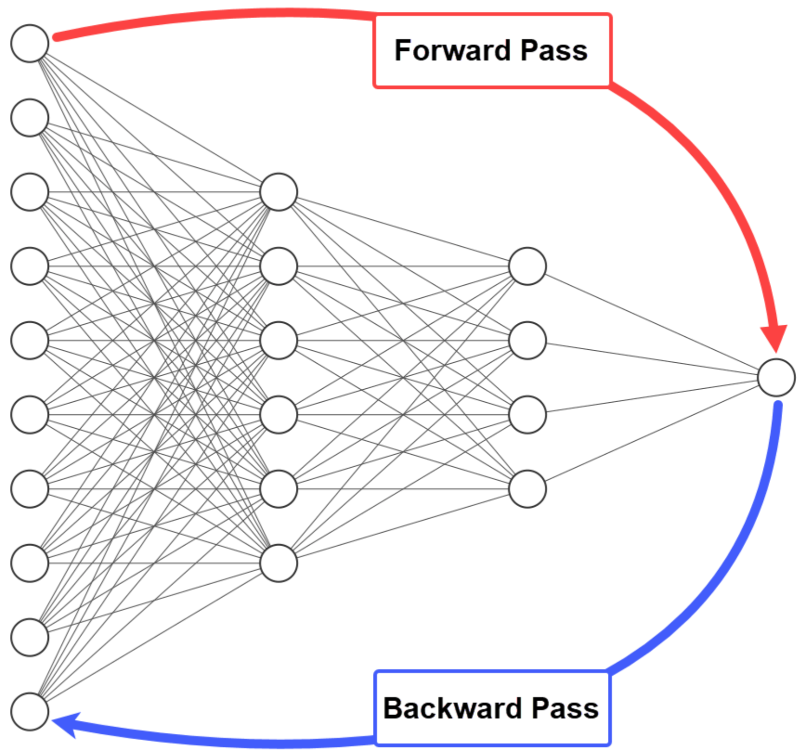
Fuzzy Logic & Neural Networks (CS-514)

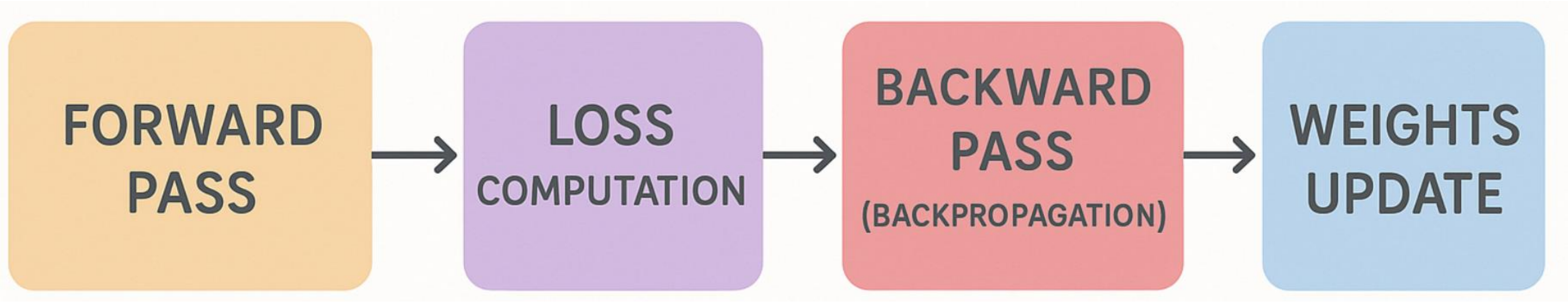
Dr. Sudeep Sharma

IIIT Surat

sudeep.sharma@iiitsurat.ac.in

Backpropagation Implementation





Source: Internet

Backpropagation Implementation

- **Importing the required libraries:**

- To import necessary libraries in Python, the import statement is used.

- Import numpy as np imports the numpy (For numerical operations and array manipulation) library and renames it to np.

- Importing the Matplotlib library for creating plots and visualizations

Backpropagation Implementation

- **Importing the required libraries:**

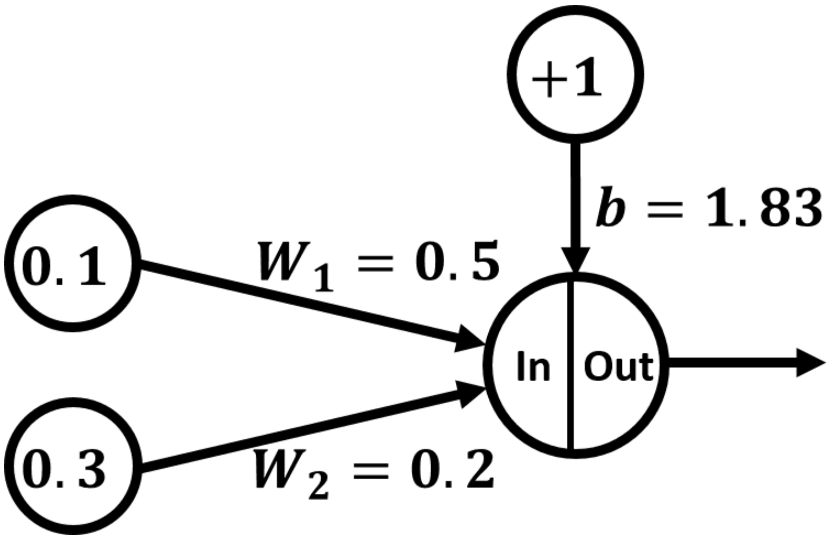
```python
import numpy as np
import matplotlib.pyplot as plt
```

Backpropagation Implementation

- A single training sample example:

X1	X2	Desired Output
0.1	0.3	0.1

W1	W2	b
0.5	0.2	1.83



Source: Internet

Backpropagation Implementation

- A single training sample example:

X1	X2	Desired Output
0.1	0.3	0.1

W1	W2	b
0.5	0.2	1.83

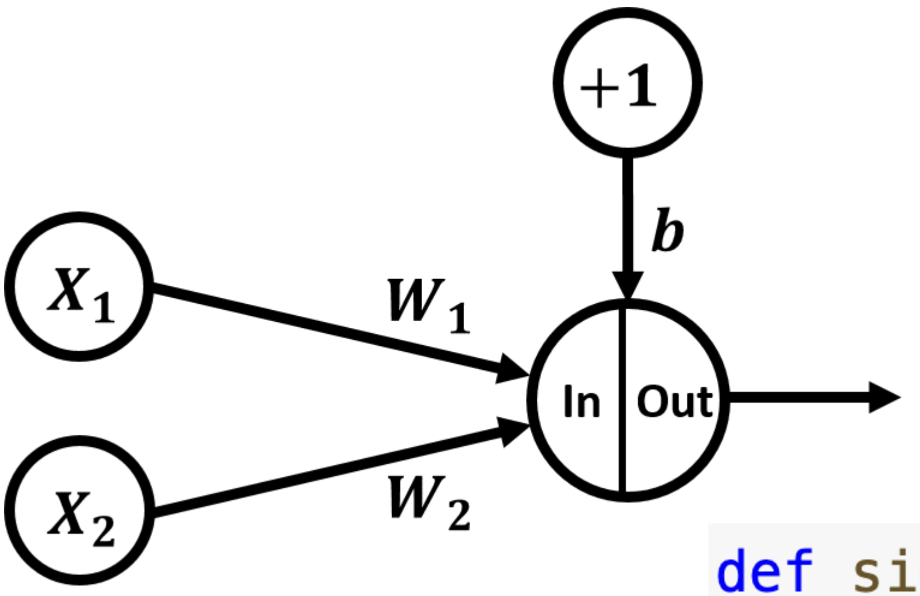
```
x1=0.1
x2=0.3

target = 0.1
```

```
w1 = 0.5
w2 = 0.2
b = 1.83
```

