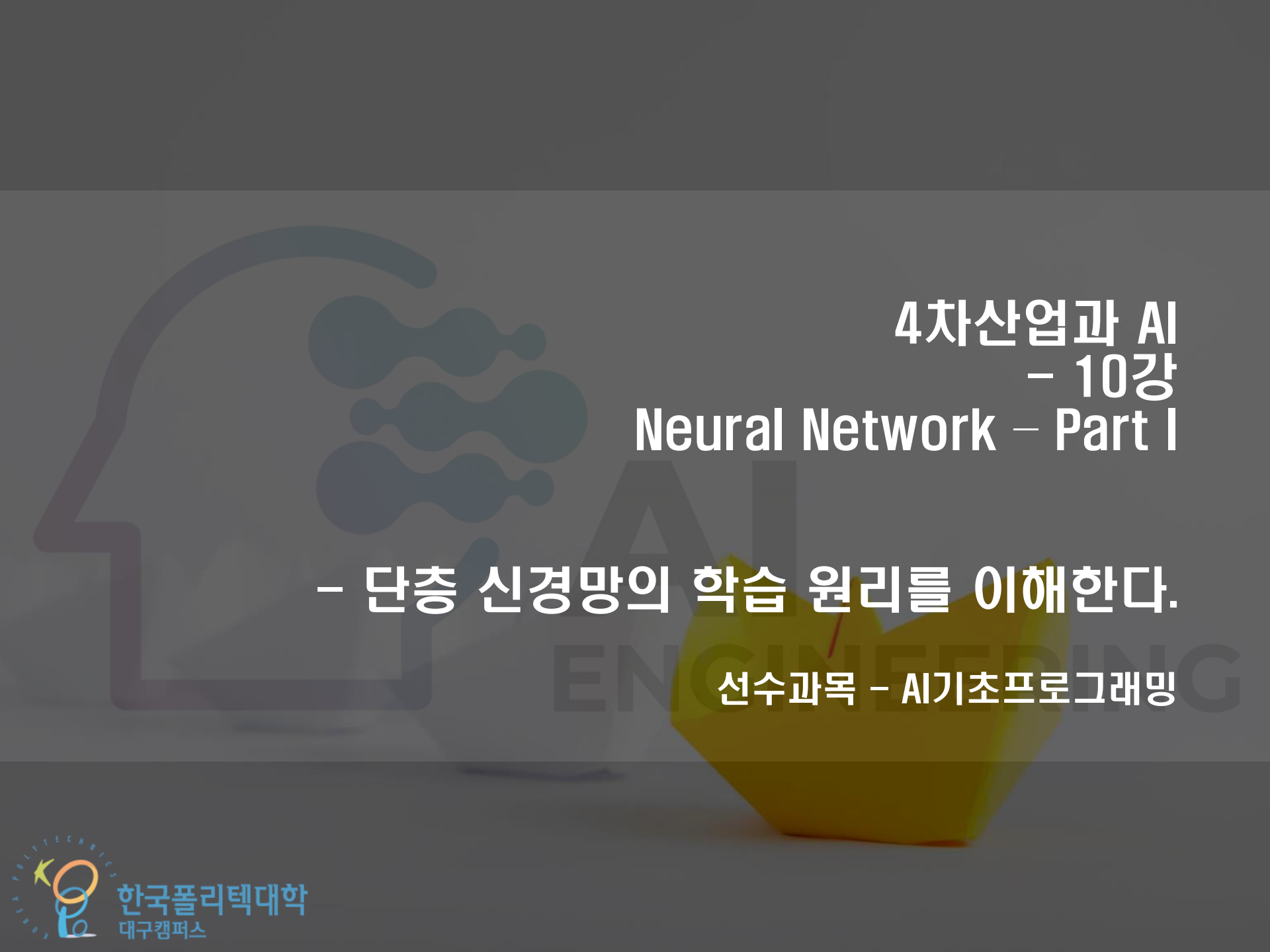
4차산업과 AI

한국폴리텍대학 대구캠퍼스
AI엔지니어링학과 강현우

4차산업과 AI
- 10강
Neural Network – Part I

- 단층 신경망의 학습 원리를 이해한다.

