A70 Pratik Jade Pratical no. 5

The McCulloch-Pitts Artificial Neuron Model

In [1]:
```
#Step 1: generate a vector of inputs and a vector of weights
import numpy as np
np.random.seed(seed=0)
I = np.random.choice([0,1], 3)# generate random vector I, sampling from {0,1}
W = np.random.choice([-1,1], 3) # generate random vector W, sampling from {-1,1}
print(f'Input vector:{I}, Weight vector:{W}')
```

```
Input vector:[0 1 1], Weight vector:[-1  1  1]
```

In [2]:
```
#Step 2: compute the dot product between the vector of inputs and weights
dot = I @ W
print(f'Dot product: {dot}')
```

```
Dot product: 2
```

In [3]:
```
#Step 3: define the threshold activation function
def linear_threshold_gate(dot: int, T: float) -> int:
    '''Returns the binary threshold output'''
    if dot >= T:
        return 1
    else:
        return 0
```

In [4]:
```
#Step 4: compute the output based on the threshold value
T = 1
activation = linear_threshold_gate(dot, T)
print(f'Activation: {activation}')
```

```
Activation: 1
```

In [5]:
```
T = 3
activation = linear_threshold_gate(dot, T)
print(f'Activation: {activation}')
```

```
Activation: 0
```

In [ ]:

Boolean algebra using the McCulloch-Pitts artificial neuron

In [6]:
```
#Step 1: generate a vector of inputs and a vector of weights
# matrix of inputs
input_table = np.array([
    [0,0], # both no
    [0,1], # one no, one yes
    [1,0], # one yes, one no
    [1,1]  # bot yes
])

print(f'input table:\n{input_table}')
```

```
input table:
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
```

In [7]:
```
# array of weights
weights = np.array([1,1])
print(f'weights: {weights}')
```

```
weights: [1 1]
```

In [8]:
```
#Step 2: compute the dot product between the matrix of inputs and weights
# dot product matrix of inputs and weights
dot_products = input_table @ weights
print(f'Dot products: {dot_products}')
```

```
Dot products: [0 1 1 2]
```

In [9]:
```
#step3: define the threshold activation function
#We defined this already, so we will reuse our linear_threshold_gate function
```

In [10]:
```
#Step 4: compute the output based on the threshold value
T = 2
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T)
    print(f'Activation: {activation}')
```

```
Activation: 0
Activation: 0
Activation: 0
Activation: 1
```

In [ ]:
```
```

The OR Function

In [11]:
```
'''Step 1: generate a vector of inputs and a vector of weights
Neither the matrix of inputs nor the array of weights changes, so we can reuse ou

Step 2: compute the dot product between the matrix of inputs and weights
Since neither the matrix of inputs nor the vector of weights changes, the dot pro

Step 3: define the threshold activation function
We can use the linear_threshold_gate function again.'''
```

Out[11]:
```
'Step 1: generate a vector of inputs and a vector of weights\nNeither the matri
x of inputs nor the array of weights changes, so we can reuse our input_table a
nd weights vector.\n\nStep 2: compute the dot product between the matrix of inp
uts and weights\nSince neither the matrix of inputs nor the vector of weights c
hanges, the dot product of those stays the same.\n\nStep 3: define the threshol
d activation function\nWe can use the linear_threshold_gate function again.'
```

In [12]:
```
#Step 4: compute the output based on the threshold value
T = 1
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T)
    print(f'Activation: {activation}')
```

```
Activation: 0
Activation: 1
Activation: 1
Activation: 1
```

The NOR function

In [13]:
```
#Step 1: generate a vector of inputs and a vector of weights
# array of weights
weights = np.array([-1,-1])
print(f'weights: {weights}')
```

```
weights: [-1 -1]
```

In [14]:
```
#Step 2: compute the dot product between the matrix of inputs and weights
# dot product matrix of inputs and weights
dot_products = input_table @ weights
print(f'Dot products: {dot_products}')
```

```
Dot products: [ 0 -1 -1 -2]
```

In [15]:
```
#Step 3: define the threshold activation function
# The function remains the same.
```

In [16]:
```
#Step 4: compute the output based on the threshold value
T = 0
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T)
    print(f'Activation: {activation}')
```

```
Activation: 1
Activation: 0
Activation: 0
Activation: 0
```

In [ ]: