XOR Problem

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from scipy import optimize
```

Bitwise Operator Dataset

```
# XOR
# X = np.array([[0,0], [0,1], [1,0], [1,1]])
# y = np.array([[0,1,1,0]]).flatten()

# OR
# X = np.array([[0,0], [0,1], [1,0], [1,1]])
# y = np.array([[0,1,1,1]]).flatten()

# AND
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0,0,0,1]]).flatten()
```

Perceptron

```
def perceptron(X, W, b):
    ## IMPLEMENT HERE
    y = X@W + b
    y = sigmoid(y) # 비선형(0에서 1사이로 만들어지는 장점)
    return y

def sigmoid(z):
    ## IMPLEMENT HERE
    z = 1 / (1 + np.exp(-z))
    return z
```

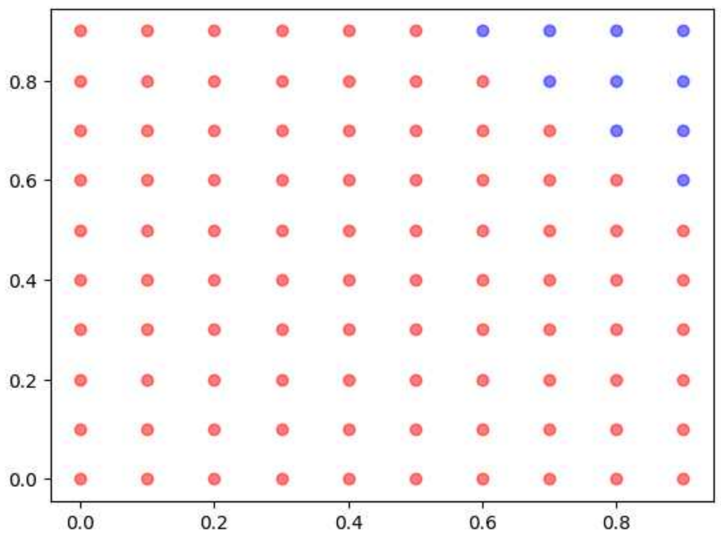
Single Perceptron Learning

```
def bce_loss(weights, args):
    ## IMPLEMENT HERE
    X = args[0]
    y = args[1]

    W, b = weights[:-1], weights[-1]

    y_hat = perceptron(X, W, b)

    # np.log(0)은 음의 무한대로 가기 때문에 입실론 값을 더해준다. 즉 아주 작은 값을 더해준다는 뜻. 그래서 1e-10을 더함
    bce = -y * np.log(y_hat + 1e-10) - (1.0 - y) * np.log(1.0 - y_hat + 1e-10)
    loss = bce.mean()

    return loss

result = optimize.minimize(fun = bce_loss, x0 = [0, 0, 0], args=[X, y])

W_opt, b_opt = result.x[:-1], result.x[-1]
y_hat = perceptron(X, W_opt, b_opt)
y_hat_cls = (y_hat > 0.5).astype('int8')

accuracy = accuracy_score(y, y_hat_cls)
print(y)
print(y_hat_cls)
print(f"Accuracy: {accuracy}")
```

```
[0 0 0 1]
[0 0 0 1]
Accuracy: 1.0
```

```
## visualization
color = ['red', 'blue']
```

```python
for x0 in np.arange(0,1,0.1):
    for x1 in np.arange(0,1,0.1):
        x = np.array([x0, x1])
        y_hat = perceptron(x, W_opt, b_opt)
        y_hat_cls = (y_hat > 0.5).astype('int8')
        plt.scatter(x0, x1, c=color[y_hat_cls], alpha=0.5)

plt.show()
```



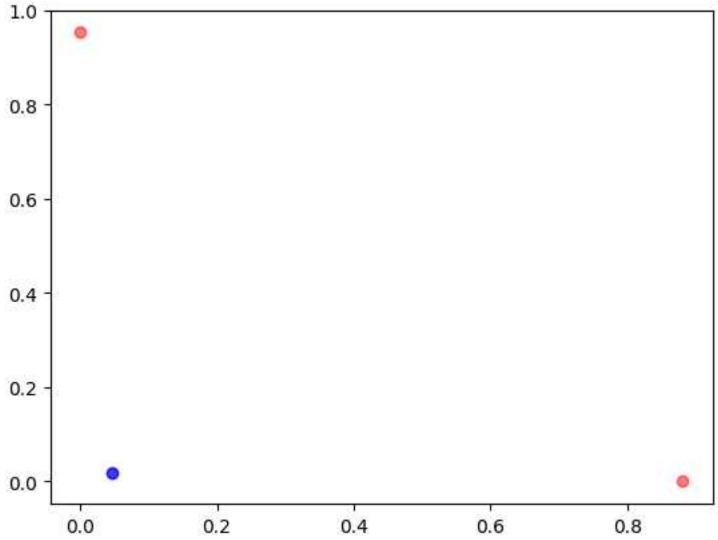
Multiple Perceptrons Learning

```python
w11 = np.array([5, 5])
w12 = np.array([-7, -7])
w2 = np.array([-15, -15])
b1 = -8
b2 = 3
b3 = 6

def predict(x):
  ## IMPLEMENT HERE
  y1 = perceptron(x, w11, b1)
  y2 = perceptron(x, w12, b2)
  x2 = np.array([y1, y2])
  y_hat = perceptron(x2, w2, b3)

  return y_hat, x2

```

Analysis of multiple perceptron

```python
# multiple perceptron
color = ['red', 'blue']
for x in X:
    y_hat, layer_x2 = predict(x)
    y_hat_cls = (y_hat > 0.5).astype('int8')
    print(f'{x} => {y_hat_cls}')
    plt.scatter(layer_x2[0], layer_x2[1], c=color[y_hat_cls], alpha=0.5)
```

```
[0 0] => 0
[0 1] => 1
[1 0] => 1
[1 1] => 0
```




```
color = ['red', 'blue']
for x0 in np.arange(0,1,0.1):
    for x1 in np.arange(0,1,0.1):
        x = np.array([x0, x1])
        y_hat, layer_x2 = predict(x)
        y_hat_cls = (y_hat > 0.5).astype('int8')
        plt.scatter(x0, x1, c=color[y_hat_cls], alpha=0.5)

plt.show()
```