Source: Internet

Backpropagation Implementation

- **Forward Pass:**



$$\hat{y}_i = \frac{1}{(1 + e^{-sop})}$$

```
def sigmoid(sop):
    return 1.0/(1+np.exp(-1*sop))
```

- where

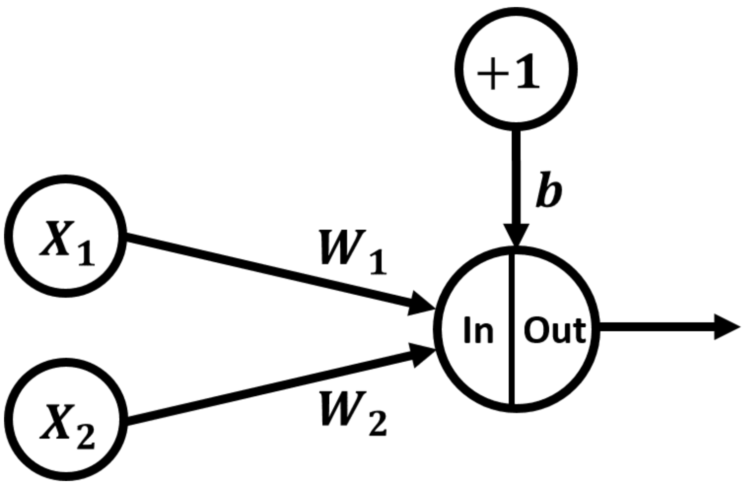
sop = X_1* W_1+ X_2*W_2+b

```
sop = w1*x1 + w2*x2 + b
```

Source: Internet

Backpropagation Implementation

- **Loss Calculations:**



$$Loss = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

```
predicted = sigmoid(sop)
```

```
def loss_mse(predicted, target):
    return 0.5*np.power(target - predicted, 2)
```

```
loss = loss_mse(predicted, target)
```

Source: Internet

Backpropagation Implementation

- **Backward Pass**

Gradient Calculations:

$$Loss = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$\frac{\partial Loss}{\partial \hat{y}_i} = -\frac{(y_i - \hat{y}_i)}{N}$$

```
def loss_predicted_deriv(predicted, target):
    return (predicted-target)
```

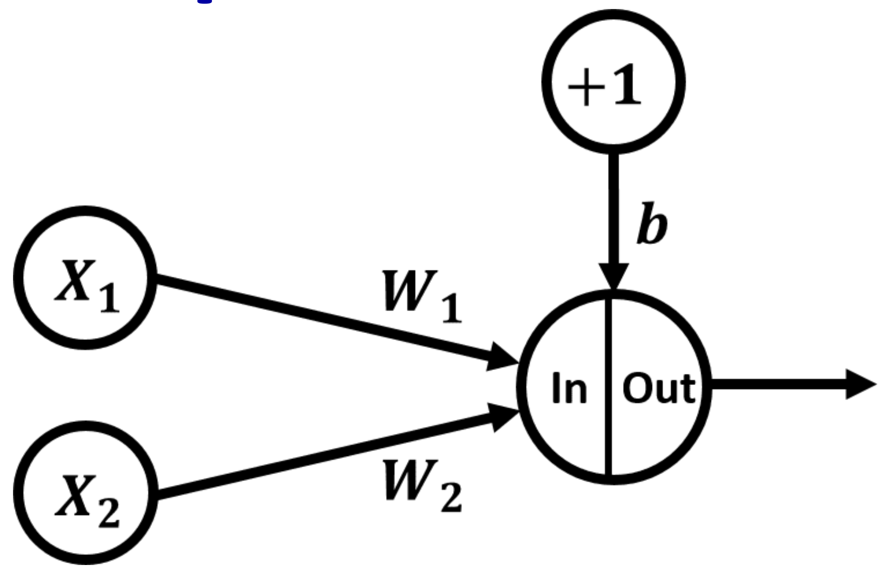
```
g1 = loss_predicted_deriv(predicted, target)
```

Backpropagation Implementation

- Gradient Calculations:

$$Loss = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$\frac{\partial Loss}{\partial \hat{y}_i} = -\frac{(y_i - \hat{y}_i)}{N}$$



$$\frac{\partial \hat{y}_i}{\partial s} = \frac{\partial}{\partial s}\hat{y}_i = \frac{\partial}{\partial s}\frac{1}{(1 + e^{-s})} = \hat{y}_i(1 - \hat{y}_i)$$

```
def sigmoid_sop_deriv(sop):
    return sigmoid(sop)*(1.0-sigmoid(sop))

g2 = sigmoid_sop_deriv(sop)
```

Source: Internet

Backpropagation Implementation

- Gradient Calculations:

$$\frac{\partial \hat{y}_i}{\partial s} = \frac{\partial}{\partial s}\hat{y}_i = \frac{\partial}{\partial s}\frac{1}{(1 + e^{-s})} = \hat{y}_i(1 - \hat{y}_i)$$

$$\frac{\partial s}{\partial w_1} = \frac{\partial}{\partial w_1}(X_1 w_1 + X_2 w_2 + b) = X_1$$

```
def sop_w_deriv(x):
    return x
```

```
g3w1 = sop_w_deriv(x1)
g3w2 = sop_w_deriv(x2)
```

Source: Internet

Backpropagation Implementation

- Gradient Calculations:

$$\frac{\partial Loss}{\partial \hat{y}_i} = -\frac{(y_i - \hat{y}_i)}{N}$$

$$\frac{\partial \hat{y}_i}{\partial s} = \frac{\partial}{\partial s}\hat{y}_i = \frac{\partial}{\partial s}\frac{1}{(1 + e^{-s})} = \hat{y}_i(1 - \hat{y}_i)$$

$$\frac{\partial s}{\partial w_1} = \frac{\partial}{\partial w_1}(X_1 w_1 + X_2 w_2 + b) = X_1$$

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s}\frac{\partial s}{\partial w_1} = -\frac{(y_i - \hat{y}_i)}{N}\hat{y}_i(1 - \hat{y}_i)X_1$$

```
gradw1 = g3w1*g2*g1
```

Source: Internet

Backpropagation Implementation

- Gradient Calculations:

$$\frac{\partial Loss}{\partial \hat{y}_i} = -\frac{(y_i - \hat{y}_i)}{N}$$

$$\frac{\partial \hat{y}_i}{\partial s} = \frac{\partial}{\partial s}\hat{y}_i = \frac{\partial}{\partial s}\frac{1}{(1 + e^{-s})} = \hat{y}_i(1 - \hat{y}_i)$$

$$\frac{\partial s}{\partial w_2} = \frac{\partial}{\partial w_2}(X_1 w_1 + X_2 w_2 + b) = X_2$$

$$\frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s}\frac{\partial s}{\partial w_2} = -\frac{(y_i - \hat{y}_i)}{N}\hat{y}_i(1 - \hat{y}_i)X_2$$

```
gradw2 = g3w2*g2*g1
```

