BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Rishabh Santosh Shenoy
UID no.	2023300222
Experiment No.	5

AIM:	To study Supervised Learning algorithm
Program 1	
PROBLEM STATEMENT :	To implement the Supervised Learning algorithm. [BPN Algorithm]
THEORY:	Supervised learning is a machine learning approach where a model learns from labeled data, meaning both input and desired output are provided. The Backpropagation Neural Network (BPN) algorithm is one of the most widely used supervised learning techniques for training multilayer feedforward neural networks. It works by minimizing the error between the actual output and the desired output using gradient descent. The algorithm consists of two phases: the forward pass and the backward pass. In the forward pass, input data is propagated through the network using weighted connections and activation functions to generate outputs. In the backward pass, the error is calculated at the output layer and then propagated backward to adjust the weights and biases. The BPN algorithm employs activation functions such as the sigmoid or hyperbolic tangent, along with their derivatives, to introduce non-linearity and aid in efficient error correction. Weights and biases are initialized with small random values, and a learning rate controls the magnitude of updates. Using error correction factors, weights between both hidden and output layers, as well as between input and hidden layers, are updated iteratively. Through repeated epochs, the network converges towards optimal weights, thereby minimizing error and achieving accurate predictions.

PROGRAM:

```
import math
import matplotlib.pyplot as plt
import numpy as np

def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))

def deriv_from_output(y):
    return y * (1.0 - y)

v11 = 0.6
v21 = -0.1
v01 = 0.3

v12 = -0.3
v22 = 0.4
v02 = 0.5

w1 = 0.4
w2 = 0.1
w0 = -0.2

x1, x2 = 0.0, 1.0
target = 1.0
alpha = 0.25
epochs = 10

fmt = lambda v: f"{v:.6f}"

epoch_list, error_list, grad_list = [], [], []

for epoch in range(1, epochs + 1):
    zin1 = v01 + x1 * v11 + x2 * v21
    z1 = sigmoid(zin1)

    zin2 = v02 + x1 * v12 + x2 * v22
    z2 = sigmoid(zin2)
```

```
    yin = w0 + z1 * w1 + z2 * w2
    y = sigmoid(yin)

    error = target - y
    fprime_yin = deriv_from_output(y)
    delta_k = error * fprime_yin

    dw1 = alpha * delta_k * z1
    dw2 = alpha * delta_k * z2
    dw0 = alpha * delta_k * 1.0

    delta_in1 = delta_k * w1
    fprime_zin1 = deriv_from_output(z1)
    delta1 = delta_in1 * fprime_zin1

    delta_in2 = delta_k * w2
    fprime_zin2 = deriv_from_output(z2)
    delta2 = delta_in2 * fprime_zin2

    dv11 = alpha * delta1 * x1
    dv21 = alpha * delta1 * x2
    dv01 = alpha * delta1 * 1.0

    dv12 = alpha * delta2 * x1
    dv22 = alpha * delta2 * x2
    dv02 = alpha * delta2 * 1.0

    print("="*72)
    print(f"Epoch {epoch}")
    print("- Forward pass -")
    print(f"  Inputs (x1, x2) = ({fmt(x1)}, {fmt(x2)}), Target = {fmt(target)}")
    print(f"  Net input z_in1 = v01 + x1*v11 + x2*v21 = {fmt(v01)} + {fmt(x1)}*{fmt(v11)} + {fmt(x2)}*{fmt(v21)} = {fmt(zin1)}")
    print(f"  z1 = f(z_in1) = {fmt(z1)}")
    print(f"  Net input z_in2 = v02 + x1*v12 + x2*v22 = {fmt(v02)} + {fmt(x1)}*{fmt(v12)} + {fmt(x2)}*{fmt(v22)} = {fmt(zin2)}")
    print(f"  z2 = f(z_in2) = {fmt(z2)}")
```

```
    print(f"  Net input y_in = w0 + z1*w1 + z2*w2 = {fmt(w0)} +
{fmt(z1)}*{fmt(w1)} + {fmt(z2)}*{fmt(w2)} = {fmt(yin)}")
    print(f"  Output y = f(y_in) = {fmt(y)}")
    print()
    print("- Error and deltas -")
    print(f"  Error (t - y) = {fmt(error)}")
    print(f"  f'(y_in) = y*(1-y) = {fmt(fprime_yin)}")
    print(f"  delta_k (output) = (t - y) * f'(y_in) = {fmt(delta_k)}")
    print()
    print("- Changes in weights (hidden -> output) -")
    print(f"  Δw1 = {fmt(dw1)}, Δw2 = {fmt(dw2)}, Δw0 = {fmt(dw0)}")
    print()
    print("- Backprop to hidden layer -")
    print(f"  δ1 = {fmt(delta1)}, δ2 = {fmt(delta2)}")
    print()
    print("- Changes in weights (input -> hidden) -")
    print(f"  Δv11 = {fmt(dv11)}, Δv21 = {fmt(dv21)}, Δv01 =
{fmt(dv01)}")
    print(f"  Δv12 = {fmt(dv12)}, Δv22 = {fmt(dv22)}, Δv02 =
{fmt(dv02)}")
    print()

    grad_mag = abs(dw1) + abs(dw2) + abs(dw0) + abs(dv11) + abs(dv21) +
abs(dv01) + abs(dv12) + abs(dv22) + abs(dv02)
    epoch_list.append(epoch)
    error_list.append(abs(error))
    grad_list.append(grad_mag)

    w1 += dw1
    w2 += dw2
    w0 += dw0

    v11 += dv11
    v21 += dv21
    v01 += dv01

    v12 += dv12
    v22 += dv22
```

```
    v02 += dv02

    print("- Updated weights (after epoch) -")
    print(f"  w1 = {fmt(w1)}, w2 = {fmt(w2)}, w0 = {fmt(w0)}")
    print(f"  v11 = {fmt(v11)}, v21 = {fmt(v21)}, v01 = {fmt(v01)}")
    print(f"  v12 = {fmt(v12)}, v22 = {fmt(v22)}, v02 = {fmt(v02)}")
    print("="*72 + "\n\n")

print("\nFinal Error and Gradient Table:")
print("Epoch | Error      | Gradient Magnitude")
print("-----------------------------------------")
for e, err, g in zip(epoch_list, error_list, grad_list):
    print(f"{e:5d} | {err:.6f} | {g:.6f}")

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(epoch_list, error_list, marker='o')
plt.title("Error vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Error (|t-y|)")

plt.subplot(1,2,2)
f = lambda w: 0.5 * w**2
w = np.linspace(-3, 3, 400)
E = f(w)

eta = 0.3
w_curr = -2.5
steps = 8
w_hist = [w_curr]
E_hist = [f(w_curr)]
for _ in range(steps):
    grad = w_curr
    w_curr = w_curr - eta * grad
    w_hist.append(w_curr)
    E_hist.append(f(w_curr))
```

```
plt.plot(w, E, label="Error Surface E(w)", color="blue")
plt.plot(w_hist, E_hist, 'ro--', label="Descent Path")
plt.title("Conceptual Gradient Descent (Error vs Weight)")
plt.xlabel("Weight (w)")
plt.ylabel("Error E(w)")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

RESULT:

