Implementation of Perceptron Algorithm for AND Logic Gate with 2-bit Binary Input

In the field of Machine Learning, the Perceptron is a Supervised Learning Algorithm for binary classifiers. The Perceptron Model implements the following function:

$$\begin{aligned} \hat{y} &= \Theta\left(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b\right) \\ &= \Theta(\mathbf{w} \cdot \mathbf{x} + b) \\ \text{where } \Theta(v) &= \begin{cases} 1 & \text{if } v \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

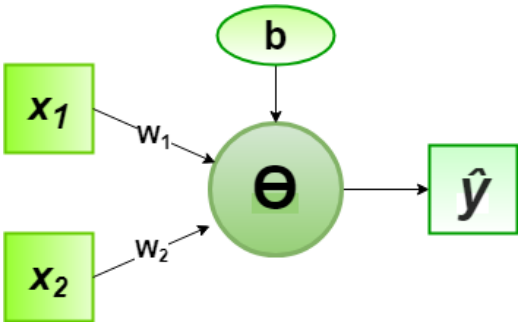
For a particular choice of the weight vector $\boldsymbol{w}$ and bias parameter $\boldsymbol{b}$, the model predicts output $\hat{\boldsymbol{y}}$ for the corresponding input vector $\boldsymbol{x}$.

AND logical function truth table for ***2-bit binary variables***, i.e, the input vector $\boldsymbol{x} : (\boldsymbol{x_1}, \boldsymbol{x_2})$ and the corresponding output $\boldsymbol{y}$ –

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Now for the corresponding weight vector $\boldsymbol{w} : (\boldsymbol{w_1}, \boldsymbol{w_2})$ of the input vector $\boldsymbol{x} : (\boldsymbol{x_1}, \boldsymbol{x_2})$, the associated Perceptron Function can be defined as:

$$\hat{\boldsymbol{y}} = \Theta\left(w_1 x_1 + w_2 x_2 + b\right)$$



For the implementation, considered weight parameters are $w_1 = 1, w_2 = 1$ and the bias parameter is $b = -1.5$.

Python Implementation:

```
# importing Python library
import numpy as np

# define Unit Step Function
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0

# design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y

# AND Logic Function
# w1 = 1, w2 = 1, b = -1.5
def AND_logicFunction(x):
    w = np.array([1, 1])
    b = -1.5
    return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test4)))
```

Output:

```
AND(0, 1) = 0
AND(1, 1) = 1
AND(0, 0) = 0
AND(1, 0) = 0
```

Here, the model predicted output ($\hat{y}$) for each of the test inputs are exactly matched with the AND logic gate conventional output (y) according to the truth table for 2-bit binary input.
Hence, it is verified that the perceptron algorithm for AND logic gate is correctly implemented.

Implementation of Perceptron Algorithm for OR Logic Gate with 2-bit Binary Input

In the field of Machine Learning, the Perceptron is a Supervised Learning Algorithm for binary classifiers. The Perceptron Model implements the following function:

$$\begin{aligned} \hat{y} &= \Theta\left(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b\right) \\ &= \Theta(\mathbf{w} \cdot \mathbf{x} + b) \\ \text{where } \Theta(v) &= \begin{cases} 1 & \text{if } v \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

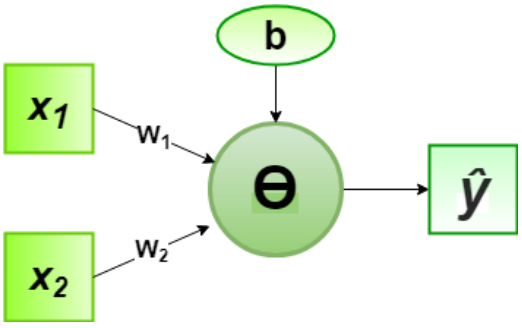
For a particular choice of the weight vector $\boldsymbol{w}$ and bias parameter $\boldsymbol{b}$, the model predicts output $\hat{\boldsymbol{y}}$ for the corresponding input vector $\boldsymbol{x}$.

OR logical function truth table for ***2-bit binary variables***, i.e, the input vector $\boldsymbol{x} : (\boldsymbol{x_1}, \boldsymbol{x_2})$ and the corresponding output $\boldsymbol{y}$ –

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{y}$
0	0	0
0	1	1
1	0	1
1	1	1

Now for the corresponding weight vector $\boldsymbol{w} : (\boldsymbol{w_1}, \boldsymbol{w_2})$ of the input vector $\boldsymbol{x} : (\boldsymbol{x_1}, \boldsymbol{x_2})$, the associated Perceptron Function can be defined as:

$$\hat{\boldsymbol{y}} = \Theta\left(w_1 x_1 + w_2 x_2 + b\right)$$



For the implementation, considered weight parameters are $w_1 = 1, w_2 = 1$ and the bias parameter is $b = -0.5$.

Python Implementation:

```
# importing Python library
import numpy as np

# define Unit Step Function
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0

# design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y

# OR Logic Function
# w1 = 1, w2 = 1, b = -0.5
def OR_logicFunction(x):
    w = np.array([1, 1])
    b = -0.5
    return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test1)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test2)))
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test4)))
```

Output:

```
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
```

Here, the model predicted output ($\hat{y}$) for each of the test inputs are exactly matched with the OR logic gate conventional output (y) according to the truth table for 2-bit binary input.
Hence, it is verified that the perceptron algorithm for OR logic gate is correctly implemented.

Implementation of Perceptron Algorithm for NOT

In the field of Machine Learning, the Perceptron is a Supervised Learning Algorithm for binary classifiers. The Perceptron Model implements the following function:

$$\begin{aligned}\hat{y} &= \Theta\left(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b\right) \\ &= \Theta(\mathbf{w} \cdot \mathbf{x} + b) \\ \text{where } \Theta(v) &= \begin{cases} 1 & \text{if } v \geqslant 0 \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

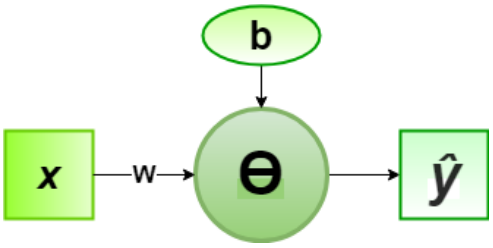
For a particular choice of the weight vector w and bias parameter b, the model predicts output $\hat{y}$ for the corresponding input vector x.

NOT logical function truth table is of only 1-bit binary input (0 or 1), i.e, the input vector x and the corresponding output y –

x	y
0	1
1	0

Now for the corresponding weight vector w of the input vector x, the associated Perceptron Function can be defined as:

$$\hat{y} = \Theta\left(wx + b\right)$$



For the implementation, considered weight parameter is $w = -1$ and the bias parameter is $b = 0.5$.

Python Implementation:

```
# importing Python library
import numpy as np
```

```
# define Unit Step Function
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0

# design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y

# NOT Logic Function
# w = -1, b = 0.5
def NOT_logicFunction(x):
    w = -1
    b = 0.5
    return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array(1)
test2 = np.array(0)

print("NOT({}) = {}".format(1, NOT_logicFunction(test1)))
print("NOT({}) = {}".format(0, NOT_logicFunction(test2)))
```

Output:

```
NOT(1) = 0
NOT(0) = 1
```

Here, the model predicted output ($\hat{y}$) for each of the test inputs are exactly matched with the NOT logic gate conventional output (y) according to the truth table.

Hence, it is verified that the perceptron algorithm for NOT logic gate is correctly implemented.