

# 인공지능

## 05. Perceptron

Dept. of Digital Contents

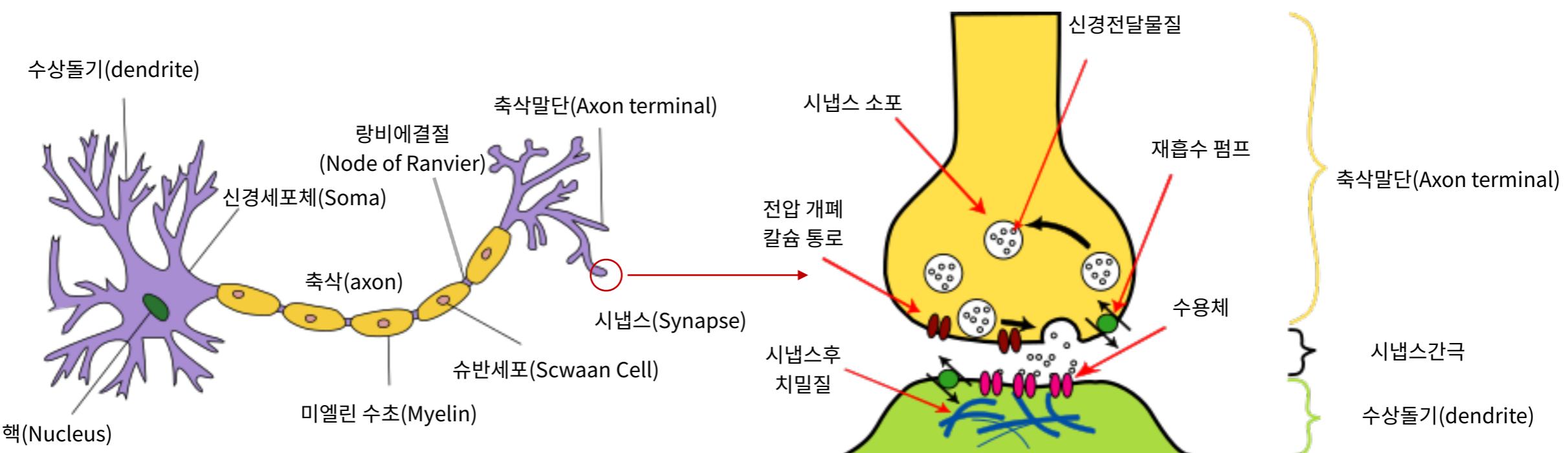


# 1. 신경망

## ○ 생물 신경망(Biological Neural Networks)

### ■ 뉴런(Neuron)

- 뇌가 수행하는 정보처리 단위(인간의 뇌:  $10^{11}$ 개)
- 신경망(Neural Network) : 서로 연결된 뉴런들이 망을 형성한 것



신경세포(뉴런) - 그림 출처 : [ko.wikipedia.org](https://ko.wikipedia.org)

시냅스(Synapse) 구조 - 그림 출처 : [ko.wikipedia.org](https://ko.wikipedia.org)

# 1. 신경망

---

## ○ 인공 신경망(ANN: Artificial Neural Networks)

- 1940년대 신경망 동작을 인공적으로 모방
  - 1943년 : 메컬록(McCulloch)과 피츠(Pitts)의 계산 모형
  - 1949년 : 헤브(Hebb) 최초의 학습 알고리즘 발표
  - 1958년 : 로젠틀랫(Rosenblatt) 퍼셉트론(Perceptron) 제안
    - 위드로(Widrow) & 호프(Hoff) : 퍼셉트론을 개선한 아달린(Adaline)과 마달린(Madaline) 발표
  - 1969년 : 민스키(Minsky)와 페퍼트(Papert) 퍼셉트론의 한계를 수학적으로 입증
  - 1986년 : 루멜하트 et al. 은닉층을 갖는 다층 퍼셉트론과 오류 역전파 알고리즘 제시