선수과목 - AI기초프로그래밍

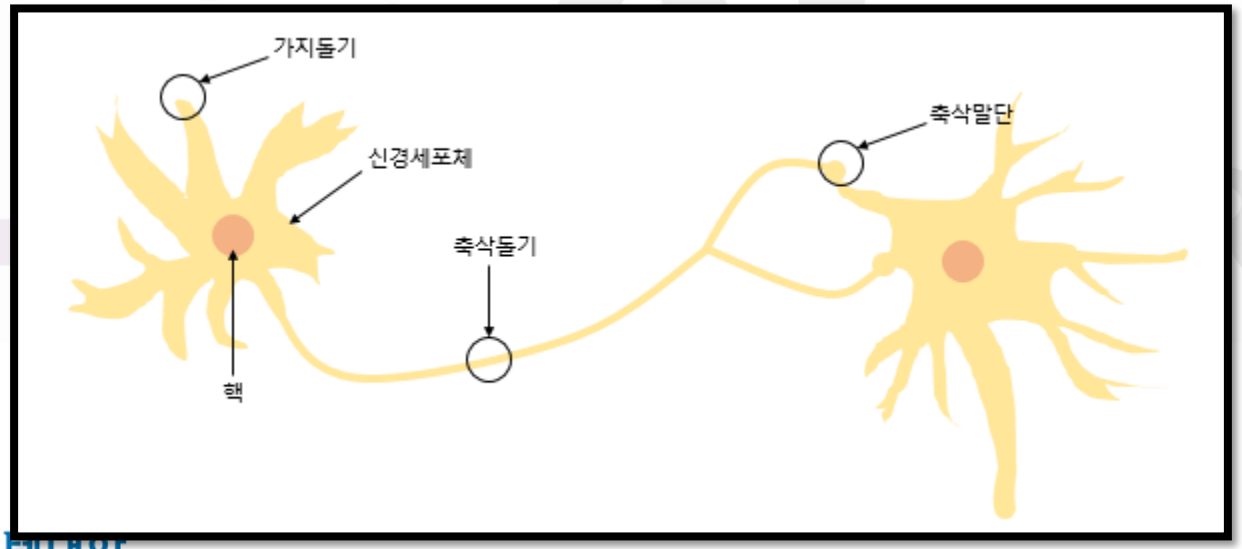


한국폴리텍대학
대구캠퍼스

Neural Network

◆ 신경망

- ➢ 인간이 뇌를 통해 문제를 처리하는 방법과 비슷한 방법으로 컴퓨터에서 문제를 해결하려는 모델
- ➢ 뉴런
  - ✓가지돌기에서 신호를 받음
  - ✓신호가 일정치 이상이면 축삭 돌기로 신호를 전달

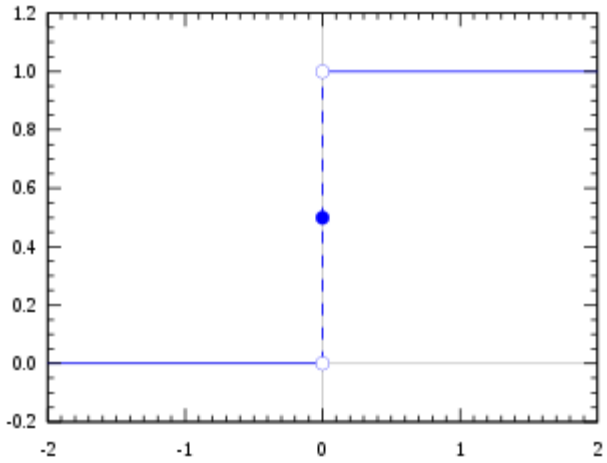


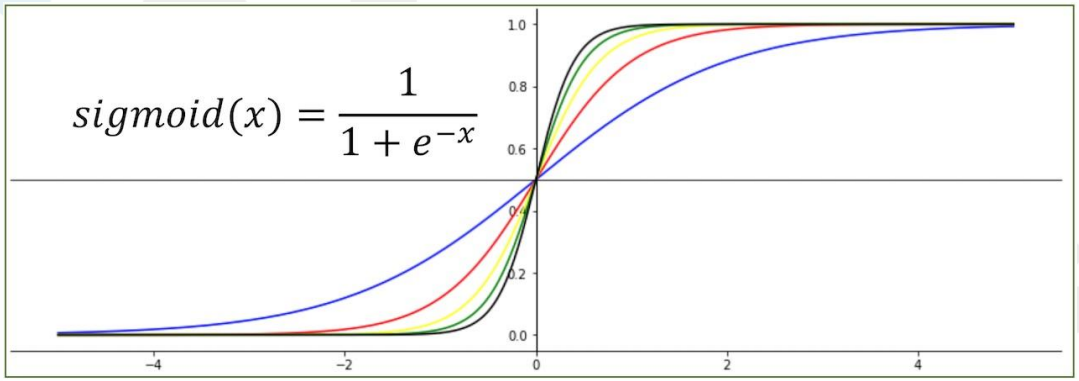
출처: https://wikidocs.net/24958

Neural Network

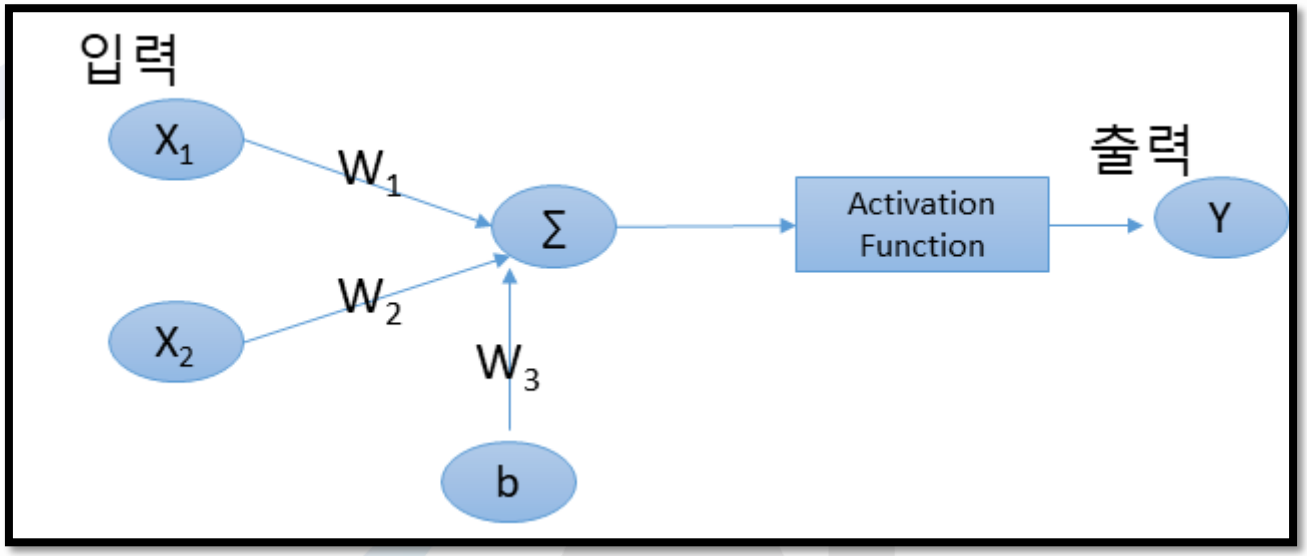
◆ Activation Function (활성화 함수)

