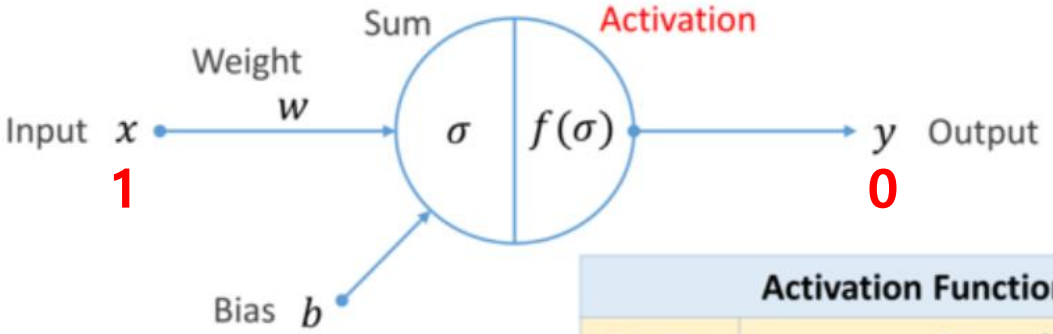
인공 신경망
퍼셉트론 실습

노트북 파일 생성

- **Ch03_study.ipynb**

뉴런 실습

- **입력과 출력**
  - 각각 1과 0
  - 가중치 w와 그 때의 결과 값을 출력
- **활성화 함수**
  - 뉴런의 출력 값을 정하는 함수
    - **시그모이드 함수 적용**

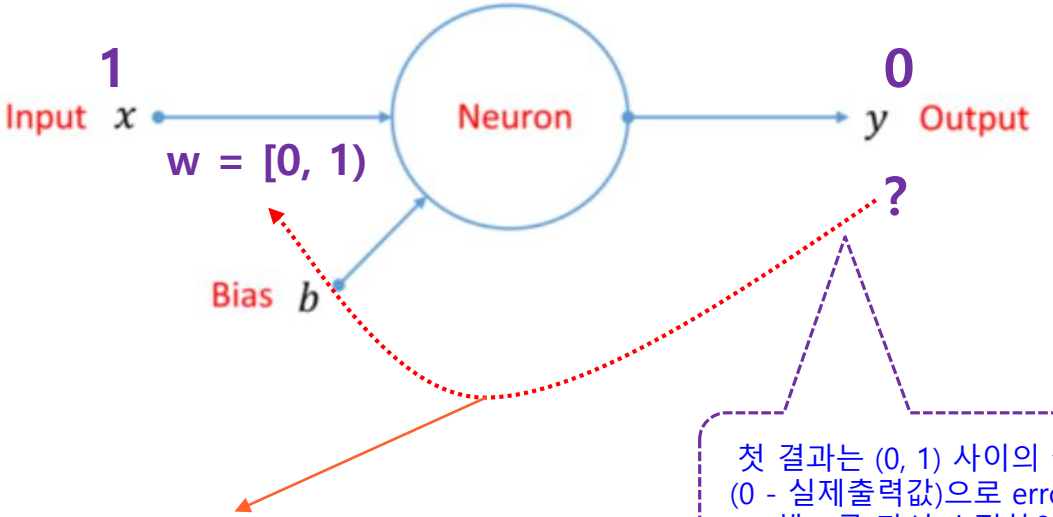


Activation Functions	
Name	**Formula**
Identity	$f(x) = x$
ReLU	$f(x) = \max(x, 0)$
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$

*ReLU: Rectified Linear Unit

여러 번 반복하여 error를 줄이는 방향

- **에러 error**
  - error = y(기대출력값) – output(실제출력값)
- **가중치 w의 조정**
  - 경사하강법(gradient descent)
    - **이전 w에 입력 x와 학습률(알파), 그리고 error를 곱하여 더함**
  - 학습률은 0.1로 설정
    - **크면 원하는 값이 안 나올 수 있고, 작으면 너무 느린 점**



