Batch: A2 Roll No: 16010121045

Experiment No. : 06

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Error Back propagation algorithm

Objective: To Write a program to implement error back propagation algorithm in MLP.

Expected Outcome of Experiment:

CO3 : Understand perceptron's and counter propagation networks

Books/ Journals/ Websites referred:

Pre Lab/ Prior Concepts:

Neural networks, sometimes referred to as connectionist models, are parallel-distributed models that have several distinguishing features-

1) A set of processing units;
2) An activation state for each unit, which is equivalent to the output of the unit;
3) Connections between the units. Generally each connection is defined by a weight w_{jk} that determines the effect that the signal of unit *j* has on unit *k;*
4) A propagation rule, which determines the effective input of the unit from its external inputs;
5) An activation function, which determines the new level of activation based on the effective input and the current activation;
6) An external input (bias, offset) for each unit;
7) A method for information gathering (learning rule);
8) An environment within which the system can operate, provide input signals and, if necessary, error signals.

Implementation Details :

The Back propagation algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set).

• Back propagation algorithm consists of the following two passes:

– **Forward pass**:

In this step the network is activated on one example and the error of (each neuron of) the output layer is computed.

– **Backward pass**:

In this step the network error is used for updating the weights. This process is more complex than the LMS algorithm for Adaline because hidden nodes are linked to the error not directly but by means of the nodes of the next layer. Therefore, starting at

the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each weight.

Back-propagation training algorithm:

Backpropagation algorithm is probably the most fundamental building block in a neural network. It was first introduced in 1960s and almost 30 years later (1989) popularized by Rumelhart, Hinton and Williams in a paper called “Learning representations by back-propagating errors”.

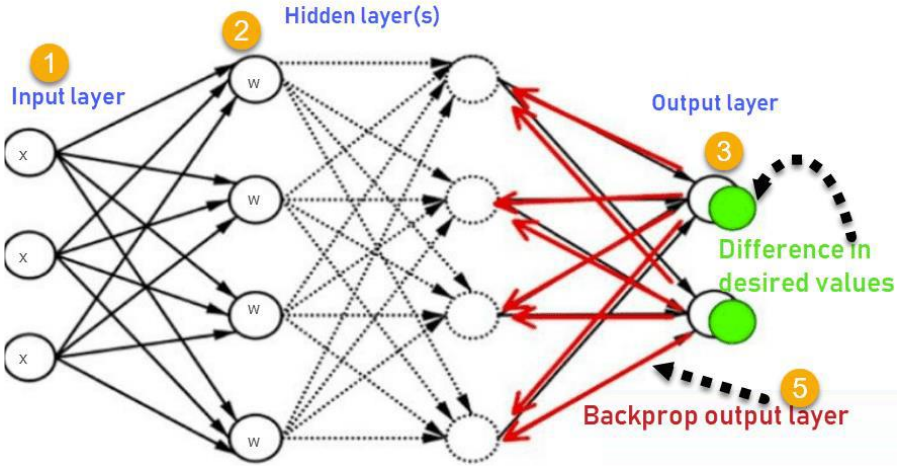
The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model’s parameters (weights and biases).

Steps for Backpropagation Algorithm:
Calculate the error – How far is your model output from the actual output.
Minimum Error – Check whether the error is minimized or not.
Update the parameters – If the error is huge then, update the parameters (weights and biases). After that again check the error. Repeat the process until the error becomes minimum.

Model is ready to make a prediction – Once the error becomes minimum, you can feed some inputs to your model and it will produce the output.

Network Architecture:

The **4-layer** neural network consists of 4 neurons for the input layer, 4 neurons for the hidden layers and 1 neuron for the output layer.



Input layer:

The neurons, colored in purple, represent the input data. These can be as simple as scalars or more complex like vectors or multidimensional matrices.
The first set of activations (a) are equal to the input values. NB: "activation" is the neuron's value after applying an activation function.

$$x_i = a_i^{(1)}, i \in 1,2,3,4$$

Hidden layers:

The final values at the hidden neurons, colored in green, are computed using z^l — weighted inputs in layer l, and a^l— activations in layer l. For layer 2 and 3 the equations are:

Layer 2:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Layer 3:

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Output layer:

The final part of a neural network is the output layer which produces the predicated value. In our simple example, it is presented as a single neuron, colored in blue and evaluated as follows:

$$s = W^{(3)}a^{(3)}$$

Algorithm: (Backpropogation)

Inputs X, arrive through the preconnected path
Input is modeled using real weights W. The weights are usually randomly selected.
Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
Calculate the error in the outputs

ErrorB= Actual Output – Desired Output

Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
Keep repeating the process until the desired output is achieved.

Code:

```
import numpy as np

def binary_sigmoid(x):
    return 1 / (1 + np.exp(-x))

def fnet(w, x):
    net = np.dot(w, x)
    return binary_sigmoid(net)

# Given weights
v12, v12, v01 = 0.6, -0.1, 0.3
v21, v22, v02 = -0.3, 0.4, 0.5
w1, w2, w0 = 0.4, 0.1, -0.2

learning_rate = 0.25

# Input
x = np.array([0, 1, 1])
desired_output = 1

W1 = np.array([v12, v12, v01])
W2 = np.array([v21, v22, v02])
W3 = np.array([w1, w2, w0])

num_iterations = 5
```

```
for i in range(num_iterations):
    hidden_output_1 = fnet(W1, x)
    hidden_output_2 = fnet(W2, x)
    y = np.array([hidden_output_1, hidden_output_2, 1])
    current_output = fnet(W3, y)

    # Error
    error = desired_output - current_output

    # Update weights
    dw3 = learning_rate * error * current_output * (1 - current_output) * y
    W3 = W3 + dw3

    dw1 = learning_rate * error * current_output * (1 - current_output) * W3[0] * hidden_output_1 * (1
- hidden_output_1) * x
    W1 = W1 + dw1

    dw2 = learning_rate * error * current_output * (1 - current_output) * W3[1] * hidden_output_2 * (1
- hidden_output_2) * x
    W2 = W2 + dw2



print("Final weights:")
print("W1:", W1)
print("W2:", W2)
print("W3:", W3)
```

Output:

```
❯ python3 -u "/Users/pargatsinghdhanjal/Desktop/Soft Computing/exo6.py"
Final weights:
W1: [-0.1        -0.09693229  0.30306771]
W2: [-0.3         0.40074117  0.50074117]
W3: [ 0.41636688  0.1211628  -0.17023305]
```

Conclusion:. Thus, we have successfully implemented back propagations algorithm.

Post Lab Descriptive Questions :

1. What is meant by local and global minima?
   The point at which a function takes the minimum value is called **global minima**. However, when the goal is to minimize the function and solved using optimization algorithms such as gradient descent, it may so happen that function may appear to have a minimum value at different points. Those several points which appear to be minima but are not the point where the function actually takes the minimum value are called **local minima**.

2. What are factors that can improve the convergence of learning in Back propagation learning algorithm?
   The factors that improve the convergence of Error Back Propotion Algorithm (EBPA) are called as learning factors they are as follows:

   Initial weights
   Steepness of activation function
   Learning constant
   Momentum
   Network architecture
   Necessary number of hidden neurons

.