- ➢ 뉴런에 가해지는 신호가 특정 임계치 까지는
- ➢ 아무런 신호를 보내지 않다가 임계치를 초과하면
- ➢ 신호를 내보내는 것을 모방





Neural Network



활성화 함수 $y = f(\sum W_i X_i)$

가중합 = $W_o X_0 + W_1 X_1 + W_3 b$

Neural Network

- **신경망**
  - 뉴런들을 여러 개 연결한 네트워크
- **신경망 학습**
  - 데이터를 통해 W의 값을 찾아내겠다
- **모델**
  - W 값들을 다 저장해둔 게 모델이지 뭐.

Weight 학습

◆어떤 입력에 대하여

- ➢ 원하는 출력값(목표값) d 와
- ➢ 모델의 출력 y 사이에 차이가 발생
- ➢ 현재의 weight에 차이를 더해서 weight를 갱신

◆ weigh를 갱신할 때 차이를 얼마나 줘?

- ➢ 그게 바로 **학습률**
- ➢ 학습률이 너무 높으면 최적의 결과를 얻지 못함
- ➢ 너무 낮으면 학습 시간이 오래 걸림

$$W_{n+1} = W_n + \eta(d - y)$$

And 게이트 학습

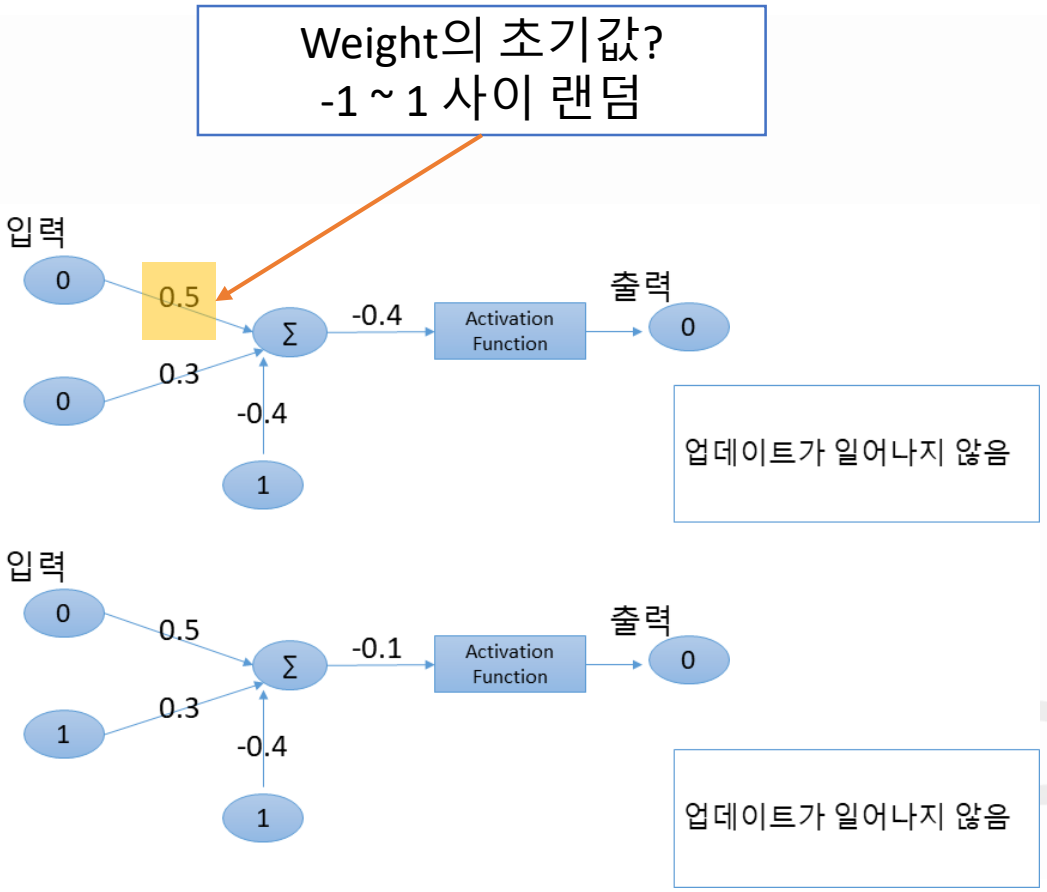
학습데이터

x1	x2	d
0	0	0
0	1	0
1	0	0
1	1	1

Activation Function

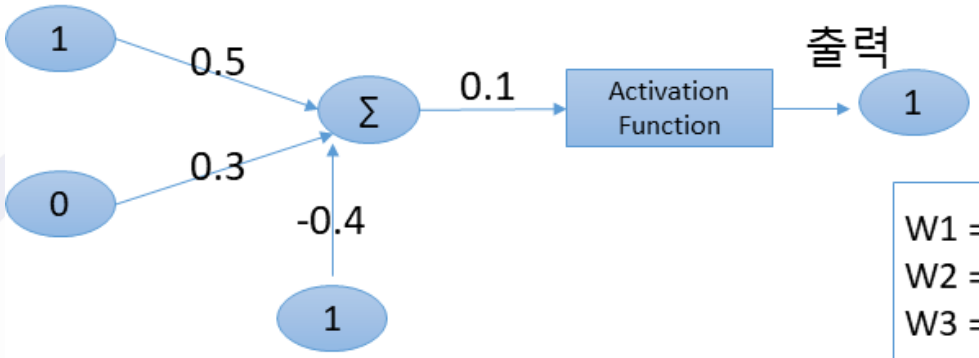
$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

