**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJ22ITL502 DATE: 29/10/24**

**COURSE NAME: Artificial Intelligence Laboratory CLASS: TY-IT**

**Name: Anish Sharma**

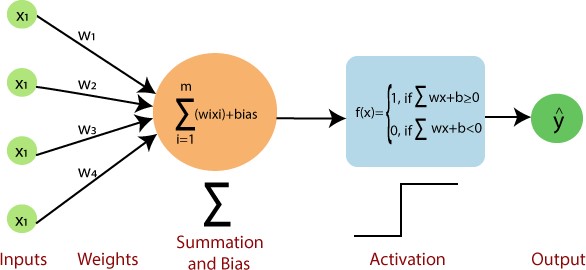
**EXPERIMENT NO.07**

**CO/LO:** Apply various AI approaches to knowledge intensive problem solving, reasoning, planning and uncertainty.

**AIM / OBJECTIVE:** Implement learning: Perceptron Learning / Backpropagation Algorithm

**DESCRIPTION OF EXPERIMENT:**

The **Perceptron** is a type of artificial neuron that performs binary classification. It maps its input x (a real-valued vector) to an output y (a binary value) using a linear transformation.



Mathematical Representation:

The perceptron works by applying a weight w to the input vector x and computing the sum z using a bias b as:



**The activation function applies a threshold to the computed value z:**



Learning Rule**: If the prediction y does not match the actual label y^, the weights are updated using:**



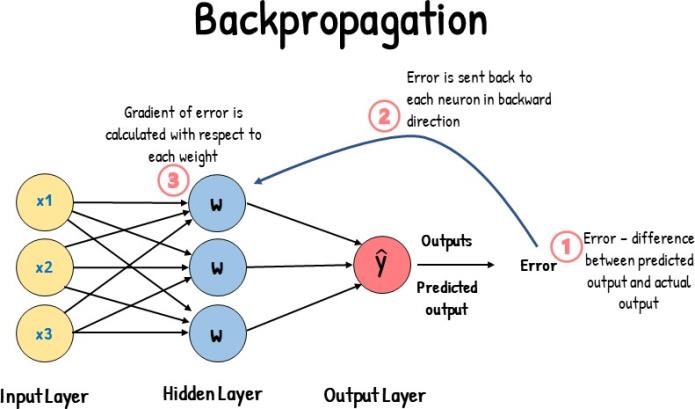
Where:

* η is the learning rate,
* y^ is the actual label, • y is the predicted label.

*2. Backpropagation Algorithm:*

Backpropagation is a supervised learning algorithm used for training multi-layered neural networks. The algorithm consists of two passes:

1. **Forward Pass**: Propagate the input through the network and compute the output.
2. **Backward Pass**: Compute the error and propagate it backward to adjust the weights.



Key Concepts:

* **Activation Function**: A differentiable function, often sigmoid, used in the neurons. For a neuron output o:



•

* **Error Calculation**: The difference between the actual and predicted output. For the output layer:



**Weight Update**: The weights are updated using the gradient of the error with respect to the weights. For the hidden layer weights:



**This process is repeated for a number of epochs to minimize the error and improve the model's accuracy.**

**Procedure:**

**Perceptron Learning Algorithm Implementation:**

1. Initialize the weights and bias to small random values or zero.
2. Input the dataset XXX with corresponding labels yyy for binary classification.
3. For each training sample:
   * Calculate the dot product of the input and weights, then apply the activation function.
   * Update the weights if the prediction is incorrect using the Perceptron learning rule.
4. Repeat this process for a fixed number of epochs until the model converges (i.e., makes accurate predictions for all data points).

**Backpropagation Algorithm Implementation:**

1. Initialize the weights of the input-hidden and hidden-output layers randomly.
2. Input the dataset XXX with corresponding labels yyy.
3. Forward Pass:
   * Compute the activations of the hidden layer using the sigmoid function.
   * Compute the output layer’s activation.

4. Backward Pass:

* + Calculate the error between the predicted and actual output.
  + Update the hidden-output and input-hidden layer weights using the gradient of the error.

1. Train the model for a number of epochs until the error is minimized.
2. Test the model using new input data and compare the predicted outputs with actual labels.

