```
import numpy as np
```

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

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

```
def AND_percep(X):
  w = np.array([1,1])
  b = -1.5
  return perceptron(X,w,b)
```

```
def OR_percep(X):
  w = np.array([1,1])
  b = -0.5
  return perceptron(X,w,b)
```

```
example1 = np.array([1,1])
example2 = np.array([1,0])
example3 = np.array([0,1])
example4 = np.array([0,0])
```

```
print("AND({},{})= {}".format(1,1,AND_percep(example1)))
print("AND({},{})= {}".format(1,0,AND_percep(example2)))
print("AND({},{})= {}".format(0,1,AND_percep(example3)))
print("AND({},{})= {}".format(0,0,AND_percep(example4)))
```

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

```
def XOR_net(X):
  gate_1 = AND_percep(X)
  gate_2 = NOT_percep(gate_1)
  gate_3 = OR_percep(X)
  new_X = np.array([gate_2, gate_3])
  output = AND_percep(new_X)
  return output

def NOT_percep(X):
  return perceptron(X,w =-1, b = 0.5)
```

```
print("XOR({},{})= {}".format(1,1,XOR_net(example1)))
print("XOR({},{})= {}".format(1,0,XOR_net(example2)))
print("XOR({},{})= {}".format(0,1,XOR_net(example3)))
print("XOR({},{})= {}".format(0,0,XOR_net(example4)))
```

```
XOR(1,1)= 0
XOR(1,0)= 1
XOR(0,1)= 1
XOR(0,0)= 0
```

Alternative Method

```
def activation_func(value):
  return(1/(1+np.exp(-value)))
def perceptron_train(in_data, labels, alpha):
  X = np.array(in_data)
  y = np.array(labels)
  print(in_data)
  weights = np.random.random(X.shape([1]))
  original = weights
  bias = np.random.random_sample()

  for key in range(X.shape[0]):
    a = activation_func(np.matmul(np.transpose(weights),X[key]))
    yn = 0
    if a>0.5:
      yn =1
    weights = weights + alpha*(yn - y[key])*X[key]
    print('Iteration +')
```