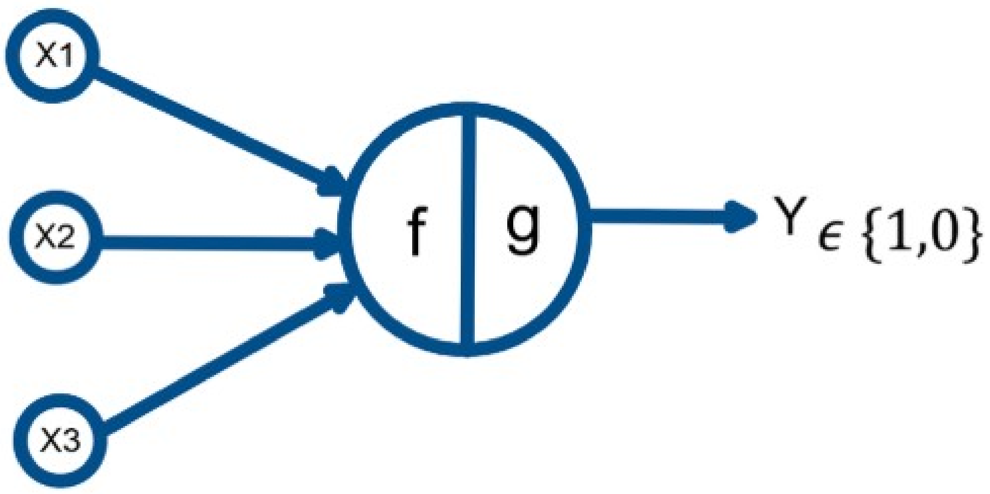
```python
import numpy as np
```

Mc Culloch pitts neuron

implementation for AND, OR gate



```python
def AND_gate(*x):
    return 1 if sum(x) == len(x) else 0

def OR_gate(*x):
    return 1 if sum(x) >= 1 else 0

# example
print('AND Output :',AND_gate(0, 0, 1)) # output is 0
print('OR Output :',OR_gate(0, 0, 1)) # output is 1
```

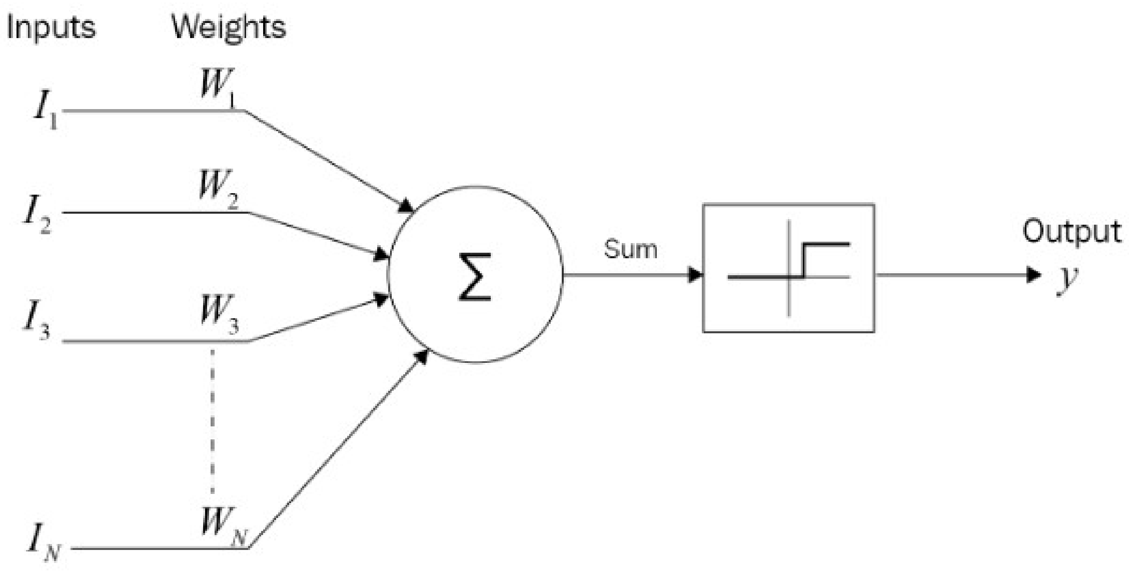
```
AND Output : 0
OR Output : 1
```

```python
def mc_and(x,w,theta): return 1 if np.dot(x,w) >= theta else 0


w,b = np.ones(3),1
x = np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
print("AND gate for 3 input : ",end=' ')
for i in x:
    print(mc_and(i,w,theta=len(x[0])),end=" ") #AND gate for theta =3
print("\nOR gate for 3 input : ",end=' ')
for i in x:
    print(mc_and(i,w,theta=1),end=" ") #OR gate for theta =1
```

```
AND gate for 3 input :  0 0 0 1
OR gate for 3 input :  1 1 1 1
```

```
len(x[0])

3
```