## 2. 퍼셉트론

### ○ 퍼셉트론의 원리

- 입력층과 출력 층으로 구성

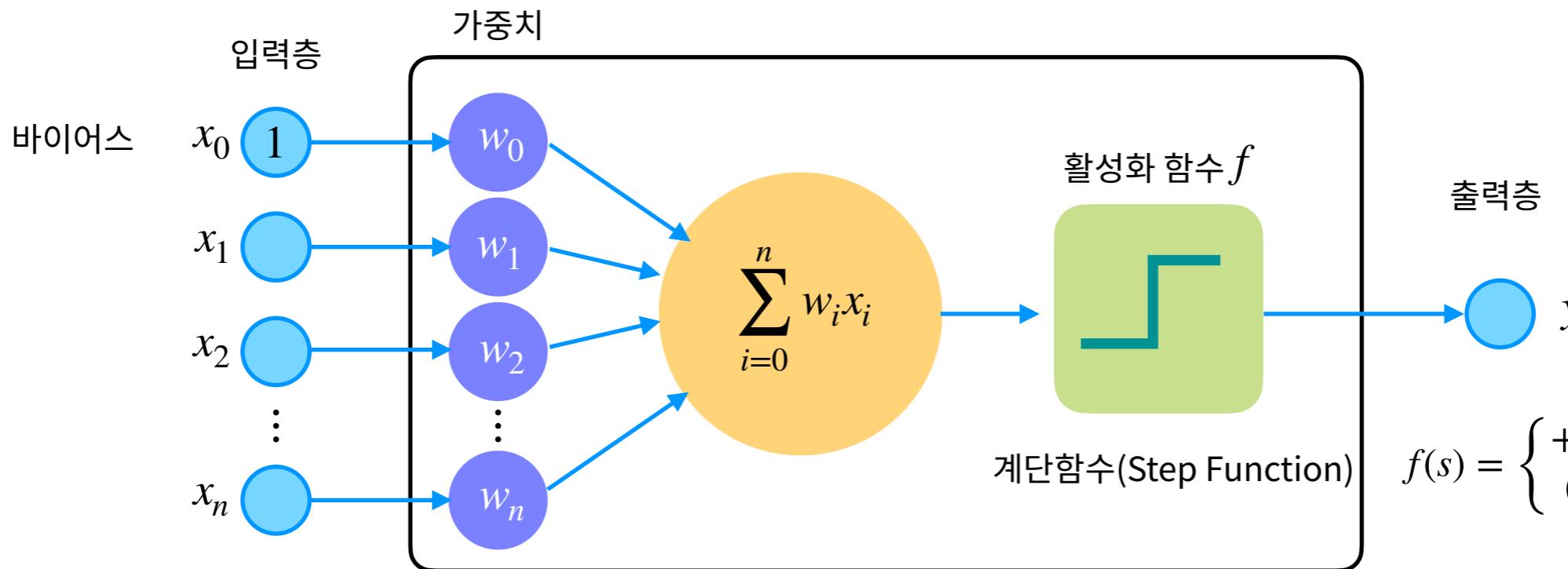
- 입력층(Input layer) : 특징 벡터의 차원  $d+1$ 개의 노드(node)로 구성

- 출력층(Output layer) : 계산 결과

특징 벡터:  $\mathbf{x} = (x_1, x_2, \dots, x_n)$

- 에지(edge) : 입력층과 출력층을 연결, 가중치  $w$ 를 가짐

출력 결과:  $y = f(s) = f\left(\sum_{i=0}^n w_i x_i\right)$



$$f(s) = \begin{cases} +1, & s > 0 \\ 0, & s \leq 0 \end{cases}$$

## 2. 퍼셉트론

### ○ 퍼셉트론으로 인식하기

#### ■ 예제 : 제품의 불량/우량 구분하기

- 제품의 특징 벡터 :  $x = (\text{크기}, \text{색상}) = (a_1, a_2)$

○ 특징 값은 0 또는 1을 갖는다고 가정

○ 생산라인에서 제품 4개를 샘플링한 결과 → 불량 3개, 정상 1개(불량은 1, 정상은 -1로 표기)

