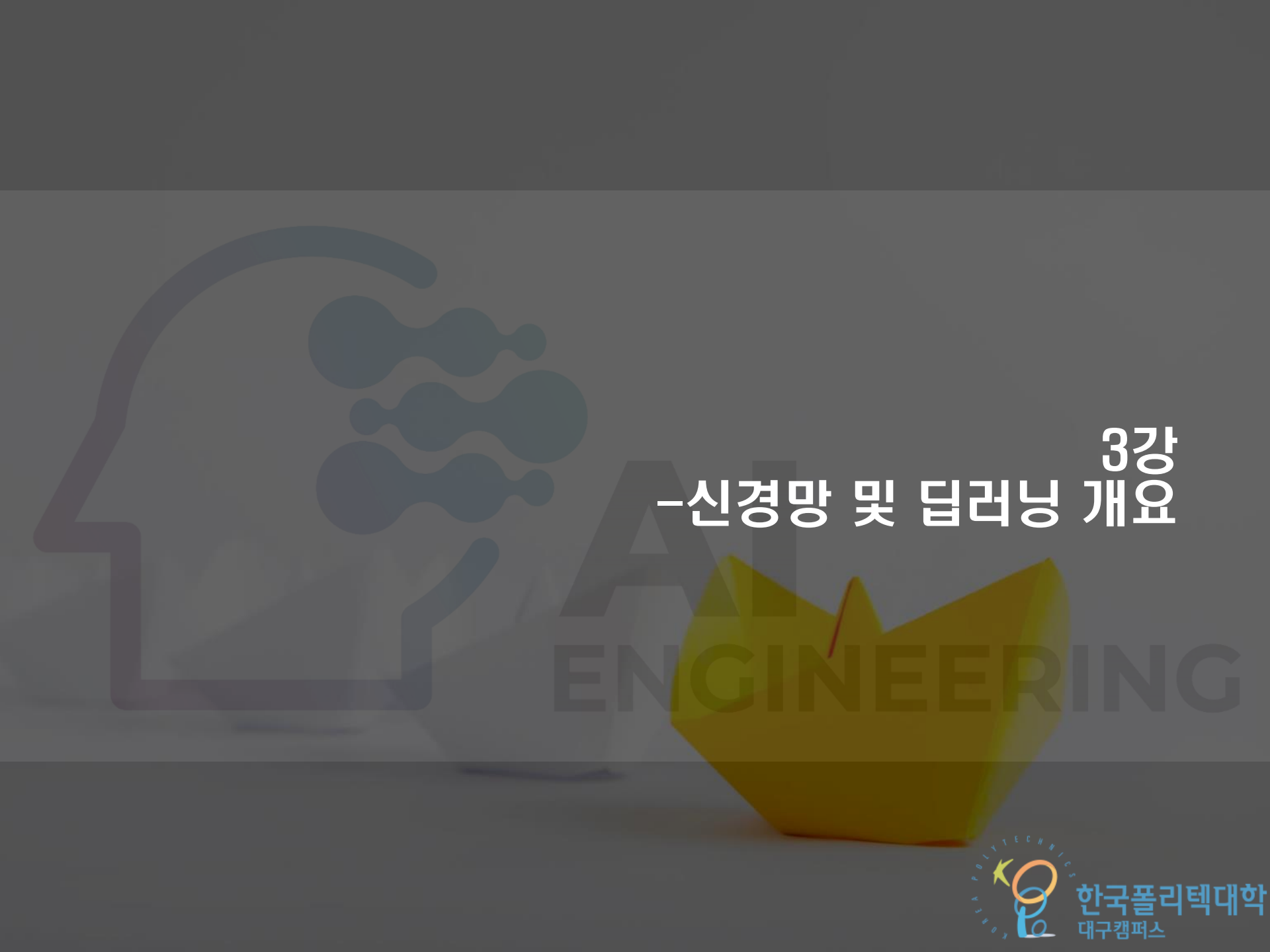
AI+x 인재양성 확대를 위한 교수자 연수

한국폴리텍대학 대구캠퍼스
AI엔지니어링학과 강현우

3강
-신경망 및 딥러닝 개요

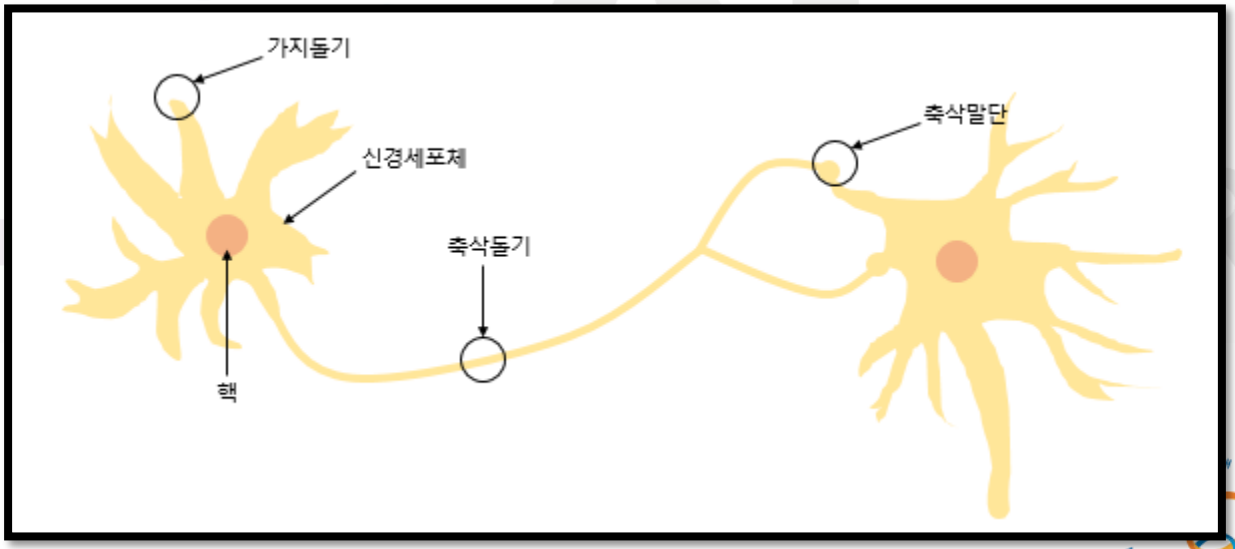


한국폴리텍대학
대구캠퍼스

Overview

◆ 신경망

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

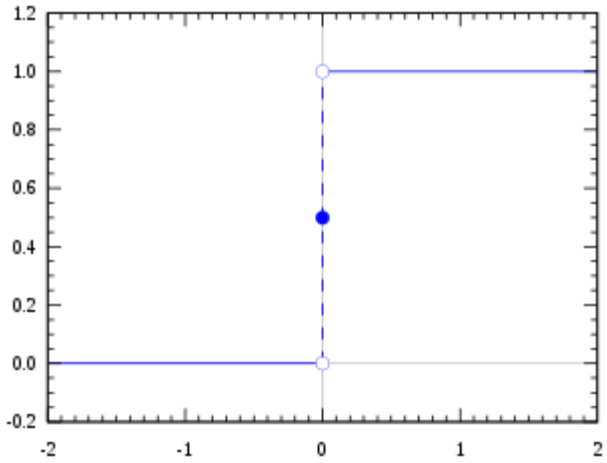


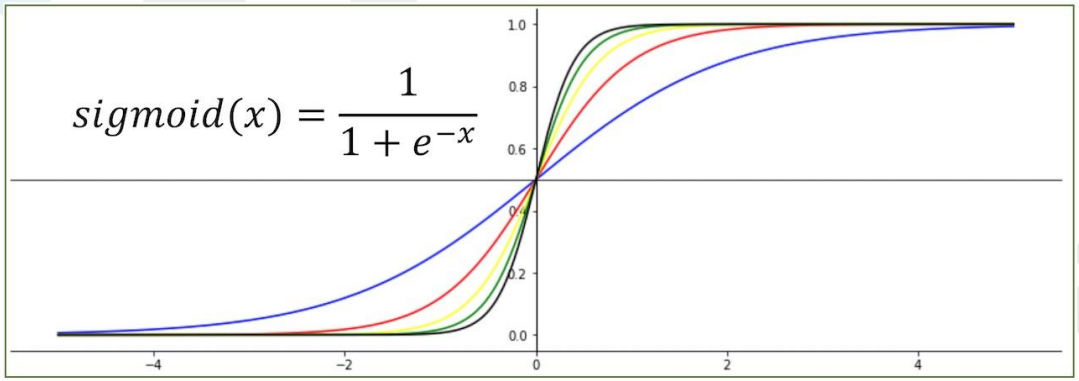
출처: https://wikidocs.net/24958

Overview

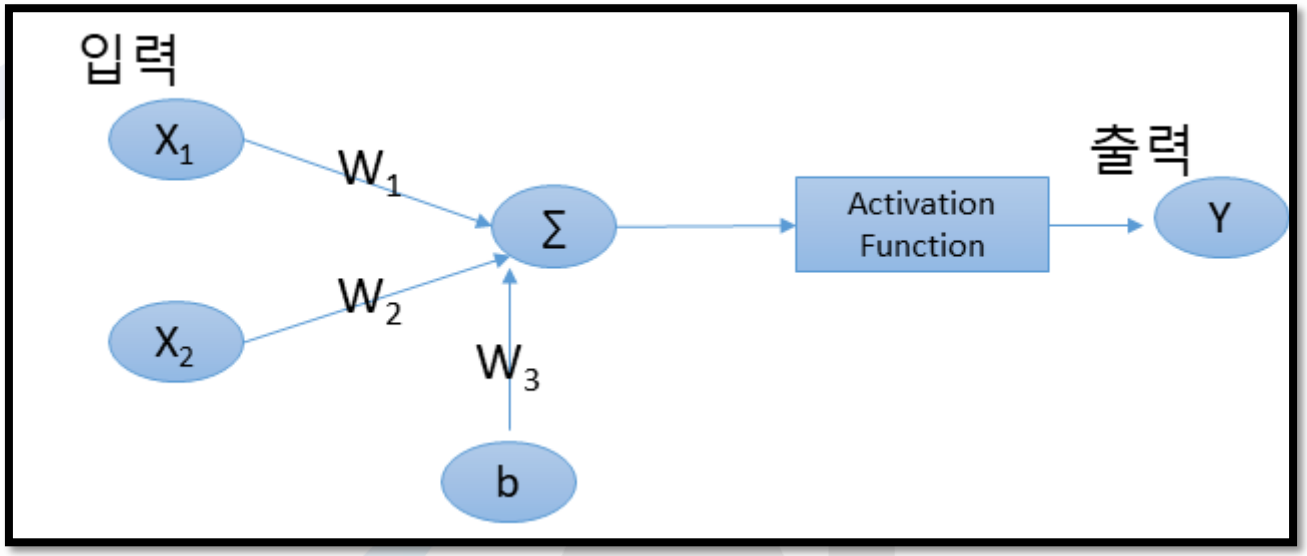
◆ Activation Function (활성화 함수)

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





Perceptron 모델



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 저장해둔게 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

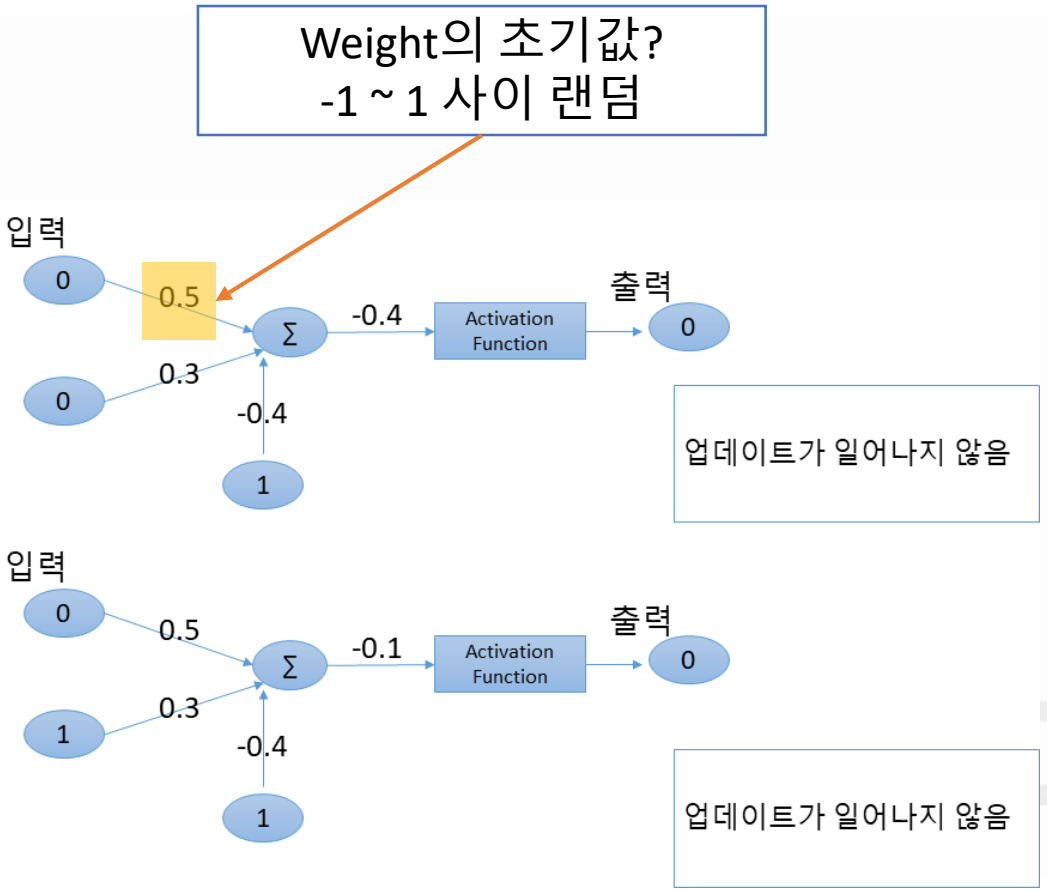
학습데이터

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)

