

| **Title:** Error Back propagation algorithm |
| --- |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

CO2 : Analyze various neural network architectures

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**Pre Lab/ Prior Concepts:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Back-propagation training algorithm :**

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.

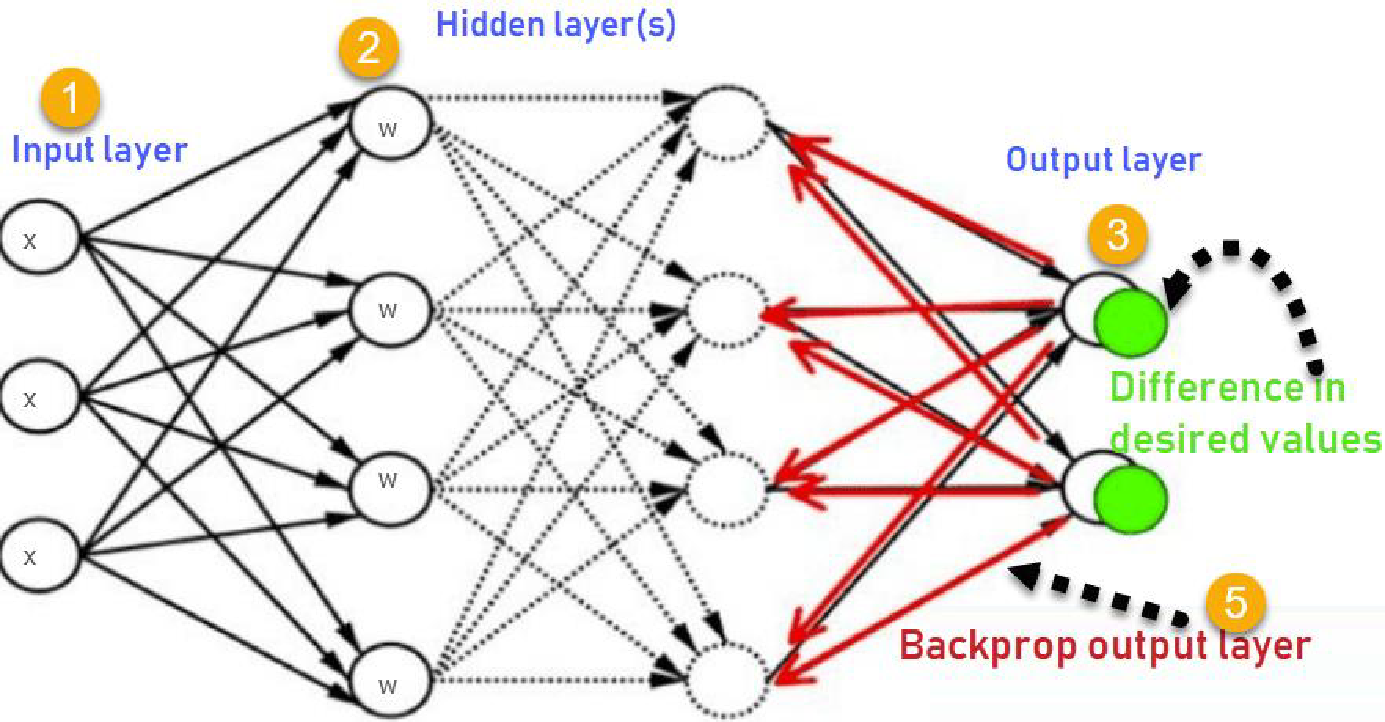
– **Backward pass**:

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.