$$x_1 = (0,0)$$

$$y_1 = -1$$

$$x_2 = (0,1)$$

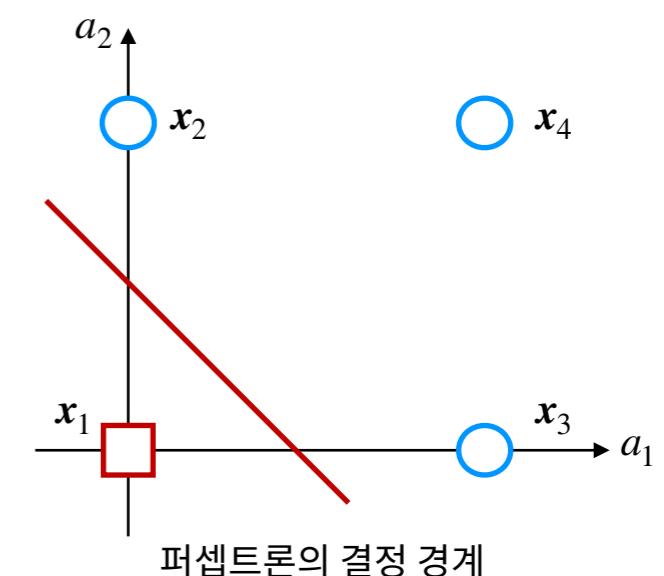
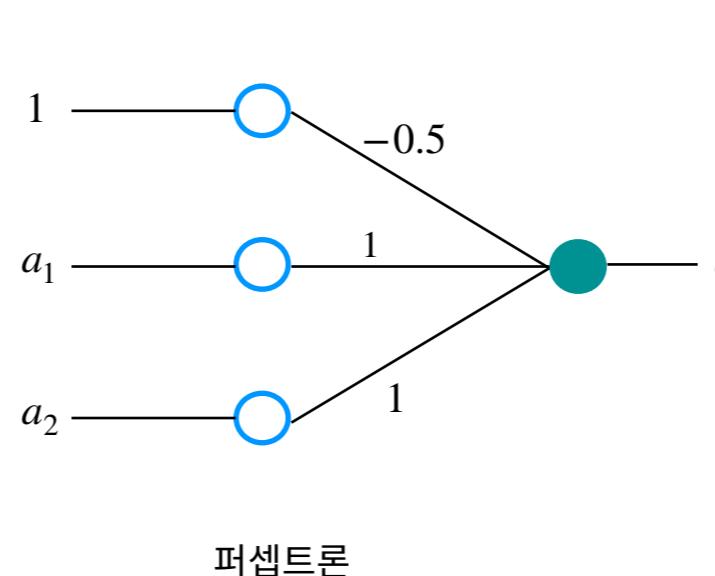
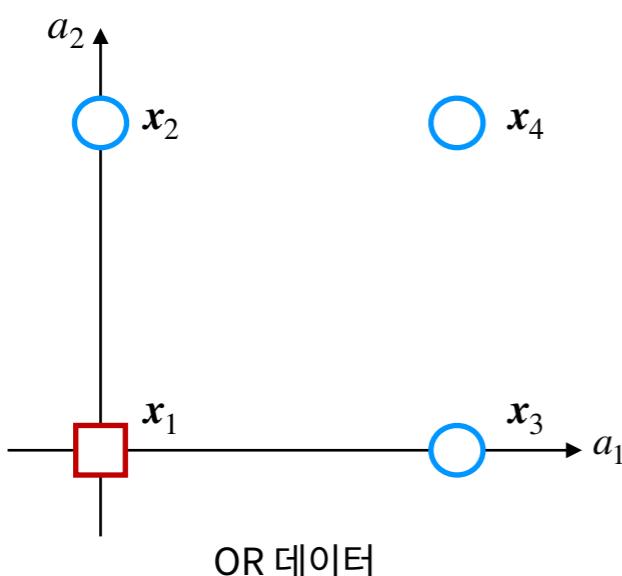
$$y_2 = 1$$

$$x_3 = (1,0)$$

$$y_3 = 1$$

$$x_4 = (1,1)$$

$$y_4 = 1$$



### 3. 퍼셉트론 학습

---

#### ○ 퍼셉트론의 학습

- 최적화문제(optimization problem)을 해결하는 최적화 알고리즘
- 특징 벡터 :  $x = (x_1, x_2, \dots, x_n)$
- 훈련 데이터 세트 : m개의 특징 데이터와 레이블(target)으로 구성
  - k번째 훈련 데이터 : [  $x^k = (x_1^k, x_2^k, \dots, x_n^k)$  ,  $y^k$  ]
- 시간 t에서의 가중치 벡터 :  $w(t) = (w_1(t), w_2(t), \dots, w_n(t))$
- 바이어스를 포함하는 입력 벡터 :  $x^k = (1, x_1^k, x_2^k, \dots, x_n^k)$
- 바이어스를 포함하는 가중치 벡터 :  $w(t) = (w_0(t), w_1(t), w_2(t), \dots, w_n(t))$

# 3. 퍼셉트론 학습

## ○ 퍼셉트론의 학습

- 최적화문제(optimization problem)을 해결하는 최적화 알고리즘

입력(Input) : 학습 데이터  $(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^m, y^m)$

