```
import numpy as np


#layer sizes
def initialize(layer_count):
    layer_dict={}
    for i in range(layer_count+1):
        print('\nEnter of nodes in layer',i,': ',end=' ')
        layer_dict['l'+str(i)]=int(input())
    return layer_dict


#parameter matrix
def param(layer_dict,layer_count):
    parameters={}
    for i in range(layer_count):
        W=np.ones([layer_dict['l'+str(i+1)],layer_dict['l'+str(i)]])
        b=np.zeros([layer_dict['l'+str(i+1)],1])
        parameters['W'+str(i+1)]=W
        parameters['b'+str(i+1)]=b
    return parameters


#activation function
def activation(X):
    return 5*X

#linear transformation function
def transform(weight,bias,x):
    return (np.dot(weight,x)+bias)


#Forward propogation
def forward_prop(X,parameters,layer_count):
    cache={}
    cache['A0']=X
    for i in range(layer_count):
        Z=transform(parameters['W'+str(i+1)],parameters['b'+str(i+1)],cache['A'+st
        A=activation(Z)
        cache['Z'+str(i+1)]=Z
        cache['A'+str(i+1)]=A
    return cache['A2'],cache


def cal_cost(final_output,target_labels):
    cost=np.square(np.subtract(target_labels,final_output)).mean()
    np.squeeze(cost)
    return cost


def back_prop(cache,parameters,X,Y,layer_count):
    m=cache['A0'].shape[1]
    updates={}
    dZ=5
    for i in reversed(range(1,1+layer_count)):
        if i==layer_count:
```

```
        if i==layer_count:
            dA=cache['A'+str(i)]-Y
        else:
            dA=dZ*np.multiply(dA.T,parameters['W'+str(i+1)])
        dW=(dZ*np.multiply(dA,cache['A'+str(i-1)]))/m
        db=(np.sum(dA*dZ))/m
        updates['dW'+str(i)]=dW
        updates['db'+str(i)]=db

    return updates


def update_param(updates,parameters,learning_rate,layer_count):
    new_parameters={}
    for i in range(layer_count):
        new_parameters['W'+str(i+1)]=parameters['W'+str(i+1)]-learning_rate*updates
        new_parameters['b'+str(i+1)]=parameters['b'+str(i+1)]-learning_rate*updates
    return new_parameters
```

```
#initilize layer size
layer_count=int(input('Enter Number of layers in Neural Network: '))
layer_dict=initialize(layer_count)
```

```
Enter Number of layers in Neural Network: 2

Enter of nodes in layer 0 :  2

Enter of nodes in layer 1 :  2

Enter of nodes in layer 2 :  1
```

```
#dataset
m=int(input('\nEnter number of input instances:'))
X,Y=[[]*m],[[]*m]
for i in range(m):
    for j in range(layer_dict['l0']):
        print('\nEnter feature value x'+str(j+1),'for instance',i+1,': ',end=' ')
        X[i].append(int(input()))
    print('\nEnter target label Y','for instance',i,':',end=' ')
    Y[i].append(int(input()))
X=np.array(X).reshape([layer_dict['l0'],m])
Y=np.array(Y).reshape([layer_dict['l'+str(layer_count)],m])
```

```
Enter number of input instances:1

Enter feature value x1 for instance 1 :  1

Enter feature value x2 for instance 1 :  0

Enter target label Y for instance 0 : 1
```

```
#initialize weights and biases
parameters=param(layer_dict,layer_count)
for key,value in parameters.items():
    print('\n',key,'=',str(value),'\n')
    print('\n',key,'=',str(value),'\n')
```

```
 W1 = [[1. 1.]
 [1. 1.]] 


 b1 = [[0.]
 [0.]] 


 W2 = [[1. 1.]] 


 b2 = [[0.]] 
```

```
#one-pass forward prop and cost calculation
output,cache=forward_prop(X,parameters,layer_count)
print('Predeiction:',output)
cost=cal_cost(output,Y)
print('\nCost: ',cost)
```

```
Predeiction: [[1.87556566e+10]
 [1.87556566e+10]]

Cost:  3.5177465258404656e+20
```

```
#single pass back prop and updated parameters display
updates=back_prop(cache,parameters,X,Y,layer_count)
for key,value in updates.items():
    print('\n',key,'=',str(value),'\n')
lr=float(input('\nEnter learning rate:'))
parameters=update_param(updates,parameters,lr,layer_count)
for key,value in parameters.items():
    print('\n',key,'=',str(value),'\n')
```

```
 dW2 = [[1225.]
 [1225.]] 


 db2 = 245.0 


 dW1 = [[1225. 1225.]
 [   0.    0.]] 