**Network Architecture :**

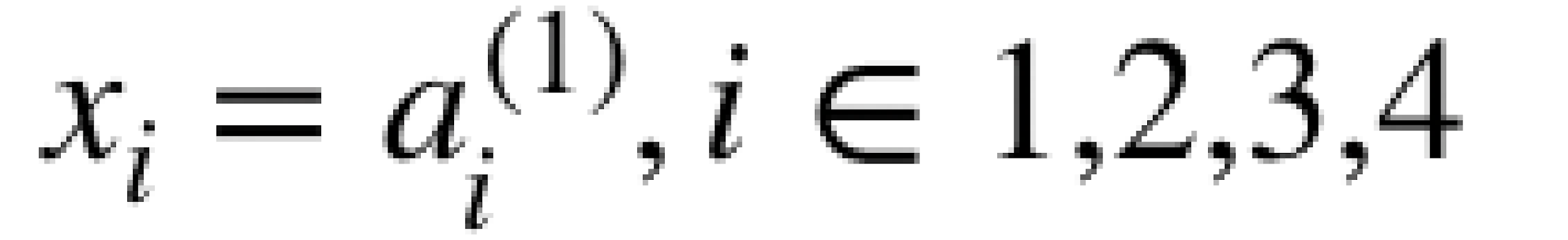
The **4-layer** neural network consists of 4 neurons for the input layer, 4 neurons for the hidden layers and 1 neuron for the output layer.



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.

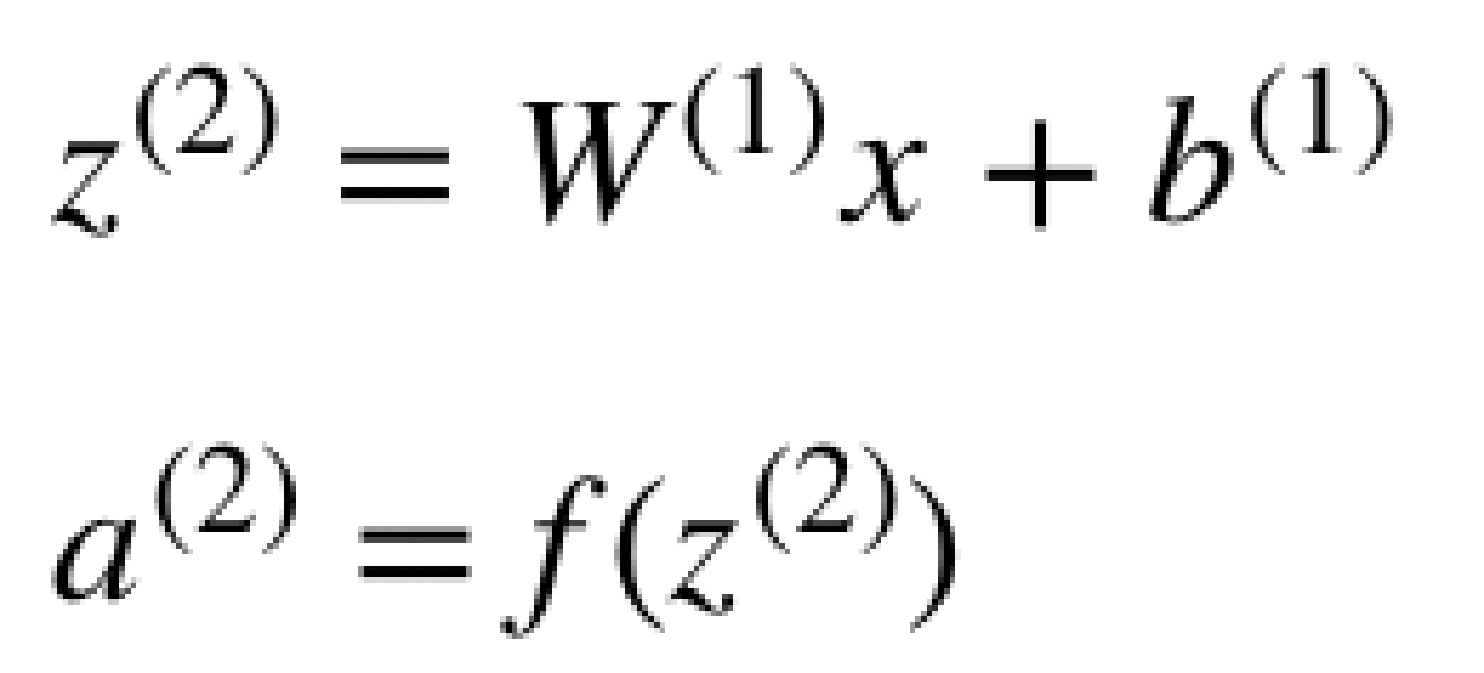
The first set of activations (a) are equal to the input values. NB: “activation” is the neuron’s value after applying an activation function.