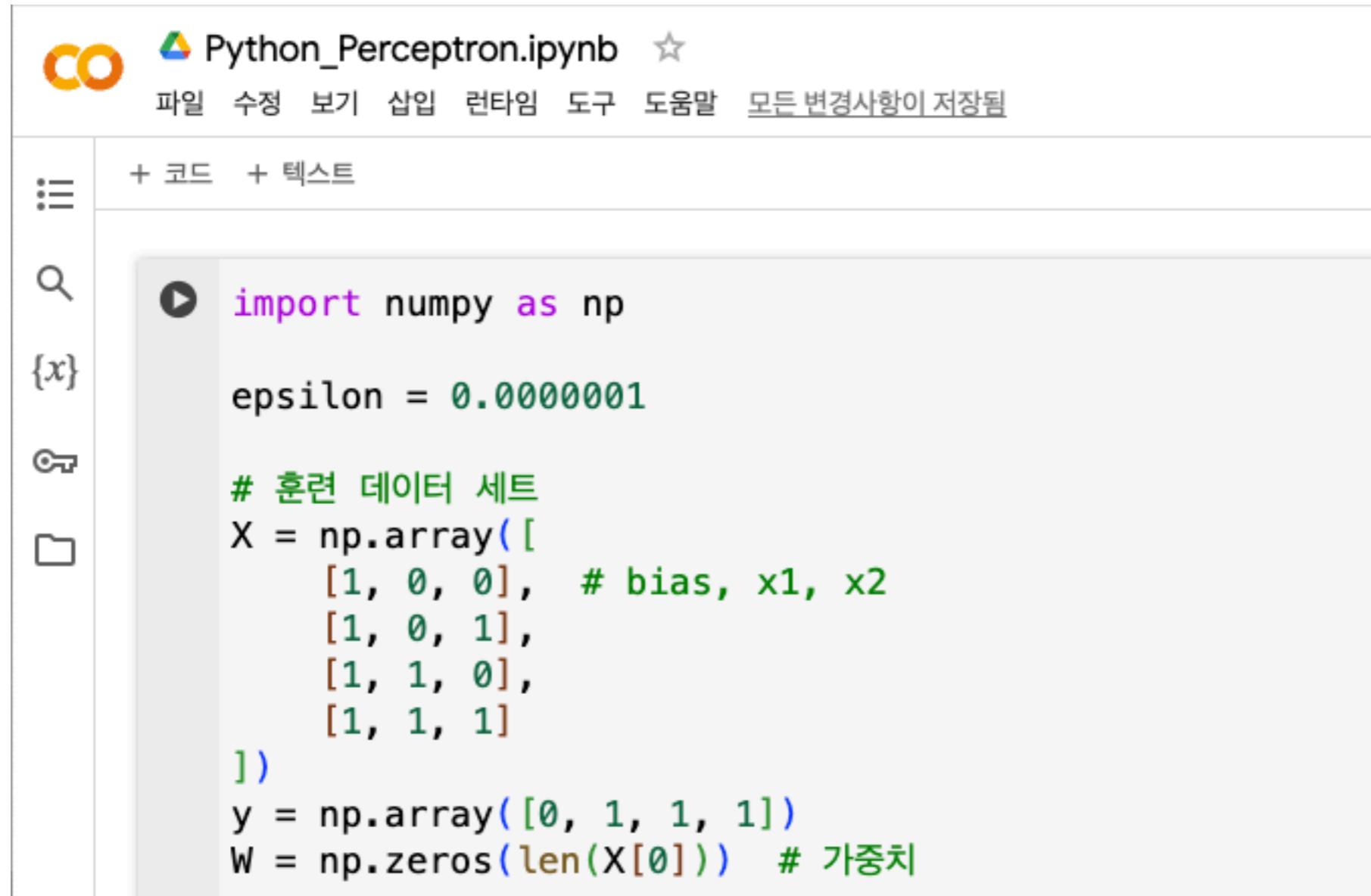
- ① 가중치 벡터  $\mathbf{w}$  의 모든 요소들을 난수로 초기화
- ② while( 가중치가 변경되지 않는 동안)
  - ③ 각 학습 데이터  $\mathbf{x}^k$  와 레이블  $y^k$ 에 대하여
  - ④ 학습 결과값 계산  $o^k(t) = f(\mathbf{w}(t) \cdot \mathbf{x}^k)$
  - ⑤ 모든 가중치에 대하여 손실함수 값을 낮추는 방향을 계산

$$w_i(t+1) = w_i(t) + \eta \cdot (y^k - o^k(t)) \cdot x_i^k$$

출력(Output) : 최적의 가중치 벡터  $\hat{\mathbf{w}}$

# 3. 퍼셉트론 학습

## ○ 퍼셉트론의 학습



The screenshot shows a Google Colab notebook titled "Python\_Perceptron.ipynb". The notebook interface includes a toolbar with file, edit, view, insert, run, tools, help, and a save button. A sidebar on the left contains icons for search, copy, and file operations. The main area displays the following Python code:

```
import numpy as np

epsilon = 0.0000001

# 훈련 데이터 세트
X = np.array([
    [1, 0, 0], # bias, x1, x2
    [1, 0, 1],
    [1, 1, 0],
    [1, 1, 1]
])
y = np.array([0, 1, 1, 1])
W = np.zeros(len(X[0])) # 가중치
```

