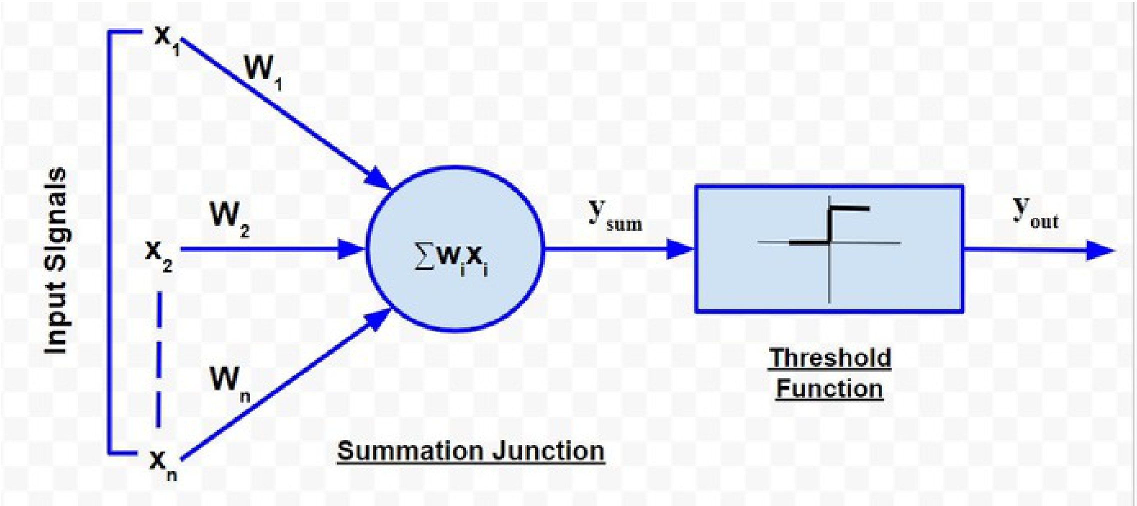
McCulloch-Pitts Neuron Model



Mathematical Summary:

1. **Inputs**:
   - Let $x_1, x_2, \ldots, x_n$ be the binary inputs to the neuron, where each x_i is either 0 or 1.
2. **Weights**:
   - Let $w_1, w_2, \ldots, w_n$ be the corresponding weights associated with each input.
3. **Threshold**:
   - Let θ be the threshold value for activation.
4. **Weighted Sum**:
   - The weighted sum of inputs is calculated as: $\text{Weighted Sum} = \sum_{i=1}^{n} w_i x_i$
5. **Activation**:
   - The neuron fires and produces an output of 1 if the weighted sum is greater than or equal to the threshold (θ): $\text{Output} = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0, & \text{otherwise} \end{cases}$
6. **Activation Function**:
   - The activation function in the McCulloch-Pitts neuron model is a step function, also known as a threshold function.
7. **Representation**:
   - In a more compact representation, we can express the output as: $y = \text{step}(\text{Weighted Sum} - \theta)$ where $\text{step}(x)$ is a step function defined as:

$$\text{step}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Limitations:

1. **Binary Logic Only**: It can only model binary logic functions, such as AND, OR, and NOT gates. It is limited to representing boolean functions that are linearly separable.
2. **Fixed Thresholds and Weights**: The model assumes fixed thresholds and weights, which cannot be adjusted or learned from data. This limits its ability to adapt to changing environments or learn from experience.
3. **No Representation of Magnitude**: It does not consider the magnitude of inputs or weights, only their binary presence or absence. This makes it unable to handle continuous data or represent complex patterns.
4. **Linearity Constraint**: The model is limited to linear decision boundaries, meaning it cannot model functions that require non-linear transformations, such as XOR.

mcculloch_pitts_neuron(inputs, weights, threshold)

