HomeWork-6

By:
Monil Shah
mds747

Epoch=1, alpha=0.2,layers = 2-4-1

In [120]:

```
import numpy as np
def tanh(x):
    return np.tanh(x)
def tanh_prime(x):
    return 1.0 - x**2
```

In [121]:

```
class NeuralNetwork:
    def __init__(self, layers, activation='tanh'):
            self.activation = tanh
            self.activation_prime = tanh_prime
            self.weights = []
            for i in range(1, len(layers) - 1):
#                  r = np.random.uniform(-0.5,0.5,(layers[i-1]+1 , layers[i

                 r=np.asarray([[0.19347555,0.31675476,-0.144748,0.36374521],
                    [ 0.30686735,0.18845288,-0.03301594,-0.48859019],
                  [-0.32243413,0.26504268,0.27330512,-0.32503622]])
                 print(r)
                 self.weights.append(r)
#                  r = np.random.uniform(-0.5,0.5, (layers[i]+1, layers[i+1

                 r=np.asarray([[-0.1401],[0.4919],[-0.2913],[-0.3979]])
                 print(r)
                 self.weights.append(r)
#                 print("Initial weight:"+'\n',self.weights[0])
#                 print("Initial Bias:"+'\n',self.weights[1])

    def fit(self, X, y, learning_rate=0.2, epochs=1):
        ones = np.atleast_2d(np.ones(X.shape[0]))
        X = np.concatenate((ones.T, X), axis=1)
#         X=np.concatenate((X,ones.T),axis=1)
        print(X)
        for k in range(epochs):
            for i in range(4):
                a = [X[i]]
                print()
                for l in range(len(self.weights)):
                        print("values",a[l],self.weights[l])
                       # dot_value = np.dot(a[l].T, self.weights[l])
                        dot_value = np.dot(a[l], self.weights[l])
                        activation = self.activation(dot_value)
                        a.append(activation)
                error = y[i] - a[-1]
                deltas = [error * self.activation_prime(a[-1])]
                for l in range(len(a) - 2, 0, -1):
                    deltas.append(deltas[-1].dot(self.weights[l].T)*self.act
                deltas.reverse()
                for j in range(len(self.weights)):
                    layer = np.atleast_2d(a[j])
                    delta = np.atleast_2d(deltas[j])
                    self.weights[j] += learning_rate * layer.T.dot(delta)

        print("===============Ans=================================")
        print('epochs:', epochs)
        print('error:',error)
        print('weight:'+'\n',self.weights[0])
        print('bias:'+'\n',self.weights[1])




    def predict(self, x):
        a = np.hstack((np.ones(1).T, np.array(x)))
        #a=np.array(x)
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
```

In [122]:

```python
if __name__ == '__main__':
    nn = NeuralNetwork([2,4,1])
    X = np.array([[1, 1],
                  [1, -1],
                  [-1, 1],
                  [-1, -1]])
    y = np.array([-1, 1, 1, -1])
    nn.fit(X, y)
    print("========================================================")
    print("Results:")
    for e in X:
        print(e,nn.predict(e))
```

```
[[ 0.19347555  0.31675476 -0.144748    0.36374521]
 [ 0.30686735  0.18845288 -0.03301594 -0.48859019]
 [-0.32243413  0.26504268  0.27330512 -0.32503622]]
[[-0.1401]
 [ 0.4919]
 [-0.2913]
 [-0.3979]]
[[ 1.  1.  1.]
 [ 1.  1. -1.]
 [ 1. -1.  1.]
 [ 1. -1. -1.]]

values [ 1.  1.  1.] [[ 0.19347555  0.31675476 -0.144748    0.36374521]
 [ 0.30686735  0.18845288 -0.03301594 -0.48859019]
 [-0.32243413  0.26504268  0.27330512 -0.32503622]]
values [ 0.17605521  0.64707498  0.09525153 -0.42180135] [[-0.1401]
 [ 0.4919]
 [-0.2913]
 [-0.3979]]

values [ 1.  1. -1.] [[ 0.22533742  0.24964589 -0.07700108  0.44051622]
 [ 0.33872922  0.12134401  0.03473098 -0.41181918]
 [-0.29057226  0.19793381  0.34105204 -0.24826521]]
values [ 0.69348518  0.17134895 -0.36558908  0.27009125] [[-0.18141961]
 [ 0.34003352]
 [-0.31365524]
 [-0.29890454]]

values [ 1. -1.  1.] [[ 0.20589242  0.31779663 -0.13311029  0.38329898]
 [ 0.31928422  0.18949475 -0.02137823 -0.46903642]
 [-0.27112726  0.12978306  0.39716124 -0.19104797]]
values [-0.36662556  0.25250345  0.27792243  0.57921955] [[-0.03822454]
 [ 0.3754147 ]
 [-0.38914431]
 [-0.24313444]]

values [ 1. -1. -1.] [[ 0.19849976  0.39632878 -0.2133422   0.34720019]
 [ 0.32667687  0.11096259  0.05885368 -0.43293763]
 [-0.27851991  0.20831522  0.31692933 -0.22714676]]
values [ 0.14922022  0.07689885 -0.52926616  0.76463656] [[-0.12014094]
 [ 0.43183241]
 [-0.32704716]
 [-0.11371745]]
================Ans=================================
epochs: 1
error: [-1.1010761]
weight:
 [[ 0.22410325  0.30276085 -0.16202577  0.35749478]
 [ 0.30107339  0.20453053  0.00753725 -0.44323221]
 [-0.30412339  0.30188315  0.26561289 -0.23744135]]
bias:
 [[-0.15266579]
 [ 0.41507112]
 [-0.21168544]
 [-0.28038177]]
=======================================================
Results:
[1 1] [ 0.29924349]
[ 1 -1] [ 0.02207902]
[-1  1] [ 0.04984269]
[-1 -1] [-0.24366162]
```

```
In [ ]:
```

```
In [ ]:
```