학습율 5% (0.05)

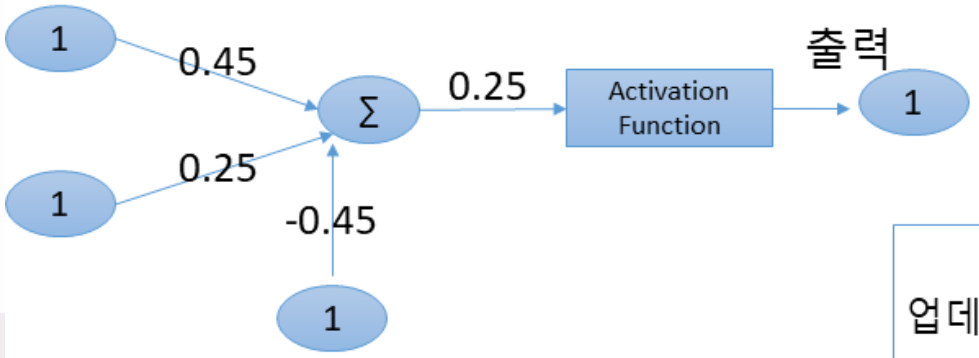


1라운드

And 게이트 학습



W1 = 0.5 + 0.05(0-1) = 0.45
W2 = 0.3 + 0.05(0-1) = 0.25
W3 = -0.4 + 0.05(0-1) = -0.45



업데이트가 일어나지 않음

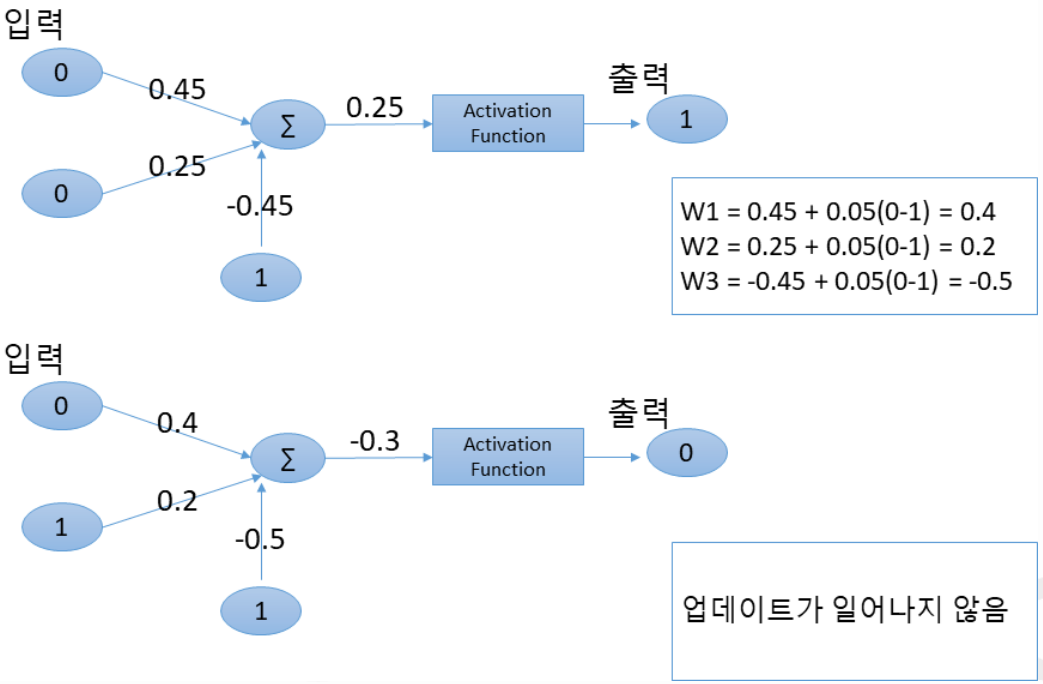
1라운드

라운드? = 모든 학습데이터를 1회 학습하면 1라운드
Epoch 라는 표현으로 많이 사용함

And 게이트 학습

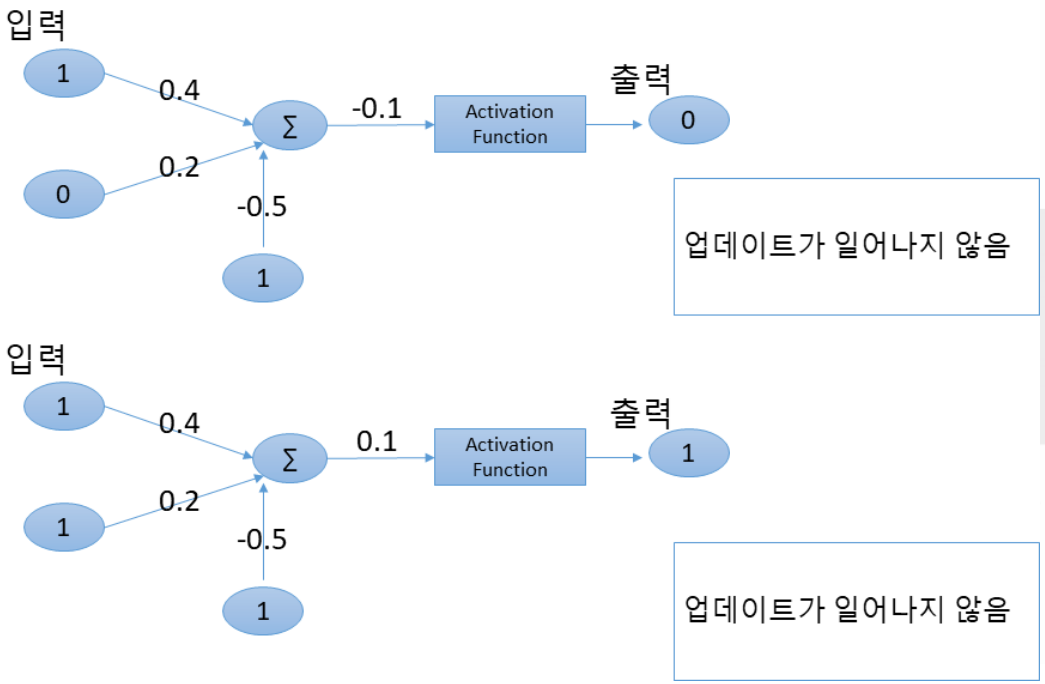
x1	x2	d	y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

1라운드 종료 후 출력값
정답률 = 75%



2라운드

And 게이트 학습

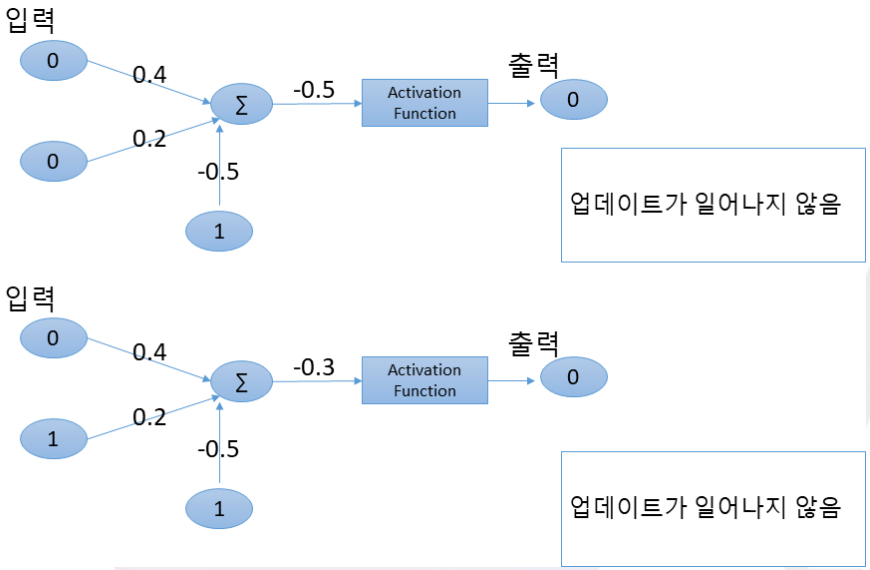


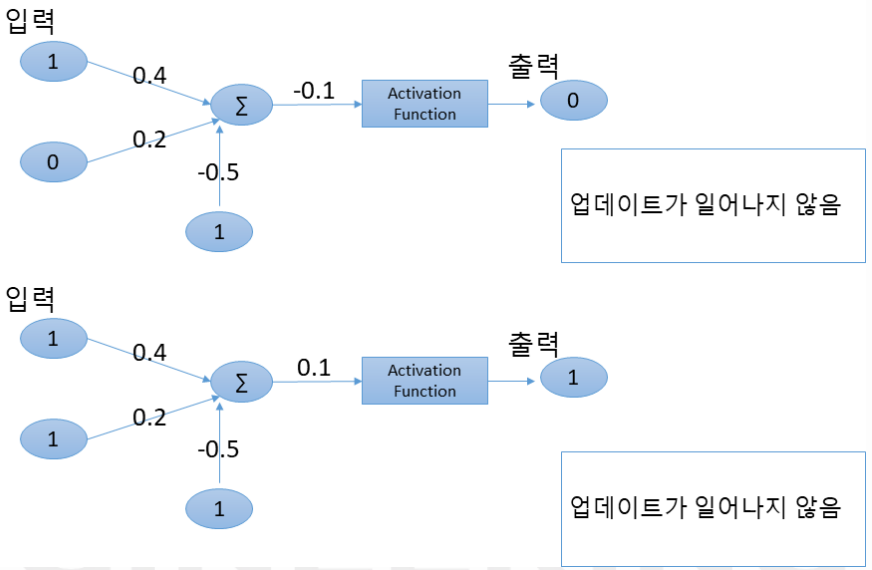
2라운드

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

2라운드 결과

And 게이트 학습





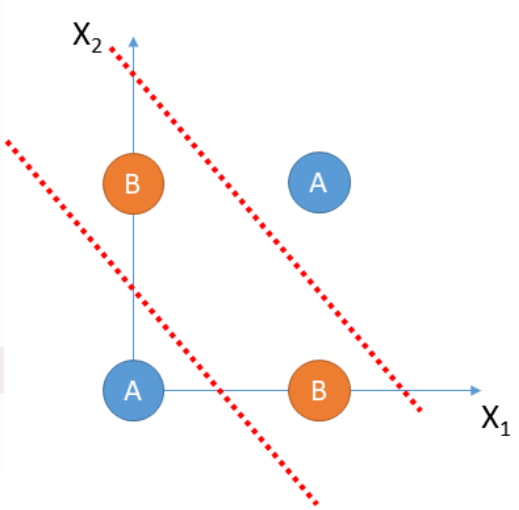
3라운드

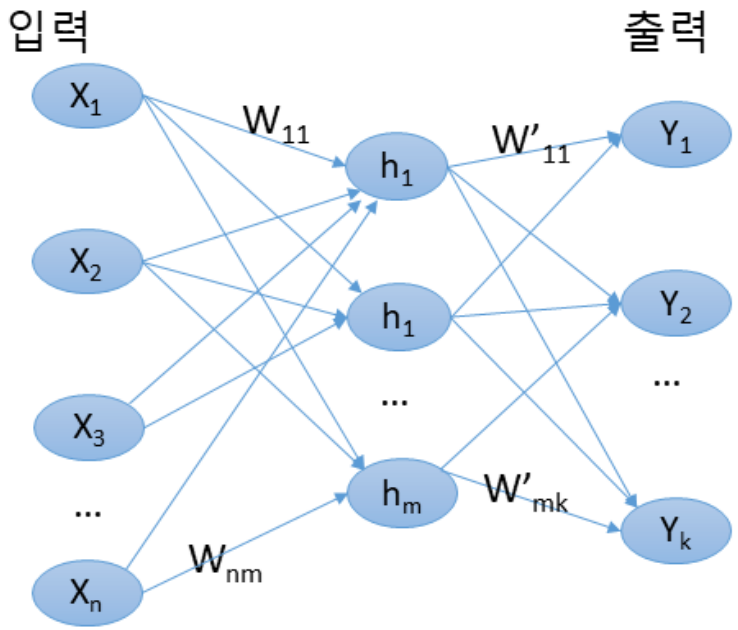
학습된 모델 구현

```
def AND_gate(x1, x2):
    w1 = 0.4
    w2 = 0.2
    # w3은 b로 표시.
    # b는 1이므로 1*w3 = w3 와 같다. 그러나! 다른 웨이트와 구별하기 위해
    b = -0.5

     w_sum = w1 * x1 + w2 * x2 + b

    # activation function
    # 0이하 0, 그 외 1
    if w_sum <= 0:
        return 0
    else:
        return 1

if __name__ == "__main__":
    print("(0, 0)", AND_gate(0, 0))
    print("(0, 1)", AND_gate(0, 1))
    print("(1, 0)", AND_gate(1, 0))
    print("(1, 1)", AND_gate(1, 1))
```

XOR 문제

- 단층 신경망으로는 XOR 문제를 해결 X
- 다층 신경망으로 발전 ~~(10년 걸림…)~~
  - Multi Layer Perceptron; MLP