```
========================================================================
Epoch 1
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.300000 + 0.000000*0.600000 + 1.000000*-0.100000 = 0.200000
  z1 = f(z_in1) = 0.549834
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.500000 + 0.000000*-0.300000 + 1.000000*0.400000 = 0.900000
  z2 = f(z_in2) = 0.710950
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.200000 + 0.549834*0.400000 + 0.710950*0.100000 = 0.091029
  Output y = f(y_in) = 0.522741

- Error and deltas -
  Error (t - y) = 0.477259
  f'(y_in) = y*(1-y) = 0.249483
  delta_k (output) = (t - y) * f'(y_in) = 0.119068

- Changes in weights (hidden -> output) -
  Δw1 = 0.016367, Δw2 = 0.021163, Δw0 = 0.029767

- Backprop to hidden layer -
  δ1 = 0.011789, δ2 = 0.002447

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002947, Δv01 = 0.002947
  Δv12 = 0.000000, Δv22 = 0.000612, Δv02 = 0.000612

- Updated weights (after epoch) -
  w1 = 0.416367, w2 = 0.121163, w0 = -0.170233
  v11 = 0.600000, v21 = -0.097053, v01 = 0.302947
  v12 = -0.300000, v22 = 0.400612, v02 = 0.500612
========================================================================
```

```
========================================================================
Epoch 2
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.302947 + 0.000000*0.600000 + 1.000000*-0.097053 = 0.205894
  z1 = f(z_in1) = 0.551292
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.500612 + 0.000000*-0.300000 + 1.000000*0.400612 = 0.901223
  z2 = f(z_in2) = 0.711201
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.170233 + 0.551292*0.416367 + 0.711201*0.121163 = 0.145478
  Output y = f(y_in) = 0.536305

- Error and deltas -
  Error (t - y) = 0.463695
  f'(y_in) = y*(1-y) = 0.248682
  delta_k (output) = (t - y) * f'(y_in) = 0.115312

- Changes in weights (hidden -> output) -
  Δw1 = 0.015893, Δw2 = 0.020503, Δw0 = 0.028828

