belonging to Class 2.

Code for the same implementation using Keras/Tensorflow:

```
import numpy as np
from random import *
from keras.models import Sequential
from keras.layers.core import Dense
inp_class1=[[1,0],[-1,0]]
inp_class2=[[0,1],[0,-1]]
out_class1=[0,0]
out_class2=[1,1]
for i in range(0,48):
    x=uniform(1,10)
    y=uniform(-1,0)
    a=uniform(-1,0)
    b=uniform(1,10)
    foo=[x,y]
    foo1=[a,b]
    inp_class1.append(foo)
    out_class1.append(0)
    inp_class2.append(foo1)
    out_class2.append(1)
inp=inp_class1+inp_class2
out=out_class1+out_class2
X = np.array(inp)
y = np.array(out)
training_data = np.array(X, "float32")
target_data = np.array(y, "float32")

model = Sequential()
model.add(Dense(3, input_dim=2, activation='sigmoid'))
model.add(Dense(3, input_dim=2, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['binary_accuracy'])

model.fit(training_data, target_data, nb_epoch=500, verbose=2)

print(model.predict(training_data).round())
```

In comparison to the homegrown neural network, the keras implementation achieved 100% classification at epochs=500, learning rate=0.2.