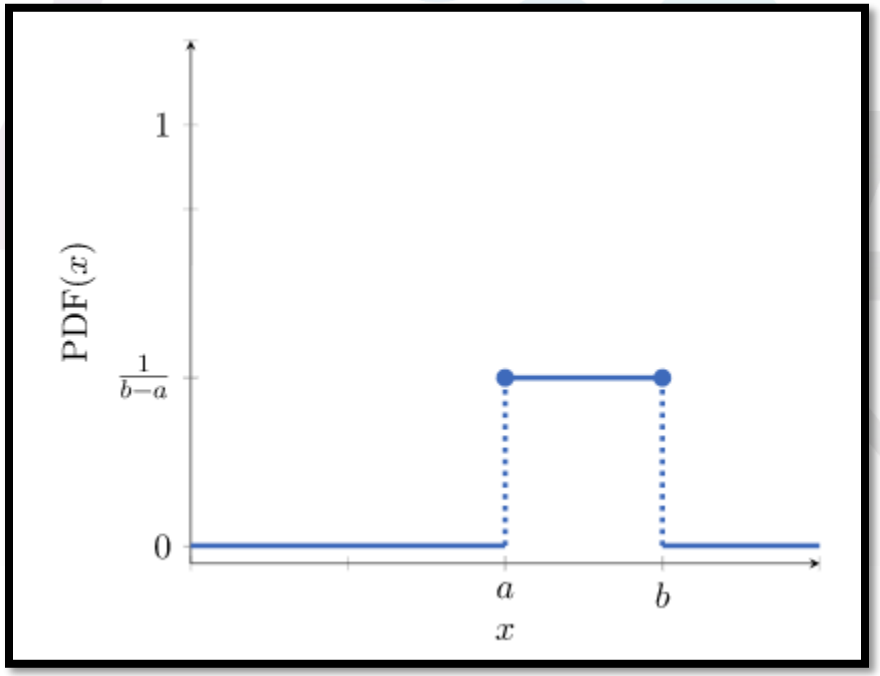


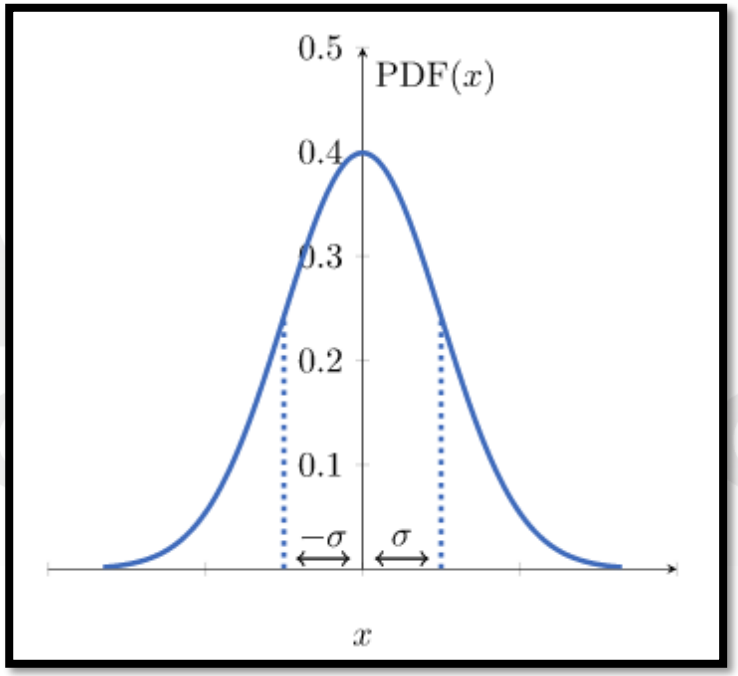
1라운드

Random

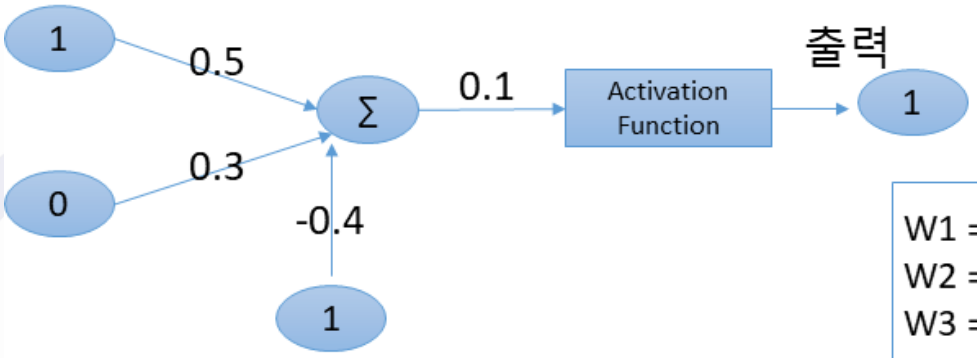
◆ Normal Distribution / Uniform Distribution

➢ 확률 밀도 함수 (Probability Density Function)

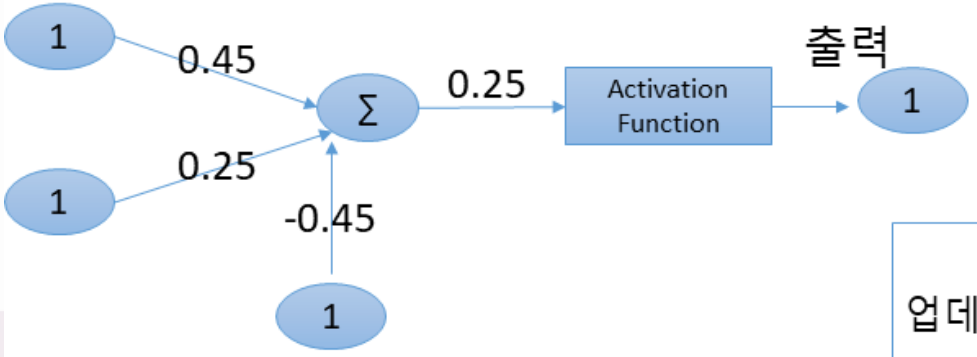




And 게이트 학습



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

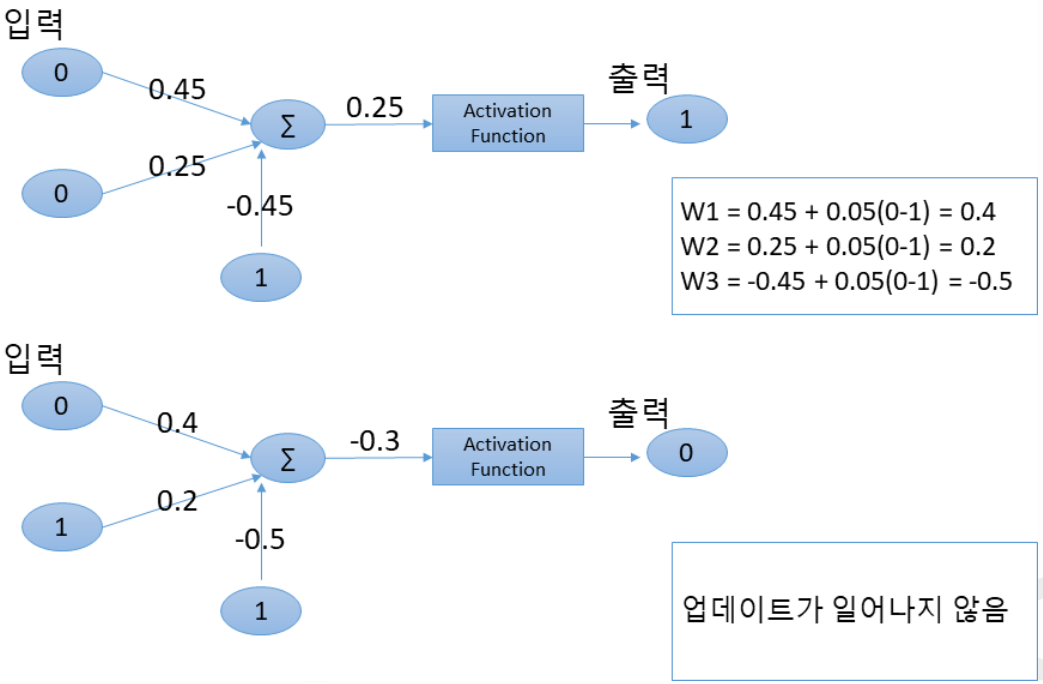
1라운드

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

And 게이트 학습

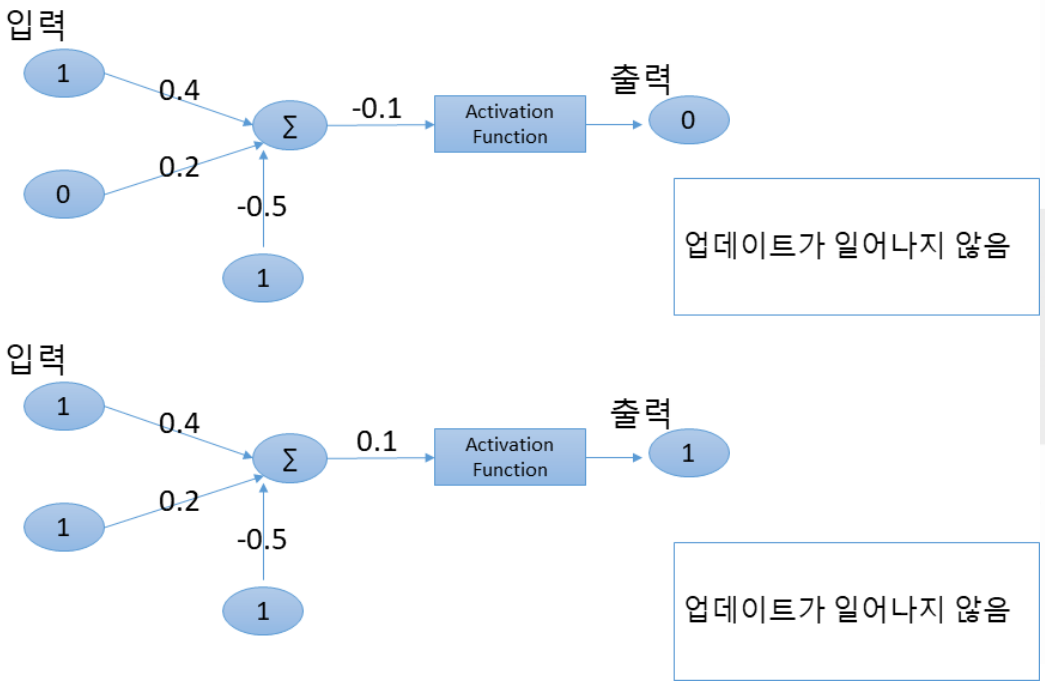
x1	x2	d	y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

1라운드 종료 후 출력값
정답률 = 75%



2라운드

And 게이트 학습

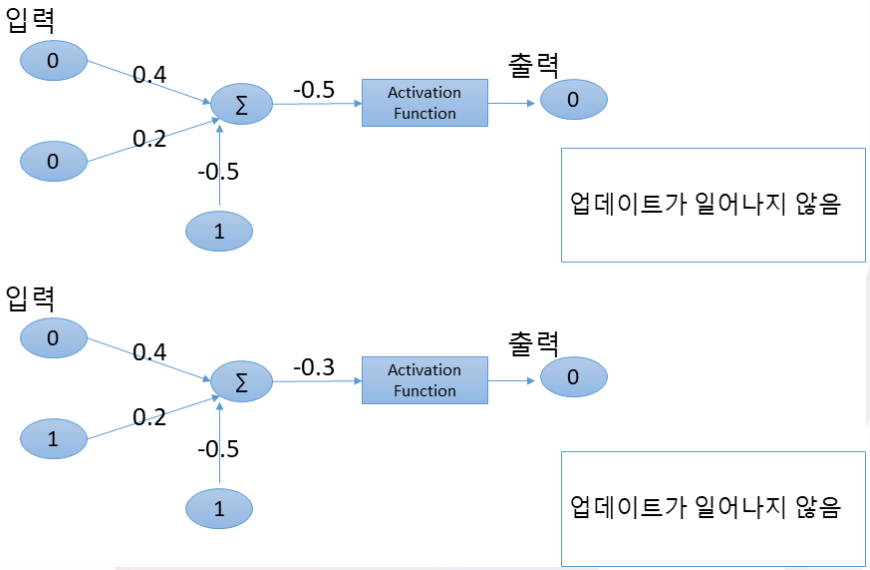


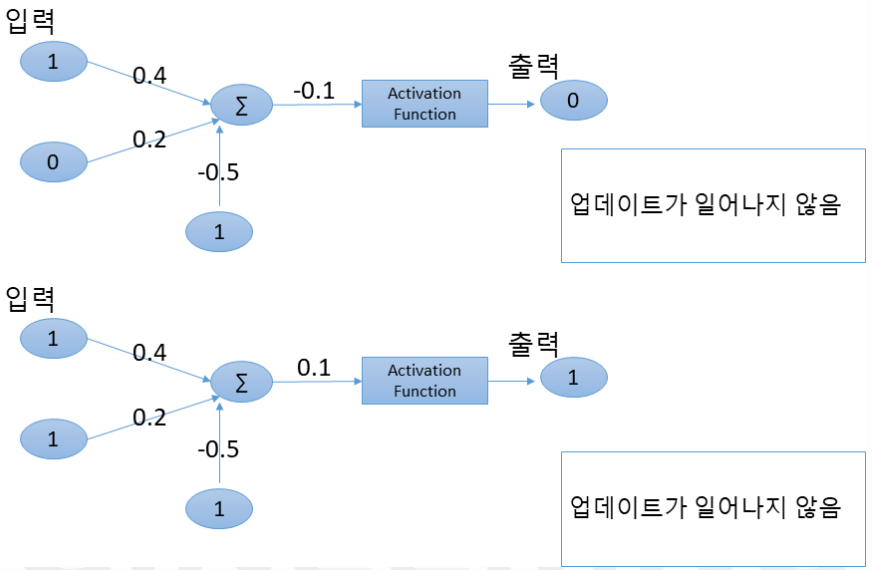
2라운드

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

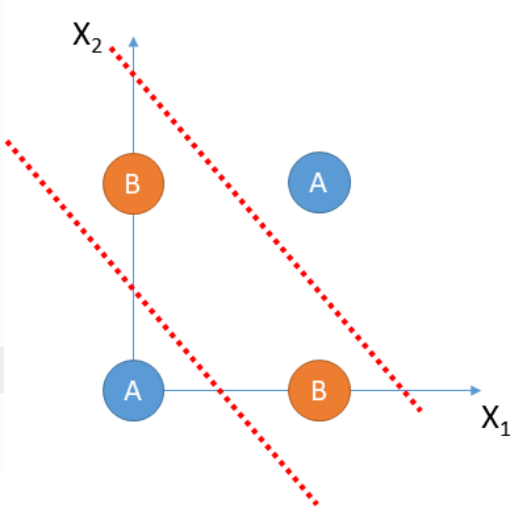
한국폴리텍대학
대구캠퍼스

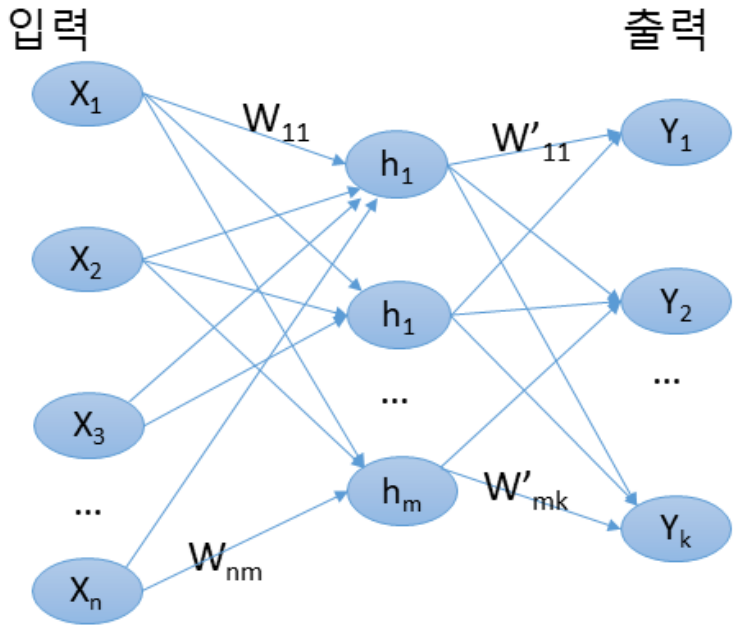
학습된 모델 구현

```
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





딥러닝 환경 구축시 고려 사항

- IDE
  - Jupyter Lab / notebook
  - Visual Studio / VS code
  - PyCharm
  - Eclipse

딥러닝 프레임워크

◆ TensorFlow (텐서 플로우)

- ➢구글, 2015년, 가장 널리 사용
- ➢개방성, 다양한 언어 지원, 많은 사용자
- ➢Python, C/C++ 등 지원
- ➢초기에 리눅스와 맥OS 만 지원, 현재 윈도우도 지원
- ➢https://www.tensorflow.org/?hl=ko

◆ Keras (케라스)

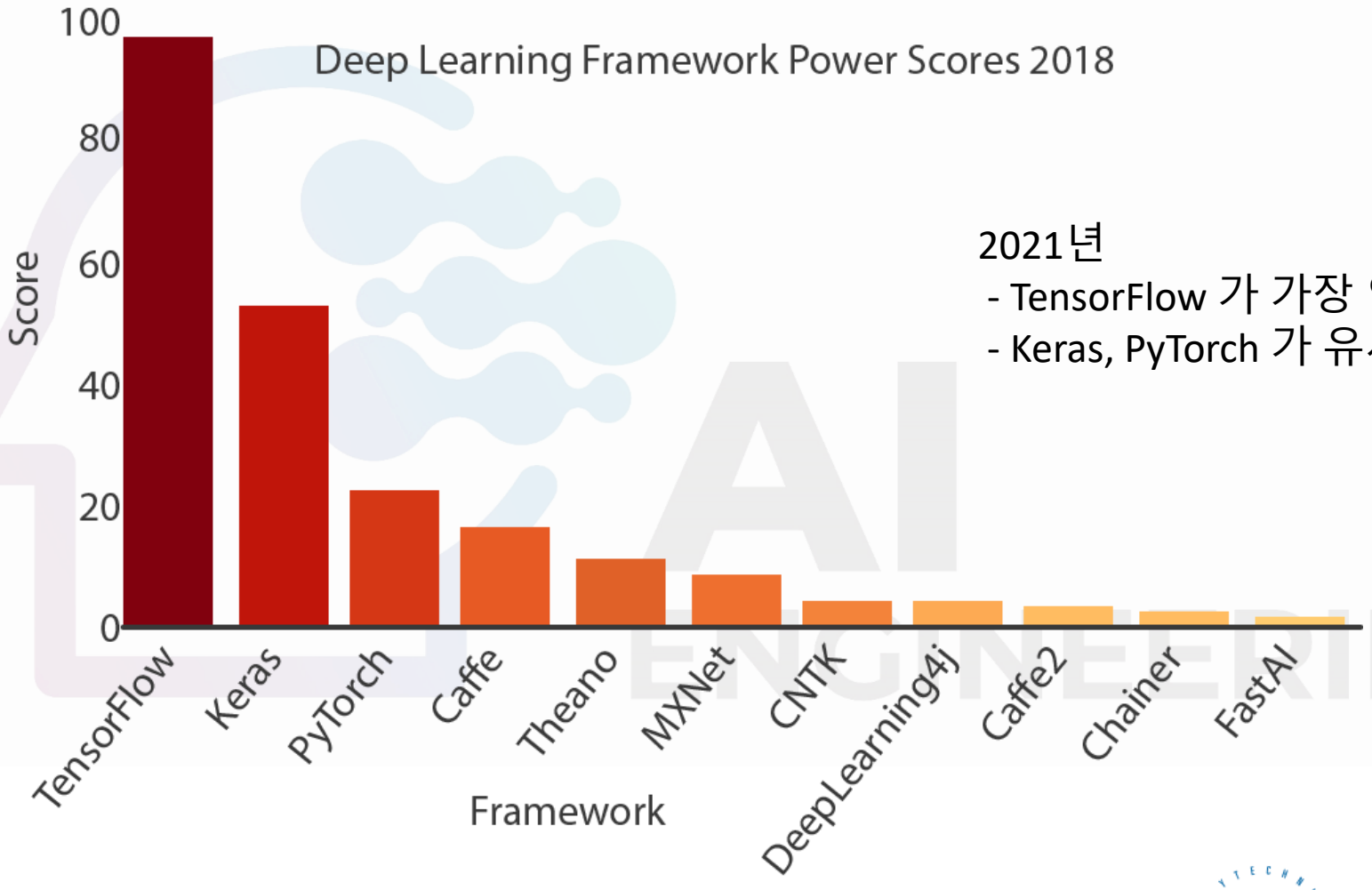
- ➢구글, 2015년, (2017년부터 TensorFlow에서 지원)
- ➢래퍼 라이브러리(MXNet, DL4J, TensorFlow, CNTK, Theano), Python 기반
- ➢사용성 우수(모듈화, 최소주의, 확장성)
- ➢https://keras.io/, https://keras.io/ko/

딥러닝 프레임워크

- **PyTorch (파이토치)**
  - 페이스북, 2016년
  - 사용성 우수(코드 구현 장벽 낮음, 사용법 쉬움)
  - 텐서 플로우에 비하여 사용자층이 작음
  - https://pytorch.org/

- **Caffe (카페)**
  - 버클리 대학, 2013년
  - C/C++ 기반, Python 인터페이스
  - 컴퓨터비전/CNN/음성 특화, 범용성 낮음
  - 모델 공유 네트워크 (커뮤니티, Model Zoo)
  - https://caffe.berkeleyvision.org/

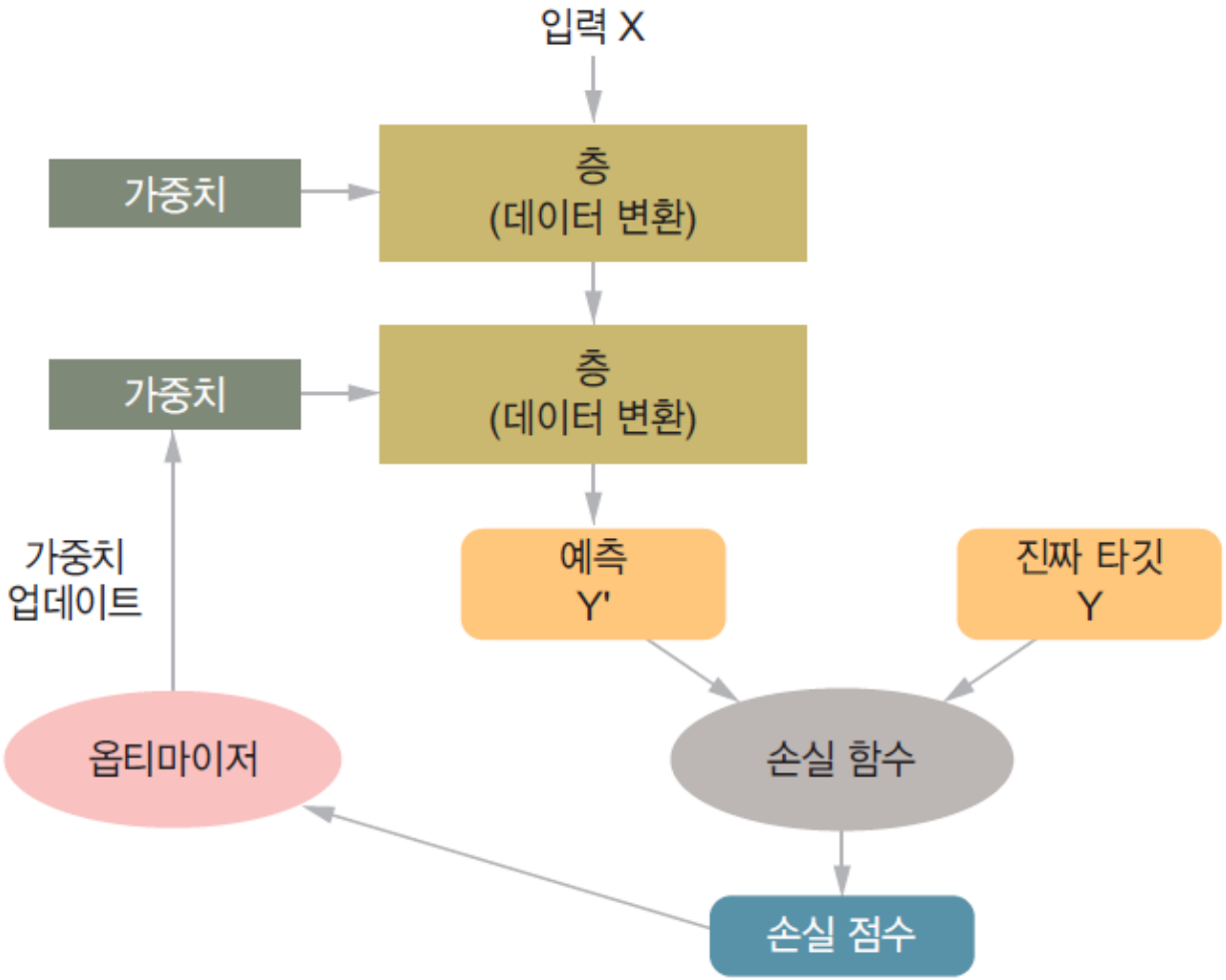
딥러닝 프레임워크



2021년
- TensorFlow 가 가장 인기
- Keras, PyTorch 가 유사한 인기

[출처] https://towardsdatascience.com/top-5-deep-learning-frameworks-to-watch-in-2021-and-why-tensorflow-98d8d6667351

학습 과정



[출처] 모두의 딥러닝

출력 – One Hot Encoding

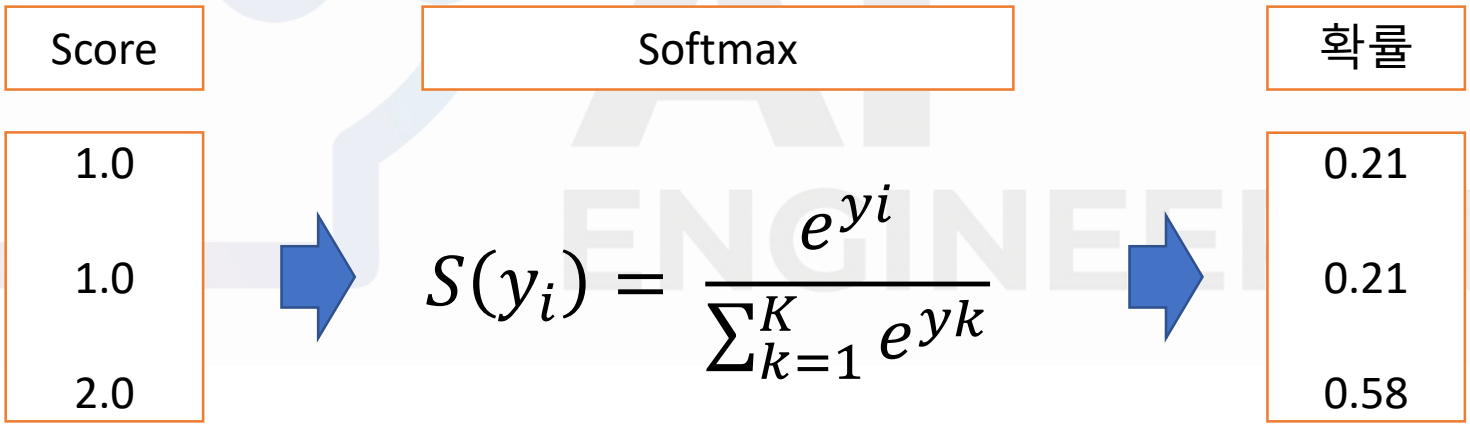
- Target 은 1로 나머지는 0으로
  - 분류(Classification) 문제에서 주로 사용

- 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



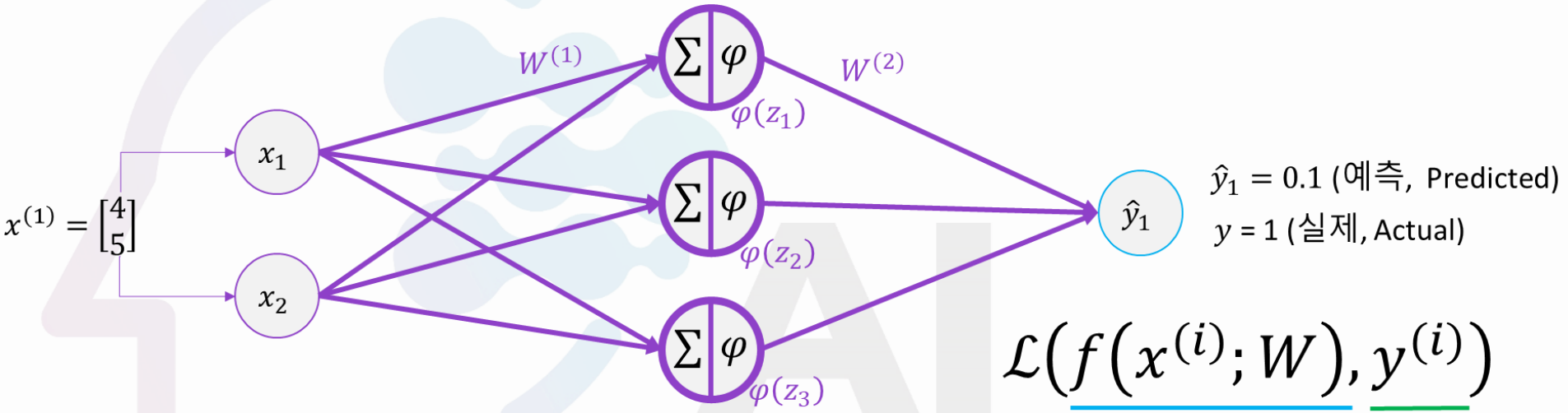
Softmax

