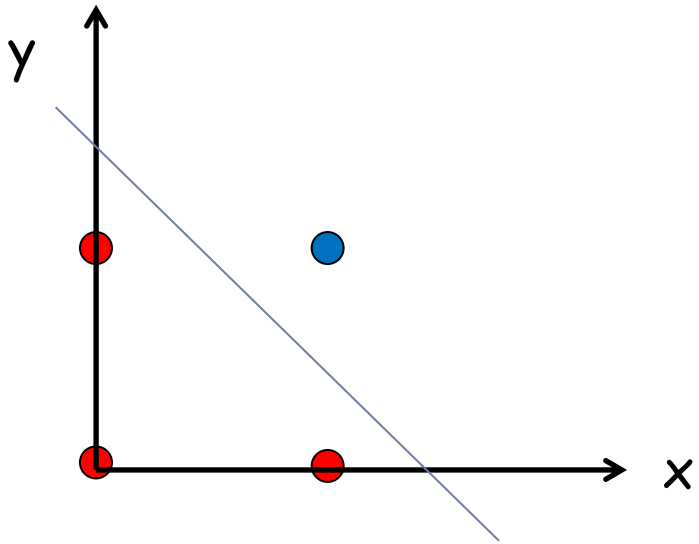
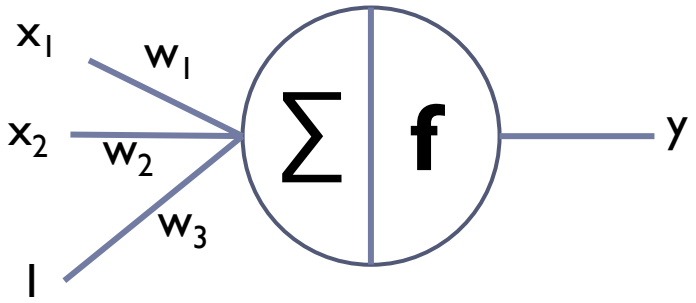
# Exercise 2

# What a Perceptron can do?

- And Operation

$(0.0, 0.0)\ (0.0, 1.0)\ (1.0, 0.0)\ (1.0, 1.0)$

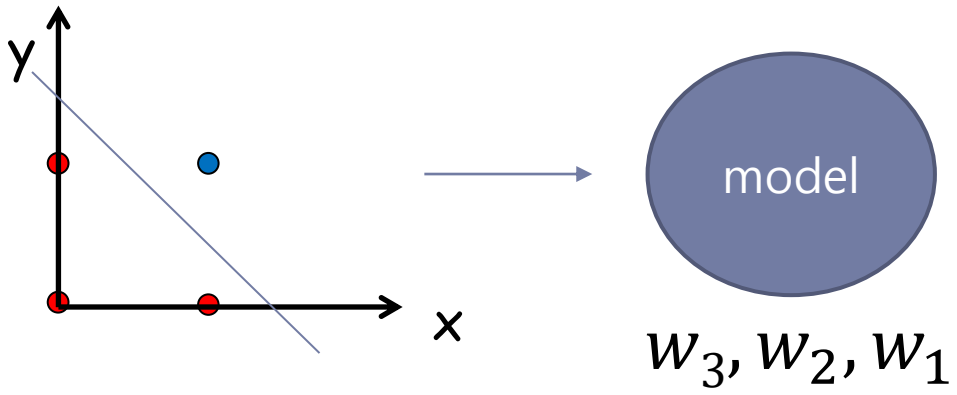



$w_1 = 1.0\ ,\ w_2 = 1.0\ ,\ w_3 = -1.5$

# Coding 전에 생각해 볼 것

- 입력 데이터 : (0.0)(0.0), (0.0)(1.0), (1.0)(0.0), (1.0)(1.0)
- 출력 데이터 : (0.0), (0.0), (0.0), (1.0)
- Optimizer: gradient descent method
- Loss function: Mean square error



# 준비 단계 1

- 입력 데이터 : (0.0)(0.0), (0.0)(1.0), (1.0)(0.0), (1.0)(1.0)
- 출력 데이터 : (0.0), (0.0), (0.0), (1.0)
- Model: nn.linear
- Loss function :F.mse_loss()
- Optimizer:SGD
- Learning rate:0.001

