Lab-1: Implementation of Perceptron Network

R Abhijit Srivathsan - 2448044

Program #1: Implementation of Simple neural network with activation function

In [1]:

```
# Inputs and weights
x1 = 0.8
x2 = 0.6
x3 = 0.4
w1 = 0.1
w2 = 0.3
w3 = -0.2
bias_input = 1.0
bias_weight = 0.35

# Compute x*w products
product1 = x1 * w1
product2 = x2 * w2
product3 = x3 * w3
bias_product = bias_input * bias_weight

# Net input
net = product1 + product2 + product3 + bias_product

# Activation functions
def binary_step(net):
    if net > 0:
        return 1
    else:
        return 0

def bipolar_step(net):
    if net > 0:
        return 1
    else:
        return -1
```

```python
def binary_sigmoid(net):
    import math
    return 1 / (1 + math.exp(-net))

def bipolar_sigmoid(net):
    import math
    return 2 / (1 + math.exp(-net)) - 1

# Print intermediate results
print('Intermediate Products:')
print(f'x1*w1 = {product1:.4f}')
print(f'x2*w2 = {product2:.4f}')
print(f'x3*w3 = {product3:.4f}')
print(f'bias*weight = {bias_product:.4f}')
print(f'Net input = {net:.4f}\n')

# Print activation outputs
print('Neuron Outputs:')
print(f'Binary Step Output = {binary_step(net)}')
print(f'Bipolar Step Output = {bipolar_step(net)}')
print(f'Binary Sigmoid Output = {binary_sigmoid(net):.4f}')
print(f'Bipolar Sigmoid Output = {bipolar_sigmoid(net):.4f}')
```

```
Intermediate Products:
x1*w1 = 0.0800
x2*w2 = 0.1800
x3*w3 = -0.0800
bias*weight = 0.3500
Net input = 0.5300

Neuron Outputs:
Binary Step Output = 1
Bipolar Step Output = 1
Binary Sigmoid Output = 0.6295
Bipolar Sigmoid Output = 0.2590
```

Conclusion: Program #1 - Single Neuron with Multiple Activation Functions

Different activation functions yield different interpretations of a neuron's behavior.

- The **step functions** (binary and bipolar) provide discrete outputs suitable for classification tasks.
- The **sigmoid functions** (binary and bipolar) offer smooth, continuous outputs that are useful for gradient-based learning and probabilistic interpretation.
- In this experiment, all activations responded to the same net input (0.53) differently, showing how the choice of activation affects the output.

This helps in understanding how neurons behave in more complex networks and aids in selecting the right activation based on the task at hand.

Program #2: Implement perceptron network for AND function

In [2]:

```
# Input samples and targets
inputs = [
  [1,  1],
  [1, -1],
  [-1, 1],
  [-1, -1]
]
targets = [1, -1, -1, -1]

# Parameters
w1 = 0
w2 = 0
bias = 0
theta = 0
alpha = 1  # Learning rate

# Activation function (bipolar step)
def activation(net):
  if net > theta:
      return 1
  else:
      return -1

# Training Loop
```

```python
epoch = 0
converged = False
while not converged:
  epoch += 1
  print(f"\nEpoch {epoch}")
  errors = 0

  for i in range(len(inputs)):
      x1, x2 = inputs[i]
      t = targets[i]

      # Calculate net input
      net = x1 * w1 + x2 * w2 + bias

      # Apply activation
      y = activation(net)

      # Calculate error
      error = t - y

      # Update weights if error
      if error != 0:
          w1 = w1 + alpha * error * x1
          w2 = w2 + alpha * error * x2
          bias = bias + alpha * error
          errors += 1

      print(f"x1={x1}, x2={x2}, target={t}, output={y}, error={error}, w1={w1}, w2={w2}, bias={bias}")

  # Check for convergence
  if errors == 0:
      converged = True

print("\nTraining complete!")
print(f"Final weights: w1={w1}, w2={w2}, bias={bias}")
```

```
Epoch 1
x1=1, x2=1, target=1, output=-1, error=2, w1=2, w2=2, bias=2
x1=1, x2=-1, target=-1, output=1, error=-2, w1=0, w2=4, bias=0
x1=-1, x2=1, target=-1, output=1, error=-2, w1=2, w2=2, bias=-2
x1=-1, x2=-1, target=-1, output=-1, error=0, w1=2, w2=2, bias=-2

Epoch 2
x1=1, x2=1, target=1, output=1, error=0, w1=2, w2=2, bias=-2
x1=1, x2=-1, target=-1, output=-1, error=0, w1=2, w2=2, bias=-2
x1=-1, x2=1, target=-1, output=-1, error=0, w1=2, w2=2, bias=-2
x1=-1, x2=-1, target=-1, output=-1, error=0, w1=2, w2=2, bias=-2

Training complete!
Final weights: w1=2, w2=2, bias=-2
```

Conclusion: Program #2 - Perceptron Learning for Bipolar AND Function

The perceptron successfully learned the **bipolar AND function**, converging in just **2 epochs**.

- It correctly classified all input patterns by adjusting weights based on the error.
- The final values were:
  - `w1 = 2`
  - `w2 = 2`
  - `bias = -2`

This confirms the perceptron's ability to handle linearly separable problems like the AND function using a simple rule-based learning algorithm.