Source: Internet

Backpropagation Implementation

- Gradient Calculations:

$$\frac{\partial Loss}{\partial \hat{y}_i} = -\frac{(y_i - \hat{y}_i)}{N}$$

$$\frac{\partial \hat{y}_i}{\partial s} = \frac{\partial}{\partial s}\hat{y}_i = \frac{\partial}{\partial s}\frac{1}{(1+e^{-s})} = \hat{y}_i(1-\hat{y}_i)$$

$$\frac{\partial s}{\partial b} = \frac{\partial}{\partial b}(X_1 w_1 + X_2 w_2 + b) = 1$$

$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s}\frac{\partial s}{\partial b} = -\frac{(y_i - \hat{y}_i)}{N}\hat{y}_i(1-\hat{y}_i)$$

```
gradb = g2*g1
```

Source: Internet

Backpropagation Implementation

- Weights Update Rule:

$$w(new) = w(old) - \eta \frac{\partial Loss}{\partial w}$$

η is the learning rate

```
def update_w(w, grad, learning_rate):
    return w - learning_rate*grad
```

Backpropagation Implementation

- Weights Update Rule:

$$w_1(new) = w_1(old) - \eta \frac{\partial Loss}{\partial w_1}$$

$$w_2(new) = w_2(old) - \eta \frac{\partial Loss}{\partial w_2}$$

$$b(new) = b(old) - \eta \frac{\partial Loss}{\partial b}$$

```
w1 = update_w(w1, gradw1, learning_rate)
w2 = update_w(w2, gradw2, learning_rate)
b = update_w(b, gradb, learning_rate)
```

Backpropagation Implementation

Full Implementation

```
# Required Libraries

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

Source: Internet

Backpropagation Implementation

Full Implementation: Required Functions

```
def sigmoid(sop):
    return 1.0/(1+np.exp(-1*sop))

def loss_mse(predicted, target):
    return 0.5*np.power(target - predicted, 2)

def loss_predicted_deriv(predicted, target):
    return (predicted-target)

def sigmoid_sop_deriv(sop):
    return sigmoid(sop)*(1.0-sigmoid(sop))

def sop_w_deriv(x):
    return x

def update_w(w, grad, learning_rate):
    return w - learning_rate*grad
```

Backpropagation Implementation

Full Implementation:

```
# Initial/given values

x1=0.1
x2=0.3

target = 0.1

learning_rate = 0.5

#w1=np.random.rand()
#w2=np.random.rand()

w1 = 0.5
w2 = 0.2
b = 1.83

print("Initial W & b : ", w1, w2, b)

predicted_output = []
network_error = []

old_err = 0
```

Backpropagation Implementation

Full Implementation: Forward Pass

```
for k in range(1000):

    # Forward Pass

    sop = w1*x1 + w2*x2 + b

    predicted = sigmoid(sop)

    loss = loss_mse(predicted, target)

    predicted_output.append(predicted)

    network_error.append(loss)
```

Backpropagation Implementation

Full Implementation: Backward Pass

```
# Backward Pass
g1 = loss_predicted_deriv(predicted, target)

g2 = sigmoid_sop_deriv(sop)

g3w1 = sop_w_deriv(x1)
g3w2 = sop_w_deriv(x2)

gradw1 = g3w1*g2*g1

gradw2 = g3w2*g2*g1

gradb = g2*g1

w1 = update_w(w1, gradw1, learning_rate)
w2 = update_w(w2, gradw2, learning_rate)
b = update_w(b, gradb, learning_rate)
```

Backpropagation Implementation

Full Implementation: Epoch Results

```
print("Parameters at Iteration ", k+1, " : ", w1, w2, b)

print("predicted output", predicted)

print("Loss fnc value", loss)
```

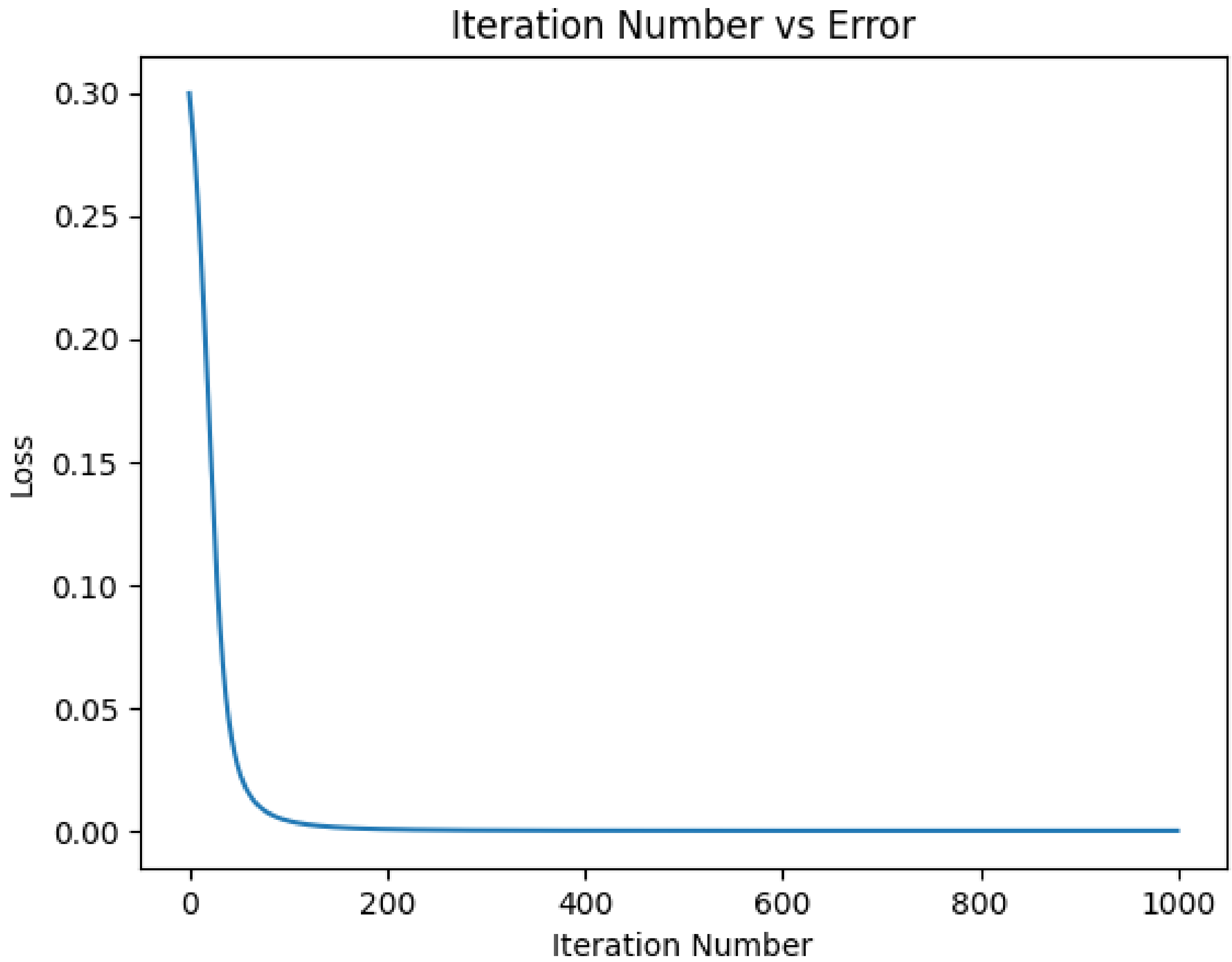
Backpropagation Implementation

Full Implementation: Plotting the Results