# Exercise 1

## And operation

```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

x_train = torch.FloatTensor([[0,0], [0,1], [1,0], [1,1]])
y_train = torch.FloatTensor([[0], [0], [0], [1]])
```

## linear model

```
class MRModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(                    ) #-> 맞는 숫자 채우기 #
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = (                                       )
        return x
```

# Exercise 1

```
model = MRModel()
optimizer = optim.SGD(model.parameters(), lr=1e-3)

nb_epochs = 50000

for epoch in range(nb_epochs + 1):
    # H(x) 계산
    prediction = model(x_train)
    # cost 계산

    cost = F.mse_loss(prediction, y_train)

    # cost로 H(x) 개선

    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    if epoch % 100 == 0 :
        print('Epoch {:4d}/{} Cost: {:.6f}'.format( epoch, nb_epochs, cost.item() ))
        #print(model.state_dict() )
```

# Exercise 1

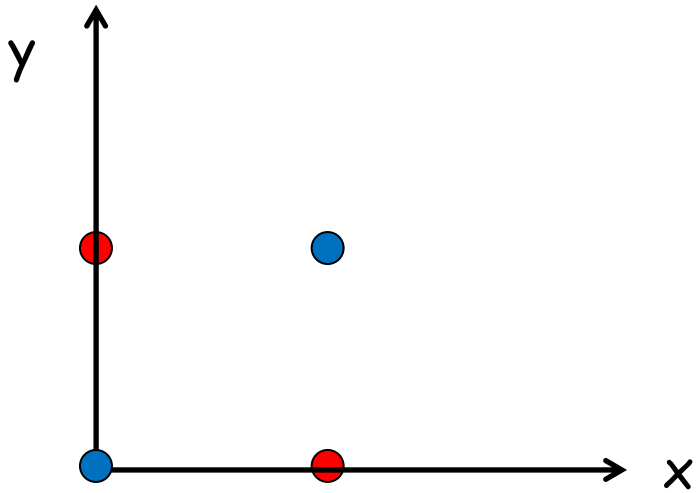
```
with torch.no_grad():
    hypothesis = model(x_train)
    predicted = (hypothesis > 0.5).float()
    accuracy = (predicted == y_train).float().mean()
    print('모델의 출력값(Hypothesis): ', hypothesis.detach().cpu().numpy())
    print('모델의 예측값(Predicted): ', predicted.detach().cpu().numpy())
    print('실제값(Y): ', y_train)
    print('정확도(Accuracy): ', accuracy.item())
```

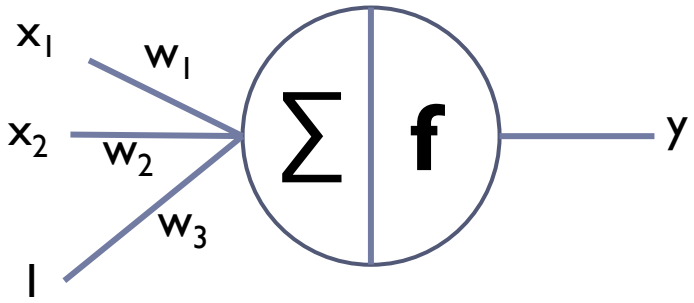
```
모델의 출력값(Hypothesis):  [[0.13428617]
 [0.3243814 ]
 [0.3044344 ]
 [0.5753231 ]]
모델의 예측값(Predicted):  [[0.]
 [0.]
 [0.]
 [1.]]
실제값(Y):  tensor([[0.],
        [0.],
        [0.],
        [1.]])
정확도(Accuracy):  1.0
```