```
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)
```

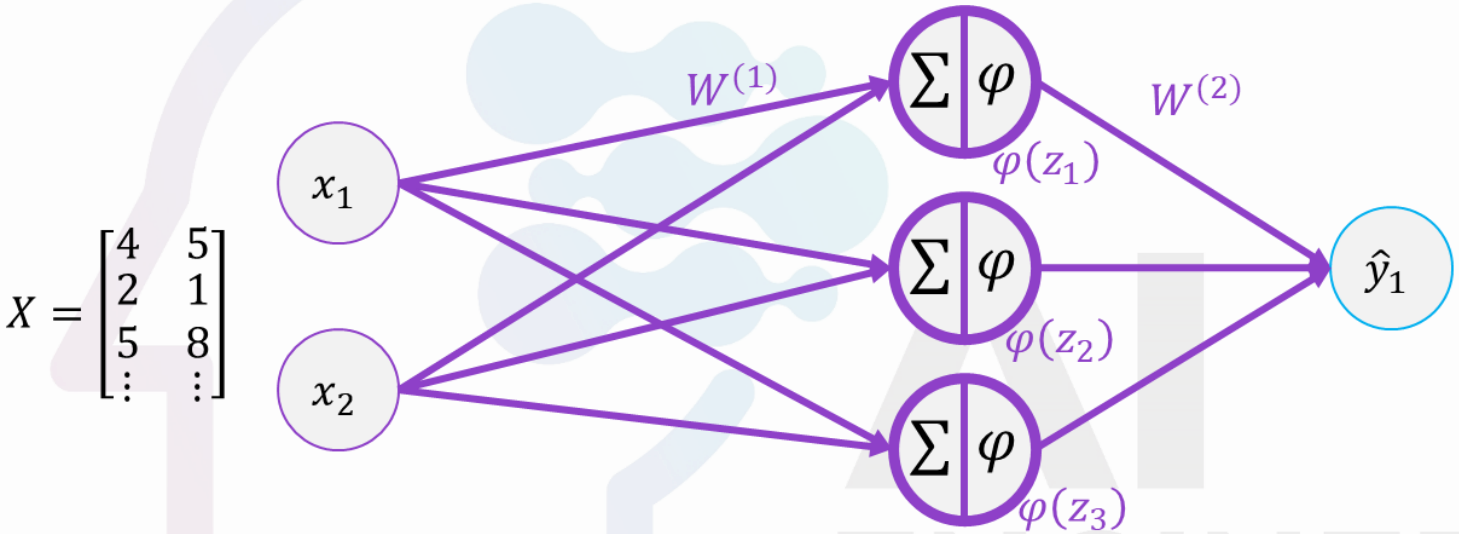
```
----------------------------------------
[0.21194156 0.21194156 0.57611688]
```

Loss (손실)



Empirical Loss

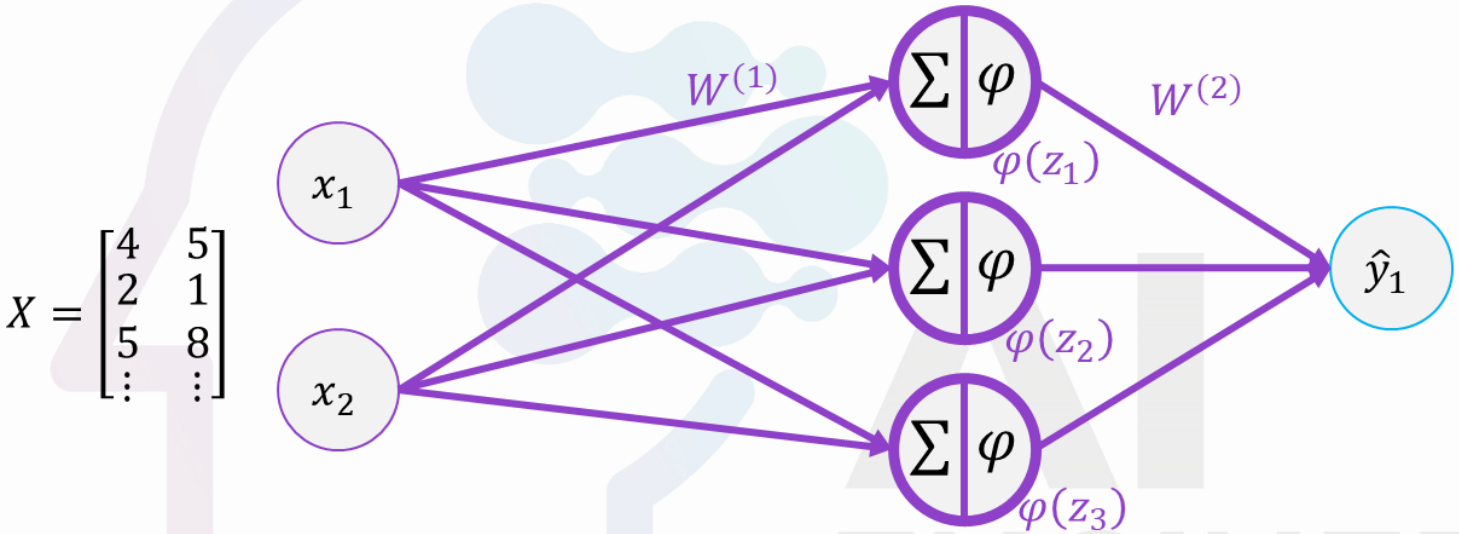
◆ 전체 데이터셋에 대한 총 손실을 측정



$$J(W) = \frac{1}{N}\sum_{i=n}^{N} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

Mean Squared Error Loss

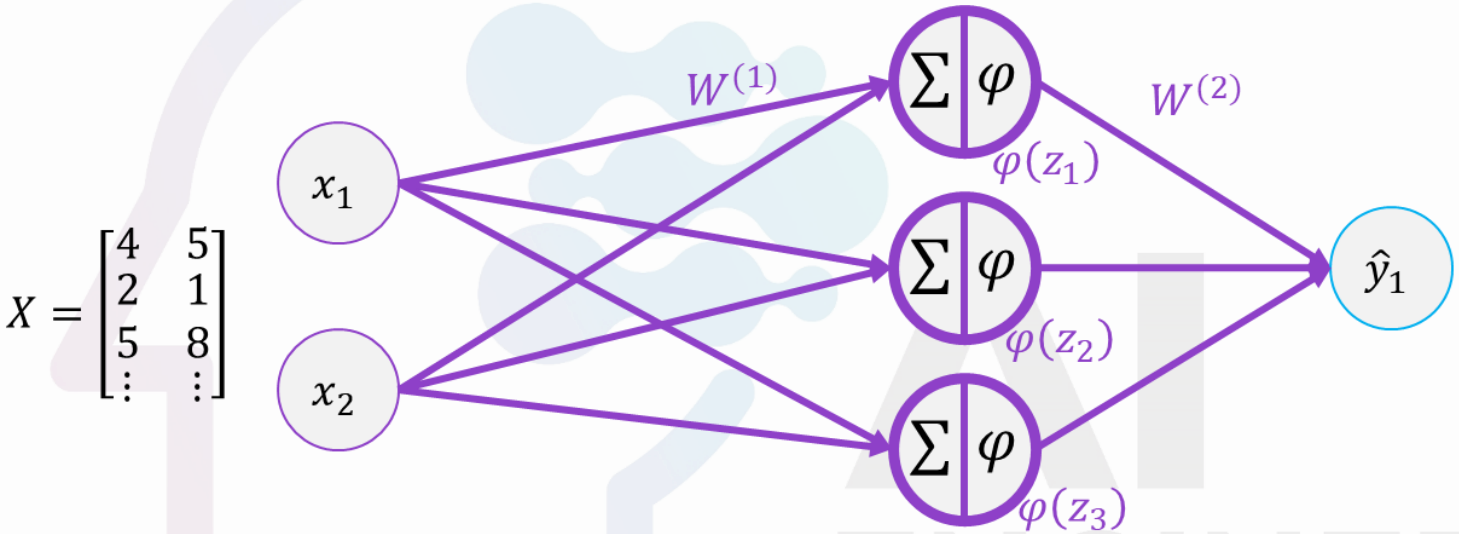
◆ Regression 문제에서 많이 사용



$$J(W) = -\frac{1}{N}\sum_{i=n}^{N}\left(y^{(i)} - f\left(x^{(i)}; W\right)\right)^2$$

Binary Cross Entropy Loss

◆ 출력이 확률(0~1) 형태일 때 사용



$f(x)$

$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix}$ **(X)** **(X)** **(O)**

$f(x)$

$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$

$$J(W) = -\frac{1}{N}\sum_{i=n}^{N} y^{(i)} log\left(f\left(x^{(i)}; W\right)\right) + \left(1 - y^{(i)}\right) log\left(1 - f\left(x^{(i)}; W\right)\right)$$

신경망의 학습

◆ Loss Optimization

➢ 손실을 최소화 하는 가중치를 찾는 것

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

$W = \{W^{(1)}.W^{(2)}, \cdots\}$

Back Propagation

◆ 오차 역전파

- ➢ MLP 에서 최적화 과정
- ➢ 모델의 예측값 과 원하는 값 사이의 오차를 계산
- ➢ 경사 하강법을 이용해 오차가 작아지는 방향으로 웨이트를 업데이트

◆ 오차가 작아지는 것?

- ➢ 미분 값 (기울기)가 0이 되는 방향으로 나아간다

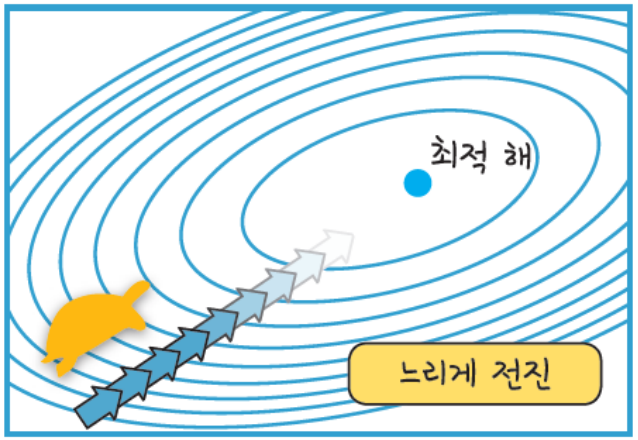
새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값!

$$W(t+1) = Wt - \frac{\partial 오차}{\partial W}$$

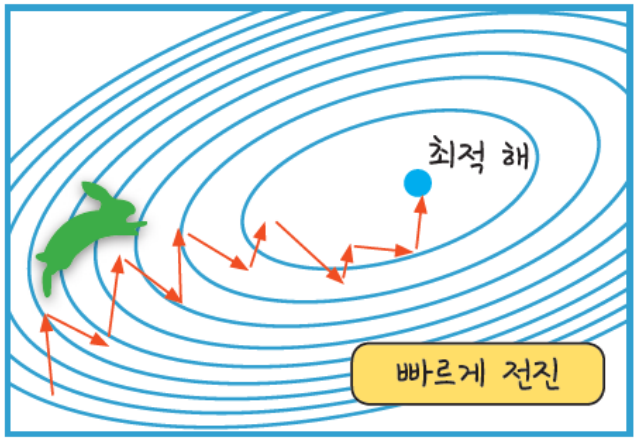
Loss Optimization

◆ Gradient Desent