뉴런 실습 x=1 y=0

- 입력 1, 출력 0, 활성화함수 sigmoid인 뉴런

```
# 3.10 sigmoid 함수
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

import tensorflow as tf
# 3.11 뉴런의 입력과 출력 정의
x = 1
y = 0
w = tf.random.normal([1],0,1)
output = sigmoid(x * w)
print('처음 :', w.numpy(), output)

# 3.12 경사 하강법을 이용한 뉴런의 학습
for i in range(1000):
    output = sigmoid(x * w)
    error = y - output
    w = w + x * 0.1 * error

    if i % 100 == 99:
        print(i, w.numpy(), error, output)
```

```
처음 : [-0.15195797] [0.4620835]
99 [-2.2167132] [-0.09914159] [0.09914159]
199 [-2.9182146] [-0.05151156] [0.05151156]
299 [-3.3361375] [-0.03446656] [0.03446656]
399 [-3.6330464] [-0.02581956] [0.02581956]
499 [-3.8630242] [-0.02061385] [0.02061385]
599 [-4.0505795] [-0.01714314] [0.01714314]
699 [-4.208868] [-0.01466669] [0.01466669]
799 [-4.3457565] [-0.01281204] [0.01281204]
899 [-4.466319] [-0.0113718] [0.0113718]
999 [-4.5740237] [-0.01022136] [0.01022136]
```

뉴런 실습 x=0 y=1

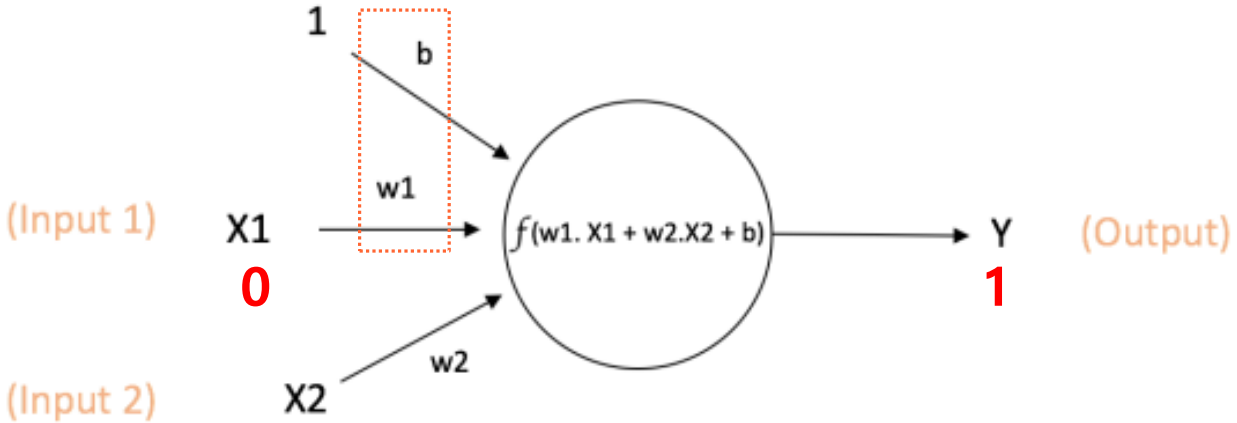
- **항상 결과가 0.5**
  - x가 0이므로 항상 w도 동일

```
# 3.13 x=0 일 때 y=1 을 얻는 뉴런의 학습
x = 0
y = 1
w = tf.random.normal([1],0,1)

for i in range(1000):
    output = sigmoid(x * w)
    error = y - output
    w = w + x * 0.1 * error

    if i % 100 == 99:
        print(i, error, output)
```

```
99 [0.5] [0.5]
199 [0.5] [0.5]
299 [0.5] [0.5]
399 [0.5] [0.5]
499 [0.5] [0.5]
599 [0.5] [0.5]
699 [0.5] [0.5]
799 [0.5] [0.5]
899 [0.5] [0.5]
999 [0.5] [0.5]
```

뉴런 실습 x=0 y=1 편향 사용

- **이제 뉴런은 w와 b를 구하는 것**
  - 입력이 하나이므로 x1과 b만 사용



Output of neuron = Y= $f(w1.\ X1 + w2.X2 + b)$

뉴런 실습 x=0 y=1 편향 사용 소스

```
# 3.10 sigmoid 함수
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

import tensorflow as tf
# 3.14 x=0 일 때 y=1 을 얻는 뉴런의 학습에 편향을 더함
x = 0
y = 1
w = tf.random.normal([1],0,1)
b = tf.random.normal([1],0,1)

for i in range(1000):
    output = sigmoid(x * w + 1 * b)
    error = y - output
    w = w + x * 0.1 * error
    b = b + 1 * 0.1 * error

    if i % 100 == 99:
        print(i, error, output)
```

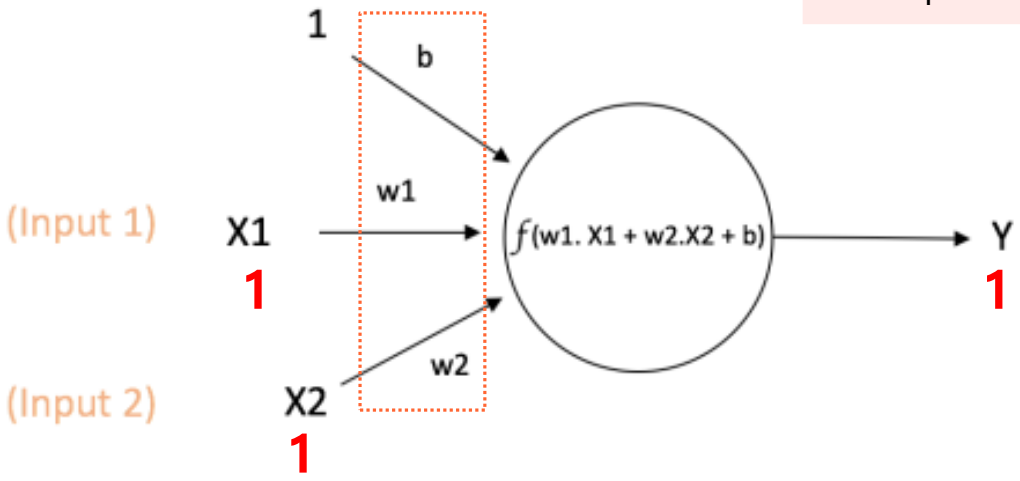
```
99 [0.07639199] [0.923608]
199 [0.04436541] [0.9556346]
299 [0.03106767] [0.96893233]
399 [0.02384973] [0.9761503]
499 [0.019333] [0.980667]
599 [0.01624507] [0.98375493]
699 [0.01400292] [0.9859971]
799 [0.01230174] [0.98769826]
899 [0.01096749] [0.9890325]
999 [0.00989318] [0.9901068]
```

논리 게이트 AND OR XOR 신경망 구현

AND 게이트 구현

- **뉴런 구조**
  - 입력 2개, 편향, 출력 1
  - 구할 값
    - **가중치 2개와 편향 1개**

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



Output of neuron = Y= $f(w1.X1 + w2.X2 + b)$

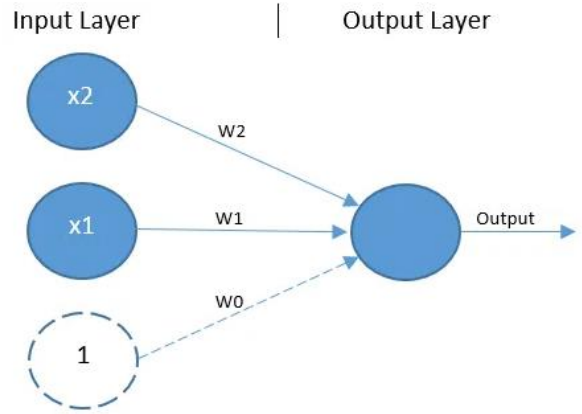


Figure 2: Single Layer Perceptron Network

AND 게이트 구현 소스

```
# 3.16 첫번째 신경망 네트워크 : AND
import tensorflow as tf
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [0], [0], [0]])
w = tf.random.normal([2],0,1)
b = tf.random.normal([1],0,1)
b_x = 1

for i in range(2000):
    error_sum = 0
    for j in range(4):
        output = sigmoid( np.sum(x[j]*w) + b_x*b )
        error = y[j][0] - output
        w = w + x[j] * 0.1 * error
        b = b + b_x * 0.1 * error
        error_sum += error

    if i % 200 == 199:
        print(i, error_sum)

# 3.20 AND 네트워크의 평가
for i in range(4):
    print('X:', x[i], 'Y:', y[i], 'Output:', sigmoid(np.sum(x[i]*w)+b))
```

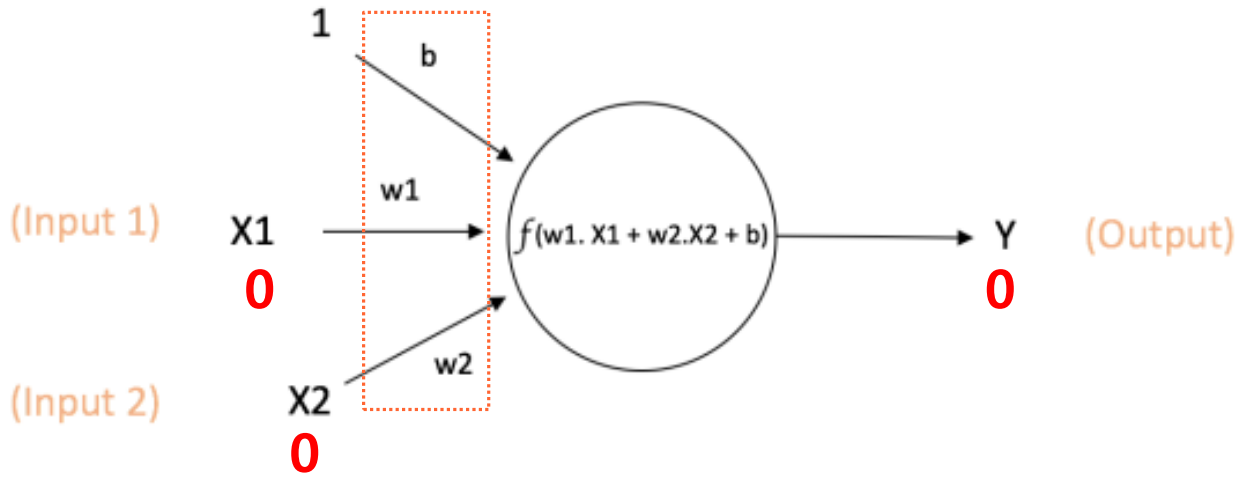
오류(손실)인 error_sum 값이 계속 줄어 듬

```
199 [-0.10670658]
399 [-0.06441742]
599 [-0.04601353]
799 [-0.03568755]
999 [-0.02909387]
1199 [-0.02452984]
1399 [-0.02118737]
1599 [-0.01863724]
1799 [-0.01662879]
1999 [-0.01500608]
X: [1 1] Y: [1] Output: [0.9649666]
X: [1 0] Y: [0] Output: [0.02482659]
X: [0 1] Y: [0] Output: [0.02490228]
X: [0 0] Y: [0] Output: [2.3604067e-05]
```

OR 게이트 구현

- **뉴런 구조**
  - 입력 2개, 편향, 출력 1
  - 구할 값
    - **가중치 2개와 편향 1개**

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1



Output of neuron = Y= $f(w1.\ X1 + w2.X2 + b)$

OR 게이트 구현 소스

```
# 3.21 두번째 신경망 네트워크 : OR
import numpy as np
import tensorflow as tf

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [1], [1], [0]])
w = tf.random.normal([2],0,1)
b = tf.random.normal([1],0,1)
b_x = 1

for i in range(2000):
    error_sum = 0
    for j in range(4):
        output = sigmoid(np.sum(x[j]*w)+b_x*b)
        error = y[j][0] - output
        w = w + x[j] * 0.1 * error
        b = b + b_x * 0.1 * error
        error_sum += error

    if i % 200 == 199:
        print(i, error_sum)

# 3.22 OR 네트워크의 평가
for i in range(4):
    print('X:', x[i], 'Y:', y[i], 'Output:', sigmoid(np.sum(x[i]*w)+b))
```

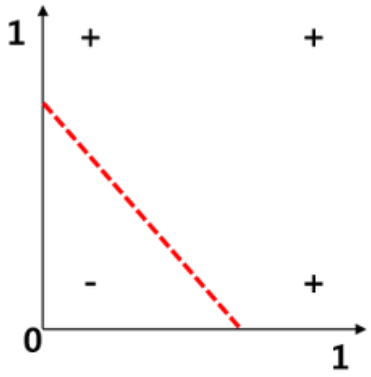
오류(손실)인 error_sum 값이 계속 줄어 듬

```
199 [-0.05037533]
399 [-0.02611086]
599 [-0.01750588]
799 [-0.01311971]
999 [-0.01047165]
1199 [-0.00870393]
1399 [-0.00744218]
1599 [-0.00649664]
1799 [-0.00576317]
1999 [-0.00517602]
X: [1 1] Y: [1] Output: [0.99999714]
X: [1 0] Y: [1] Output: [0.98973095]
X: [0 1] Y: [1] Output: [0.9897216]
X: [0 0] Y: [0] Output: [0.02566187]
```

XOR 문제

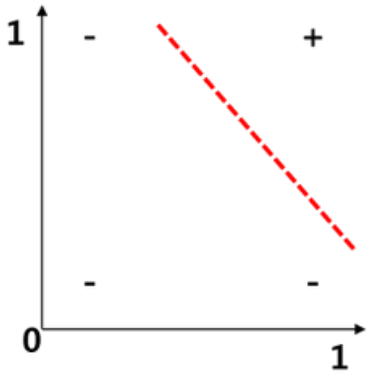
- **하나의 퍼셉트론으로는 XOR 게이트는 불가능**
  - 마빈 민스키와 시모어 페퍼트가 증명
  - 첫 AI 겨울의 계기

OR



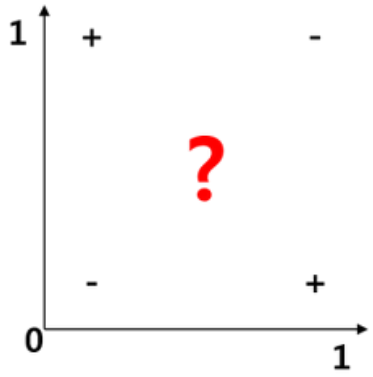
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

AND



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

XOR



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

XOR 해결

- **뉴런 3 개의 2층으로 가능**
  - 모델이 구해야 할 총 매개변수(가중치와 편향)
    - **3 * 2 + 3 * 1 = 9개**

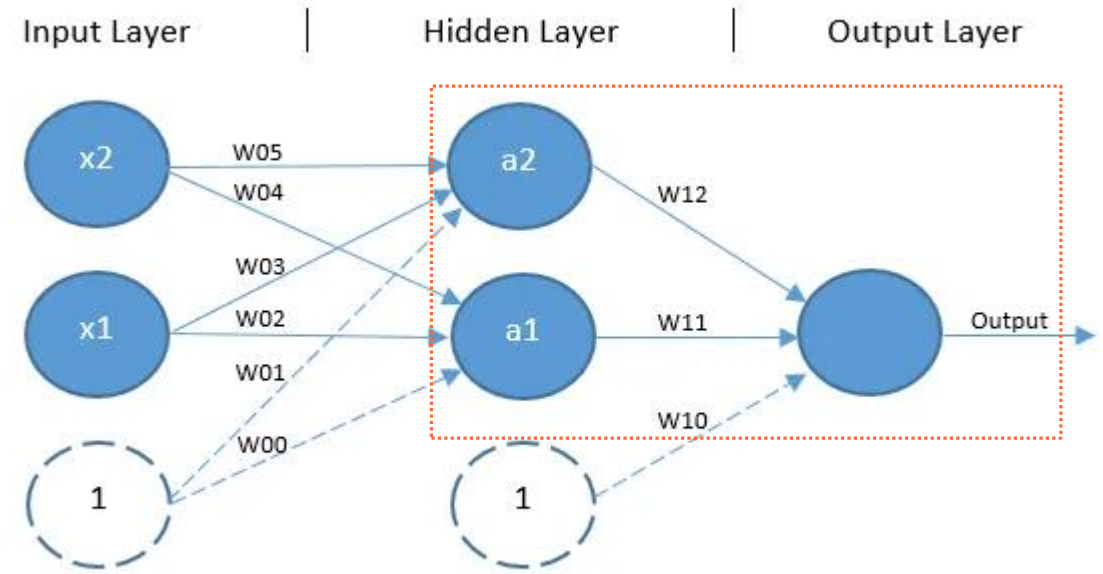


Figure 4: Multilayer Pereceptron Architecture for XOr

Sequential 모델

- **Dense 층**
  - 가장 기본적인 층
  - 인자 units, activation
    - **뉴런 수와 활성화 함수**
  - 인자 input_shape
    - **첫 번째 층에서만 정의**
    - **입력의 차원을 명시**
      - **(2,)**
      - **2개의 입력을 받는 1차원**

