22510064 – Parshwa Herwade (TY CSE)

Soft Computing

Assignment No. 1

Implement simple backpropagation algorithm for the following dataset.

SR. NO.	INPUTS		OUTPUT
	I1	I2	O
1	0.4	-0.7	0.1
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12

ANSWER:

1. Overview of Backpropagation

Backpropagation is a method used to train multi-layer neural networks. It works in two main phases:

Forward Pass: You take your input data, pass it through the network layer by layer, and produce an output.

Backward Pass: You compare the network's output with the desired or target output to determine how far off the prediction is (the error). Then you send this error information backward through the network so that each connection (weight) can be adjusted in a way that reduces future error.

The key idea is that each weight in the network has some responsibility for the overall error. Backpropagation figures out how to distribute that responsibility so that each weight gets adjusted just enough to make the overall output more accurate.

2. Basic Steps in Backpropagation

Initialization

Begin by setting the weights (the numeric values connecting neurons) to small random numbers. Each neuron typically also has an extra value called a bias, which is also set to a small random value.

Forward Pass

Send the input data (for example, your two inputs I1 and I2) into the first layer of the network.

Each neuron in the hidden layer takes those inputs, multiplies them by its respective weights, adds its bias, and then passes the result through an activation function (often something like a Sigmoid, which squashes the result into a specific range).

The outputs of the hidden layer become inputs to the next layer (the output layer). The output layer does the same process (multiply, add bias, apply activation) to produce the final network output.

Error Calculation

Compare the network's output to the target value. The difference between these two is the error.

Typically, you want a single measure of error for each data point, so you combine the difference in some way (for instance, by squaring the difference).

Backward Pass

Starting from the output layer, figure out how much each connection contributed to the error. This involves taking the network's output, the target, and seeing which weights led to that discrepancy.

Adjust the weights between the hidden layer and the output layer first.

Move one step backward to the hidden layer. Figure out how much each hidden neuron contributed to the error, and then adjust the weights between the input layer and the hidden layer.

In practice, this is done by computing signals (often called "error signals" or "gradients") that tell each weight which direction and by how much to change.

Update Weights

Once you know how each weight affects the error, you adjust the weights by a small amount (controlled by something called the learning rate).

A larger learning rate makes the weights change more drastically. A smaller learning rate makes them change more slowly.

Repeat

You repeat these forward and backward passes many times (through all your training samples). Each full pass through the entire dataset is often called an "epoch." Over many epochs, the weights gradually settle into values that minimize the overall error.

3. Applying Backpropagation to the Given Dataset

You have a dataset of five entries, each with two inputs (I1, I2) and one target output (O). An example neural network to handle this might have:

Input Layer: 2 neurons (one for each of I1 and I2)

Hidden Layer: a small number of neurons (often 2 to 5 in a simple example)

Output Layer: 1 neuron (since there is a single target value)

A typical training approach would be:

Set up the Network

Decide how many hidden neurons you want. Initialize the weights and biases.

Train Over Multiple Epochs

For each input-target pair (e.g., 0.4 and -0.7, target 0.1), do a forward pass to get the predicted output.

Compare the predicted output to the target to get the error.

Perform a backward pass to update the weights, slightly nudging them in the direction that reduces the error.

Move to the next input-target pair.

After you've processed all five data points, you've completed one epoch.

Monitor the Error

Keep track of the overall error across all data points. If it's going down over time, that's a good sign your network is learning.

Stop and Evaluate

After a certain number of epochs or after the error is sufficiently small, stop training.

Test the network on the same five inputs to see if the outputs are close to the targets.

CODE:

```python
import numpy as np

training_data = [
    ([0.4, -0.7], 0.1),
    ([0.3, -0.5], 0.05),
    ([0.6,  0.1], 0.3),
    ([0.2,  0.4], 0.25),
    ([0.1,  0.2], 0.12)
]

X = np.array([x for (x, _) in training_data], dtype=np.float32)
y = np.array([t for (_, t) in training_data], dtype=np.float32)

input_neurons = 2
hidden_neurons = 2
output_neurons = 1
eta = 0.5

def sigmoid(z):
    # Sigmoid activation function
    return 1.0 / (1.0 + np.exp(-z))

def sigmoid_deriv(a):
    # Derivative of sigmoid function
    return a * (1 - a)

W_input_hidden = np.random.uniform(-0.5, 0.5, (input_neurons, hidden_neurons))
W_hidden_output = np.random.uniform(-0.5, 0.5, (hidden_neurons, output_neurons))
b_hidden = np.random.uniform(-0.5, 0.5, (hidden_neurons,))
b_output = np.random.uniform(-0.5, 0.5, (output_neurons,))

```

```python

epochs = 10000

for epoch in range(epochs):
    total_error = 0.0
    for i in range(len(X)):
        net_hidden = np.dot(X[i], W_input_hidden) + b_hidden
        out_hidden = sigmoid(net_hidden)
        net_output = np.dot(out_hidden, W_hidden_output) + b_output
        out_output = sigmoid(net_output)
        target = y[i]
        error = 0.5 * (target - out_output) ** 2
        total_error += error
        delta_output = (out_output - target) * sigmoid_deriv(out_output)
        delta_hidden = delta_output * W_hidden_output[:, 0] * sigmoid_deriv(out_hidden)
        W_hidden_output[:, 0] -= eta * delta_output * out_hidden
        b_output -= eta * delta_output
        for j in range(hidden_neurons):
            W_input_hidden[:, j] -= eta * delta_hidden[j] * X[i]
        b_hidden -= eta * delta_hidden
    if epoch % 2000 == 0:
        print(f"Epoch {epoch}, Error: {total_error}")

print("\nTraining complete.")
for i in range(len(X)):
    net_hidden = np.dot(X[i], W_input_hidden) + b_hidden
    out_hidden = sigmoid(net_hidden)
    net_output = np.dot(out_hidden, W_hidden_output) + b_output
    out_output = sigmoid(net_output)
    print(f"Input: {X[i]}, Predicted: {out_output}, Target: {y[i]}")

```

OUTPUT:

```
PS C:\Users\Parshwa\Desktop> python -u "c:\Users\Parshwa\Desktop\22510064_SC_ISE_
Epoch 0, Error: [0.35131357]
Epoch 2000, Error: [0.00268076]
Epoch 4000, Error: [0.0012226]
Epoch 6000, Error: [0.00118472]
Epoch 8000, Error: [0.00113526]

Training complete.
Input: [ 0.4 -0.7], Predicted: [0.06615524], Target: 0.10000000149011612
Input: [ 0.3 -0.5], Predicted: [0.07151205], Target: 0.05000000074505806
Input: [0.6 0.1], Predicted: [0.3032979], Target: 0.30000001192092896
Input: [0.2 0.4], Predicted: [0.23674917], Target: 0.25
Input: [0.1 0.2], Predicted: [0.13837056], Target: 0.11999999731779099
PS C:\Users\Parshwa\Desktop>
```