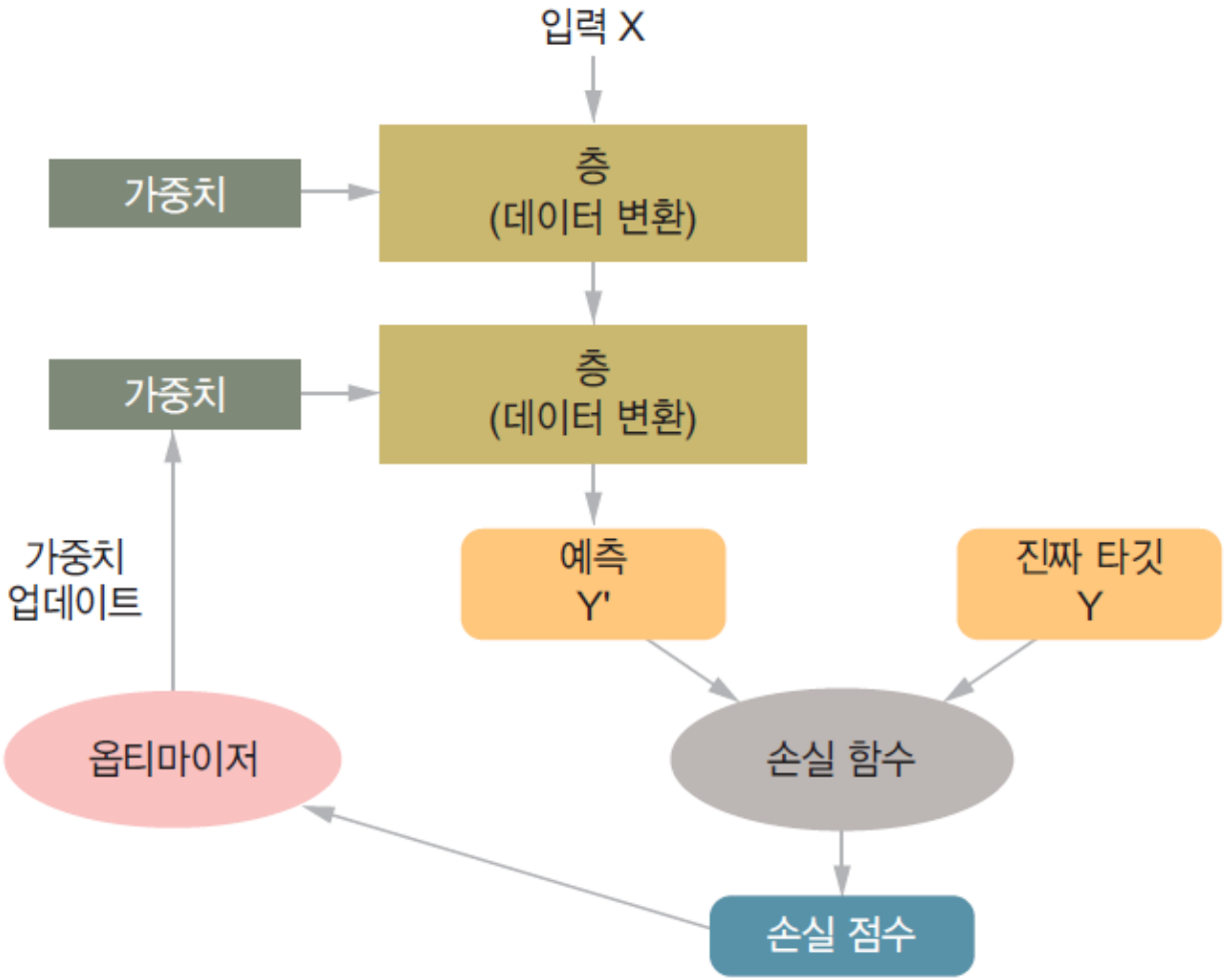


TensorFlow – Keras 로 모델 만들기

```python
# keras tesnor flow 신경망
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


def make_model():
    model = Sequential()
    model.add(Dense(2, input_dim=2, kernel_initializer='normal', activation="sigmoid"))
    model.summary()