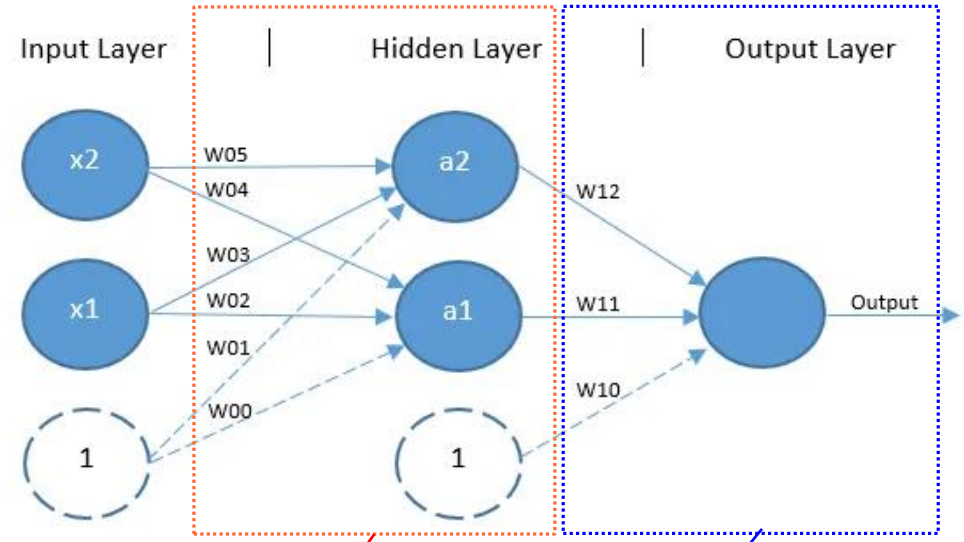


Figure 4: Multilayer Pereceptron Architecture for XOr

```
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

Sequential 모델과 딥러닝 구조

- **입력, 은닉, 출력 층**
  - 패러미터 수
    - **(입력측 뉴런 수 + 1) * (출력측 뉴런 수)**

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 2)                 6
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