```
plt.figure()
plt.plot(network_error)
plt.title("Iteration Number vs Error")
plt.xlabel("Iteration Number")
plt.ylabel("Loss")

plt.figure()
plt.plot(predicted_output)
plt.title("Iteration Number vs Prediction")
plt.xlabel("Iteration Number")
plt.ylabel("Prediction")
```

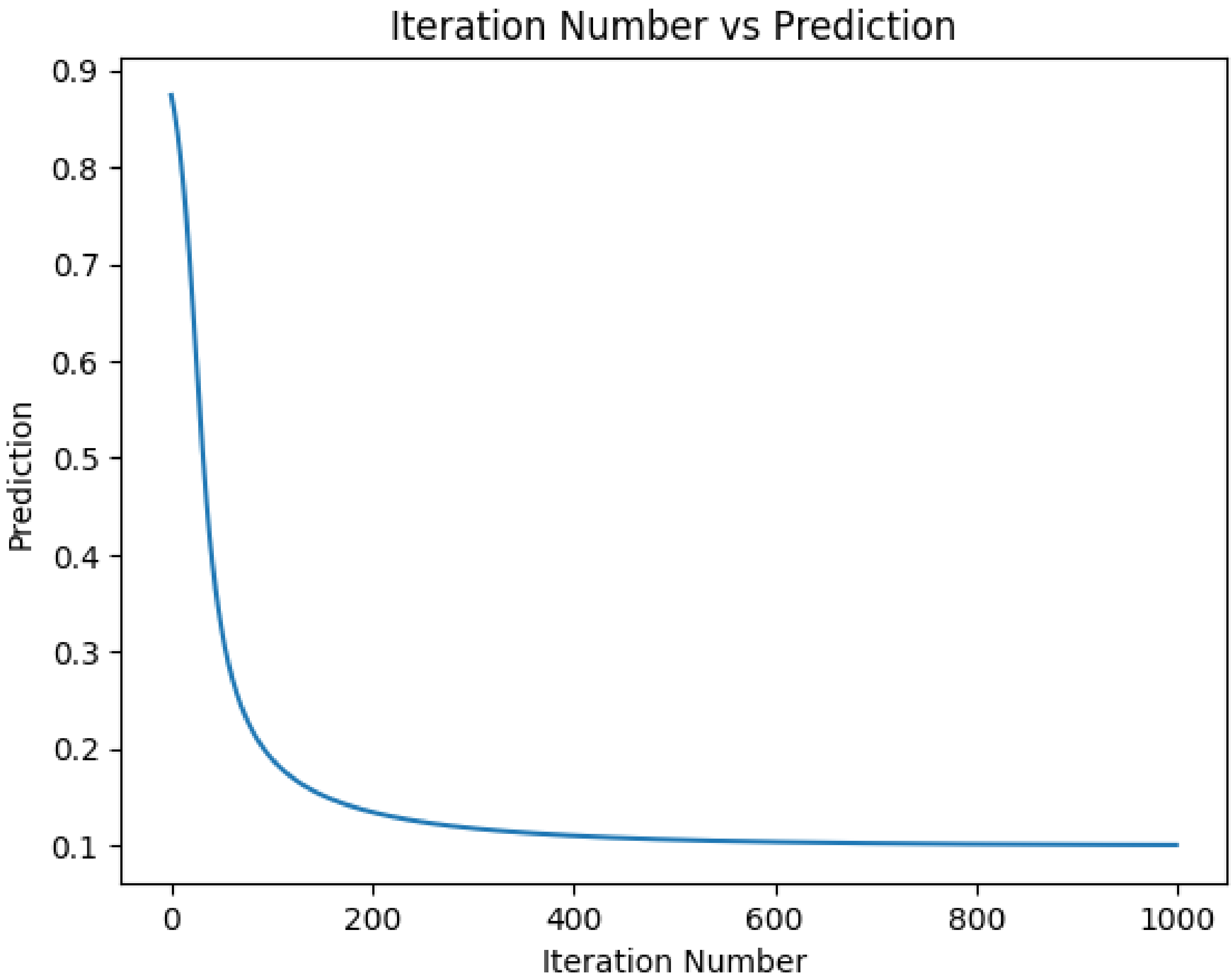
Backpropagation Implementation

Full Implementation: Plotting the Results



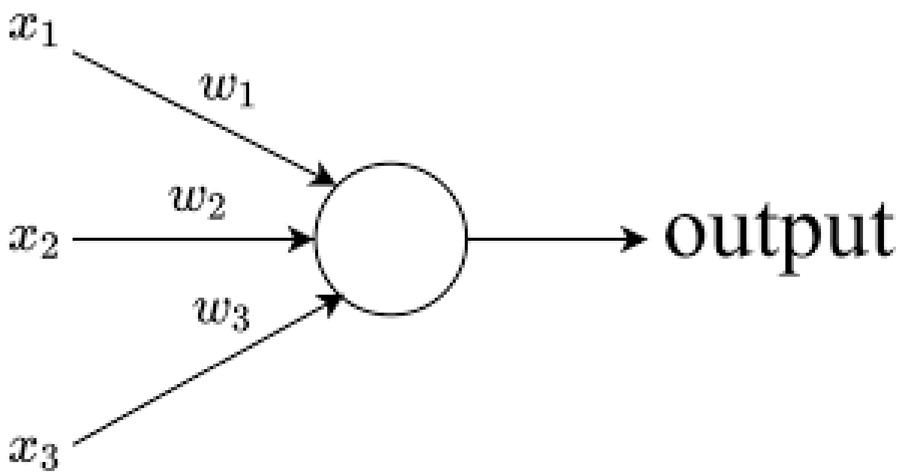
Backpropagation Implementation

Full Implementation: Plotting the Results



Backpropagation Implementation

➢ A single training sample example with 3 features:



X1	X2	X3	Desired Output
0.1	0.3	0.5	0.1

W1	W2	W2	b
0.5	0.2	-0.1	1.83

Source: Internet

Backpropagation Implementation

Full Implementation

```
# Required Libraries

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

Source: Internet

Backpropagation Implementation

Full Implementation: Required Functions

```python
def sigmoid(sop):
    return 1.0/(1+np.exp(-1*sop))

def loss_mse(predicted, target):
    return 0.5*np.mean(np.power(predicted-target, 2))

def loss_predicted_deriv(predicted, target):
    return (predicted-target)

def sigmoid_sop_deriv(sop):
    return sigmoid(sop)*(1.0-sigmoid(sop))

def update_w(w, grad, learning_rate):
    return w - learning_rate*grad
```

Backpropagation Implementation

Full Implementation:

```
x = np.array([0.1, 0.3, 0.5])    # Any number of inputs
target = 0.1
learning_rate = 0.5

w = np.array([0.5, 0.2, -0.1])  # One weight per input
b = 1.83



print("Initial W & b : ", w, b)

predicted_output = []
network_error = []
```

Backpropagation Implementation

Forward and Backward Pass

```
for k in range(3000):
    # Forward Pass
    y = np.dot(w, x) + b        # Dot product for all inputs
    predicted = sigmoid(y)
    loss = loss_mse(predicted, target)

    predicted_output.append(predicted)
    network_error.append(loss)

    # Backward Pass
    g1 = loss_predicted_deriv(predicted, target)
    g2 = sigmoid_sop_deriv(y)

    grad_w = x * g2 * g1        # Vectorized weight gradient
    grad_b = g2 * g1

    w = update_w(w, grad_w, learning_rate)
    b = update_w(b, grad_b, learning_rate)
```

