Practical 7 (a)

Implement A Perceptron For Binary AND Operation

In [33]:

```
import numpy as np
```

In [34]:

```
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
```

In [35]:

```
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
```

In [36]:

```
def AND_Logic(x):
    w = np.array([0.5, 0.5])
    b = -1
    return perceptronModel(x, w, b)
```

In [37]:

```
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])
```

In [38]:

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

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

In [ ]:

```

```

Practical 7 (b)

Implement A Perceptron For Binary OR Operation

In [39]:

```
import numpy as np
```

In [40]:

```
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
```

In [41]:

```
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
```

In [42]:

```
def OR_Logic(x):
    w = np.array([1, 1])
    b = -0.5
    return perceptronModel(x, w, b)
```

In [43]:

```
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])
```

In [44]:

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

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