JHU_NeuralNet

February 20, 2022

[6]:

```
#Johns Hopkins University Neural Network Code
# Written by: Lucas Buccafusca 2/24/2022
# With guidance from 'A Neural Network in 11 lines of Python' from iamtrask,␣
 ↪'What is a Neural Network?' by 3blue1brown and 'Neural networks from scratch␣
 ↪in Python' by Cristian Dima

from joblib.numpy_pickle_utils import xrange
import numpy as np

#NETWORK SIZE
#INPUT LAYER=3
#HIDDEN LAYER=4
#OUTPUT LAYER=1

def sigmoid(x): #Sigmoid Activation Function
        return 1/(1+np.exp(-x))

def sigmoid_deriv(x): #Derivative of Sigmoid function
        return x*(1-x)

def arctan(x): #Arctan Activation Function
        return numpy.arctan(x)

def arctan_deriv(x): #Derivative of Arctan function
        return 1/(1+x^2)

def ReLU(x): #ReLU Activation Function
        return max(0.0, x)

def ReLU_deriv(x): #Derivative of ReLU function
        return 1*(x>0)

X = np.array([[0,0,1],
            [0,1,1],
            [1,0,1],
            [1,1,1]]) #Half Dataset for Training
```

```
y = np.array([[0],
                        [1],
                        [1],
                        [0]])

np.random.seed(92) #This is a good performing random seed. In general, make␣
 ↪sure to fix your random seed in some fashion to ensure code works properly␣
 ↪first!

# randomly initialize our weights with mean 0
weights_0 = 2*np.random.random((3,4)) - 1
weights_1 = 2*np.random.random((4,1)) - 1

#############################
# TRAINING #
#############################
for j in xrange(60000): #60,000 training iterations

        # Feed forward through neural network
    layer_0 = X
    layer_1 = sigmoid(np.dot(layer_0,weights_0))
    layer_2 = sigmoid(np.dot(layer_1,weights_1))

    # How much did we miss the target value?
    layer_2_true_error = 0.5*np.sum(np.power((y-layer_2),2)) #L2 norm error
    layer_2_error = y - layer_2 #Partial derivative of L2 norm error (for␣
 ↪backpropagation)

    if (j% 5000) == 0:
        print ("Error:" + str(layer_2_true_error)) #Print error every 5000␣
 ↪iterations
    # In what direction is the target value?
    layer_2_delta = layer_2_error*sigmoid_deriv(layer_2)

    # How much did each layer_1 value contribute to the layer_2 error?
    layer_1_error = layer_2_delta.dot(weights_1.T)

    # In what direction is the target layer_1?
    layer_1_delta = layer_1_error * sigmoid_deriv(layer_1)

    # Update the weights in the neural network in the direction of gradient␣
 ↪descent
    weights_1 += layer_1.T.dot(layer_2_delta)
    weights_0 += layer_0.T.dot(layer_1_delta)


#############################
```

```python
# TESTING #
##############################
X_new=np.array([[0,0,0],
            [0,1,0],
            [1,0,0],
            [1,1,0],[0,0,1],
            [0,1,1],
            [1,0,1],
            [1,1,1]]) #Full dataset for testing, normally is a different set of␣
 ↪data than used for training
#Feedforward using the new weights
layer_0 = X_new
layer_1 = sigmoid(np.dot(layer_0,weights_0))
layer_2 = sigmoid(np.dot(layer_1,weights_1))
print ("Output of Full Dataset After Training:")
print (layer_2)
```

```
Error:0.4989274862080605
Error:0.0005404889275701138
Error:0.0002172190601097974
Error:0.0001303239755950017
Error:9.145419026283027e-05
Error:6.977940335659718e-05
Error:5.6083696997266e-05
Error:4.6700949534547984e-05
Error:3.989816268325927e-05
Error:3.4754648194137755e-05
Error:3.073801064592162e-05
Error:2.751988455571167e-05
Output of Full Dataset After Training:
[[0.32379775]
 [0.9269275 ]
 [0.92116144]
 [0.00229311]
 [0.0018005 ]
 [0.996365  ]
 [0.99632265]
 [0.00444932]]
```

[ ]: