**Task 10**

**Apply back propagation neural network on image data. The idea is to build a Artificial Neural Network model that can effectively analyze and extract features from an image.**

**Tools: Google co-lab, Python, Scikitlearn, Anaconda navigator**

**Task 10:  Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

**Reference:** [**https://github.com/profthyagu/Python-ANN-Backpropagation**](https://github.com/profthyagu/Python-ANN-Backpropagation)

**Aim:**

To implement the backpropagation algorithm for a neural network from scratch with Python.

**Algorithms:**

**Step 1:** Initialize Network

**Step 2:**Forward Propagate

**Step 3:**Neuron Activation

**Step 4:**Neuron Transfer

**Step 5:**Forward Propagation

**Step 6:**Forward Propagation

**Step 7:**Transfer Derivative

**Step 8:**Error Backpropagation

**Step 9:**Train Network

**Step 10:**Update Weights

**Step 11:**Train Network

**Step 12:**Predict

**Step 13:**Dataset

**Extensions:**

This section lists extensions to the tutorial that you may wish to explore.

* **Tune Algorithm Parameters.** Try larger or smaller networks trained for longer or shorter. See if you can get better performance on the seeds dataset.
* **Additional Methods.** Experiment with different weight initialization techniques (such as small random numbers) and different transfer functions (such as tanh).
* **More Layers.** Add support for more hidden layers, trained in just the same way as the one hidden layer used in this tutorial.
* **Regression.** Change the network so that there is only one neuron in the output layer and that a real value is predicted. Pick a regression dataset to practice on. A linear transfer function could be used for neurons in the output layer, or the output values of the chosen dataset could be scaled to values between 0 and 1.
* **Batch Gradient Descent.** Change the training procedure from online to batch gradient descent and update the weights only at the end of each epoch.

**Outcomes:**

* How to forward-propagate an input to calculate an output.
* How to back-propagate error and train a network.

How to apply the backpropagation algorithm to a real-world predictive modeling problem.

**Python Program to Implement and Demonstrate Backpropagation Algorithm Machine Learning**

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=5 #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\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

#Forward Propogation

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

wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**Training Examples:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Example** | **Sleep** | **Study** | **Expected % in** **Exams** |
| **1** | 2 | 9 | 92 |
| **2** | 1 | 5 | 86 |
| **3** | 3 | 6 | 89 |

**Normalize the input**

|  |  |  |  |
| --- | --- | --- | --- |
| **Example** | **Sleep** | **Study** | **Expected % in Exams** |
| **1** | 2/3 = 0.66666667 | 9/9 = 1 | 0.92 |
| **2** | 1/3 = 0.33333333 | 5/9 = 0.55555556 | 0.86 |
| **3** | 3/3 = 1 | 6/9 = 0.66666667 | 0.89 |

**Output**

———–Epoch- 1 Starts———-  
Input:  
[[0.66666667 1. ]  
[0.33333333 0.55555556]  
[1. 0.66666667]]  
Actual Output:  
[[0.92]  
[0.86]  
[0.89]]  
Predicted Output:  
[[0.81951208]  
[0.8007242 ]  
[0.82485744]]  
———–Epoch- 1 Ends———-

———–Epoch- 2 Starts———-  
Input:  
[[0.66666667 1. ]  
[0.33333333 0.55555556]  
[1. 0.66666667]]  
Actual Output:  
[[0.92]  
[0.86]  
[0.89]]  
Predicted Output:  
[[0.82033938]  
[0.80153634]  
[0.82568134]]  
———–Epoch- 2 Ends———-

———–Epoch- 3 Starts———-  
Input:  
[[0.66666667 1. ]  
[0.33333333 0.55555556]  
[1. 0.66666667]]  
Actual Output:  
[[0.92]  
[0.86]  
[0.89]]  
Predicted Output:  
[[0.82115226]  
[0.80233463]  
[0.82649072]]  
———–Epoch- 3 Ends———-

———–Epoch- 4 Starts———-  
Input:  
[[0.66666667 1. ]  
[0.33333333 0.55555556]  
[1. 0.66666667]]  
Actual Output:  
[[0.92]  
[0.86]  
[0.89]]  
Predicted Output:  
[[0.82195108]  
[0.80311943]  
[0.82728598]]  
———–Epoch- 4 Ends———-

———–Epoch- 5 Starts———-  
Input:  
[[0.66666667 1. ]  
[0.33333333 0.55555556]  
[1. 0.66666667]]  
Actual Output:  
[[0.92]  
[0.86]  
[0.89]]  
Predicted Output:  
[[0.8227362 ]  
[0.80389106]  
[0.82806747]]  
———–Epoch- 5 Ends———-

Input:  
[[0.66666667 1. ]  
[0.33333333 0.55555556]  
[1. 0.66666667]]  
Actual Output:  
[[0.92]  
[0.86]  
[0.89]]  
Predicted Output:  
[[0.8227362 ]  
[0.80389106]  
[0.82806747]]