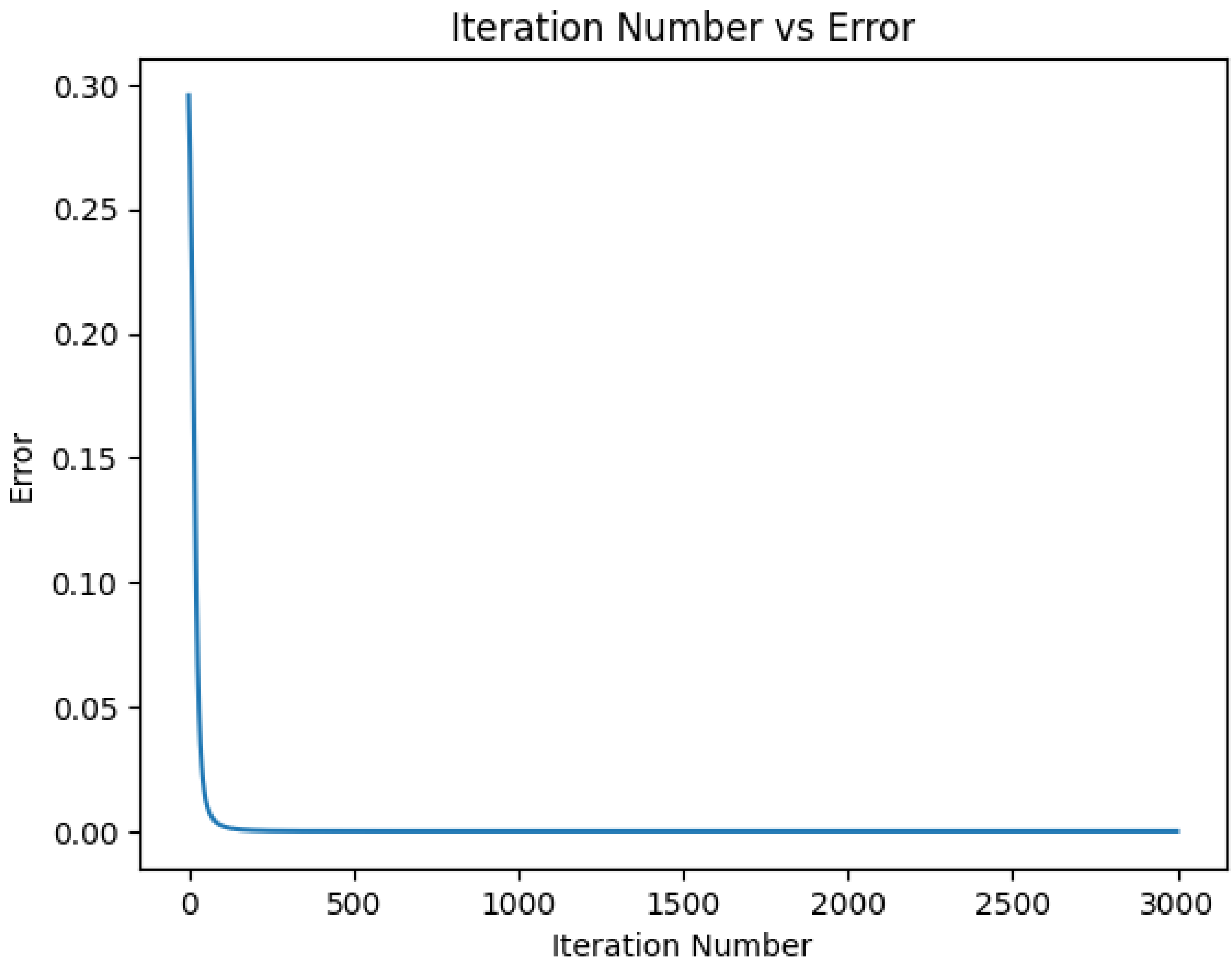
Backpropagation Implementation

Full Implementation: Plotting the Results

```
print("Final Loss Value",loss)

print("Final predicted output",predicted)

plt.figure()
plt.plot(network_error)
plt.title("Iteration Number vs Error")
plt.xlabel("Iteration Number")
plt.ylabel("Error")

plt.figure()
plt.plot(predicted_output)
plt.title("Iteration Number vs Prediction")
plt.xlabel("Iteration Number")
plt.ylabel("Prediction")
plt.show()
```

