IV Back-propagation Algorithm

Implement Back-propagation algorithm to experiment with the use of Neural Networks for a multi-class classification problem, and try and interpret the high-level or hidden representations learnt by it.

Back propagation Algorithm:

1.Load data set
2. Assign all network inputs and output
3.Initialize all weights with small random numbers, typically between -1 and 1
repeat
for every pattern in the training set
Present the pattern to the network
// Propagated the input forward through the network:
for each layer in the network
for every node in the layer
1. Calculate the weight sum of the inputs to the node
2. Add the threshold to the sum
3. Calculate the activation for the node
end
end
// Propagate the errors backward through the network
for every node in the output layer
calculate the error signal
end
for all hidden layers
for every node in the layer
1. Calculate the node's signal error
2. Update each node's weight in the network
end
end
// Calculate Global Error
Calculate the Error Function
end
while ((maximum number of iterations < than specified) AND
(Error Function is > than specified))

Source Code:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally y
= y/100

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

#Derivative of Sigmoid Function
```

```
def derivatives_sigmoid(x):
return x * (1 - x)

#Variable initialization
epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_n
eurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neu
rons))
bout=np.random.uniform(size=(1,output_neurons))

# draws a random range of numbers uniformly of dim x*y
#Forward Propagation
for i in range(epoch):
hinp1=np.dot(X,wh)
hinp=hinp1 + bh
hlayer_act = sigmoid(hinp)
outinp1=np.dot(hlayer_act,wout)
outinp= outinp1+ bout
output = sigmoid(outinp)
#Backpropagation
EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)
#how much hidden layer wts contributed to error
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr
# dotproduct of nextlayererror and currentlayerop
bout += np.sum(d_output, axis=0,keepdims=True) *lr
wh += X.T.dot(d_hiddenlayer) *lr
#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:

[[0.92]
[0.86]
[0.89]]
Predicted Output:
[[0.89559591]
[0.88142069]
[0.8928407]]