# 3. 퍼셉트론 학습

## ○ 퍼셉트론의 학습

```
def step_func(t):          # 퍼셉트론의 활성화 함수
    if t > epsilon : return 1
    else: return 0

# 퍼셉트론 학습 알고리즘 구현
def perceptron_fit(X, Y, epochs=10):
    global W
    eta = 0.2      # 학습률

    for t in range(epochs):
        print('epoch=', t, '=====')
        for i in range(len(X)):
            predict = step_func(np.dot(X[i], W))
            error = Y[i] - predict      # 오차 계산
            W += eta * error * X[i]    # 가중치 업데이트
            print('현재 처리 입력=', X[i], '정답=', Y[i],
                  '출력=', predict, '변경된 가중치=', W)
        print('=====')
```

### 3. 퍼셉트론 학습

---

#### ○ 퍼셉트론의 학습

```
# 예측
def perceptron_predict(X):
    global W
    for x in X:
        print(x[1], x[2], '→', step_func(np.dot(x, W)))

perceptron_fit(X, y, 6)
perceptron_predict(X)
```

### 3. 퍼셉트론 학습

#### ○ sklearn에서 퍼셉트론 학습

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** SKLearn\_Perceptron.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cell:** + 코드 + 텍스트
- Code Content:**

```
▶ from sklearn.linear_model import Perceptron
{x}
X = [[0,0], [0,1], [1,0], [1,1]]
y = [0, 1, 1, 1]      # 학습 데이터 변경

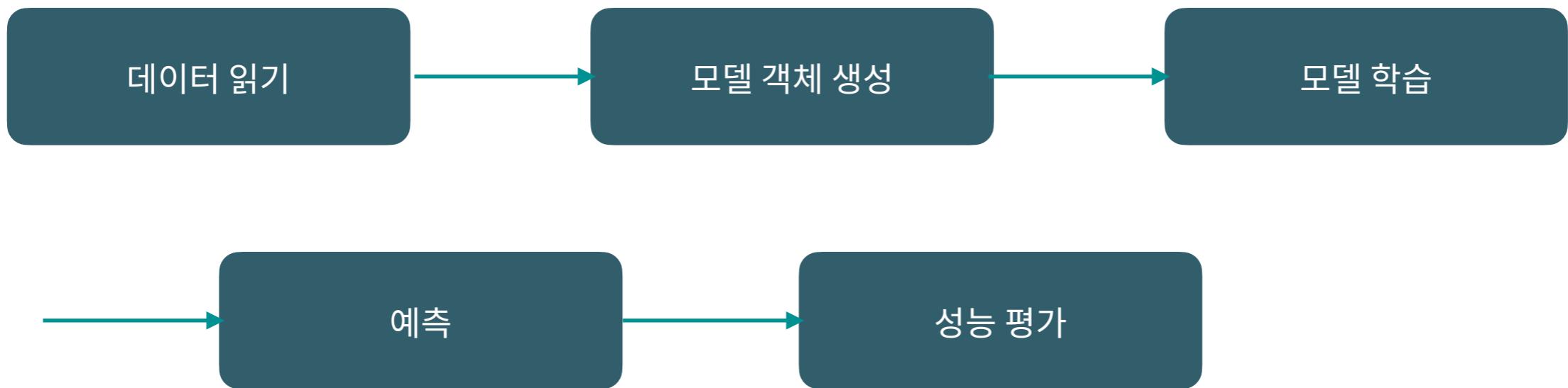
p = Perceptron()
p.fit(X, y)

print('학습된 퍼셉트론의 매개 변수 : ', p.coef_, p.intercept_)
print('훈련 집합에 대한 예측 : ', p.predict(X))
print('정확률 측정 : ', p.score(X,y)*100, '%')
```