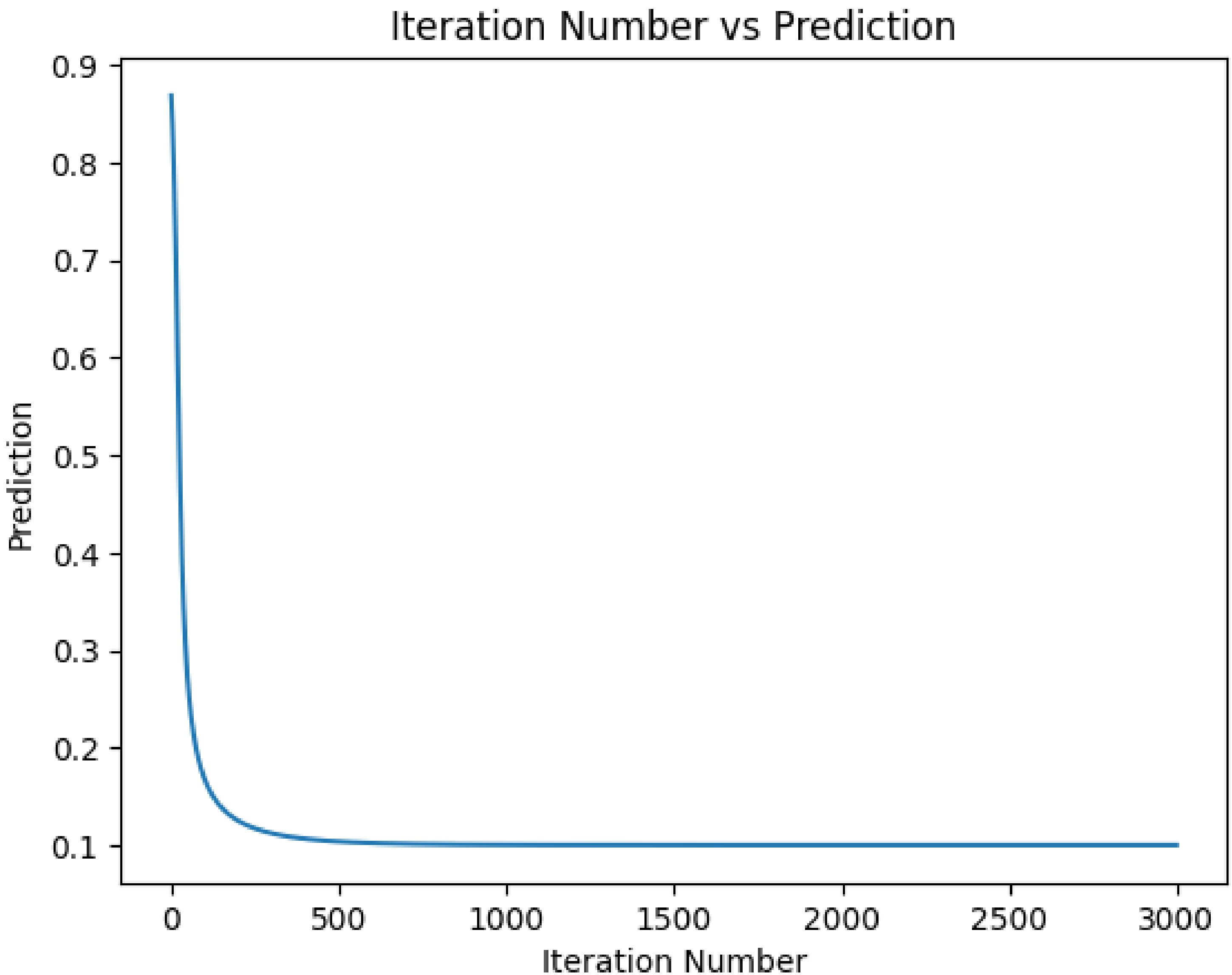
Backpropagation Implementation

Full Implementation: Plotting the Results



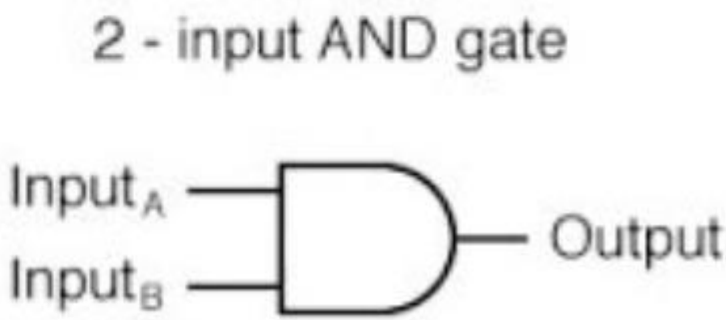
Backpropagation Implementation

Full Implementation: Plotting the Results



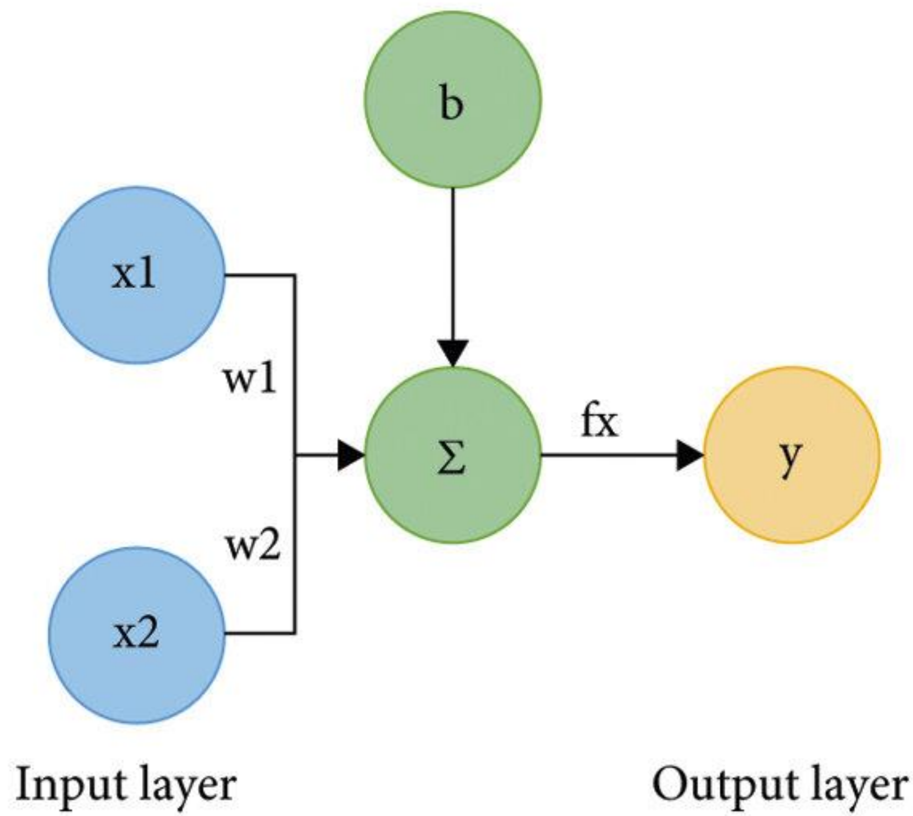
Backpropagation Implementation

Implementing Simple Logic Circuits: AND Gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Fig: Two Input AND Gate



Source: Google Images

Backpropagation Implementation

Implementing Simple Logic Circuits: AND Gate

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

# --- Activation ---
def sigmoid(sop):
    return 1.0 / (1 + np.exp(-1 * sop))

# --- Loss (MSE over batch) ---
def loss_mse(predicted, target):
    return 0.5 * np.mean((predicted - target) ** 2)

# --- Derivatives ---
def loss_predicted_deriv(predicted, target):
    N = predicted.shape[0]
    return (predicted - target) / N

def sigmoid_sop_deriv(sop):
    s = sigmoid(sop)
    return s * (1.0 - s)

def update_w(w, grad, learning_rate):
    return w - learning_rate * grad
```

Backpropagation Implementation

Full Implementation: Data

```
# ---------- Data for 2-input AND gate ----------
# Inputs: [x1, x2]
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])  # shape: (4, 2)