- Backprop to hidden layer -
  δ1 = 0.011877, δ2 = 0.002870

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002969, Δv01 = 0.002969
  Δv12 = 0.000000, Δv22 = 0.000717, Δv02 = 0.000717

- Updated weights (after epoch) -
  w1 = 0.432260, w2 = 0.141665, w0 = -0.141405
  v11 = 0.600000, v21 = -0.094084, v01 = 0.305916
  v12 = -0.300000, v22 = 0.401329, v02 = 0.501329
========================================================================
```

```
========================================================================
Epoch 3
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.305916 + 0.000000*0.600000 + 1.000000*-0.094084 = 0.211833
  z1 = f(z_in1) = 0.552761
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.501329 + 0.000000*-0.300000 + 1.000000*0.401329 = 0.902658
  z2 = f(z_in2) = 0.711495
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.141405 + 0.552761*0.432260 + 0.711495*0.141665 = 0.198326
  Output y = f(y_in) = 0.549420

- Error and deltas -
  Error (t - y) = 0.450580
  f'(y_in) = y*(1-y) = 0.247558
  delta_k (output) = (t - y) * f'(y_in) = 0.111545

- Changes in weights (hidden -> output) -
  Δw1 = 0.015414, Δw2 = 0.019841, Δw0 = 0.027886

- Backprop to hidden layer -
  δ1 = 0.011920, δ2 = 0.003244

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002980, Δv01 = 0.002980
  Δv12 = 0.000000, Δv22 = 0.000811, Δv02 = 0.000811

- Updated weights (after epoch) -
  w1 = 0.447674, w2 = 0.161506, w0 = -0.113519
  v11 = 0.600000, v21 = -0.091104, v01 = 0.308896
  v12 = -0.300000, v22 = 0.402140, v02 = 0.502140
========================================================================
```

```
========================================================================
Epoch 4
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.308896 + 0.000000*0.600000 + 1.000000*-0.091104 = 0.217793
  z1 = f(z_in1) = 0.554234
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.502140 + 0.000000*-0.300000 + 1.000000*0.402140 = 0.904280
  z2 = f(z_in2) = 0.711828
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.113519 + 0.554234*0.447674 + 0.711828*0.161506 = 0.249562
  Output y = f(y_in) = 0.562069

- Error and deltas -
  Error (t - y) = 0.437931
  f'(y_in) = y*(1-y) = 0.246147
  delta_k (output) = (t - y) * f'(y_in) = 0.107796

- Changes in weights (hidden -> output) -
  Δw1 = 0.014936, Δw2 = 0.019183, Δw0 = 0.026949

- Backprop to hidden layer -
  δ1 = 0.011922, δ2 = 0.003571

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002981, Δv01 = 0.002981
  Δv12 = 0.000000, Δv22 = 0.000893, Δv02 = 0.000893

- Updated weights (after epoch) -
  w1 = 0.462610, w2 = 0.180689, w0 = -0.086570
  v11 = 0.600000, v21 = -0.088123, v01 = 0.311877
  v12 = -0.300000, v22 = 0.403033, v02 = 0.503033
========================================================================
```

```
========================================================================
Epoch 5
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.311877 + 0.000000*0.600000 + 1.000000*-0.088123 = 0.223754
  z1 = f(z_in1) = 0.555706
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.503033 + 0.000000*-0.300000 + 1.000000*0.403033 = 0.906066
  z2 = f(z_in2) = 0.712194
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.086570 + 0.555706*0.462610 + 0.712194*0.180689 = 0.299191
  Output y = f(y_in) = 0.574245

- Error and deltas -
  Error (t - y) = 0.425755
  f'(y_in) = y*(1-y) = 0.244488
  delta_k (output) = (t - y) * f'(y_in) = 0.104092

- Changes in weights (hidden -> output) -
  Δw1 = 0.014461, Δw2 = 0.018533, Δw0 = 0.026023

- Backprop to hidden layer -
  δ1 = 0.011889, δ2 = 0.003855

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002972, Δv01 = 0.002972
  Δv12 = 0.000000, Δv22 = 0.000964, Δv02 = 0.000964

- Updated weights (after epoch) -
  w1 = 0.477071, w2 = 0.199223, w0 = -0.060547
  v11 = 0.600000, v21 = -0.085151, v01 = 0.314849
  v12 = -0.300000, v22 = 0.403997, v02 = 0.503997
========================================================================
```

```
=========================================================================
Epoch 6
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.314849 + 0.000000*0.600000 + 1.000000*-0.085151 = 0.229698
  z1 = f(z_in1) = 0.557173
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.503997 + 0.000000*-0.300000 + 1.000000*0.403997 = 0.907993
  z2 = f(z_in2) = 0.712589
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.060547 + 0.557173*0.477071 + 0.712589*0.199223 = 0.347228
  Output y = f(y_in) = 0.585945