# What a Perceptron can do?

- Xor Operation

$(0.0, 0.0)\ (0.0, 1.0)\ (1.0, 0.0)\ (1.0, 1.0)$




$w_1 = ?, w_2 = ?, w_3 = ?$

# Exercise 2

## Xor operation

```
x_train = torch.FloatTensor([[0,0], [0,1], [1,0], [1,1]])
y_train = torch.FloatTensor([[0], [1], [1], [0]])
```

```
model = MRModel()
optimizer = optim.SGD(model.parameters(), lr=1e-3)

nb_epochs = 50000

for epoch in range(nb_epochs + 1):
    # H(x) 계산
    prediction = model(x_train)
    # cost 계산

    cost = F.mse_loss(prediction, y_train)

    # cost로 H(x) 개선

    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    if epoch % 100 == 0 :
        print('Epoch {:4d}/{} Cost: {:.6f}'.format( epoch, nb_epochs, cost.item() ))
        #print(model.state_dict() )
```

```
Epoch 48200/50000 Cost: 0.250283
Epoch 48300/50000 Cost: 0.250282
Epoch 48400/50000 Cost: 0.250280
Epoch 48500/50000 Cost: 0.250279
Epoch 48600/50000 Cost: 0.250278
Epoch 48700/50000 Cost: 0.250276
Epoch 48800/50000 Cost: 0.250275
Epoch 48900/50000 Cost: 0.250274
Epoch 49000/50000 Cost: 0.250273
Epoch 49100/50000 Cost: 0.250271
Epoch 49200/50000 Cost: 0.250270
Epoch 49300/50000 Cost: 0.250269
Epoch 49400/50000 Cost: 0.250268
Epoch 49500/50000 Cost: 0.250266
Epoch 49600/50000 Cost: 0.250265
Epoch 49700/50000 Cost: 0.250264
Epoch 49800/50000 Cost: 0.250263
Epoch 49900/50000 Cost: 0.250262
Epoch 50000/50000 Cost: 0.250260
```

# Exercise 2

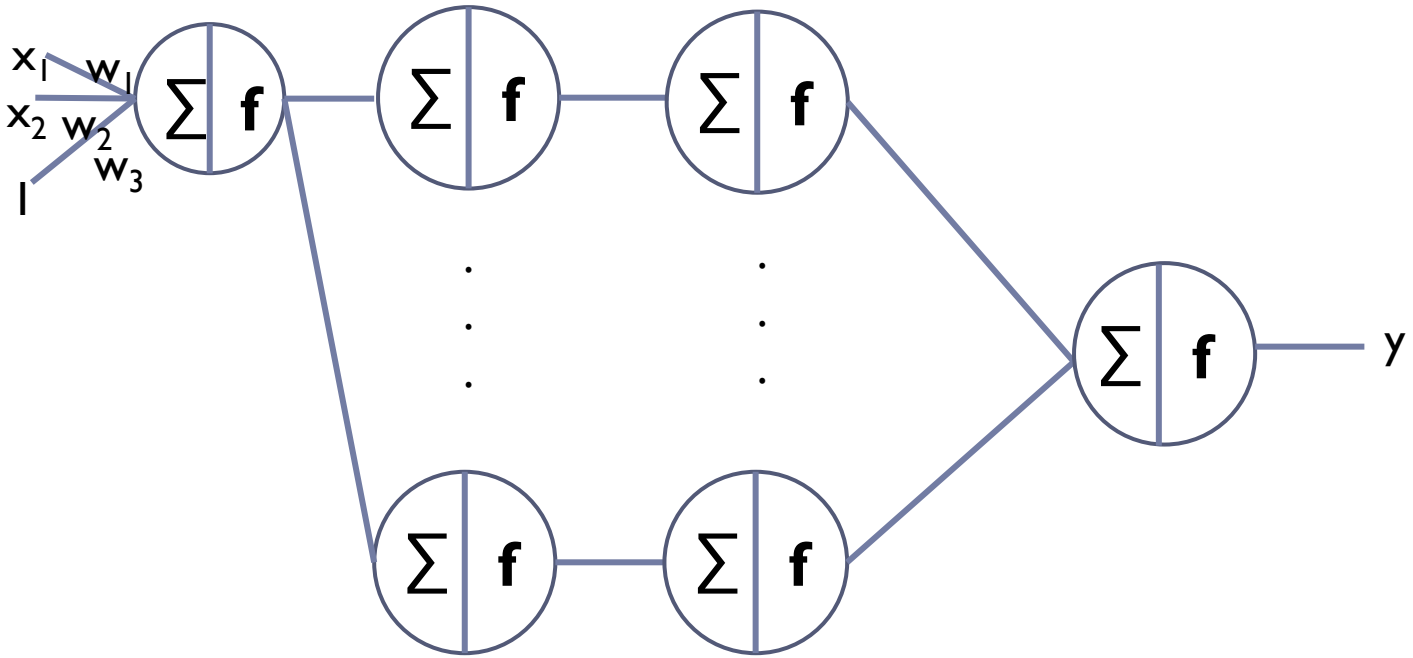
```
with torch.no_grad():
    hypothesis = model(x_train)
    predicted = (hypothesis > 0.5).float()
    accuracy = (predicted == y_train).float().mean()
    print('모델의 출력값(Hypothesis): ', hypothesis.detach().cpu().numpy())
    print('모델의 예측값(Predicted): ', predicted.detach().cpu().numpy())
    print('실제값(Y): ', y_train)
    print('정확도(Accuracy): ', accuracy.item())
```

```
모델의 출력값(Hypothesis):  [[0.5220674 ]
 [0.5160707 ]
 [0.49109605]
 [0.48509398]]
모델의 예측값(Predicted):  [[1.]
 [1.]
 [0.]
 [0.]]
실제값(Y):  tensor([[0.],
        [1.],
        [1.],
        [0.]])
정확도(Accuracy):  0.5
```