### 3. 퍼셉트론 학습

---

#### ○ 기계학습 디자인 패턴



### 3. 퍼셉트론 학습

#### ○ 퍼셉트론으로 필기 숫자 인식하기

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Digits\_Perceptron.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말
- Code Cells:**
  - [ ] from sklearn import datasets  
from sklearn.linear\_model import Perceptron  
from sklearn.model\_selection import train\_test\_split  
import numpy as np
  - [ ] # 데이터 읽고 훈련 데이터와 테스트 데이터로 나누기  
digit = datasets.load\_digits()  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 digit.data, digit.target, train\_size=0.6)
  - [ ] # 모델 객체 생성 : Perceptron  
p = Perceptron(max\_iter=100, eta0=0.001, verbose=0)

### 3. 퍼셉트론 학습

---

#### ○ 퍼셉트론으로 필기 숫자 인식하기

```
[ ] # 모델 학습  
p.fit(X_train, y_train)
```

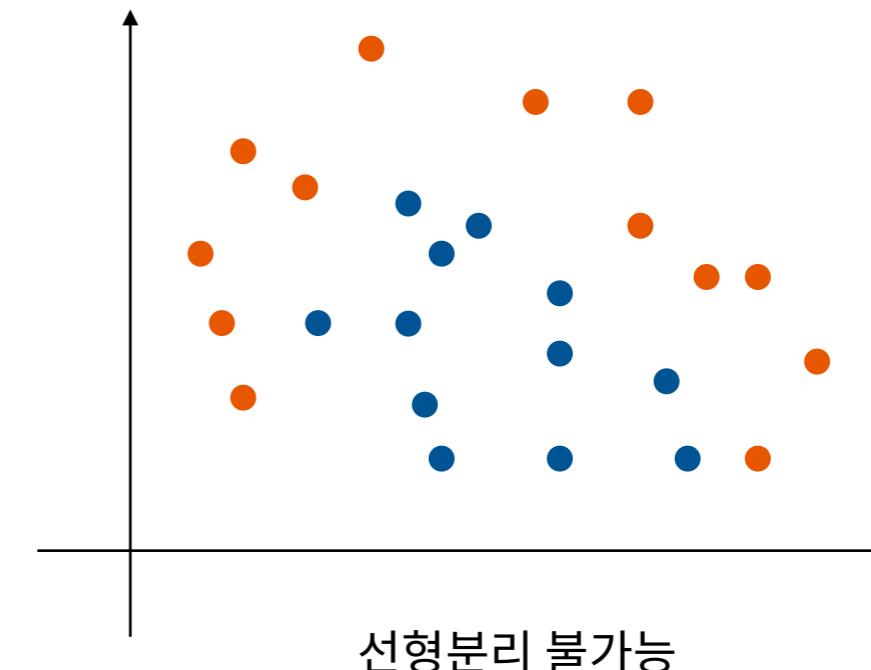
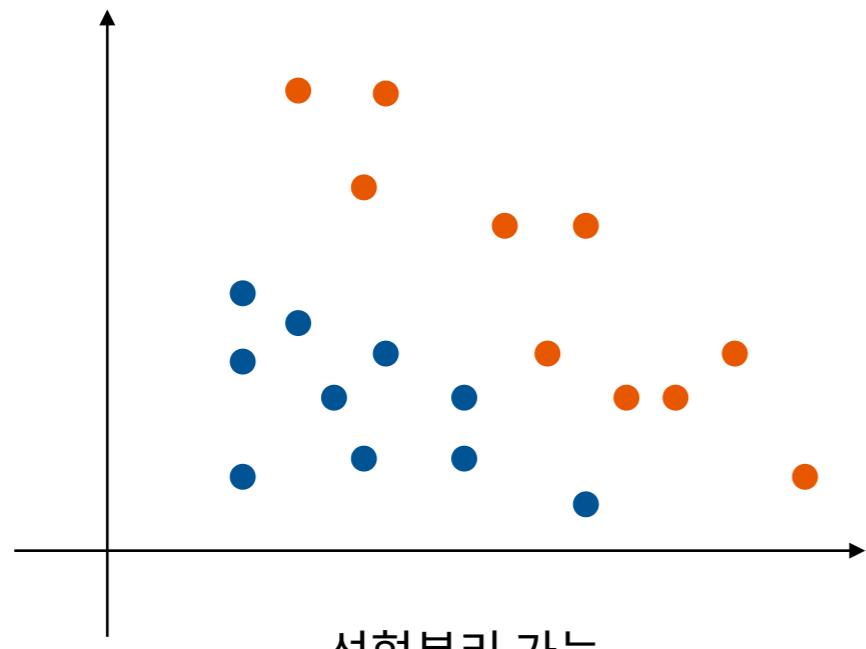
```
[ ] # 학습된 모델로 예측  
res = p.predict(X_test)
```

```
▶ # 성능 평가  
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score  
  
disp = ConfusionMatrixDisplay.from_estimator(p, X_test, y_test)  
  
print(f'정확도는 {accuracy_score(y_test, res) * 100:.3f}%')
```