- Error and deltas -
  Error (t - y) = 0.414055
  f'(y_in) = y*(1-y) = 0.242613
  delta_k (output) = (t - y) * f'(y_in) = 0.100455

- Changes in weights (hidden -> output) -
  Δw1 = 0.013993, Δw2 = 0.017896, Δw0 = 0.025114

- Backprop to hidden layer -
  δ1 = 0.011824, δ2 = 0.004099

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002956, Δv01 = 0.002956
  Δv12 = 0.000000, Δv22 = 0.001025, Δv02 = 0.001025

- Updated weights (after epoch) -
  w1 = 0.491064, w2 = 0.217119, w0 = -0.035433
  v11 = 0.600000, v21 = -0.082195, v01 = 0.317805
  v12 = -0.300000, v22 = 0.405021, v02 = 0.505021
=========================================================================
```

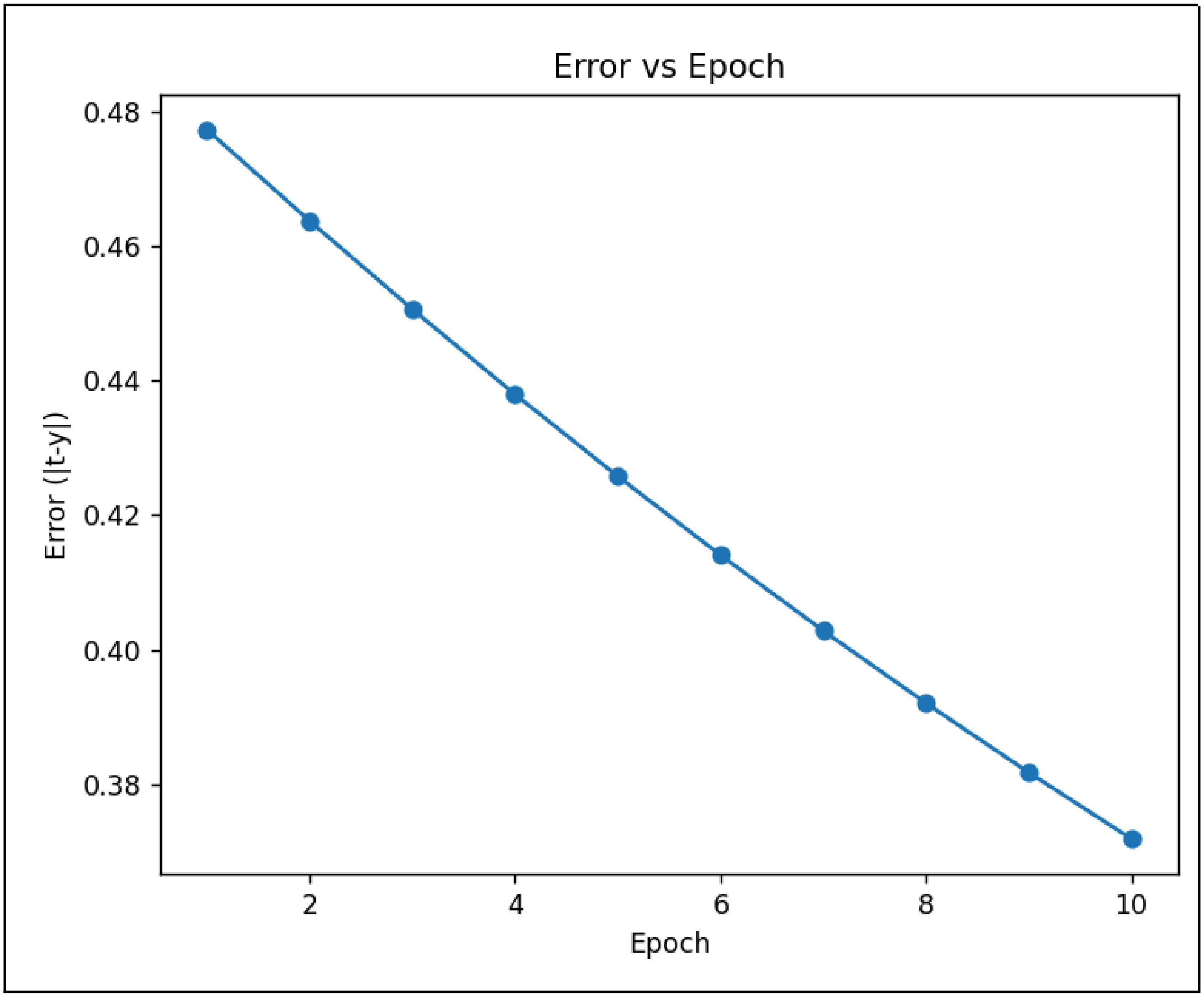
```
=========================================================================
Epoch 7
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.317805 + 0.000000*0.600000 + 1.000000*-0.082195 = 0.235610
  z1 = f(z_in1) = 0.558632
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.505021 + 0.000000*-0.300000 + 1.000000*0.405021 = 0.910043
  z2 = f(z_in2) = 0.713009
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.035433 + 0.558632*0.491064 + 0.713009*0.217119 = 0.393698
  Output y = f(y_in) = 0.597173