# How to fix it?

- **Xor Operation**

$$(0.0, 0.0)\ (0.0, 1.0)\ (1.0, 0.0)\ (1.0, 1.0)$$



# Exercise 2

## Xor operation

```
x_train = torch.FloatTensor([[0,0], [0,1], [1,0], [1,1]])
y_train = torch.FloatTensor([[0], [1], [1], [0]])
```

```
criterion = torch.nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1.0)
```

```
class MRModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear( 2,32 , bias=True )   #-> 맞는 숫자 채우기 #
        self.linear2 = nn.Linear(32, 64)
        self.linear3 = nn.Linear(64, 128)
        self.linear4 = nn.Linear(128, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.sigmoid(self.linear(x))
        x = self.sigmoid(self.linear2(x))
        x = self.sigmoid(self.linear3(x))
        x = self.sigmoid(self.linear4(x))
        return x
```

# Exercise 2

```python
model = MRModel()
optimizer = optim.SGD(model.parameters(), lr=0.1)

nb_epochs = 50000

for epoch in range(nb_epochs + 1):
    # H(x) 계산
    prediction = model(x_train)
    # cost 계산

    cost =criterion(prediction, y_train)

    # cost로 H(x) 개선

    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    if epoch % 1000 == 0 :
        print('Epoch {:4d}/{} Cost: {:.6f}'.format( epoch, nb_epochs, cost.item() ))
        #print(model.state_dict() )
```

```
Epoch 32000/50000 Cost: 0.000290
Epoch 33000/50000 Cost: 0.000264
Epoch 34000/50000 Cost: 0.000242
Epoch 35000/50000 Cost: 0.000223
Epoch 36000/50000 Cost: 0.000206
Epoch 37000/50000 Cost: 0.000192
Epoch 38000/50000 Cost: 0.000180
Epoch 39000/50000 Cost: 0.000169
Epoch 40000/50000 Cost: 0.000159
Epoch 41000/50000 Cost: 0.000150
Epoch 42000/50000 Cost: 0.000142
Epoch 43000/50000 Cost: 0.000135
Epoch 44000/50000 Cost: 0.000128
Epoch 45000/50000 Cost: 0.000122
Epoch 46000/50000 Cost: 0.000117
Epoch 47000/50000 Cost: 0.000111
Epoch 48000/50000 Cost: 0.000107
Epoch 49000/50000 Cost: 0.000103
Epoch 50000/50000 Cost: 0.000099
```

# Exercise 2

```
with torch.no_grad():
    hypothesis = model(x_train)
    predicted = (hypothesis > 0.5).float()
    accuracy = (predicted == y_train).float().mean()
    print('모델의 출력값(Hypothesis): ', hypothesis.detach().cpu().numpy())
    print('모델의 예측값(Predicted): ', predicted.detach().cpu().numpy())
    print('실제값(Y): ', y_train)
    print('정확도(Accuracy): ', accuracy.item())
```

```
모델의 출력값(Hypothesis):  [[6.9294612e-05]
 [9.9989522e-01]
 [9.9990714e-01]
 [1.2702450e-04]]
모델의 예측값(Predicted):  [[0.]
 [1.]
 [1.]
 [0.]]
실제값(Y):  tensor([[0.],
        [1.],
        [1.],
        [0.]])
정확도(Accuracy):  1.0
```

# Question and Answer