# 4. 퍼셉트론의 한계

## ○ 퍼셉트론의 한계

- 선형분리가 가능한 데이터에서 정확



- XOR 데이터

- 퍼셉트론의 정확률 상한은 75%

## 4. 퍼셉트론의 한계

### ○ sklearn에서 퍼셉트론 학습 : XOR 문제

The screenshot shows a Jupyter Notebook cell with the following code:

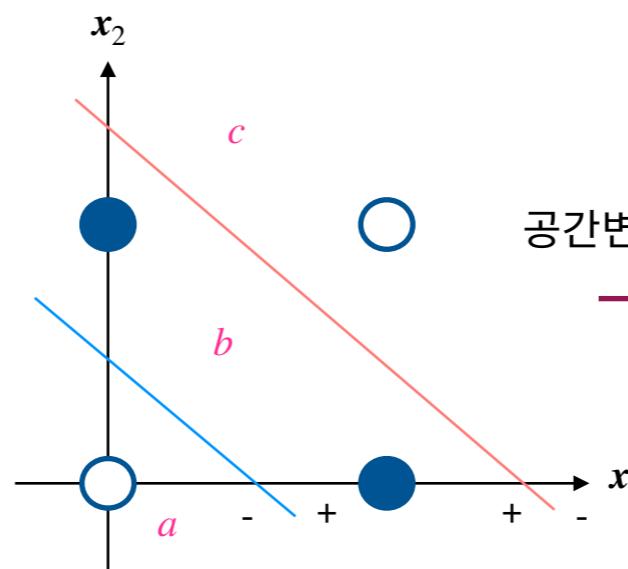
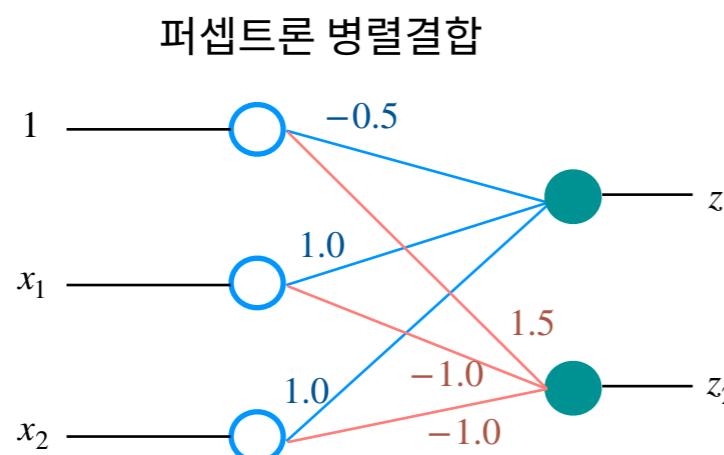
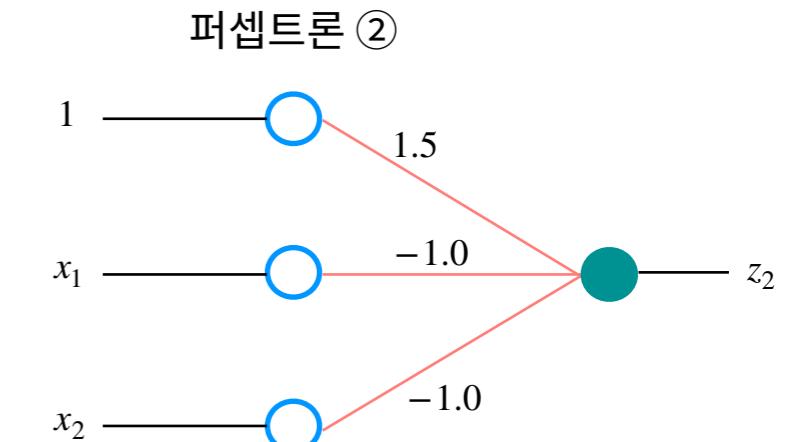
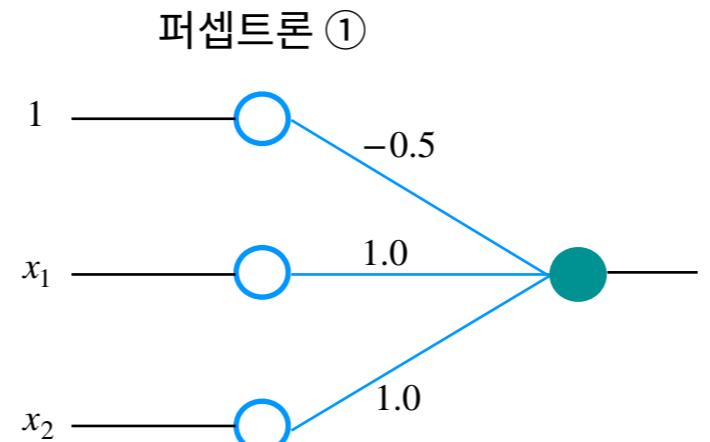
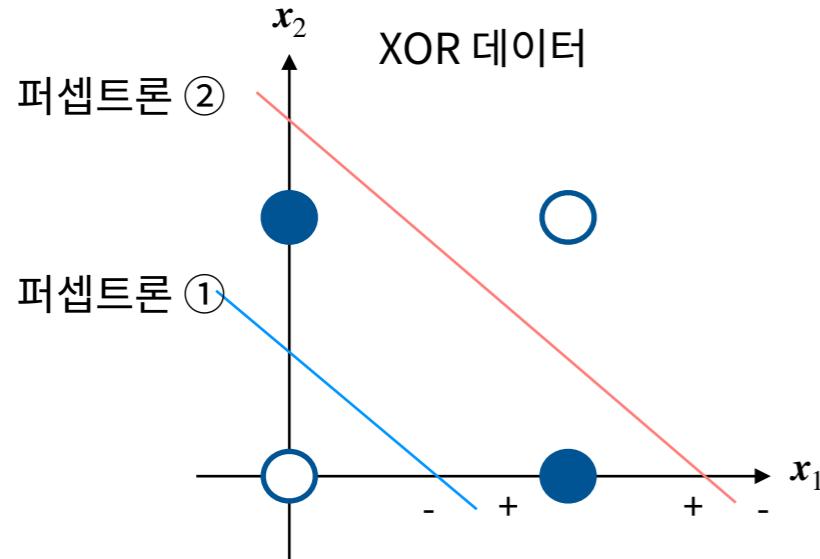
```
▶ from sklearn.linear_model import Perceptron  
X = [[0,0], [0,1], [1,0], [1,1]]  
y = [0, 1, 1, 0] # 학습 데이터 변경  
p = Perceptron()  
p.fit(X, y)  
  
print('학습된 퍼셉트론의 매개 변수 : ', p.coef_, p.intercept_)  
print('훈련 집합에 대한 예측 : ', p.predict(X))  
print('정확률 측정 : ', p.score(X,y)*100, '%')
```