- Error and deltas -
  Error (t - y) = 0.402827
  f'(y_in) = y*(1-y) = 0.240557
  delta_k (output) = (t - y) * f'(y_in) = 0.096903

- Changes in weights (hidden -> output) -
  Δw1 = 0.013533, Δw2 = 0.017273, Δw0 = 0.024226

- Backprop to hidden layer -
  δ1 = 0.011733, δ2 = 0.004305

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002933, Δv01 = 0.002933
  Δv12 = 0.000000, Δv22 = 0.001076, Δv02 = 0.001076

- Updated weights (after epoch) -
  w1 = 0.504597, w2 = 0.234392, w0 = -0.011207
  v11 = 0.600000, v21 = -0.079262, v01 = 0.320738
  v12 = -0.300000, v22 = 0.406098, v02 = 0.506098
=========================================================================
```

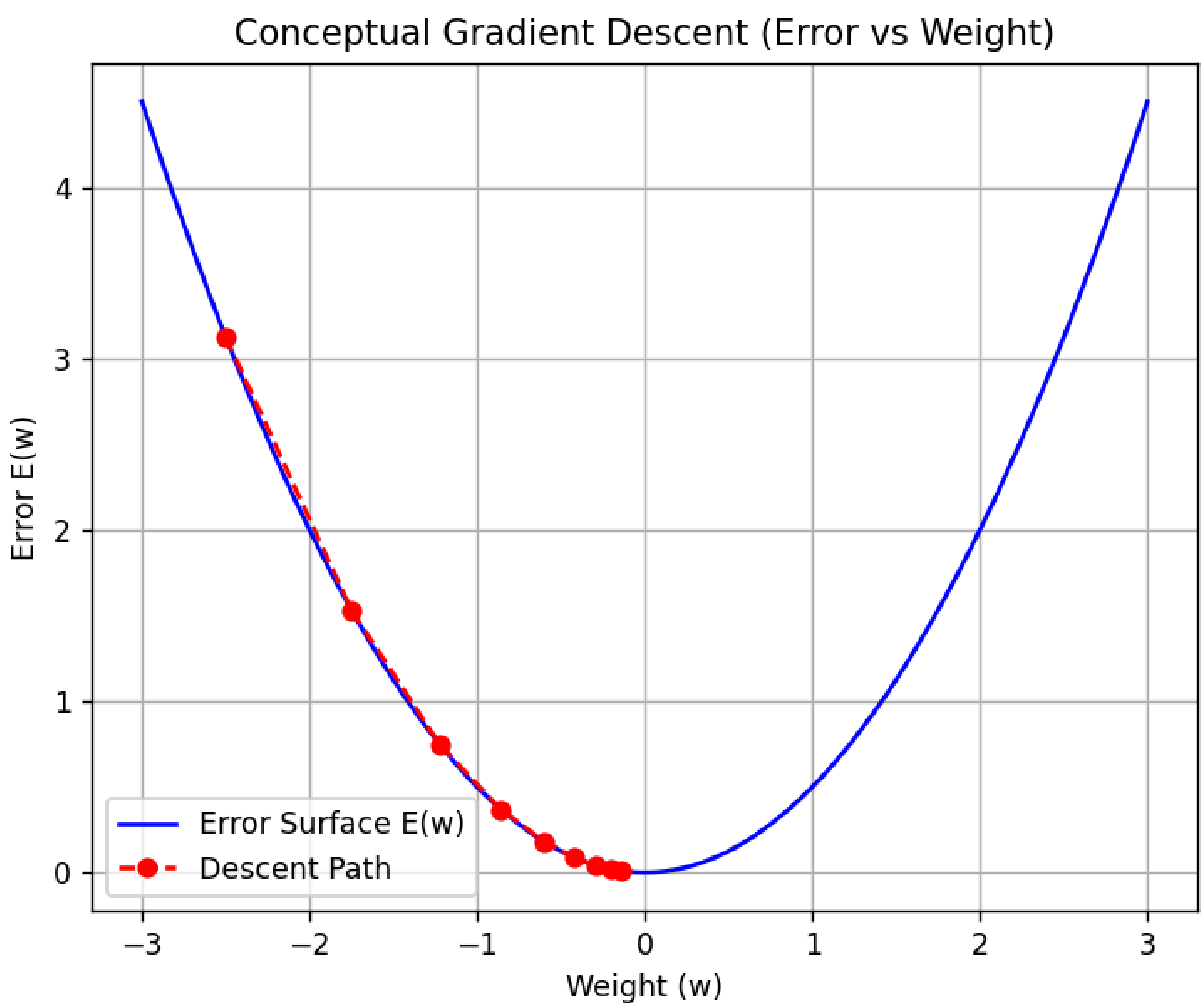
```
========================================================================
Epoch 8
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.320738 + 0.000000*0.600000 + 1.000000*-0.079262 = 0.241477
  z1 = f(z_in1) = 0.560078
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.506098 + 0.000000*-0.300000 + 1.000000*0.406098 = 0.912195
  z2 = f(z_in2) = 0.713449
  Net input y_in = w0 + z1*w1 + z2*w2 = -0.011207 + 0.560078*0.504597 + 0.713449*0.234392 = 0.438633
  Output y = f(y_in) = 0.607933

