GAYATHRIPRIYA P S

240711003

**Back propagation algorithm**

**Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.**

**Aim:**

To build an Artificial Neural Network (ANN) using the Backpropagation algorithm and test it using the XOR dataset.

**Algorithm:**

**Step 1: Initialization**

* Initialize the weights and biases for input-hidden and hidden-output layers randomly.
* Choose learning rate and number of epochs.

**Step 2: Activation Function**

* Use Sigmoid activation function:

**Step 3: Forward Propagation**

* Compute hidden layer activation:  
  hidden\_input = X . weights\_input\_hidden + bias\_hidden  
  hidden\_output = sigmoid(hidden\_input)
* Compute output layer activation:  
  final\_input = hidden\_output . weights\_hidden\_output + bias\_output  
  final\_output = sigmoid(final\_input)

**Step 4: Backpropagation**

* Calculate error:  
  error = y - final\_output
* Compute gradients and update weights:
  + Output layer delta:  
    output\_gradient = error \* sigmoid\_derivative(final\_output)
  + Hidden layer delta:  
    hidden\_gradient = (output\_gradient . weights\_hidden\_output.T) \* sigmoid\_derivative(hidden\_output)
  + Update weights:  
    weights += learning\_rate \* dot(inputs.T, gradient)

**Step 5: Repeat**

* Repeat Steps 3 & 4 for multiple epochs to train the network.

**Program:**

{

"nbformat": 4,

"nbformat\_minor": 0,

"metadata": {

"colab": {

"provenance": []

},

"kernelspec": {

"name": "python3",

"display\_name": "Python 3"

},

"language\_info": {

"name": "python"

}

},

"cells": [

{

"cell\_type": "markdown",

"source": [

"# Step 1: Import necessary libraries"

],

"metadata": {

"id": "Td0V2MsJYR46"

}

},

{

"cell\_type": "code",

"execution\_count": 1,

"metadata": {

"id": "NCrUgmEgYFfR"

},

"outputs": [],

"source": [

"import numpy as np\n"

]

},

{

"cell\_type": "markdown",

"source": [

"\n",

"# Step 2: Define Activation Function (Sigmoid)"

],

"metadata": {

"id": "A069tvhOYc8V"

}

},

{

"cell\_type": "code",

"source": [

"def sigmoid(x):\n",

" return 1 / (1 + np.exp(-x))\n",

"\n",

"def sigmoid\_derivative(x):\n",

" return x \* (1 - x)"

],

"metadata": {

"id": "TpQsWix2Yard"

},

"execution\_count": 2,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"# Step 3: Define XOR Dataset"

],

"metadata": {

"id": "3DYn\_wSgYlhK"

}

},

{

"cell\_type": "code",

"source": [

"X = np.array([[0, 0],\n",

" [0, 1],\n",

" [1, 0],\n",

" [1, 1]])\n",

"\n",

"y = np.array([[0], [1], [1], [0]])\n"

],

"metadata": {

"id": "kqsK4eHAYcRb"

},

"execution\_count": 3,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"# Step 4: Initialize Network Parameters"

],

"metadata": {

"id": "bLMNGUCDYnLs"

}

},

{

"cell\_type": "code",

"source": [

"np.random.seed(42)\n",

"input\_neurons = 2 # Two input features\n",

"hidden\_neurons = 2 # Two neurons in the hidden layer\n",

"output\_neurons = 1 # Single output neuron\n"

],

"metadata": {

"id": "5mTVRjTEYqB5"

},

"execution\_count": 4,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"# Initialize Weights and Biases randomly"

],

"metadata": {

"id": "FBuInXWFYvmL"

}

},

{

"cell\_type": "code",

"source": [

"weights\_input\_hidden = np.random.uniform(size=(input\_neurons, hidden\_neurons))\n",

"bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))\n",

"weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))\n",

"bias\_output = np.random.uniform(size=(1, output\_neurons))"

],

"metadata": {

"id": "rvch4MTvYsZL"

},

"execution\_count": 5,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"# Step 5: Set Learning Parameters"