    model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

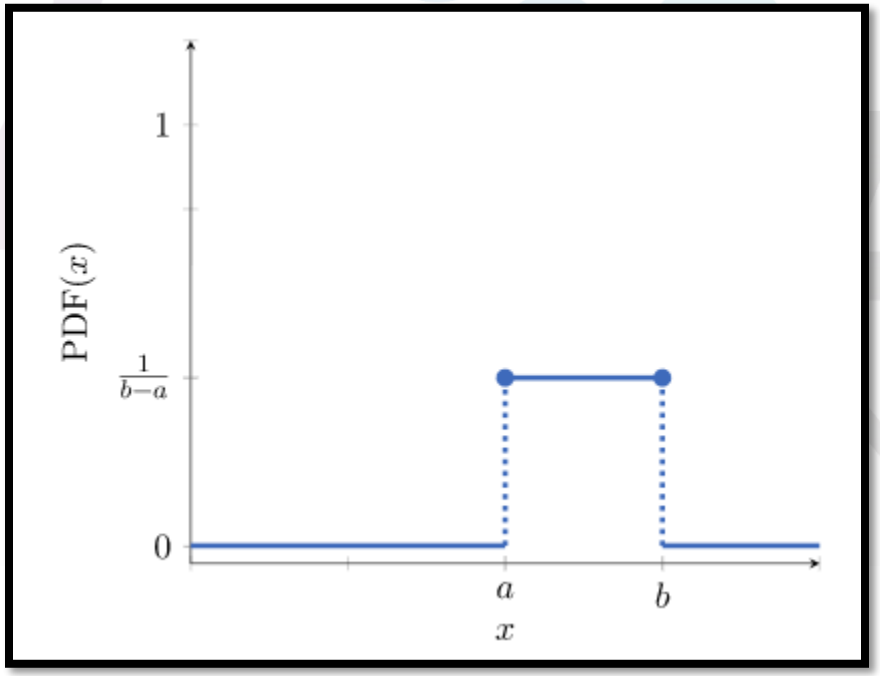
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
```

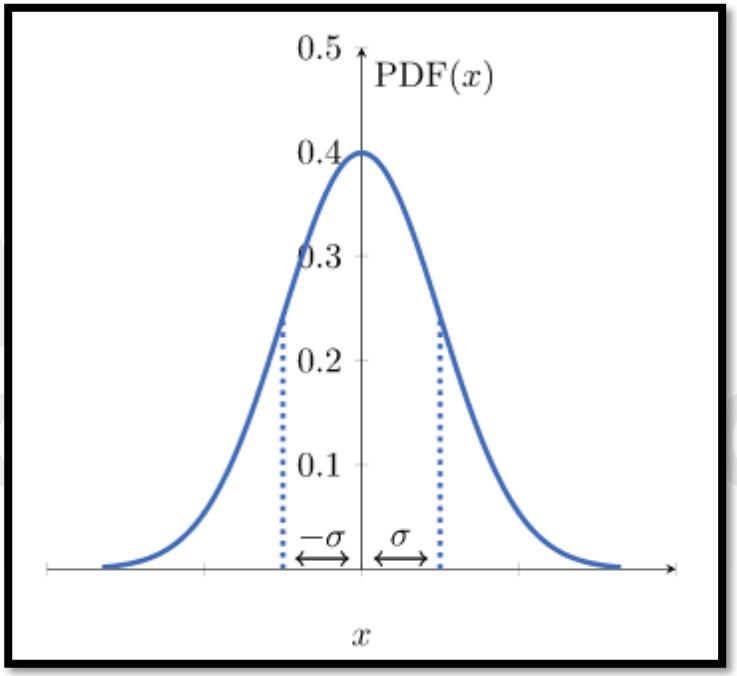
학습 과정



Random

- **Normal Distribution / Uniform Distribution**
  - 확률 밀도 함수 (Probability Density Function)