- Error and deltas -
  Error (t - y) = 0.392067
  f'(y_in) = y*(1-y) = 0.238350
  delta_k (output) = (t - y) * f'(y_in) = 0.093449

- Changes in weights (hidden -> output) -
  Δw1 = 0.013085, Δw2 = 0.016668, Δw0 = 0.023362

- Backprop to hidden layer -
  δ1 = 0.011618, δ2 = 0.004478

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002905, Δv01 = 0.002905
  Δv12 = 0.000000, Δv22 = 0.001119, Δv02 = 0.001119

- Updated weights (after epoch) -
  w1 = 0.517682, w2 = 0.251060, w0 = 0.012155
  v11 = 0.600000, v21 = -0.076357, v01 = 0.323643
  v12 = -0.300000, v22 = 0.407217, v02 = 0.507217
========================================================================
```

```
========================================================================
Epoch 9
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.323643 + 0.000000*0.600000 + 1.000000*-0.076357 = 0.247286
  z1 = f(z_in1) = 0.561508
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.507217 + 0.000000*-0.300000 + 1.000000*0.407217 = 0.914434
  z2 = f(z_in2) = 0.713907
  Net input y_in = w0 + z1*w1 + z2*w2 = 0.012155 + 0.561508*0.517682 + 0.713907*0.251060 = 0.482071
  Output y = f(y_in) = 0.618237

- Error and deltas -
  Error (t - y) = 0.381763
  f'(y_in) = y*(1-y) = 0.236020
  delta_k (output) = (t - y) * f'(y_in) = 0.090104

- Changes in weights (hidden -> output) -
  Δw1 = 0.012649, Δw2 = 0.016081, Δw0 = 0.022526

- Backprop to hidden layer -
  δ1 = 0.011485, δ2 = 0.004620

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002871, Δv01 = 0.002871
  Δv12 = 0.000000, Δv22 = 0.001155, Δv02 = 0.001155

- Updated weights (after epoch) -
  w1 = 0.530330, w2 = 0.267141, w0 = 0.034681
  v11 = 0.600000, v21 = -0.073486, v01 = 0.326514
  v12 = -0.300000, v22 = 0.408372, v02 = 0.508372
========================================================================
```

```
========================================================================
Epoch 10
- Forward pass -
  Inputs (x1, x2) = (0.000000, 1.000000), Target = 1.000000
  Net input z_in1 = v01 + x1*v11 + x2*v21 = 0.326514 + 0.000000*0.600000 + 1.000000*-0.073486 = 0.253028
  z1 = f(z_in1) = 0.562922
  Net input z_in2 = v02 + x1*v12 + x2*v22 = 0.508372 + 0.000000*-0.300000 + 1.000000*0.408372 = 0.916744
  z2 = f(z_in2) = 0.714378
  Net input y_in = w0 + z1*w1 + z2*w2 = 0.034681 + 0.562922*0.530330 + 0.714378*0.267141 = 0.524055
  Output y = f(y_in) = 0.628096

- Error and deltas -
  Error (t - y) = 0.371904
  f'(y_in) = y*(1-y) = 0.233592
  delta_k (output) = (t - y) * f'(y_in) = 0.086874

- Changes in weights (hidden -> output) -
  Δw1 = 0.012226, Δw2 = 0.015515, Δw0 = 0.021718

- Backprop to hidden layer -
  δ1 = 0.011336, δ2 = 0.004735

- Changes in weights (input -> hidden) -
  Δv11 = 0.000000, Δv21 = 0.002834, Δv01 = 0.002834
  Δv12 = 0.000000, Δv22 = 0.001184, Δv02 = 0.001184

