

EXPERIMENT-4

CODE:-

```
import math
import random

# ----- Dataset (XOR problem) -----
# Input -> Output
dataset = [
    ([0, 0], [0]),
    ([0, 1], [1]),
    ([1, 0], [1]),
    ([1, 1], [0])
]

# ----- Neural Network Parameters -----
input_size = 2
hidden_size = 2
output_size = 1
learning_rate = 0.5
epochs = 5000

# ----- Initialize Weights -----
random.seed(1)
w_input_hidden = [[random.uniform(-1, 1) for _ in range(hidden_size)] for _ in
range(input_size)]
w_hidden_output = [[random.uniform(-1, 1) for _ in range(output_size)] for _ in
range(hidden_size)]
b_hidden = [random.uniform(-1, 1) for _ in range(hidden_size)]
b_output = [random.uniform(-1, 1) for _ in range(output_size)]
```

```

# ----- Activation Function -----

def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# ----- Training using Backpropagation -----

for epoch in range(epochs):
    total_error = 0

    for inputs, target in dataset:
        # ---- Forward Pass ----

        hidden_input = [sum(inputs[i] * w_input_hidden[i][j] for i in range(input_size)) +
                        b_hidden[j]
                        for j in range(hidden_size)]

        hidden_output = [sigmoid(x) for x in hidden_input]

        final_input = [sum(hidden_output[j] * w_hidden_output[j][k] for j in
                        range(hidden_size)) + b_output[k]
                        for k in range(output_size)]

        final_output = [sigmoid(x) for x in final_input]

        # ---- Error ----

        error = [target[k] - final_output[k] for k in range(output_size)]
        total_error += sum(e ** 2 for e in error)

    # ---- Backpropagation ----

```

```
    output_delta = [error[k] * sigmoid_derivative(final_output[k]) for k in
range(output_size)]
```

```
    hidden_error = [sum(output_delta[k] * w_hidden_output[j][k] for k in
range(output_size))
                    for j in range(hidden_size)]
    hidden_delta = [hidden_error[j] * sigmoid_derivative(hidden_output[j])
                    for j in range(hidden_size)]
```

```
# ---- Update Weights and Biases ----
```

```
for j in range(hidden_size):
    for k in range(output_size):
        w_hidden_output[j][k] += learning_rate * output_delta[k] * hidden_output[j]
```

```
for i in range(input_size):
    for j in range(hidden_size):
        w_input_hidden[i][j] += learning_rate * hidden_delta[j] * inputs[i]
```

```
for k in range(output_size):
    b_output[k] += learning_rate * output_delta[k]
```

```
for j in range(hidden_size):
    b_hidden[j] += learning_rate * hidden_delta[j]
```

```
# Optional: print error every 1000 epochs
```

```
if epoch % 1000 == 0:
```

```
    print(f"Epoch {epoch}, Error: {total_error:.4f}")
```

```
# ----- Testing the Network -----
```

```
print("\nTesting ANN after Training:")
```

for inputs, _ in dataset:

```
    hidden = [sigmoid(sum(inputs[i] * w_input_hidden[i][j] for i in range(input_size)) +  
b_hidden[j])
```

```
        for j in range(hidden_size)]
```

```
    output = [sigmoid(sum(hidden[j] * w_hidden_output[j][k] for j in  
range(hidden_size)) + b_output[k])
```

```
        for k in range(output_size)]
```

```
    print(f"Input: {inputs} -> Output: {round(output[0])}")
```

OUTPUT:-

Epoch 0, Error: 1.1937

Epoch 1000, Error: 0.2256

Epoch 2000, Error: 0.0128

Epoch 3000, Error: 0.0060

Epoch 4000, Error: 0.0038

Testing ANN after Training:

Input: [0, 0] -> Output: 0

Input: [0, 1] -> Output: 1

Input: [1, 0] -> Output: 1

Input: [1, 1] -> Output: 0