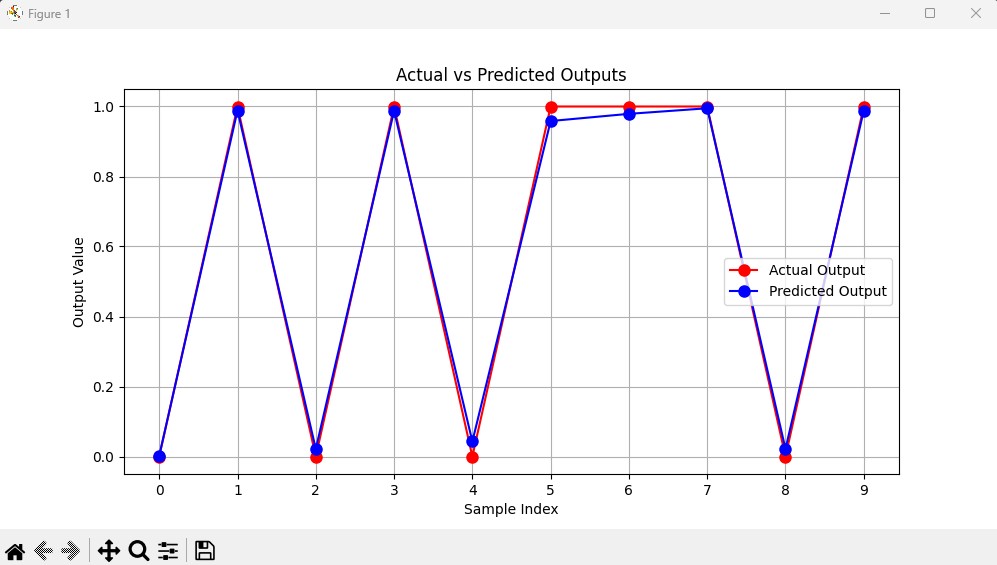
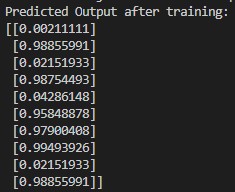
**EXPLANATION / SOLUTIONS (DESIGN):**

**Code:**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt    def sigmoid(x):  return 1 / (1 + np.exp(-x))    def sigmoid\_derivative(x):  return x \* (1 - x)  input\_size = 3 hidden\_size = 4 output\_size = 1 learning\_rate = 0.1  epochs = 10000    np.random.seed(42) weights\_input\_hidden = np.random.rand(input\_size, hidden\_size) weights\_hidden\_output = np.random.rand(hidden\_size, output\_size)    X = np.array([[0, 0, 0], [0, 1, 0],  [1, 0, 0],  [1, 1, 0],  [0, 0, 1],  [1, 0, 1],  [0, 1, 1],  [1, 1, 1],  [1, 0, 0], [0, 1, 0]])  y = np.array([[0], [1], [0], [1], [0], [1], [1], [1], [0], [1]])    for epoch in range(epochs):  hidden\_layer\_input = np.dot(X, weights\_input\_hidden) hidden\_layer\_output = sigmoid(hidden\_layer\_input) |

|  |
| --- |
| output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) predicted\_output = sigmoid(output\_layer\_input)  error = y - predicted\_output  d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)    weights\_hidden\_output += hidden\_layer\_output.T.dot(d\_predicted\_output) \*  learning\_rate  error\_hidden\_layer = d\_predicted\_output.dot(weights\_hidden\_output.T) d\_hidden\_layer\_output = error\_hidden\_layer \*  sigmoid\_derivative(hidden\_layer\_output) weights\_input\_hidden += X.T.dot(d\_hidden\_layer\_output) \* learning\_rate    hidden\_layer\_input = np.dot(X, weights\_input\_hidden) hidden\_layer\_output = sigmoid(hidden\_layer\_input) output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) predicted\_output = sigmoid(output\_layer\_input)  print("Predicted Output after training:") print(predicted\_output)  plt.figure(figsize=(10, 5)) plt.plot(range(len(y)), y, 'ro-', label='Actual Output', markersize=8)  plt.plot(range(len(predicted\_output)), predicted\_output, 'bo-', label='Predicted Output', markersize=8) plt.title('Actual vs Predicted Outputs') plt.xlabel('Sample Index') plt.ylabel('Output Value') plt.xticks(range(len(y))) plt.legend() plt.grid() plt.show() |

**Output:**



**CONCLUSION:** Implementing the Perceptron Learning Algorithm or Backpropagation allows a neural network to learn from its errors, iteratively adjusting weights to minimize the difference between predicted and actual outputs, ultimately improving accuracy in classification tasks.

**REFERENCES:**

[1] Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, 2nd Edition,Pearson Education, 2010