One Hot Encoding

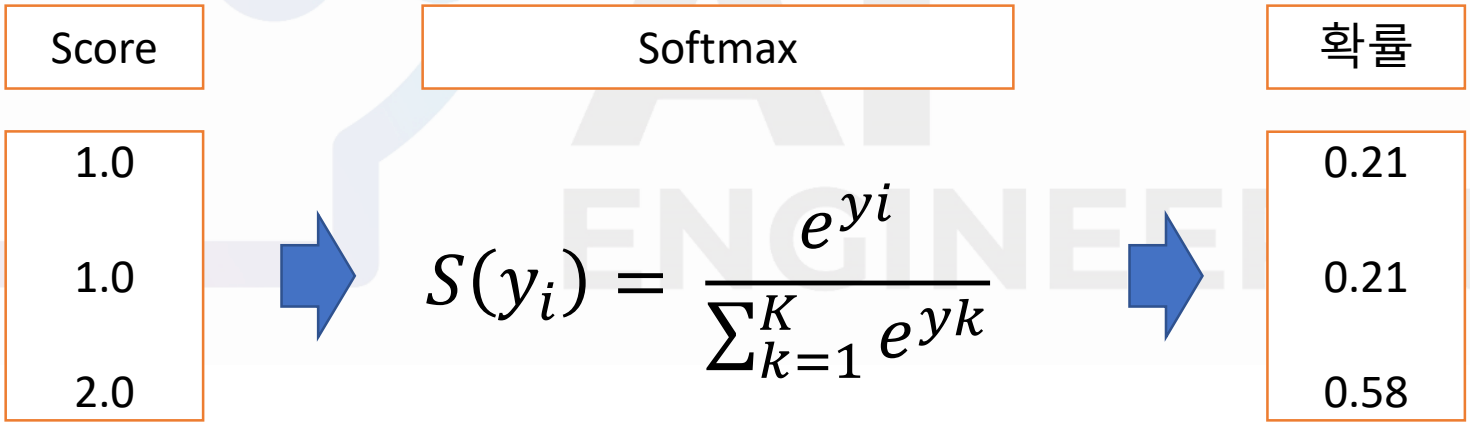
- Target 은 1로 나머지는 0으로

- MNIST 예
  - 1 → [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  - 5 → [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
  - 9 → [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

- 우리 문제
  - 0 → [1, 0]
  - 1 → [0, 1]

Softmax

- 출력값들의 합이 1이 되도록 정규화
- 확률과 동일한 개념
- multi class 분류 문제에서 많이 사용



Softmax

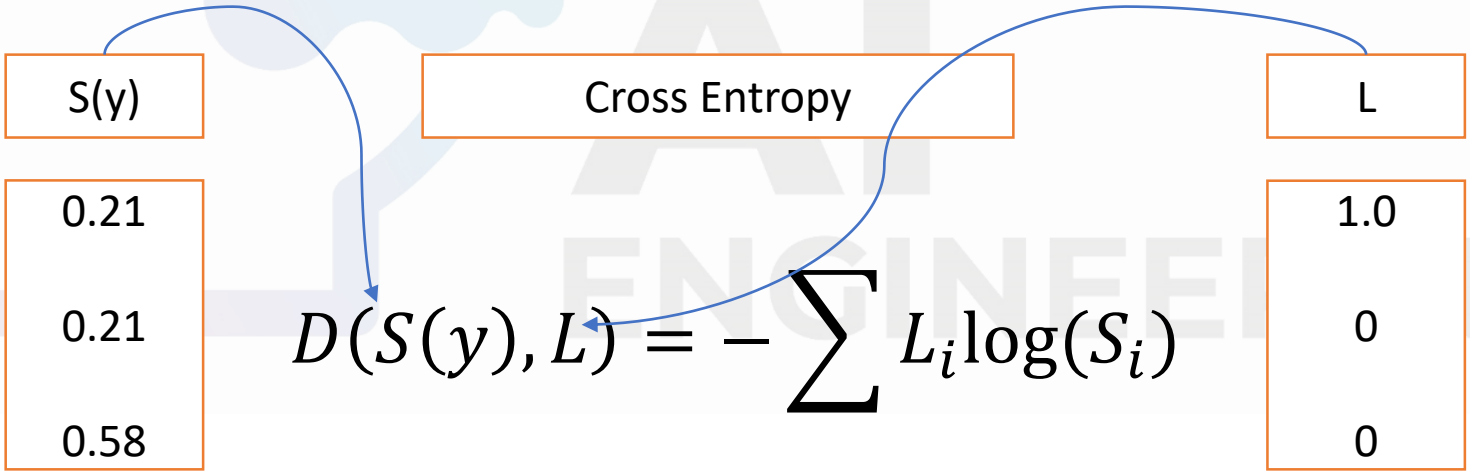
```
def softmax(x):
    e_x = np.exp(x)
    sum_e_x = np.sum(e_x)
    y = e_x / sum_e_x

    return y
```

```
if __name__ == "__main__":
    a = np.array([1.0, 1.0, 2.0])
    sm_a = softmax(a)
    print(sm_a)
```

```
----------------------------------------
[0.21194156 0.21194156 0.57611688]
```

Cross-entropy

- 분류 문제에서 loss를 계산할 때 주로 사용
  - Softmax 와 궁합이 좋다
  - S(y) 가 모델의 출력값, L 이 정답이라고 할 때



학습 해보기

```
if __name__ == "__main__":
    perceptron_model = make_model()

    x = [[0, 0], [0, 1], [1, 0], [1, 1]]
    y = [[1 , 0], [1 , 0], [1 , 0], [0, 1] ]

    perceptron_model.fit(x, y, epochs=100)

    result = perceptron_model.predict(x)
    print(result)
```

```
Epoch 100/100
1/1 [==============================] - 0s 0s/step - loss: 0.6457 - accuracy:
1.0000
[[0.54956883 0.4504311 ]
 [0.53250307 0.4674969 ]
 [0.51435405 0.48564604]
 [0.49717474 0.50282526]]
```

Summary

◆ 신경망에 대하여 알아보았다.

➢ 신경망?

- ✓ 인간의 신경을 모방
- ✓ 입력에 대한 가중치 합과 활성화 함수로 뉴런을 표현
- ✓ 뉴런들을 여러 개 연결하여 신경망을 구성

➢ 활성화 함수

- ✓ Step 함수 → 미분 가능하지 않다.
- ✓ Sigmoid 함수 → 고전적으로 많이 사용
- ✓ ReLu, Leaky ReLu 등은 나중에…

◆ Keras

➢ 신경망, 딥러닝 구현을 위한 라이브러리

➢ 잘 쓰면 된다!