- ➢ 1차 근사값 발견용 최적화 알고리즘
- ➢ 함수의 기울기를 구하고 기울기의 절대값이 낮은 쪽으로 계속 이동시켜 극값에 이를때 까지 반복



경사 하강법



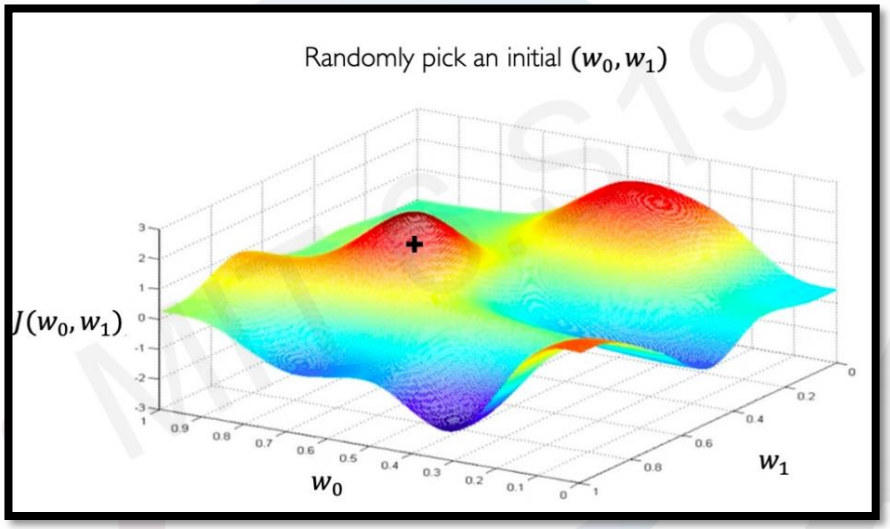
확률적 경사 하강법

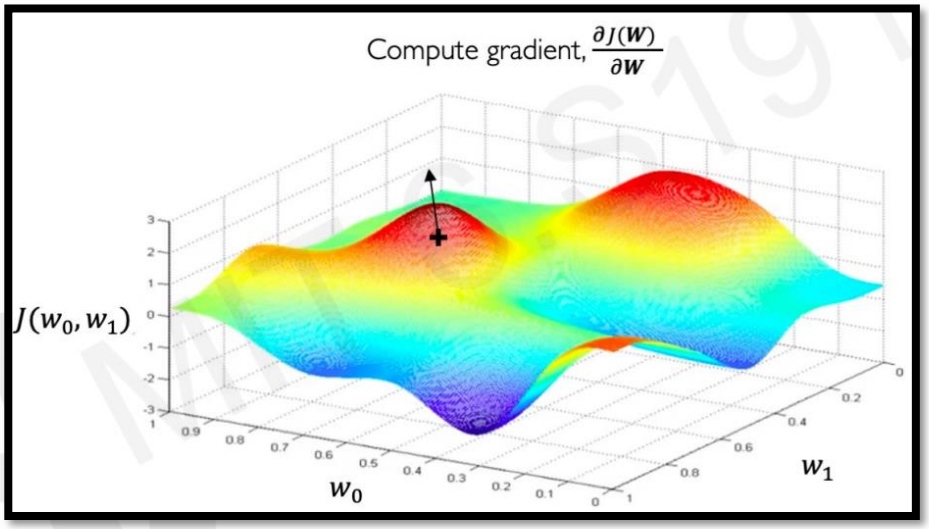
$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

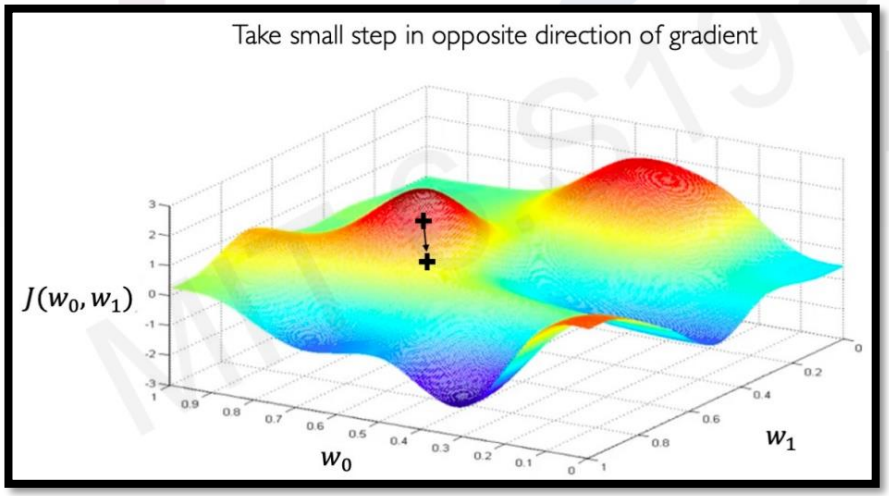
[출처] 모두의 딥러닝

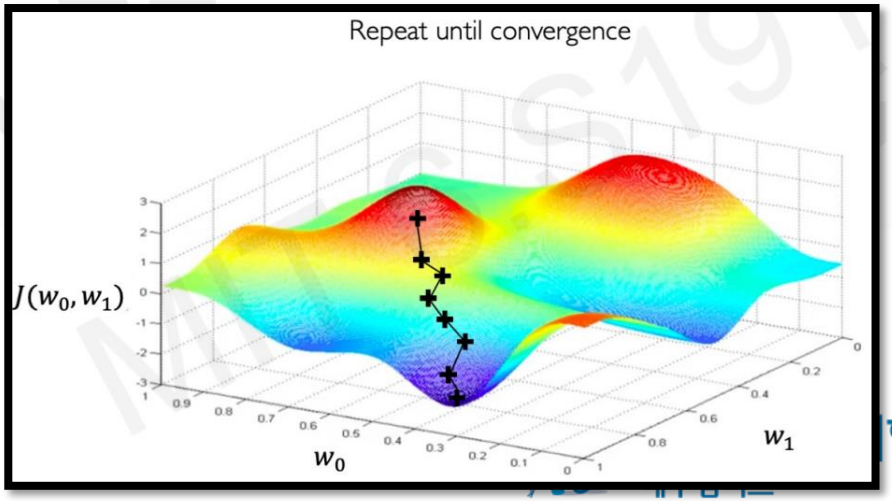
Loss Optimization

◆ 가중치에 따른 손실 분포









Back Propagagtion

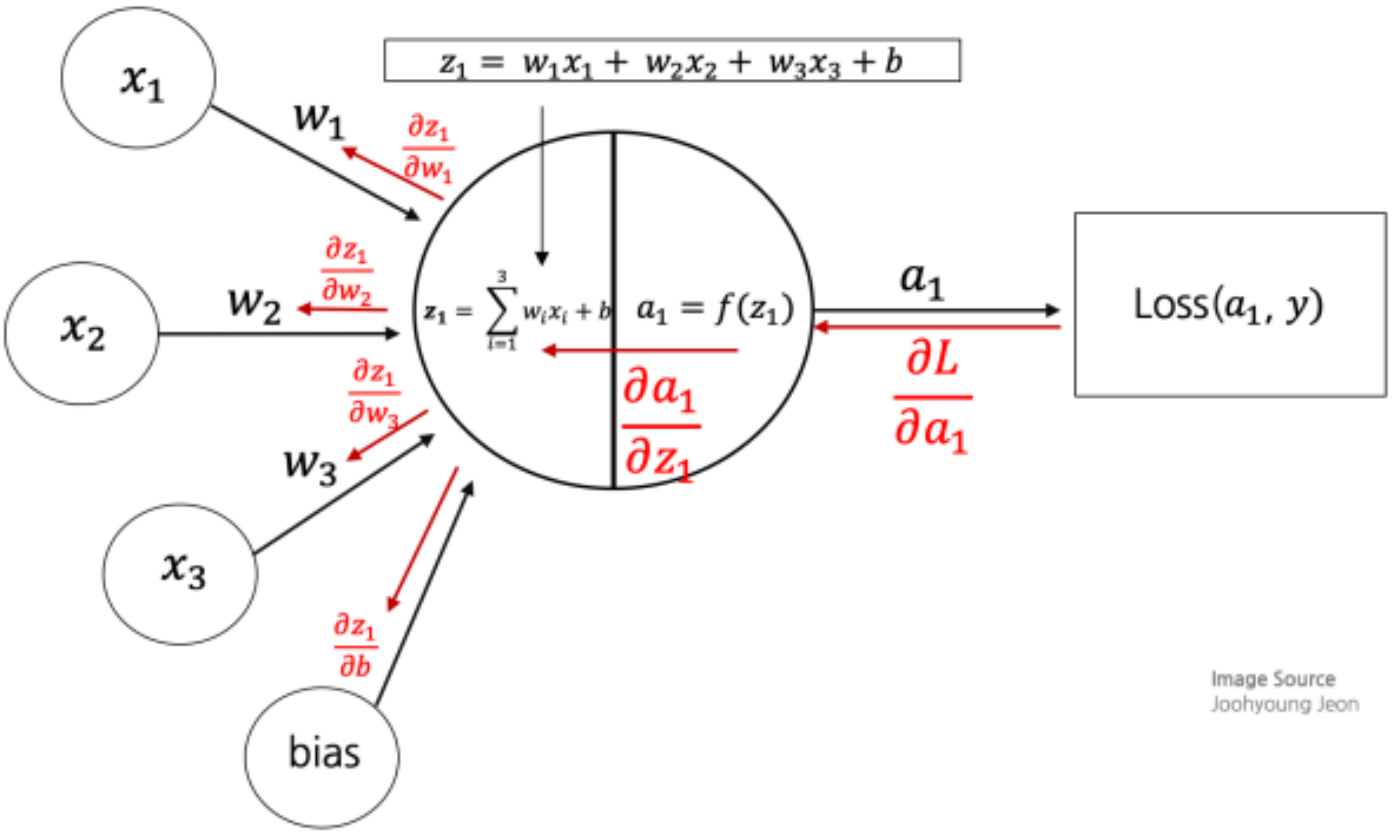


Image Source
Joohyoung Jeon

[출처] https://ds-academy.net/chapter-1-introduction-to-dl/

Optimization

고급 경사 하강법	개요	효과	케라스 사용법
확률적 경사 하강법 (SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	keras.optimizers.SGD(lr = 0.1) 케라스 최적화 함수를 이용합니다.
모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가합니다.
네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True) 네스테로프 옵션을 추가합니다.
아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6) 아다그라드 함수를 사용합니다. ※ 참고: 여기서 epsilon, rho, decay 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 lr, 즉 learning rate(학습률) 값만 적절히 조절하면 됩니다.

[출처] 모두의 딥러닝

Optimization

고급 경사 하강법	개요	효과	케라스 사용법
알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e - 08, decay = 0.0) 알엠에스프롭 함수를 사용합니다.
아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0) 아담 함수를 사용합니다.

[출처] 모두의 딥러닝

잘 모르겠으면,
그냥 Adam 을 사용한다!

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 해보기

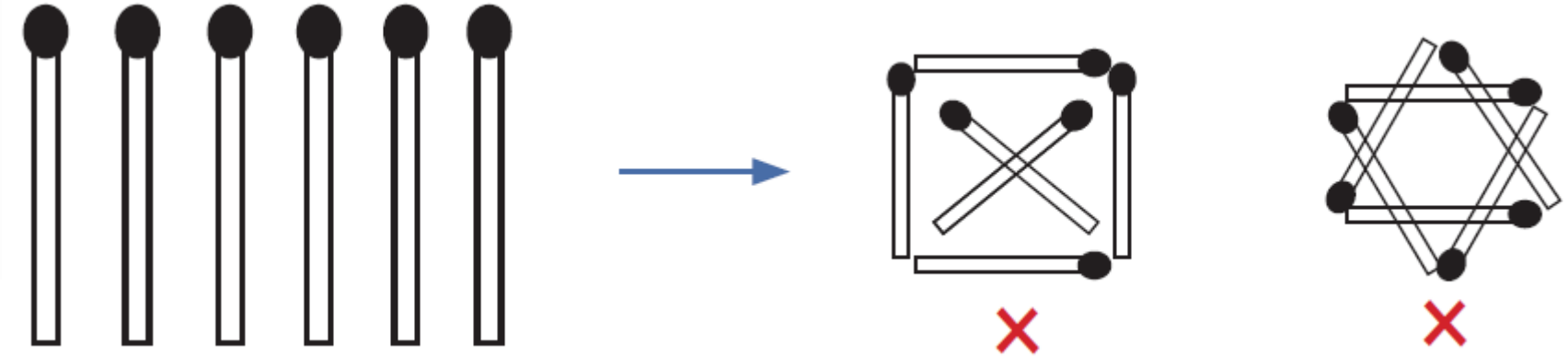
```
if __name__ == "__main__":
    perceptron_model = make_model()