# Targets as column vector (N x 1)
T = np.array([[0],
              [0],
              [0],
              [1]])  # shape: (4, 1)
```

Backpropagation Implementation

Full Implementation: Parameters

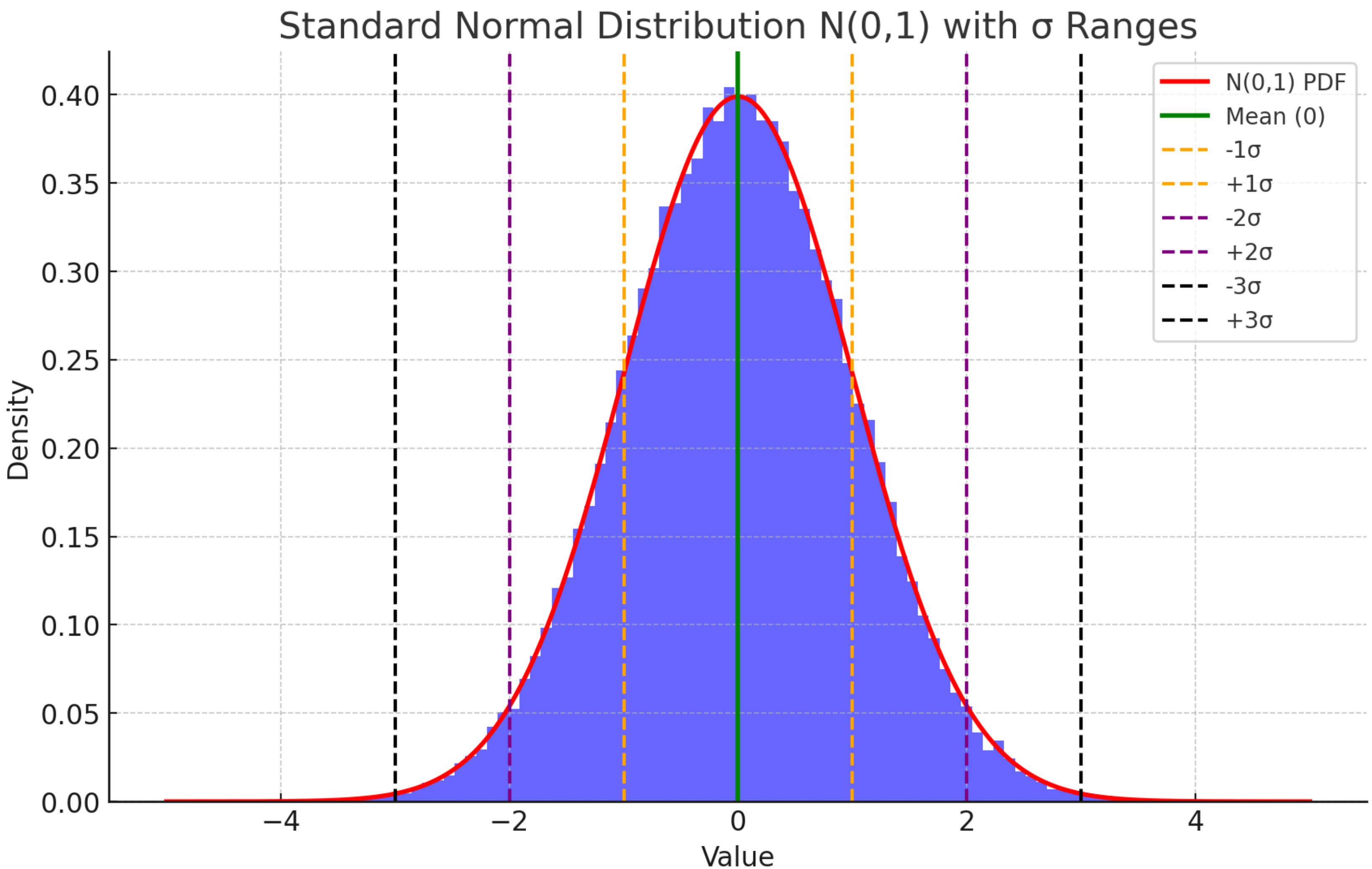
```
learning_rate = 0.3

# Weights (D x 1) and bias (scalar)
w = np.random.randn(2, 1)  # 2 inputs
b = np.random.randn()      # scalar bias

print("Initial W & b:", w, b)

network_error = []

num_iters = 50000
```

Backpropagation Implementation

Full Implementation: Parameters

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

np.random.randn()



Backpropagation Implementation

Forward and Backward Pass

```
for k in range(num_iters):
    # ---------- Forward Pass ----------
    y = np.dot(X, w) + b            # (N x 1)
    predicted = sigmoid(y)          # (N x 1)
    loss = loss_mse(predicted, T)

    network_error.append(loss)

    # ---------- Backward Pass ----------
    g1 = loss_predicted_deriv(predicted, T)  # (N x 1)
    g2 = sigmoid_sop_deriv(y)                # (N x 1)
    grad = g1 * g2                           # (N x 1)

    grad_w = np.dot(X.T, grad)               # (D x 1)
    grad_b = np.sum(grad)                    # scalar

    w = update_w(w, grad_w, learning_rate)   # (D x 1)
    b = update_w(b, grad_b, learning_rate)   # scalar
```

Backpropagation Implementation

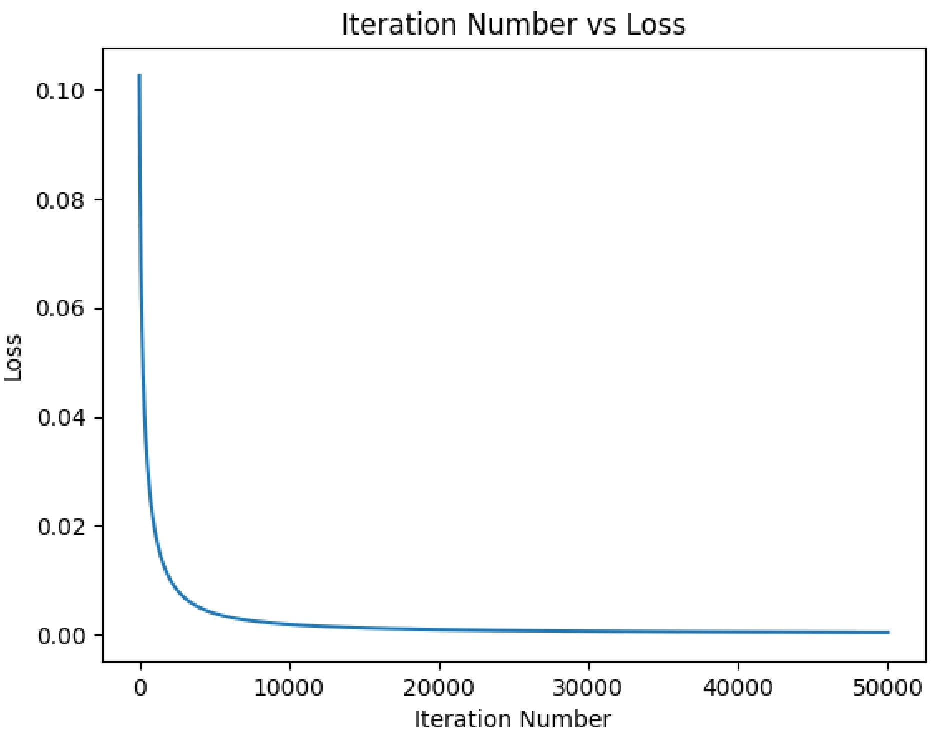
Full Implementation: Plotting the Results

```
print("\nFinal Loss Value:", loss)
print("Final W & b:", w, b)
print("Final predicted outputs:\n", predicted.round(2))
print("Targets:\n", T)

# ---------- Plot ----------
# Loss curve
plt.figure()
plt.plot(network_error)
plt.title("Iteration Number vs Loss")
plt.xlabel("Iteration Number")
plt.ylabel("Loss")
plt.show()
```

Backpropagation Implementation

Full Implementation: Plotting the Results

Implementing AND Gate

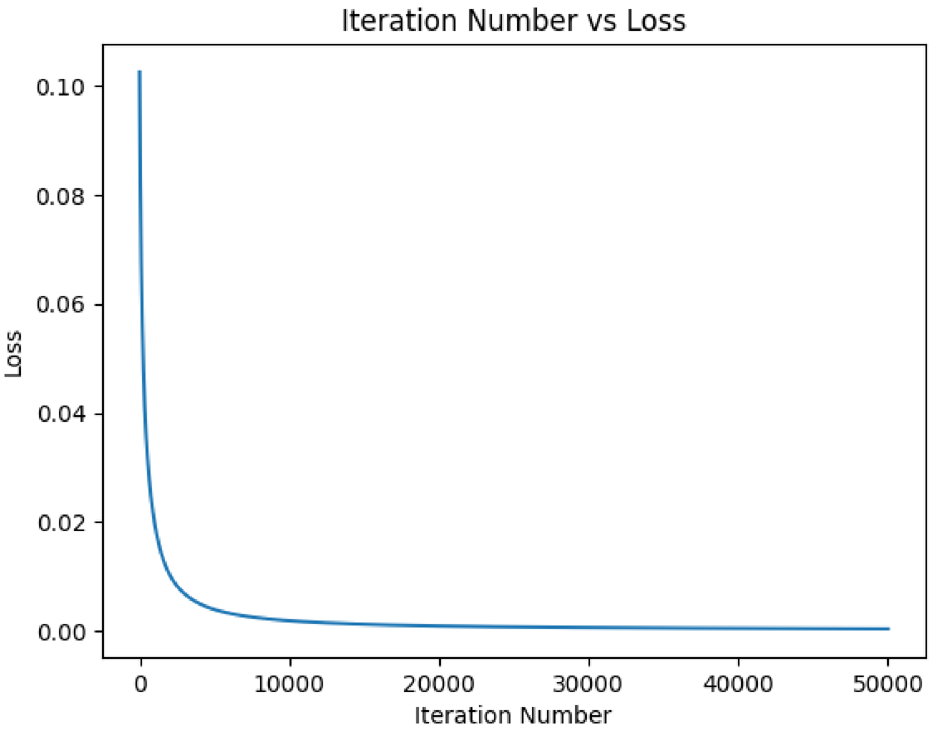