 db1 = 2450.0 

Enter learning rate:10

 W1 = [[-1.2249e+04 -1.2249e+04]
 [ 1.0000e+00  1.0000e+00]] 
 b1 = [[-24500.]
 [-24500.]]


 W2 = [[-12249. -12249.]
 [-12249. -12249.]]


 b2 = [[-2450.]]
```

```
#combined forward and back prop for multiple epochs
iterations=int(input('Enter number of epochs: '))
for i in range(iterations):
    output,cache=forward_prop(X,parameters,layer_count)
    print('\nPredeiction:',output)
    cost=cal_cost(output,Y)
    print('\nCost: ',cost)
    updates=back_prop(cache,parameters,X,Y,layer_count)
    lr=float(input('\nEnter learning rate:'))
    parameters=update_param(updates,parameters,lr,layer_count)
    for key,value in parameters.items():
        print('\n',key,'=',str(value),'\n')
```

```
 b1 = [[2.29738037e+17]
 [2.29738037e+17]]


 W2 = [[1.72312906e+17 1.72312906e+17]
 [1.14873707e+17 1.14873707e+17]]


 b2 = [[-1.87556566e+12]]


Predeiction: [[2.22675911e+36]
 [1.48448588e+36]]

Cost:  3.5810772383982626e+72

Enter learning rate:10

 W1 = [[-9.59248333e+55 -6.39490188e+55]
 [ 1.00000000e+00  1.00000000e+00]]


 b1 = [[-2.6645497e+56]
 [-2.6645497e+56]]


 W2 = [[-1.59866021e+56 -1.59866021e+56]
 [-8.52607179e+55 -8.52607179e+55]]


 b2 = [[-1.8556225e+38]]


Predeiction: [[2.51323283e+114]
 [1.34027261e+114]]
```

```
  [1.34037261e+114]]

Cost:  4.0564689903162776e+228

Enter learning rate:10
<ipython-input-8-a76d8b8e47f5>:10: RuntimeWarning: invalid value encountere
  dW=(dZ*np.multiply(dA,cache['A'+str(i-1)]))/m

 W1 = [[1.00445133e+173 5.35700089e+172]
 [1.00000000e+000 1.00000000e+000]]


 b1 = [[2.36155434e+173]
 [2.36155434e+173]]


 W2 = [[2.27686205e+173 2.27686205e+173]
 [8.92872356e+172 8.92872356e+172]]


 b2 = [[-1.92680272e+116]]


Predeiction: [[inf]
```

```
cache['A'+str(i-1)].T
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-6-523355c0104a> in <module>
----> 1 cache['A'+str(i-1)].T

KeyError: 'A-1'
```

SEARCH STACK OVERFLOW

```
output
```

```
array([[521589.704]])
```

```
output
```

```
array([[2.]])
```

```
output-Y
```

```
array([[1.]])
```

```
loss=np.multiply(Y,np.log(output))+np.multiply((1-Y),np.log(1-output))
```

```
<ipython-input-40-32fbfe961744>:1: RuntimeWarning: invalid value encountered
  loss=np.multiply(Y,np.log(output))+np.multiply((1-Y),np.log(1-output))
```

```
1-output
```

```
     array([[-1.]])
```

```
len(output)
```

```
     1
```

```
for i in range(layer_count):
    print(i)
```

```
     0
     1
```

```
def back_prop(cache,parameters,X,Y,layer_count):
    m=cache['A0'].shape[1]
    updates={}
    for i in reversed(range(1,1+layer_count)):
        if i==layer_count:
            dZ=cache['A2']-Y
        else:
            dZ=np.multiply(np.dot(parameters['W2'].T, dZ_layer_ahead), 1 - np.powe
        dW=(1/m)*np.dot(dZ,cache['A'+str(i-1)].T)
        db=(1/m)*np.sum(dZ,axis=1,keepdims=1)
        updates['dW'+str(i)]=dW
        updates['db'+str(i)]=db
        dZ_layer_ahead=dZ
        print(dZ)
        print(dW)
        print(db)
    return updates
```

```
a=np.array([5,10,15])
```

```
a/5
```

```
     array([1., 2., 3.])
```

```
if i==layer_count:
            dZ=cache['A2']-Y
        else:
            dZ=np.multiply(np.dot(parameters['W2'].T, dZ_layer_ahead), 1 - np.powe
        dW=(1/m)*np.dot(dZ,cache['A'+str(i-1)].T)
        db=(1/m)*np.sum(dZ,axis=1,keepdims=1)
        updates['dW'+str(i)]=dW
        updates['db'+str(i)]=db
        dZ_layer_ahead=dZ
```

● ×