```
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

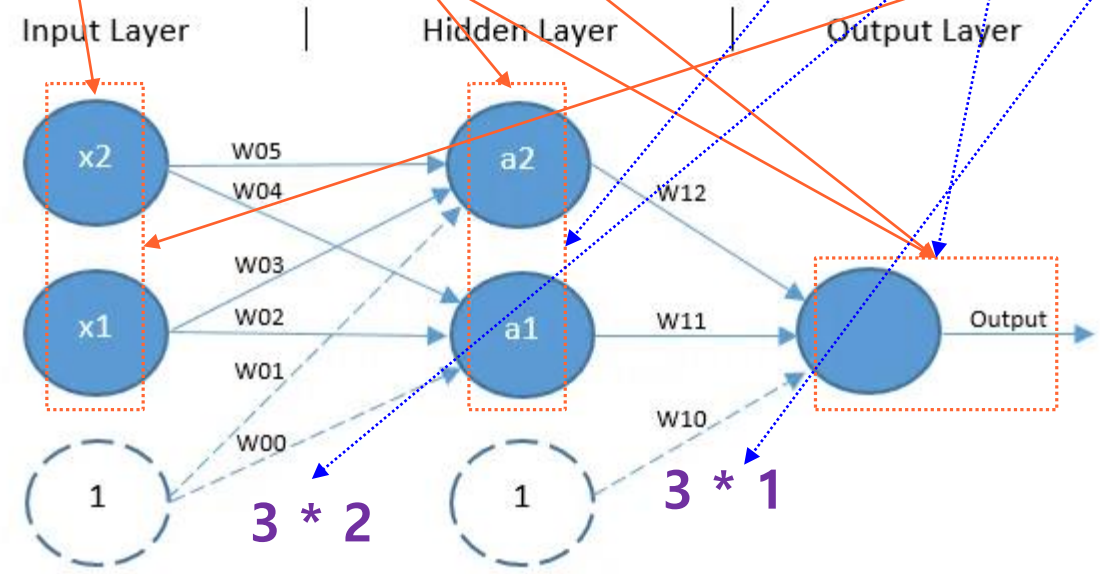


Figure 4: Multilayer Pereceptron Architecture for XOr

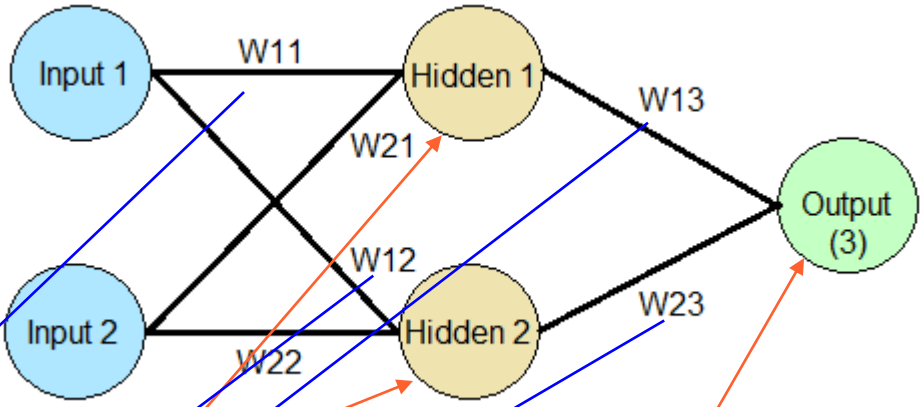
XOR 게이트 구현 소스

```
# 3.27 tf.keras 를 이용한 XOR 네트워크 계산
import tensorflow as tf
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()

# 3.28 tf.keras 를 이용한 XOR 네트워크 학습
history = model.fit(x, y, epochs=2000, batch_size=1)

# 3.29 tf.keras 를 이용한 XOR 네트워크 평가
print(model.predict(x))

# 3.30 XOR 네트워크의 가중치와 편향 확인
for weight in model.weights:
    print(weight)
```

```
Epoch 1999/2000
4/4 [==============================] - 0s 2ms/step - loss: 0.0017
Epoch 2000/2000
4/4 [==============================] - 0s 1ms/step - loss: 0.0017
[[0.04060324]
 [0.9609237 ]
 [0.96031225]
 [0.04571233]]
<tf.Variable 'dense_2/kernel:0' shape=(2, 2) dtype=float32, numpy=
array([[ 5.378626 , -5.299518 ],
       [-5.518024 ,  5.0764313]], dtype=float32)>
<tf.Variable 'dense_2/bias:0' shape=(2,) dtype=float32, numpy=array([-3.0511274, -2.8576972], dtype=float32)>
<tf.Variable 'dense_3/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[7.6903753],
       [7.7518315]], dtype=float32)>
<tf.Variable 'dense_3/bias:0' shape=(1,) dtype=float32, numpy=array([-3.8067687], dtype=float32)>
```

가중치와 model.weights

- **가중치 결과**
  - 층/kernel
- **편향 결과**
  - 층/bias




```
<tf.Variable 'dense_2/kernel:0' shape=(2, 2) dtype=float32, numpy=
array([[ 5.378626 , -5.299518 ],
       [-5.518024 ,  5.0764313]], dtype=float32)>
<tf.Variable 'dense_2/bias:0' shape=(2,) dtype=float32, numpy=array([-3.0511274, -2.8576972], dtype=float32)>
<tf.Variable 'dense_3/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[7.6903753],
       [7.7518315]], dtype=float32)>
<tf.Variable 'dense_3/bias:0' shape=(1,) dtype=float32, numpy=array([-3.8067687], dtype=float32)>
```

XOR 모델의 학습 과정 시각화

- **손실(loss) 또는 오류 값의 변화**
  - 가로는 에폭의 수
    - **학습 횟수가 증가하면서 계속 손실은 작아짐**

```
# 3.34 2-레이어 XOR 네트워크의 loss 변화를 선 그래프로 표시
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
```