Experiment 2

Prinshi Jha

CSE(DS)/23

Back propogation in Deep Learning

Backpropagation is a crucial component of deep learning and forms the basis for training neural networks. It is an optimization algorithm used to adjust the parameters (weights and biases) of a neural network so that it can learn to make accurate predictions on a given task. The process involves two main steps: the forward pass and the backward pass.

- Static backpropagation. Static backpropagation is a network developed to map static inputs for static outputs. Static backpropagation networks can solve static classification problems, such as optical character recognition (OCR).
- Recurrent backpropagation. The recurrent backpropagation network is used for fixed-point learning. Recurrent backpropagation activation feeds forward until it reaches a fixed value.

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)
```

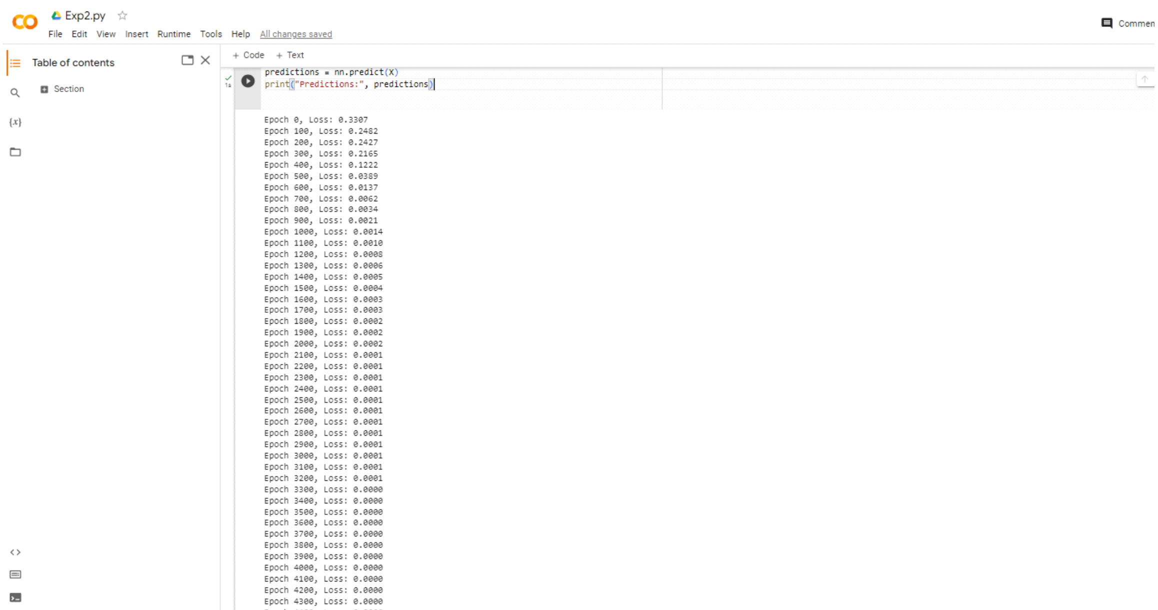
```python
        dZ1 = np.dot(self.W2.T, dZ2) * self.sigmoid_derivative(self.a1)
        dW1 = (1 / m) * np.dot(dZ1, X.T)
        db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

        # 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):
```

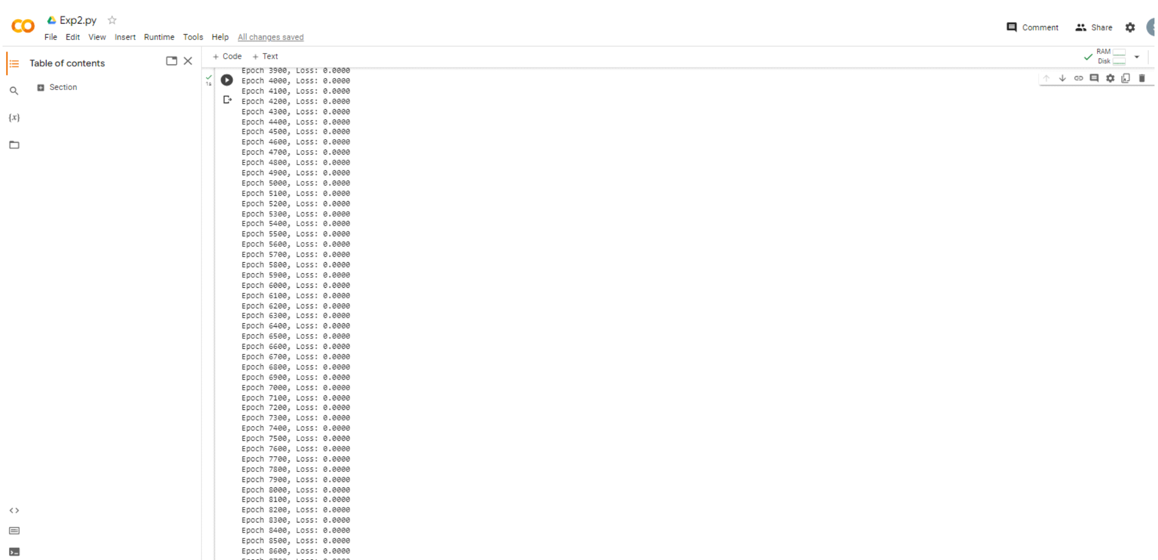
```
        return self.forward(X)

# Example usage:
input_size = 2
hidden_size = 4
output_size = 1
learning_rate = 0.5
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:





Exp2.py
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Section
+ Code + Text
RAM
Disk
Epoch 6800, Loss: 0.0000
Epoch 6900, Loss: 0.0000
Epoch 7000, Loss: 0.0000
Epoch 7100, Loss: 0.0000
Epoch 7200, Loss: 0.0000
Epoch 7300, Loss: 0.0000
Epoch 7400, Loss: 0.0000
Epoch 7500, Loss: 0.0000
Epoch 7600, Loss: 0.0000
Epoch 7700, Loss: 0.0000
Epoch 7800, Loss: 0.0000
Epoch 7900, Loss: 0.0000
Epoch 8000, Loss: 0.0000
Epoch 8100, Loss: 0.0000
Epoch 8200, Loss: 0.0000
Epoch 8300, Loss: 0.0000
Epoch 8400, Loss: 0.0000
Epoch 8500, Loss: 0.0000
Epoch 8600, Loss: 0.0000
Epoch 8700, Loss: 0.0000
Epoch 8800, Loss: 0.0000
Epoch 8900, Loss: 0.0000
Epoch 9000, Loss: 0.0000
Epoch 9100, Loss: 0.0000
Epoch 9200, Loss: 0.0000
Epoch 9300, Loss: 0.0000
Epoch 9400, Loss: 0.0000
Epoch 9500, Loss: 0.0000
Epoch 9600, Loss: 0.0000
Epoch 9700, Loss: 0.0000
Epoch 9800, Loss: 0.0000
Epoch 9900, Loss: 0.0000
Predictions: [[9.07148066e-04 9.98292920e-01 9.98376135e-01 3.22089908e-03]]
[ ]