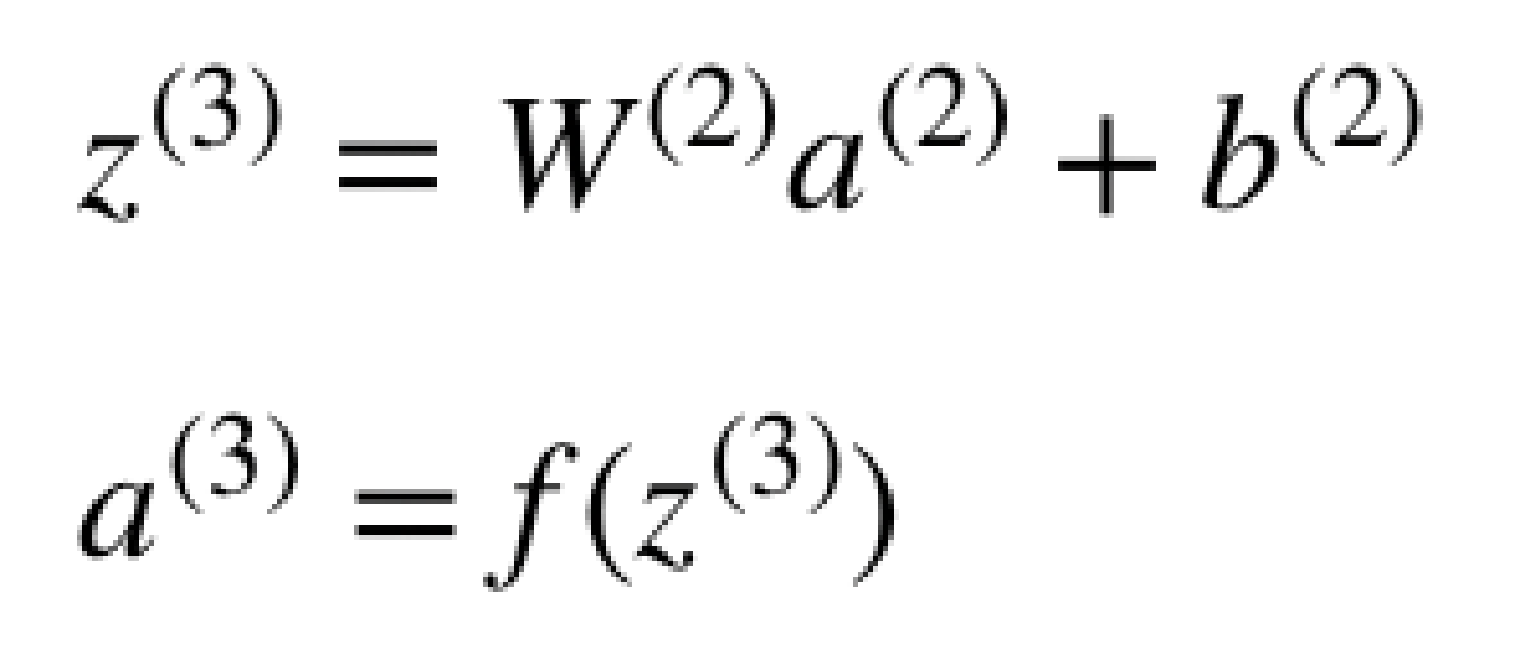
Hidden layers:

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:

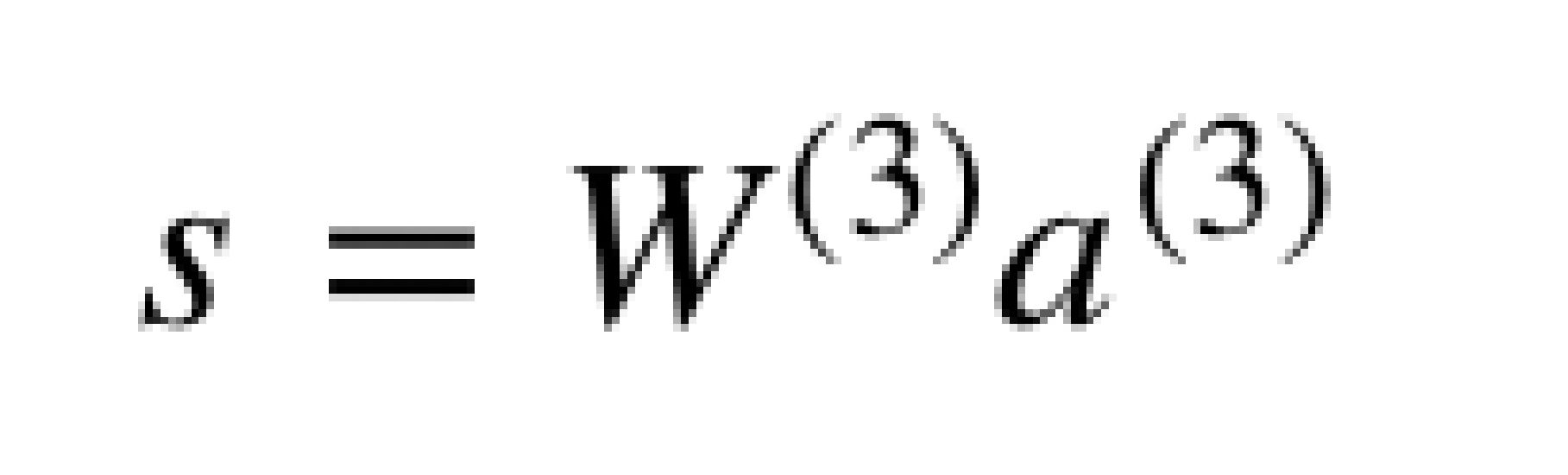


Layer 3:



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:



**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

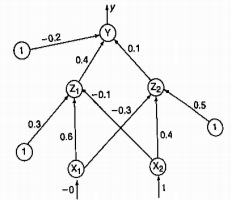
ErrorB= Actual Output – Desired Output

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.

**Implementation Details :**

To implementEBPT algorithm to find the new weights for the network given for given input output pair for certain no. of epochs.



It is presented with the input pattern [0, 1] and the target output is 1.

Use a learning rare a = 0.25 and binary sigmoidal activation function.

**Program:**

**(Forward pass)**

**import math**

**X = [0.05, 0.10]**

**W = [0.6, -0.1, -0.3, 0.4, 0.4, 0.1]**

**Hin = [0, 0,]**

**Hout = [0, 0]**

**O = [0, 0]**

**B = [0.3, 0.5, -0.2]**

**E = 0**

**t = 1**

**def sigmoid(x):**

**return 1 / (1 + math.exp(-x))**

**def forward\_pass(X, W, Hin, Hout, O):**

**Hin[0] = W[0] \* X[0] + W[1] \* X[1] + B[0]**

**Hin[1] = W[2] \* X[0] + W[3] \* X[1] + B[1]**

**Hout[0] = sigmoid(Hin[0])**

**Hout[1] = sigmoid(Hin[1])**

**O[0] = W[4] \* Hout[0] + W[5] \* Hout[1] + B[2]**

**O[1] = sigmoid(O[0])**

**E = 0.5 \* (t - O[1]) \*\* 2**

**return Hin, Hout, O, E**

**print("Inputs: ", X)**

**print("Weights: ", W)**

**print("Targets: ", t)**

**print("Biases: ", B)**

**Hin, Hout, O, E = forward\_pass(X, W, Hin, Hout, O)**

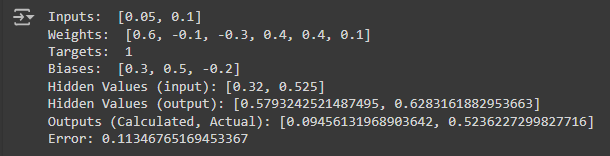
**print("Hidden Values (input):", Hin)**

**print("Hidden Values (output):", Hout)**

**print("Outputs (Calculated, Actual):", O)**

**print("Error:", E)**

**output:**

****

**Conclusion:**.

**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**.

1. 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