- Updated weights (after epoch) -
  w1 = 0.542556, w2 = 0.282656, w0 = 0.056399
  v11 = 0.600000, v21 = -0.070652, v01 = 0.329348
  v12 = -0.300000, v22 = 0.409556, v02 = 0.509556
========================================================================
```

```
Final Error and Gradient Table:
Epoch | Error       | Gradient Magnitude
----------------------------------------
    1 | 0.477259 | 0.074414
    2 | 0.463695 | 0.072597
    3 | 0.450580 | 0.070723
    4 | 0.437931 | 0.068815
    5 | 0.425755 | 0.066890
    6 | 0.414055 | 0.064964
    7 | 0.402827 | 0.063051
    8 | 0.392067 | 0.061163
    9 | 0.381763 | 0.059308
   10 | 0.371904 | 0.057495
```


Error vs Epoch
Error (|t-y|)
0.48
0.46
0.44
0.42
0.40
0.38
2
4
6
8
10
Epoch



DATASET STUDY:	We demonstrate these steps on the Breast Cancer Wisconsin dataset (binary classification). The dataset (569 samples, 30 features) is available via scikit-learn (originally from UCI/Kaggle datasetsearch.research.google.com). We first normalize the inputs to [0,1] range for stable sigmoid operation, then train a network with 5 hidden neurons:
PROGRAM:	from sklearn.datasets import load_breast_cancer import numpy as np import matplotlib.pyplot as plt data = load_breast_cancer() X = data.data

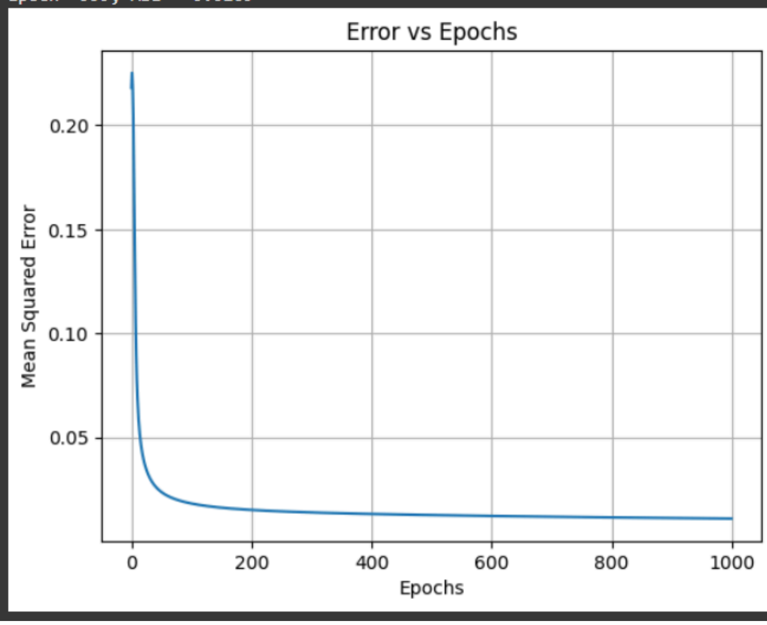
```
y = data.target
X_min = X.min(axis=0); X_max = X.max(axis=0)
X_norm = (X - X_min) / (X_max - X_min)

n_input  = X_norm.shape[1]
n_hidden = 5
n_output = 1
np.random.seed(42)
W_input_hidden  = np.random.normal(scale=0.1, size=(n_input,
n_hidden))
b_hidden        = np.zeros(n_hidden)
W_hidden_output = np.random.normal(scale=0.1, size=(n_hidden,
n_output))
b_output        = np.zeros(n_output)

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))
def sigmoid_derivative(y):
    return y * (1.0 - y)

learning_rate = 0.1
epochs = 1000
loss_history = []

for epoch in range(epochs):
    total_loss = 0.0
    for i in range(X_norm.shape[0]):
        x_i = X_norm[i]
        t_i = y[i]

        z_hidden = np.dot(x_i, W_input_hidden) + b_hidden
        a_hidden = sigmoid(z_hidden)
        z_out    = np.dot(a_hidden, W_hidden_output) + b_output
        a_out    = sigmoid(z_out)