In [10]:

```
def mcculloch_pitts_neuron(inputs, weights, threshold):
    """
    McCulloch-Pitts Neuron Model

    Args:
    inputs (list): List of binary input values (0 or 1)
    weights (list): List of corresponding weights for each input
    threshold (int): Threshold value for activation

    Returns:
    int: Output of the neuron (0 or 1)
    """
    # Calculate weighted sum of inputs
    weighted_sum = sum([x * w for x, w in zip(inputs, weights)])

    # Determine the output based on the threshold function
    output = threshold_function(weighted_sum, threshold)

    print(f'{inputs=}, {weights=}, {weighted_sum=}, {threshold=}, {output=}')

    return output


def threshold_function(weighted_sum, threshold):
    """
    Threshold function for McCulloch-Pitts neuron model.
```

```
    Args:
    weighted_sum (int): Weighted sum of inputs
    threshold (int): Threshold value for activation

    Returns:
    int: Output of the neuron (0 or 1)
    """
    return 1 if weighted_sum >= threshold else 0
```

In [11]:

```
def test_logic_gate(name, weights, threshold, input_combinations):
    """
    Test a logic gate and print its truth table.

    Args:
    name (str): Name of the logic gate.
    weights (list): List of weights for the neuron.
    threshold (int): Threshold value for activation.
    input_combinations (list of tuples): List of input combinations to test.

    Returns:
    None
    """
    print(f"{name} Truth Table:")
    print()
    for inputs in input_combinations:
      output = mcculloch_pitts_neuron(inputs, weights, threshold)
      print(f"Input: {inputs}, Output: {output}")
      print()
```

In [12]:

```
# All possible combinations of input values for two inputs
input_combinations = [(0, 0), (0, 1), (1, 0), (1, 1)]
```

AND Gate

In [13]:

```
# Create a neuron for AND gate
and_weights = [1, 1]
and_threshold = 2

# Test AND gate
test_logic_gate("AND Gate", and_weights, and_threshold, input_combinations)
```

```
AND Gate Truth Table:

inputs=(0, 0), weights=[1, 1], weighted_sum=0, threshold=2, output=0
Input: (0, 0), Output: 0

inputs=(0, 1), weights=[1, 1], weighted_sum=1, threshold=2, output=0
Input: (0, 1), Output: 0

inputs=(1, 0), weights=[1, 1], weighted_sum=1, threshold=2, output=0
Input: (1, 0), Output: 0

inputs=(1, 1), weights=[1, 1], weighted_sum=2, threshold=2, output=1
Input: (1, 1), Output: 1
```

OR Gate

In [14]:

```
# Create a neuron for OR gate
or_weights = [1, 1]
or_threshold = 1

# Test OR gate
test_logic_gate("OR Gate", or_weights, or_threshold, input_combinations)
```

```
OR Gate Truth Table:

inputs=(0, 0), weights=[1, 1], weighted_sum=0, threshold=1, output=0
Input: (0, 0), Output: 0

inputs=(0, 1), weights=[1, 1], weighted_sum=1, threshold=1, output=1
Input: (0, 1), Output: 1

inputs=(1, 0), weights=[1, 1], weighted_sum=1, threshold=1, output=1
Input: (1, 0), Output: 1

inputs=(1, 1), weights=[1, 1], weighted_sum=2, threshold=1, output=1
Input: (1, 1), Output: 1
```

NOT gate

In [15]:

```
# Create a neuron for NOT gate
not_weights = [-1]
not_threshold = 0

# Test NOT gate
test_logic_gate("NOT Gate", not_weights, not_threshold, [(0,), (1,)])
```

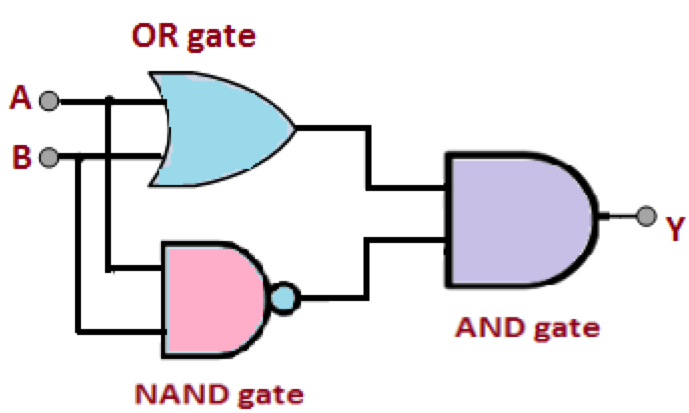
```
NOT Gate Truth Table:

inputs=(0,), weights=[-1], weighted_sum=0, threshold=0, output=1
Input: (0,), Output: 1

inputs=(1,), weights=[-1], weighted_sum=-1, threshold=0, output=0
Input: (1,), Output: 0
```