    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]]
```

Multi-layer Perceptron

- **XOR 문제**
  - 단층 신경망으로는 풀 수 없다
  - 새로운 접근이 필요

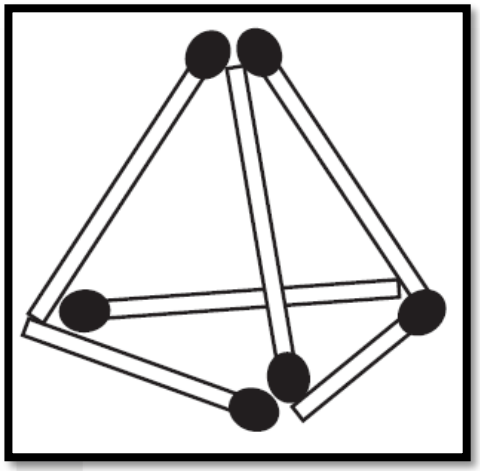
- **성냥개비 6개로 정삼각형 3개를 만들어보라**



[출처] 모두의 딥러닝

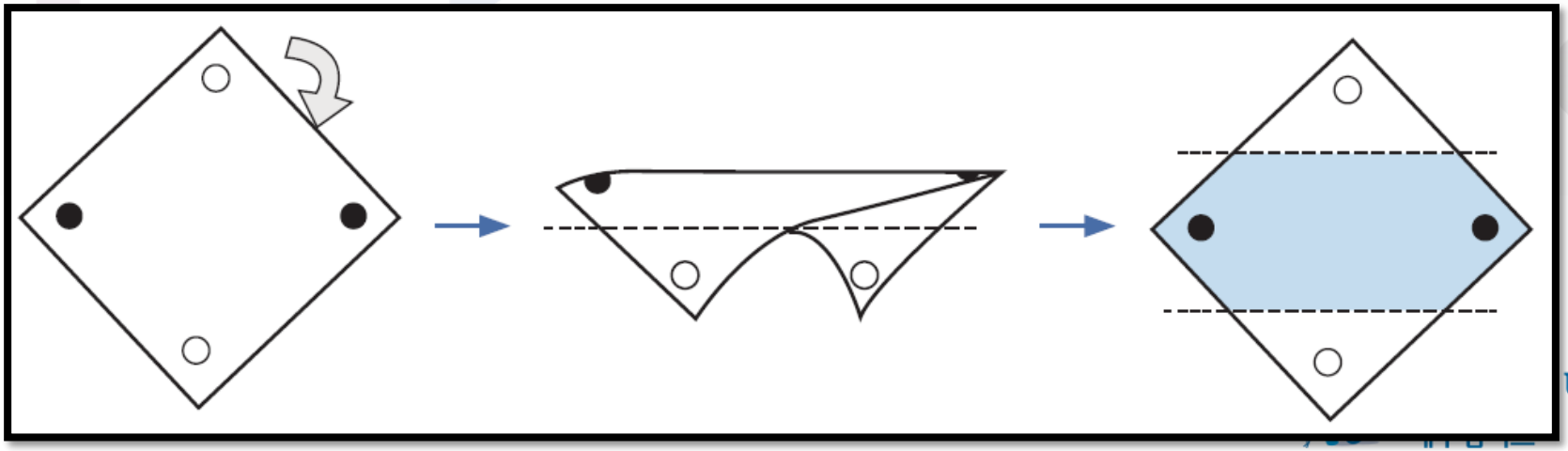
Multi Layer Perceptron

- 2차원 평면 노노
  - 3차원에서는 쉽게 해결



[출처] 모두의 딥러닝

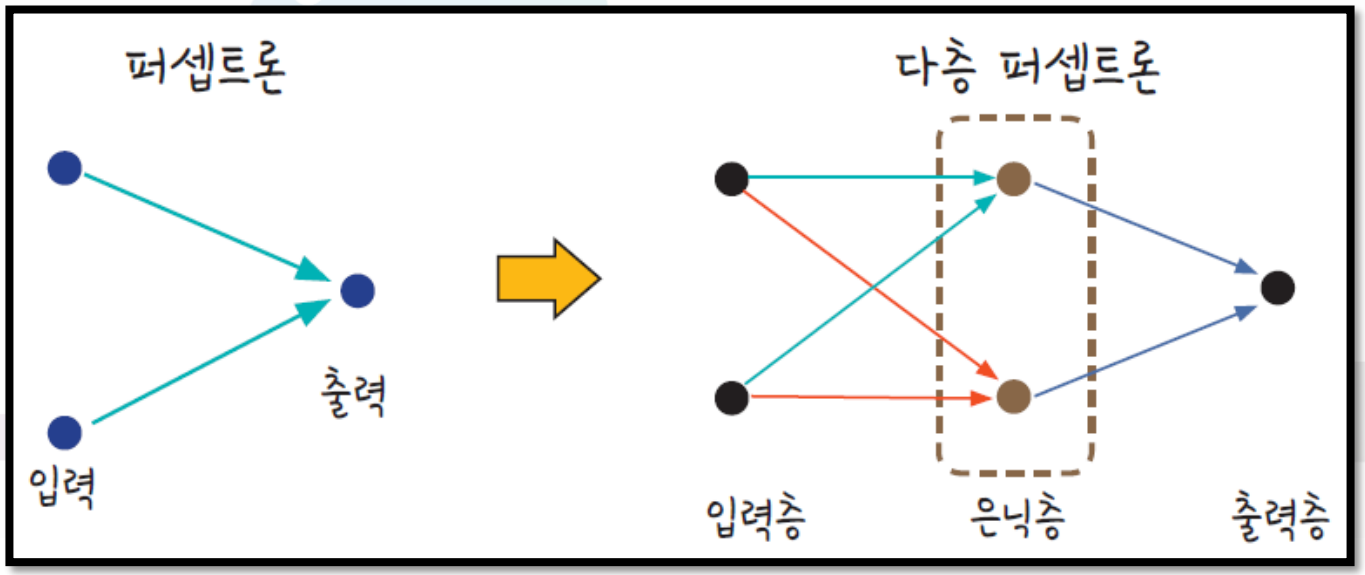
- XOR 문제의 해결은 평면을 휘어 주는 것



Multi Layer Perceptron

◆ 좌표 평면 자체에 변화

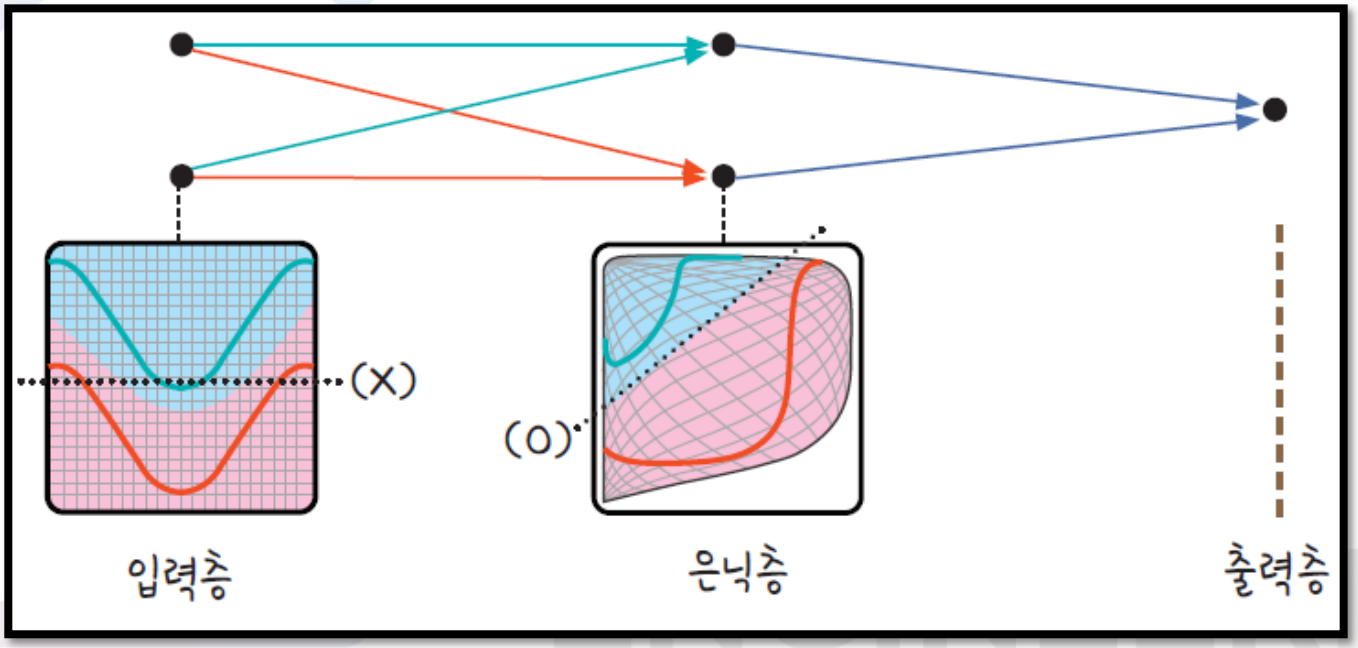
- ➢ 2개의 perceptron을 한 번에 계산할 수 있어야 함
- ➢ 은닉층 (Hidden layer)를 추가!



[출처] 모두의 딥러닝

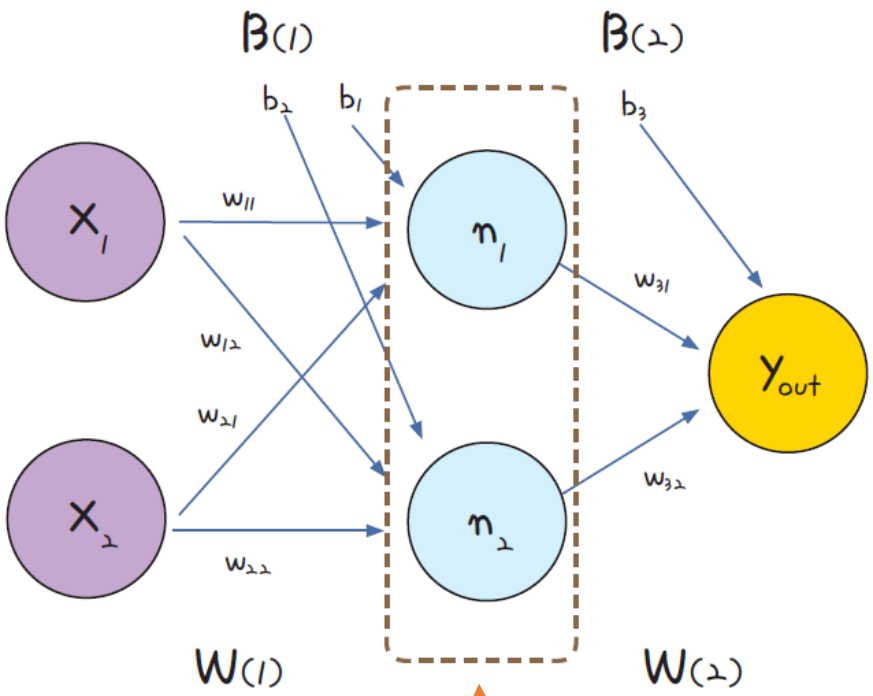
Multi Layer Perceptron

◆ 은닉층이 좌표 평면을 왜곡 시키는 결과



http://colah.github.io/posts/2015-09-NN-Types-FP/

MLP 구조



[출처] 모두의 딥러닝

$$n_1 = w_{11}x_1 + w_{21}x_2 + b_1$$

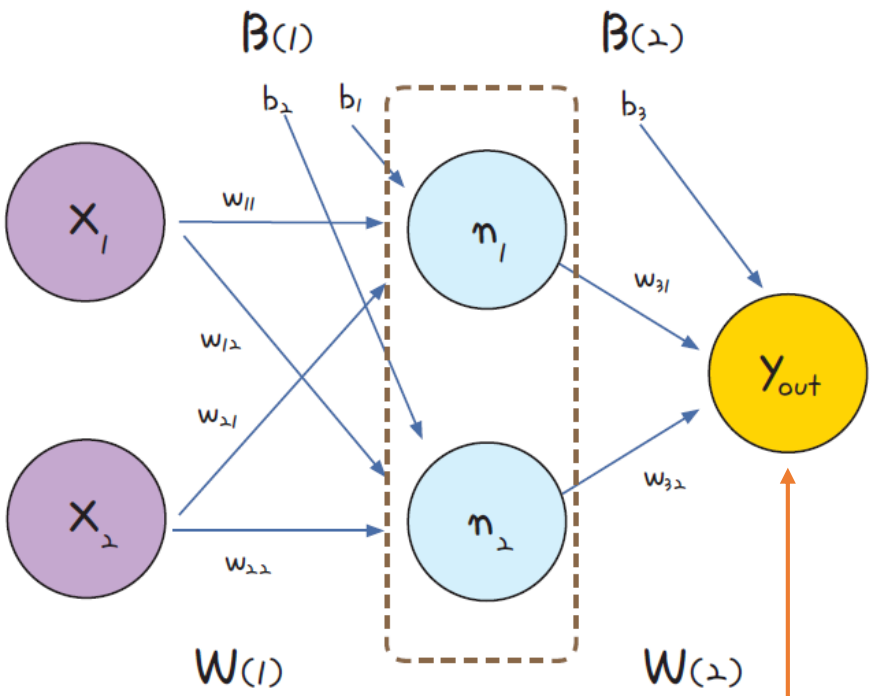
$$n_2 = w_{12}x_1 + w_{22}x_2 + b_2$$

$$W1 = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$B1 = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$N1 = W_1X + B1$$

MLP 구조



$$y_{out} = w_{31}n_1 + w_{32}n_2 + b_3$$

$$W2 = \begin{bmatrix} w_{31} \\ w_{32} \end{bmatrix} \quad N = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$$

$$B2 = [b_3]$$

$$y_{out} = W_2{}^T N + B2$$

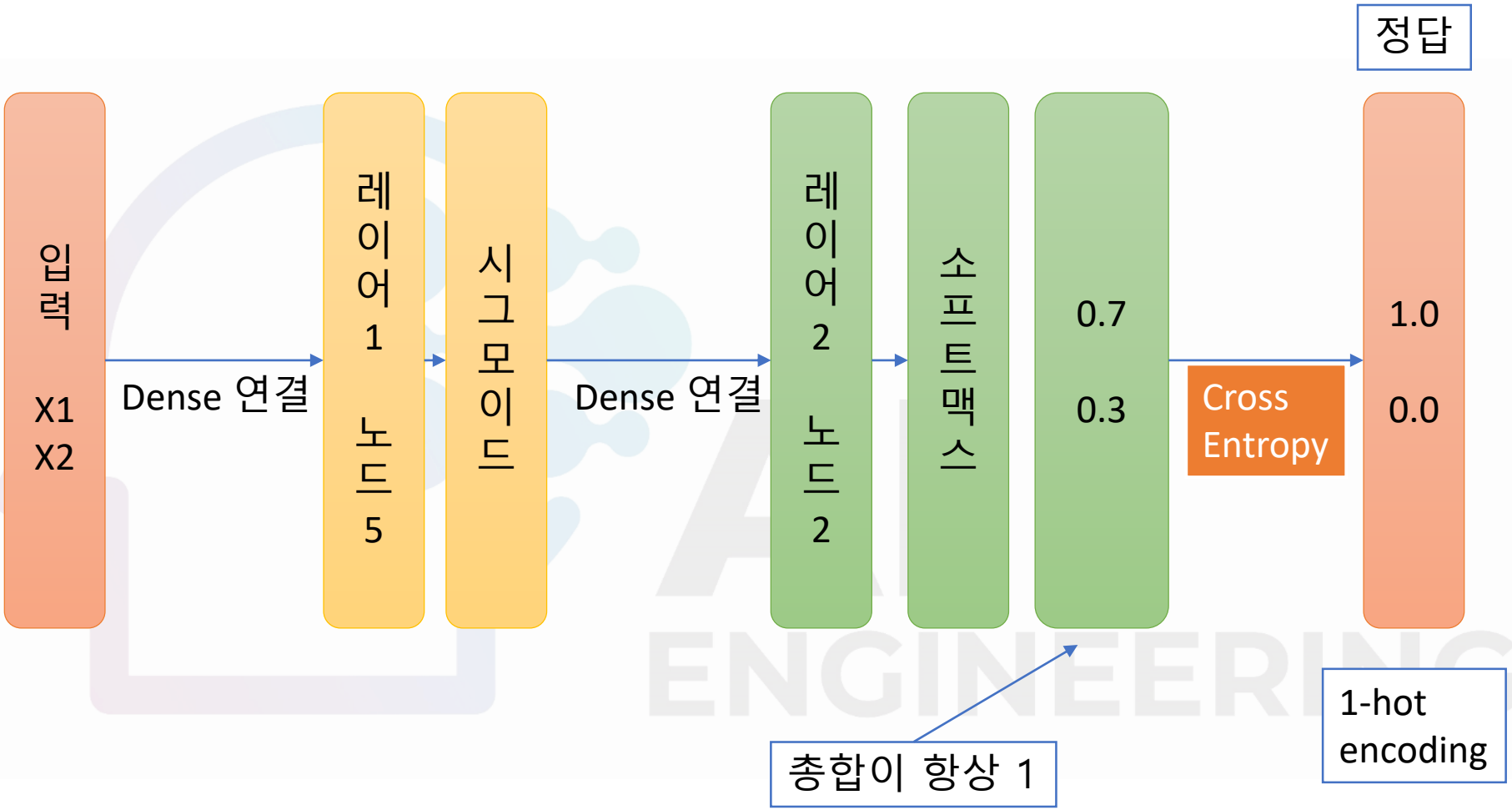
[출처] 모두의 딥러닝

XOR 문제 풀기

- 은닉층에 노드 5개 추가
  - 적절한 개수? → 실험적으로 찾는다.
  - 일반적으로 히든 노드의 개수를 늘이면 학습 잘 됨

- 출력은 One-hot Encoding으로
  - XOR 게이트의 출력 0과 1을 각각의 클래스로 간주
  - [1, 0] , [0, 1], [0, 1], [1, 0]

모델 설계



모델 구현