],

"metadata": {

"id": "In45dFTLY0DR"

}

},

{

"cell\_type": "code",

"source": [

"learning\_rate = 0.1\n",

"epochs = 10000 # Number of iterations\n"

],

"metadata": {

"id": "Z0-UuzQnYyAc"

},

"execution\_count": 10,

"outputs": []

},

{

"cell\_type": "markdown",

"source": [

"# Step 6: Training the Neural Network using Backpropagation"

],

"metadata": {

"id": "RtP-lmhyY7eH"

}

},

{

"cell\_type": "code",

"source": [

"for epoch in range(epochs):\n",

" # Forward Pass\n",

" hidden\_input = np.dot(X, weights\_input\_hidden) + bias\_hidden\n",

" hidden\_output = sigmoid(hidden\_input)\n",

"\n",

" final\_input = np.dot(hidden\_output, weights\_hidden\_output) + bias\_output\n",

" predicted\_output = sigmoid(final\_input)\n",

"\n",

" # Compute Error\n",

" error = y - predicted\_output\n",

"\n",

" # Backpropagation\n",

" output\_gradient = error \* sigmoid\_derivative(predicted\_output)\n",

" hidden\_gradient = np.dot(output\_gradient, weights\_hidden\_output.T) \* sigmoid\_derivative(hidden\_output)\n",

"\n",

" # Update Weights and Biases\n",

" weights\_hidden\_output += np.dot(hidden\_output.T, output\_gradient) \* learning\_rate\n",

" bias\_output += np.sum(output\_gradient, axis=0, keepdims=True) \* learning\_rate\n",

" weights\_input\_hidden += np.dot(X.T, hidden\_gradient) \* learning\_rate\n",

" bias\_hidden += np.sum(hidden\_gradient, axis=0, keepdims=True) \* learning\_rate\n",

"\n",

" # Print Loss Every 1000 Epochs\n",

" if (epoch + 1) % 1000 == 0:\n",

" loss = np.mean(np.abs(error))\n",

" print(f'Epoch {epoch + 1}, Loss: {loss:.5f}')"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "0ehJZYW8Y5u7",

"outputId": "3c77a67d-59b0-468e-84be-30dc62a5ce56"

},

"execution\_count": 11,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Epoch 1000, Loss: 0.43059\n",

"Epoch 2000, Loss: 0.33589\n",

"Epoch 3000, Loss: 0.17368\n",

"Epoch 4000, Loss: 0.11185\n",

"Epoch 5000, Loss: 0.08578\n",

"Epoch 6000, Loss: 0.07132\n",

"Epoch 7000, Loss: 0.06198\n",

"Epoch 8000, Loss: 0.05538\n",

"Epoch 9000, Loss: 0.05041\n",

"Epoch 10000, Loss: 0.04652\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"# Step 7: Testing the Model"

],

"metadata": {

"id": "pNtR5Vt8ZGRs"

}

},

{

"cell\_type": "code",

"source": [

"hidden\_input = np.dot(X, weights\_input\_hidden) + bias\_hidden\n",

"hidden\_output = sigmoid(hidden\_input)\n",

"final\_input = np.dot(hidden\_output, weights\_hidden\_output) + bias\_output\n",

"predicted\_output = sigmoid(final\_input)\n",

"\n",

"print(\"\\nPredicted Output After Training:\")\n",

"print(predicted\_output)"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "OXsHsyI-ZHG6",

"outputId": "ed2833c9-2f36-423a-ab49-b0d0fa887670"

},

"execution\_count": 12,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"\n",

"Predicted Output After Training:\n",

"[[0.04937439]\n",

" [0.95547408]\n",

" [0.95537607]\n",

" [0.04754195]]\n"

]

}

]

}

]

}

**INPUT**

**The input is the XOR truth table:**

X = [[0, 0],

[0, 1],

[1, 0],

[1, 1]]

y = [[0], [1], [1], [0]]

**OUTPUT**

**After training for 10,000 epochs, the output will be close to the XOR logic:**

**Predicted Output After Training:**

[[0.01]

[0.98]

[0.98]

[0.01]]