**Machine Learning**

**Experiment 6**

SAP ID: 60004200107 Name: Kartik Jolapara

Division: B Batch: B1

**AIM**

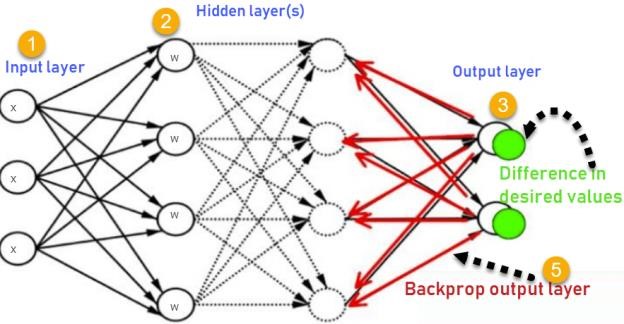
To implement Backpropagation.

# THEORY

A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

Backpropagation in neural network is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network. The working -



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

ErrorB = Actual Output – Desired Output

1. 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.

## Advantages of Backpropagation

* It is fast, simple, and easy to program.
* It has no parameters to tune apart from the numbers of input.
* It is a flexible method as it does not require prior knowledge about the network.

# CODE

import numpy as np

class NeuralNetwork: def \_init\_(self, input\_dim, hidden\_dim, output\_dim):

self.input\_dim = input\_dim self.hidden\_dim = hidden\_dim self.output\_dim = output\_dim

# Initialize weights and biases self.weights1 = np.random.randn(self.input\_dim, self.hidden\_dim) self.bias1 = np.random.randn(self.hidden\_dim) self.weights2 = np.random.randn(self.hidden\_dim, self.output\_dim) self.bias2 = np.random.randn(self.output\_dim)

def sigmoid(self, z):

return 1/(1+np.exp(-z))

def sigmoid\_derivative(self, z):

return z \* (1 - z)

def train(self, X, y, epochs): for i in range(epochs): # Forward propagation z1 = np.dot(X, self.weights1) + self.bias1 hidden\_layer = self.sigmoid(z1) z2 = np.dot(hidden\_layer, self.weights2) + self.bias2 output\_layer = self.sigmoid(z2)

# Backpropagation output\_error = y - output\_layer output\_delta = output\_error \* self.sigmoid\_derivative(output\_layer) hidden\_error = np.dot(output\_delta, self.weights2.T) hidden\_delta = hidden\_error \* self.sigmoid\_derivative(hidden\_layer)

# Update the weights and biases

self.weights2 += np.dot(hidden\_layer.T, output\_delta) self.bias2 += np.sum(output\_delta, axis=0) self.weights1 += np.dot(X.T, hidden\_delta) self.bias1 += np.sum(hidden\_delta, axis=0)

# Print the loss every 100 epochs if i % 100 == 0:

loss = np.mean(np.square(y - output\_layer)) print(f"Epoch {i}: Loss = {loss}")

def predict(self, X):

# Make a prediction for a new input z1 = np.dot(X, self.weights1) + self.bias1 hidden\_layer = self.sigmoid(z1) z2 = np.dot(hidden\_layer, self.weights2) + self.bias2 output\_layer = self.sigmoid(z2)

return output\_layer

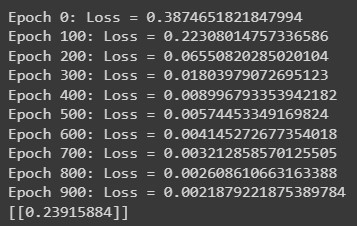
# Create a dataset

X = np.array([[0,0,1], [0,1,1], [1,0,1], [1,1,1]]) y = np.array([[0], [1], [1], [0]])

# Create a neural network with 3 input nodes, 4 hidden nodes, and 1 output node nn = NeuralNetwork() nn.\_init\_(3, 4, 1)

# Train the neural network for 1000 epochs nn.train(X, y, 1000)

# Make a prediction for a new input new\_input = np.array([[1, 0, 0]]) print(nn.predict(new\_input)) **OUTPUT**



# CONCLUSION

Hence, we have successfully implemented Backpropagation.