```
def make_model():
    model = Sequential()
    model.add(Dense(5, input_dim=2, kernel_initializer='normal',
activation="sigmoid"))
    model.add(Dense(2, input_dim=5, kernel_initializer='normal',
activation="softmax"))
    model.summary()
    model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=['accuracy'])
    return model
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 5)                 15
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 12
=================================================================
Total params: 27
Trainable params: 27
Non-trainable params: 0
```

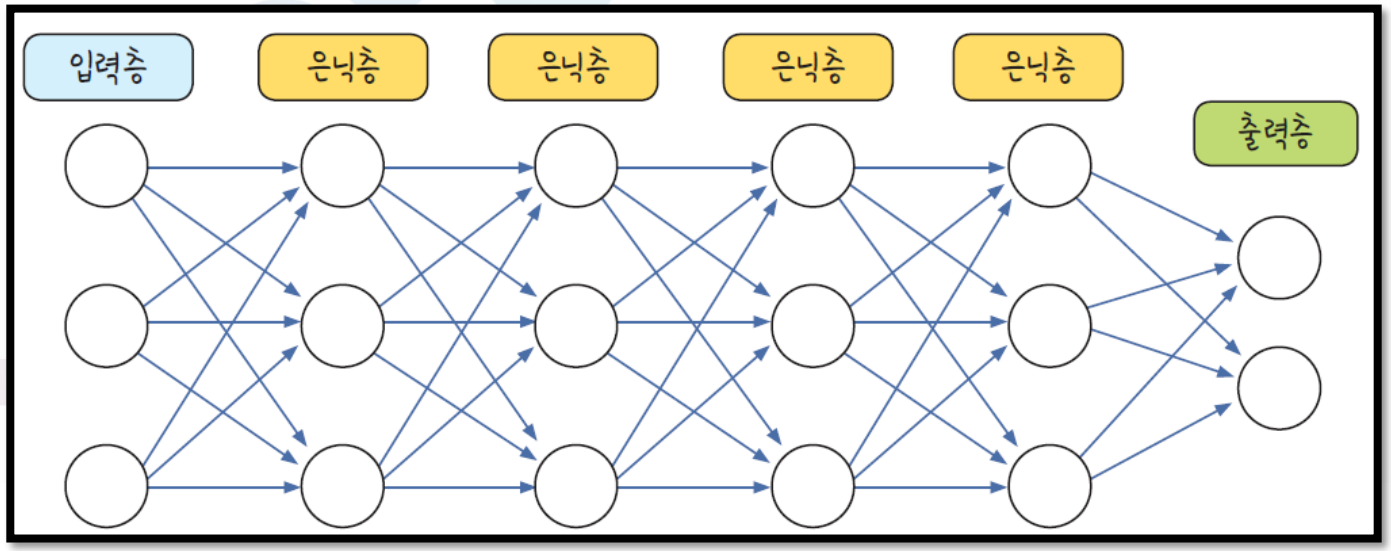
모델 학습 및 테스트

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

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

    perceptron_model.fit(x, y, epochs=1000)

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