        error = t_i - a_out[0]
        total_loss += error**2
```

	`` delta_out = error * sigmoid_derivative(a_out)[0] grad_hidden_output = np.expand_dims(a_hidden,1) * delta_out W_hidden_output += learning_rate * grad_hidden_output b_output += learning_rate * delta_out delta_hidden = (W_hidden_output.flatten() * delta_out) * sigmoid_derivative(a_hidden) grad_input_hidden = np.outer(x_i, delta_hidden) W_input_hidden += learning_rate * grad_input_hidden b_hidden += learning_rate * delta_hidden mse = total_loss / X_norm.shape[0] loss_history.append(mse) if epoch % 100 == 0 or epoch == epochs-1: print(f"Epoch {epoch:4d}, MSE = {mse:.4f}") plt.plot(range(epochs), loss_history) plt.xlabel("Epochs") plt.ylabel("Mean Squared Error") plt.title("Error vs Epochs") plt.grid(True) plt.show() ```
PROGRAM-2:	``` X_xor = np.array([[0,0],[0,1],[1,0],[1,1]]) y_xor = np.array([0,1,1,0]) np.random.seed(1) W_input_hidden = np.random.uniform(-1,1,(2, 2)) b_hidden = np.random.uniform(-1,1,2) W_hidden_output = np.random.uniform(-1,1,(2, 1)) b_output = np.random.uniform(-1,1,1) def sigmoid(x): return 1/(1+np.exp(-x)) def sigmoid_derivative(y): return y*(1-y) learning_rate = 0.5 epochs = 10000 ```

```
for epoch in range(epochs):
    total_loss = 0.0
    for i in range(X_xor.shape[0]):
        x_i = X_xor[i]
        t_i = y_xor[i]

        z_hidden = np.dot(x_i, W_input_hidden) + b_hidden
        a_hidden = sigmoid(z_hidden)
        z_out    = np.dot(a_hidden, W_hidden_output) + b_output
        a_out    = sigmoid(z_out)

        error = t_i - a_out[0]
        total_loss += error**2
        delta_out = error * sigmoid_derivative(a_out)[0]

        grad_hidden_output = np.expand_dims(a_hidden,1) * delta_out
        W_hidden_output += learning_rate * grad_hidden_output
        b_output        += learning_rate * delta_out

        delta_hidden = (W_hidden_output.flatten() * delta_out) *
sigmoid_derivative(a_hidden)
        grad_input_hidden = np.outer(x_i, delta_hidden)
        W_input_hidden += learning_rate * grad_input_hidden
        b_hidden       += learning_rate * delta_hidden

    if epoch % 2000 == 0:
        mse = total_loss / X_xor.shape[0]
        print(f"Epoch {epoch:4d}, MSE = {mse:.6f}")

print("\nFinal network outputs for XOR:")
for x_i, t_i in zip(X_xor, y_xor):
    a_hidden = sigmoid(np.dot(x_i, W_input_hidden) + b_hidden)
    a_out    = sigmoid(np.dot(a_hidden, W_hidden_output) + b_output)[0]
    print(f"Input {x_i.tolist()} -> Predicted {a_out:.3f}, Target {t_i}")
```

OUTPUT 1:

```
Epoch    0, MSE = 0.2181
Epoch  100, MSE = 0.0181
Epoch  200, MSE = 0.0151
Epoch  300, MSE = 0.0139
Epoch  400, MSE = 0.0131
Epoch  500, MSE = 0.0126
Epoch  600, MSE = 0.0122
Epoch  700, MSE = 0.0118
Epoch  800, MSE = 0.0115
Epoch  900, MSE = 0.0112
Epoch  999, MSE = 0.0109
```



OUTPUT 2:

```
Epoch    0, MSE = 0.269411
Epoch 2000, MSE = 0.004854
Epoch 4000, MSE = 0.001108
Epoch 6000, MSE = 0.000607
Epoch 8000, MSE = 0.000414

Final network outputs for XOR:
Input [0, 0] -> Predicted 0.016, Target 0
Input [0, 1] -> Predicted 0.983, Target 1
Input [1, 0] -> Predicted 0.983, Target 1
Input [1, 1] -> Predicted 0.021, Target 0
```

CONCLUSION:

The experiment on implementing the Backpropagation Neural Network (BPN) highlights the effectiveness of supervised learning in training multilayer neural networks. By systematically propagating inputs forward and errors backward, the algorithm adjusts weights and biases to minimize

	the difference between predicted and actual outputs. This iterative process ensures that the network gradually improves its ability to recognize patterns and relationships within the dataset. The use of activation functions introduces non-linearity, enabling the model to handle complex problems beyond linear separability. Overall, the BPN algorithm demonstrates how supervised learning can be applied to achieve accurate predictions and classification results. The step-by-step computation of net inputs, activations, error correction factors, and weight updates reinforces the importance of mathematical rigor in neural network training. The experiment confirms that with proper learning rate and sufficient iterations, the network converges towards optimal weights, showcasing the practical significance of backpropagation in modern artificial intelligence applications.