```
Initial W & b: [[ 1.32879525]
 [-0.53083813]] -0.6732405866184608

Final Loss Value: 0.00031765450785214196
Final W & b: [[6.96956151]
 [6.96956151]] -10.542454906649445
Final predicted outputs:
 [[0.  ]
 [0.03]
 [0.03]
 [0.97]]
Targets:
 [[0]
 [0]
 [0]
 [1]]
```

Backpropagation Implementation

Full Implementation: Plotting the Results

Implementing OR Gate

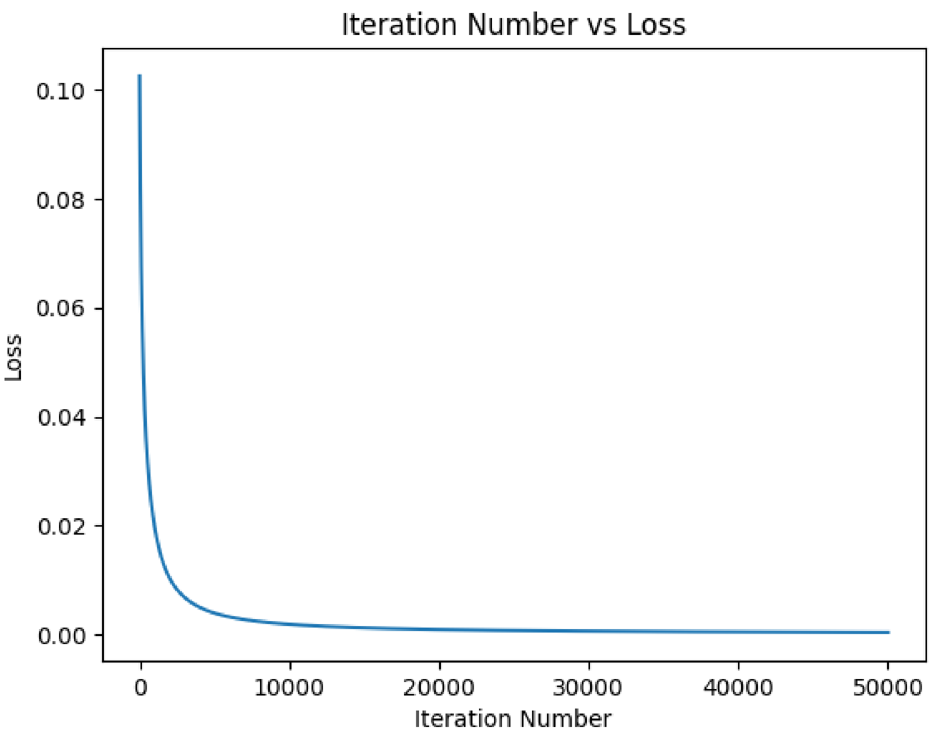



```
Initial W & b: [[ 0.28042288]
 [-1.44871042]] -0.6414387815914916

Final Loss Value: 0.00016461271036776994
Final W & b: [[7.63442379]
 [7.63440965]] -3.580000516192813
Final predicted outputs:
 [[0.03]
 [0.98]
 [0.98]
 [1.  ]]
Targets:
 [[0]
 [1]
 [1]
 [1]]
```

Backpropagation Implementation

Full Implementation: Plotting the Results

Implementing NOR Gate

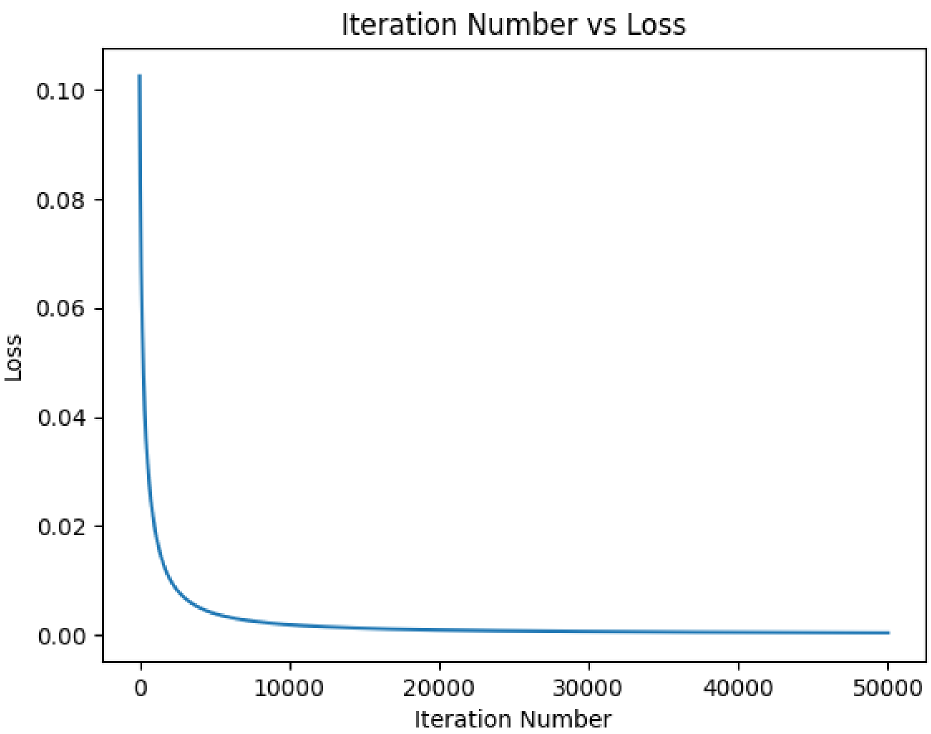



```
Initial W & b: [[-0.01972788]
 [ 0.09287627]] -0.17525700293936647

Final Loss Value: 0.00016459426244359986
Final W & b: [[-7.63453167]
 [-7.6345309 ]] 3.580058272157754
Final predicted outputs:
 [[0.97]
 [0.02]
 [0.02]
 [0.  ]]
Targets:
 [[1]
 [0]
 [0]
 [0]]
```

Backpropagation Implementation

Full Implementation: Plotting the Results

Implementing NAND Gate




```
Initial W & b: [[-0.44228946]
 [ 0.51781045]] -0.21116044500793954

Final Loss Value: 0.00031833882067594
Final W & b: [[-6.96734277]
 [-6.96734277]] 10.53913130529956
Final predicted outputs:
 [[1.  ]
 [0.97]
 [0.97]
 [0.03]]
Targets:
 [[1]
 [1]
 [1]
 [0]]
```