```
Epoch 1000/1000
1/1 [==============================] - 0s 0s/step - loss: 0.3597 - accuracy: 1.0000
[[0.6848706  0.3151294 ]
 [0.2230476  0.77695245]
 [0.380831   0.619169  ]
 [0.7218615  0.27813852]]
```

Deep Learning

- MLP 는 XOR 문제를 가볍게 해결
- 신경망을 차곡 차곡 쌓기만 하면 되겠네?
  - ~~응 아니야.~~

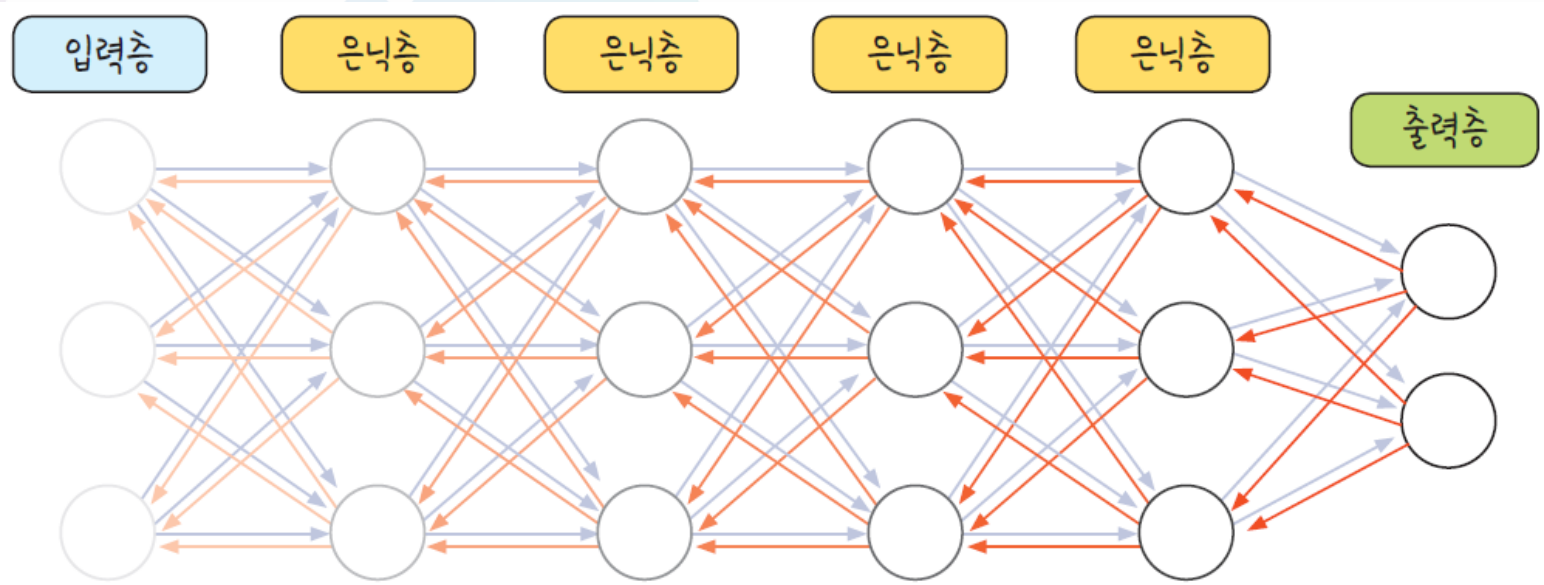


[출처] 모두의 딥러닝

Gradient Vanishing

◆ Back Propagation

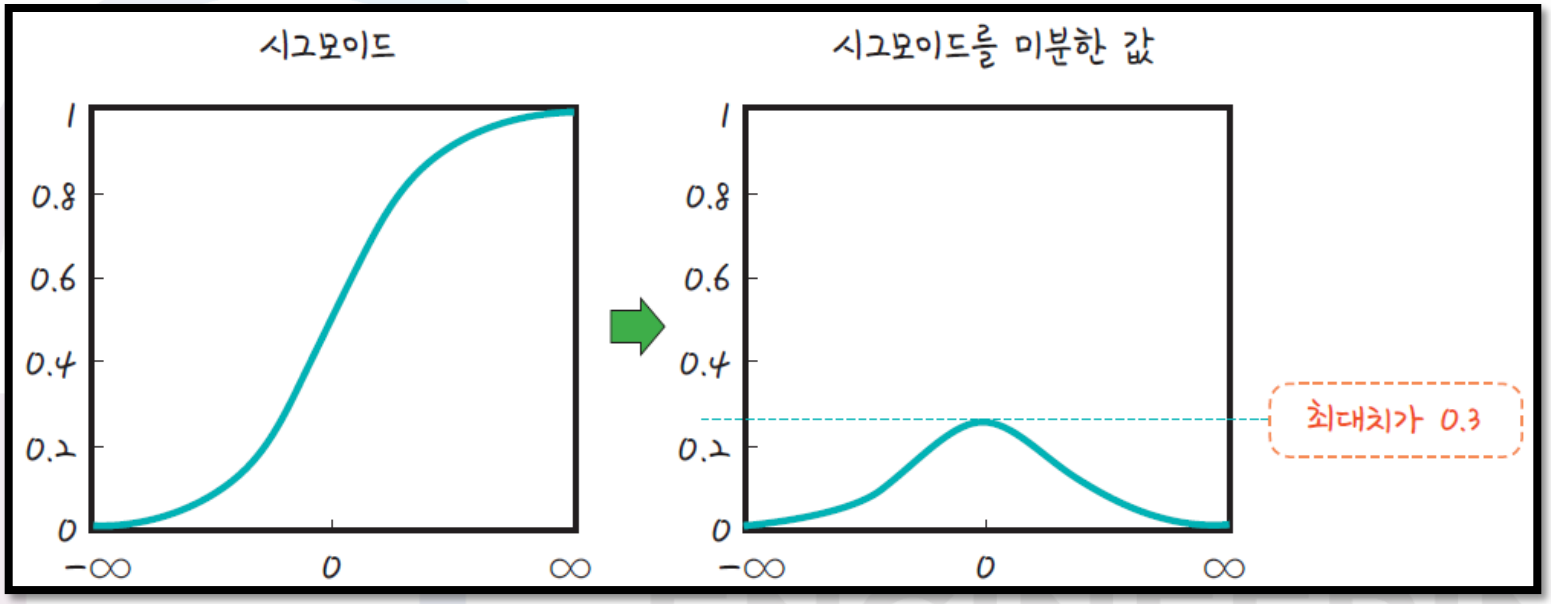
- ➢ 출력층으로 되돌아가며 가중치를 수정
- ➢ 이 때 미분을 사용하는데…



[출처] 모두의 딥러닝

Gradient Vanishing

◆ Sigmoid 함수를 미분하면

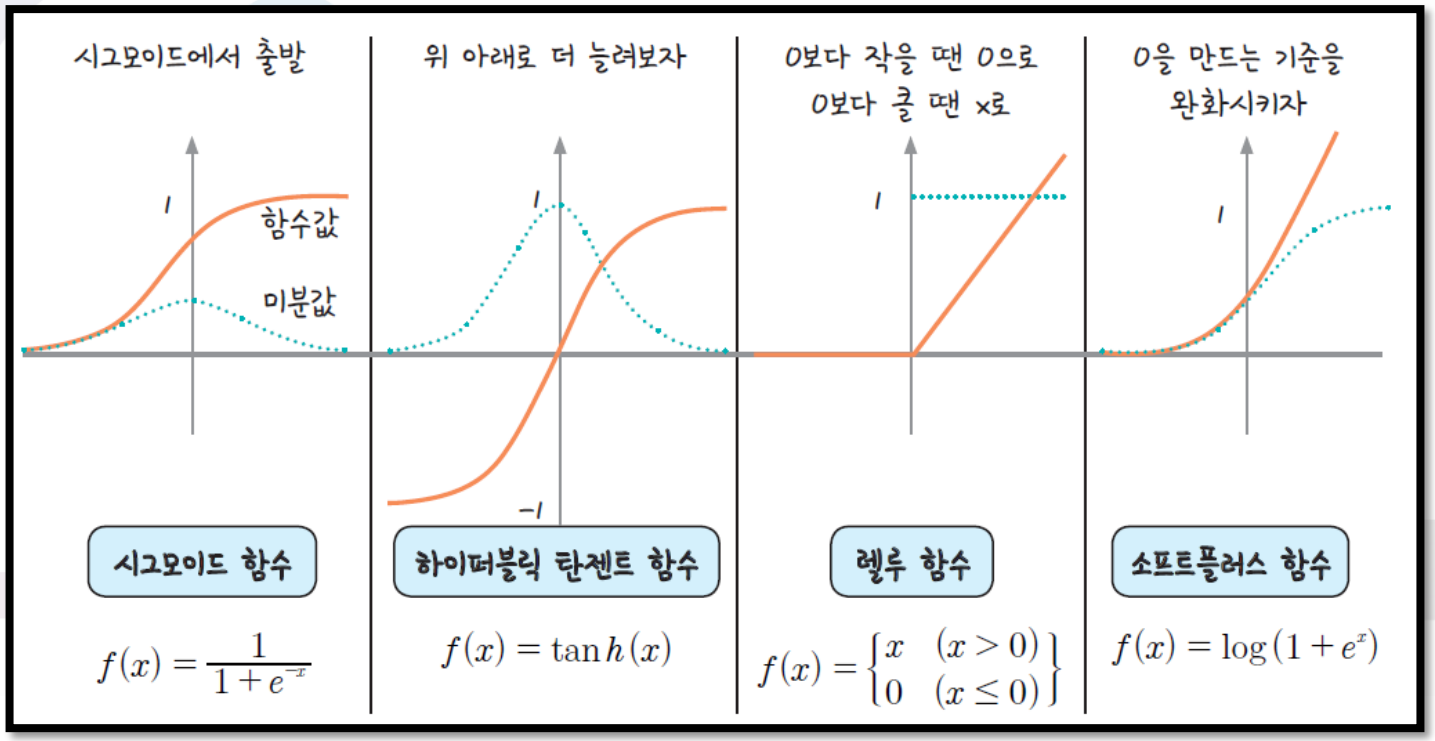


- 여러 층을 거칠 수록 기울기가 사라짐
- 가중치를 수정할 수 없게 됨

[출처] 모두의 딥러닝

Activation Function

◆ 활성화 함수를 다른 걸 써야겠구나?!

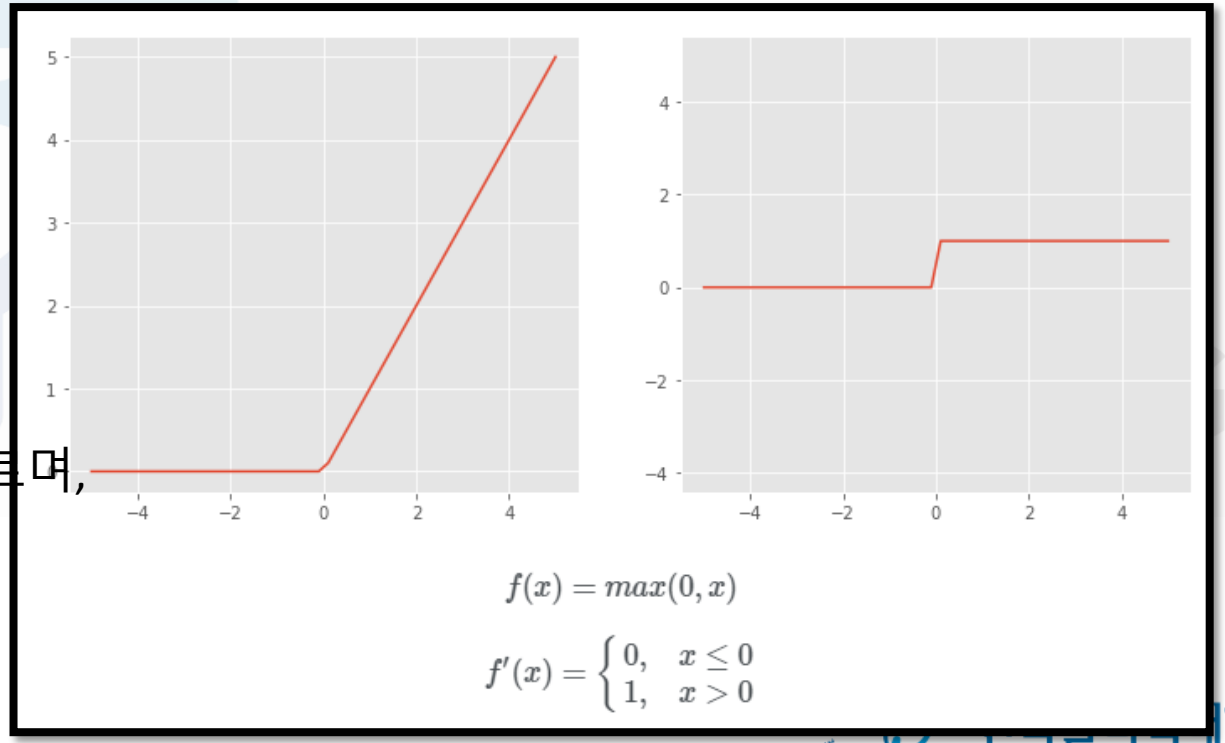


[출처] 모두의 딥러닝

Activation Function

- Keras 는 다양한 활성화 함수를 지원
  - https://keras.io/ko/activations/
  - https://subinium.github.io/introduction-to-activation/

Rectified Linear Unit
제프리 힌튼
시그모이드의 대안으로 떠오르며,
현재 가장 널리 사용



Summary

- **신경망이란?**
  - 인간의 뉴런을 모방
  - Activation Function
- **학습이란?**
  - 손실이 최소가 되는 최적의 Weight를 찾는 것
  - 오류 역전파를 통해 손실을 앞으로 전달
- **다음 시간?**
  - MNIST 손글씨 인식
  - Cifar-10 이미지 분류 실습