NAND Gate

In [16]:
```
# Define weights and threshold for NAND gate (negated AND gate)
nand_weights = [-1, -1]  # Negated weights of AND gate
nand_threshold = -1.9  # Negated threshold of AND gate

# Test NAND gate using the test_logic_gate function
test_logic_gate("NAND Gate", nand_weights, nand_threshold, input_combinations)
```

```
NAND Gate Truth Table:

inputs=(0, 0), weights=[-1, -1], weighted_sum=0, threshold=-1.9, output=1
Input: (0, 0), Output: 1

inputs=(0, 1), weights=[-1, -1], weighted_sum=-1, threshold=-1.9, output=1
Input: (0, 1), Output: 1

inputs=(1, 0), weights=[-1, -1], weighted_sum=-1, threshold=-1.9, output=1
Input: (1, 0), Output: 1

inputs=(1, 1), weights=[-1, -1], weighted_sum=-2, threshold=-1.9, output=0
Input: (1, 1), Output: 0
```

XOR gate



In [17]:
```
# Define functions for AND, OR, and NOT gates
def AND_gate(inputs):
  return mcculloch_pitts_neuron(inputs, and_weights, and_threshold)
```

```python


def OR_gate(inputs):
  return mcculloch_pitts_neuron(inputs, or_weights, or_threshold)


def NOT_gate(input):
  return mcculloch_pitts_neuron([input], not_weights, not_threshold)
```

In [18]:

```python
# Define function for XOR gate using AND, OR, and NOT gates
def XOR_gate(input1, input2):
  return AND_gate([OR_gate([input1, input2]), NOT_gate(AND_gate([input1, input2]))]


# Test the XOR gate
print("XOR Gate Truth Table:")
print()
for input1, input2 in input_combinations:
  output = XOR_gate(input1, input2)
  print(f"Input: {input1, input2}, Output: {output}")
  print()
```

```
XOR Gate Truth Table:

inputs=[0, 0], weights=[1, 1], weighted_sum=0, threshold=1, output=0
inputs=[0, 0], weights=[1, 1], weighted_sum=0, threshold=2, output=0
inputs=[0], weights=[-1], weighted_sum=0, threshold=0, output=1
inputs=[0, 1], weights=[1, 1], weighted_sum=1, threshold=2, output=0
Input: (0, 0), Output: 0

inputs=[0, 1], weights=[1, 1], weighted_sum=1, threshold=1, output=1
inputs=[0, 1], weights=[1, 1], weighted_sum=1, threshold=2, output=0
inputs=[0], weights=[-1], weighted_sum=0, threshold=0, output=1
inputs=[1, 1], weights=[1, 1], weighted_sum=2, threshold=2, output=1
Input: (0, 1), Output: 1

inputs=[1, 0], weights=[1, 1], weighted_sum=1, threshold=1, output=1
inputs=[1, 0], weights=[1, 1], weighted_sum=1, threshold=2, output=0
inputs=[0], weights=[-1], weighted_sum=0, threshold=0, output=1
inputs=[1, 1], weights=[1, 1], weighted_sum=2, threshold=2, output=1
Input: (1, 0), Output: 1

inputs=[1, 1], weights=[1, 1], weighted_sum=2, threshold=1, output=1
inputs=[1, 1], weights=[1, 1], weighted_sum=2, threshold=2, output=1
inputs=[1], weights=[-1], weighted_sum=-1, threshold=0, output=0
inputs=[1, 0], weights=[1, 1], weighted_sum=1, threshold=2, output=0
Input: (1, 1), Output: 0
```