Perceptron



Implementation for AND, OR gate

```
import numpy as np

class Perceptron:
    def __init__(self, lr=0.1, epochs=10):
        self.lr = lr
        self.epochs = epochs

    def fit(self, X, y):
        self.w = np.zeros(X.shape[1])
        self.b = 0

        for _ in range(self.epochs):
            for xi, yi in zip(X, y):
                y_pred = 1 if np.dot(xi, self.w) + self.b >= 0 else 0
                error = yi - y_pred
                self.w += self.lr * error * xi
                self.b += self.lr * error

    def predict(self, X):
        return np.array([
            1 if np.dot(xi, self.w) + self.b >= 0 else 0
            for xi in X
        ])

# AND data
X_and = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y_and = np.array([0, 0, 0, 1])

p_and = Perceptron()
p_and.fit(X_and, y_and)

print("AND predictions:", p_and.predict(X_and))
print("AND weights:", p_and.w, "bias:", p_and.b)
```

```
AND predictions: [0 0 0 1]
AND weights: [0.2 0.1] bias: -0.20000000000000004
```

```
# OR data
X_or = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y_or = np.array([0, 1, 1, 1])

p_or = Perceptron()
p_or.fit(X_or, y_or)

print("OR predictions:", p_or.predict(X_or))
print("OR weights:", p_or.w, "bias:", p_or.b)
```

```
OR predictions: [0 1 1 1]
OR weights: [0.1 0.1] bias: -0.1
```

for weight and bais

w += Error_rate*x

b += error_rate

Here, Error_rate = target - output

```
import numpy as np
# for AND gate
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_list = np.array([0,0,0,1])
epochs = 8
w,b = np.zeros(2),0


for _ in range(epochs):
  for x,y in zip(x_list,y_list):
    # print('y og :',y,",",end=" ")
    y_p = np.dot(x,w) + b
    ### condition area for y_pred
    y_pred = 1 if y_p >= 0 else 0
    er = y - y_pred
    print(er,"<- at epo",_)
    w += er*x
    b += er
    print(w,b)
```

```
-1 <- at epo 0
[0. 0.] -1
0 <- at epo 0
[0. 0.] -1
0 <- at epo 0
[0. 0.] -1
1 <- at epo 0
[1. 1.] 0
-1 <- at epo 1
[1. 1.] -1
-1 <- at epo 1
[1. 0.] -2
0 <- at epo 1
[1. 0.] -2
1 <- at epo 1
[2. 1.] -1
0 <- at epo 2
[2. 1.] -1
-1 <- at epo 2
```

```
[2. 0.] -2
-1 <- at epo 2
[1. 0.] -3
1 <- at epo 2
[2. 1.] -2
0 <- at epo 3
[2. 1.] -2
0 <- at epo 3
[2. 1.] -2
-1 <- at epo 3
[1. 1.] -3
1 <- at epo 3
[2. 2.] -2
0 <- at epo 4
[2. 2.] -2
-1 <- at epo 4
[2. 1.] -3
0 <- at epo 4
[2. 1.] -3
0 <- at epo 4
[2. 1.] -3
0 <- at epo 5
[2. 1.] -3
0 <- at epo 5
[2. 1.] -3
0 <- at epo 5
[2. 1.] -3
0 <- at epo 5
[2. 1.] -3
0 <- at epo 6
[2. 1.] -3
0 <- at epo 6
[2. 1.] -3
0 <- at epo 6
[2. 1.] -3
0 <- at epo 6
[2. 1.] -3
0 <- at epo 7
[2. 1.] -3
0 <- at epo 7
[2. 1.] -3
0 <- at epo 7
[2. 1.] -3
0 <- at epo 7
[2. 1.] -3
```

```python
import numpy as np
# for AND gate
x_list = np.array([[2,2],[1,-2],[-2,2],[-1,1]])
y_list = np.array([0,1,0,1])
```

```
epochs = 3
w,b = np.zeros(2),0


for _ in range(epochs):
  for x,y in zip(x_list,y_list):
    # print('y og :',y,",",end=" ")
    y_p = np.dot(x,w) + b
    ### condition area for y_pred
    y_pred = 1 if y_p >= 0 else 0
    er = y - y_pred
    print("error rate : ",er,"<- at epo",_)
    w += er*x
    b += er
    print(w,b)
```

```
error rate :  -1 <- at epo 0
[-2. -2.] -1
error rate :  0 <- at epo 0
[-2. -2.] -1
error rate :  0 <- at epo 0
[-2. -2.] -1
error rate :  1 <- at epo 0
[-3. -1.] 0
error rate :  0 <- at epo 1
[-3. -1.] 0
error rate :  1 <- at epo 1
[-2. -3.] 1
error rate :  0 <- at epo 1
[-2. -3.] 1
error rate :  0 <- at epo 1
[-2. -3.] 1
error rate :  0 <- at epo 2
[-2. -3.] 1
error rate :  0 <- at epo 2
[-2. -3.] 1
error rate :  0 <- at epo 2
[-2. -3.] 1
error rate :  0 <- at epo 2
[-2. -3.] 1
```

my test

```
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
w = np.array([2,1])
b = -3
y_p = np.dot(x_list,w) + b
print(y_p)
#
```

```
[-3 -2 -1  0]
```