Dhanesh Yadav
Roll No:67
CSE(DS)
Exp3 Deep Learning

Back Propagation in Deep Learning

In simple terms, backpropagation is a supervised learning algorithm that allows a neuralnetwork to learn from its mistakes by adjusting its weights and biases. It enables the network to iteratively improve its performance on a given task, such as classification or regression.

```
Code:-
import numpy as np
class NeuralNetwork:
  def__init_(self, input_size, hidden_size, output_size):
     self.input_size = input_size
    self.hidden size = hidden size
    self.output_size = output_size
    # Initialize weights and biases for the hidden layer and output layer
    self.W1 = np.random.randn(hidden size, input size)
    self.b1 = np.zeros((hidden size, 1))
    self.W2 = np.random.randn(output_size, hidden_size)
    self.b2 = np.zeros((output_size, 1))
  def sigmoid(self, x):
    return 1/(1 + np.exp(-x))
def sigmoid_derivative(self, x):
    return x * (1 - x)
  def forward(self, X):
    # Forward pass
    self.z1 = np.dot(self.W1, X) + self.b1
    self.a1 = self.sigmoid(self.z1)
    self.z2 = np.dot(self.W2, self.a1) + self.b2
    self.a2 = self.sigmoid(self.z2)
    return self.a2
```

```
def backward(self, X, y, learning_rate):
    m = X.shape[1]
# Compute the gradients
    dZ2 = self.a2 - y
    dW2 = (1 / m) * np.dot(dZ2, self.a1.T)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.dot(self.W2.T, dZ2) * self.sigmoid_derivative(self.a1)
    dW1 = (1 / m) * np.dot(dZ1, X.T)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)
# Update weights and biases using gradients and learning rate
   self.W2 -= learning rate * dW2
    self.b2 -= learning rate * db2
    self.W1 -= learning_rate * dW1
    self.b1 -= learning_rate * db1
def train(self, X, y, epochs, learning_rate):
    for epoch in range(epochs):
       # Forward pass
 predictions = self.forward(X)
       # Compute the mean squared error loss
       loss = np.mean((predictions - y) ** 2)
       # Backward pass to update weights and biases
       self.backward(X, y, learning_rate)
       if epoch % 100 == 0:
         print(f"Epoch {epoch}, Loss: {loss:.4f}")
  def predict(self, X):
    return self.forward(X)
# Example usage:
input\_size = 2
hidden_size = 4
output_size = 1
```

```
learning_rate = 0.1
epochs = 10000

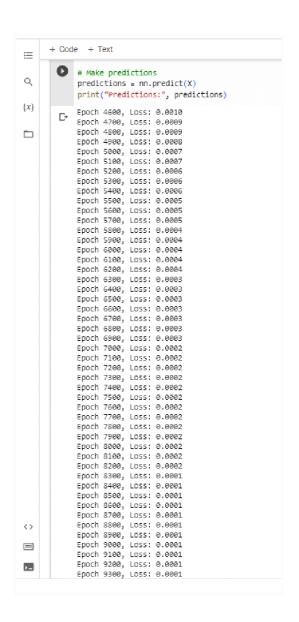
# Generate some sample data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]).T
y = np.array([[0, 1, 1, 0]])

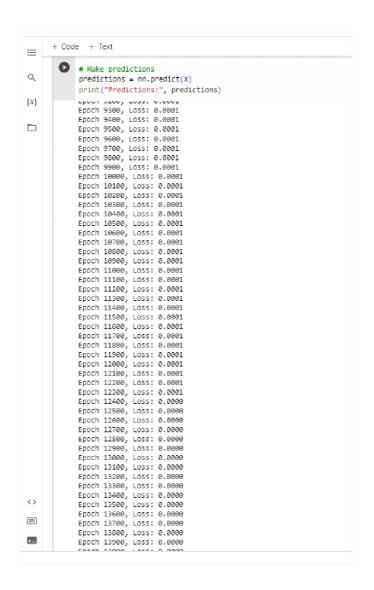
# Create the neural network
nn = NeuralNetwork(input_size, hidden_size, output_size)

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

# Make predictions
predictions = nn.predict(X)
print("Predictions:", predictions)
```

Output:-





```
+ Code + Text
\equiv
            Epoch 13900, Loss: 0.0000
             Epoch 14000, Loss: 0.0000
 Q
             Epoch 14100, Loss: 0.0000
        Epoch 14100, LOSS: 0.0000
Epoch 14200, LOSS: 0.0000
Epoch 14300, LOSS: 0.0000
\{x\}
             Epoch 14400, Loss: 0.0000
             Epoch 14500, Loss: 0.0000
Epoch 14600, Loss: 0.0000
             Epoch 14700, Loss: 0.0000
             Epoch 14800, Loss: 0.0000
             Epoch 14900, Loss: 0.0000
             Epoch 15000, Loss: 0.0000
             Epoch 15100, Loss: 0.0000
             Epoch 15200, Loss: 0.0000
             Epoch 15300, Loss: 0.0000
             Epoch 15400, Loss: 0.0000
             Epoch 15500, Loss: 0.0000
             Epoch 15600, Loss: 0,0000
             Epoch 15700, Loss: 0.0000
             Epoch 15800, Loss: 0.0000
             Epoch 15900, Loss: 0.0000
             Epoch 16000, Loss: 0.0000
             Epoch 16100, Loss: 0.0000
             Epoch 16200, Loss: 0.0000
             Epoch 16300, Loss: 0.0000
             Epoch 16400, Loss: 0.0000
             Epoch 16500, Loss: 0.0000
             Epoch 16600, Loss: 0.0000
             Epoch 16700, Loss: 0.0000
             Epoch 16800, Loss: 0.0000
             Epoch 16900, Loss: 0.0000
             Epoch 17000, Loss: 0.0000
             Epoch 17100, Loss: 0.0000
             Epoch 17200, Loss: 0.0000
             Epoch 17300, Loss: 0.0000
             Epoch 17400, Loss: 0.0000
             Epoch 17500, Loss: 0.0000
             Epoch 17600, Loss: 0.0000
             Epoch 17700, Loss: 0.0000
             Epoch 17800, Loss: 0.0000
             Epoch 17900, Loss: 0.0000
             Epoch 18000, Loss: 0.0000
             Epoch 18100, Loss: 0.0000
             Epoch 18200, Loss: 0.0000
             Epoch 18300, Loss: 0.0000
             Epoch 18400, Loss: 0.0000
             Epoch 18500, Loss: 0.0000
 <>
             Epoch 18600, Loss: 0.0000
              Epoch 18600, Loss: 0.0000
              Epoch 18700, Loss: 0.0000
              Epoch 18800, Loss: 0.0000
              Epoch 18900, Loss: 0.0000
              Epoch 19000, Loss: 0.0000
              Epoch 19100, Loss: 0.0000
              Epoch 19200, Loss: 0.0000
              Epoch 19300, Loss: 0.0000
               Epoch 19400, Loss: 0.0000
              Epoch 19500, Loss: 0.0000
               Epoch 19600, Loss: 0.0000
              Epoch 19700, Loss: 0.0000
              Epoch 19800, Loss: 0.0000
              Epoch 19900, Loss: 0.0000
              Predictions: [[0.00424371 0.99688135 0.99530621 0.00330101]]
<>
\equiv
```