The variable `y` is highlighted with a red box around the value `0`, indicating it is being modified for the XOR problem.

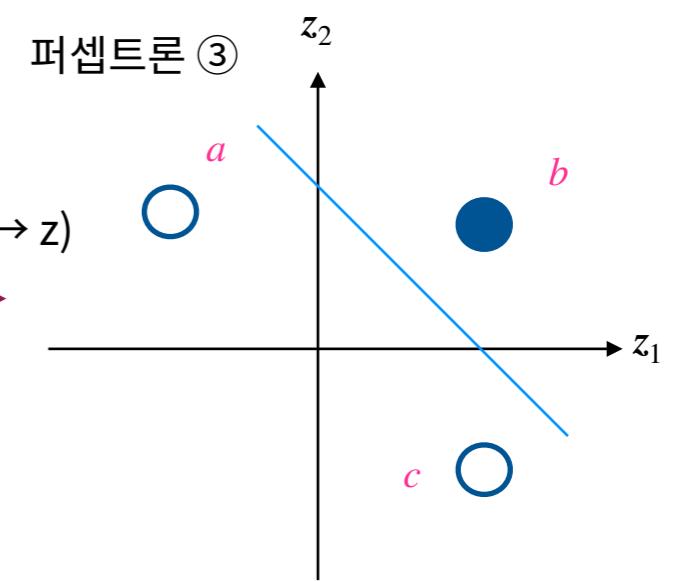
# 4. 퍼셉트론의 한계

## ○ 특징 공간의 변환

- 2개의 퍼셉트론으로 부분 공간 나누기(XOR 데이터)



공간변화( $x \rightarrow z$ )

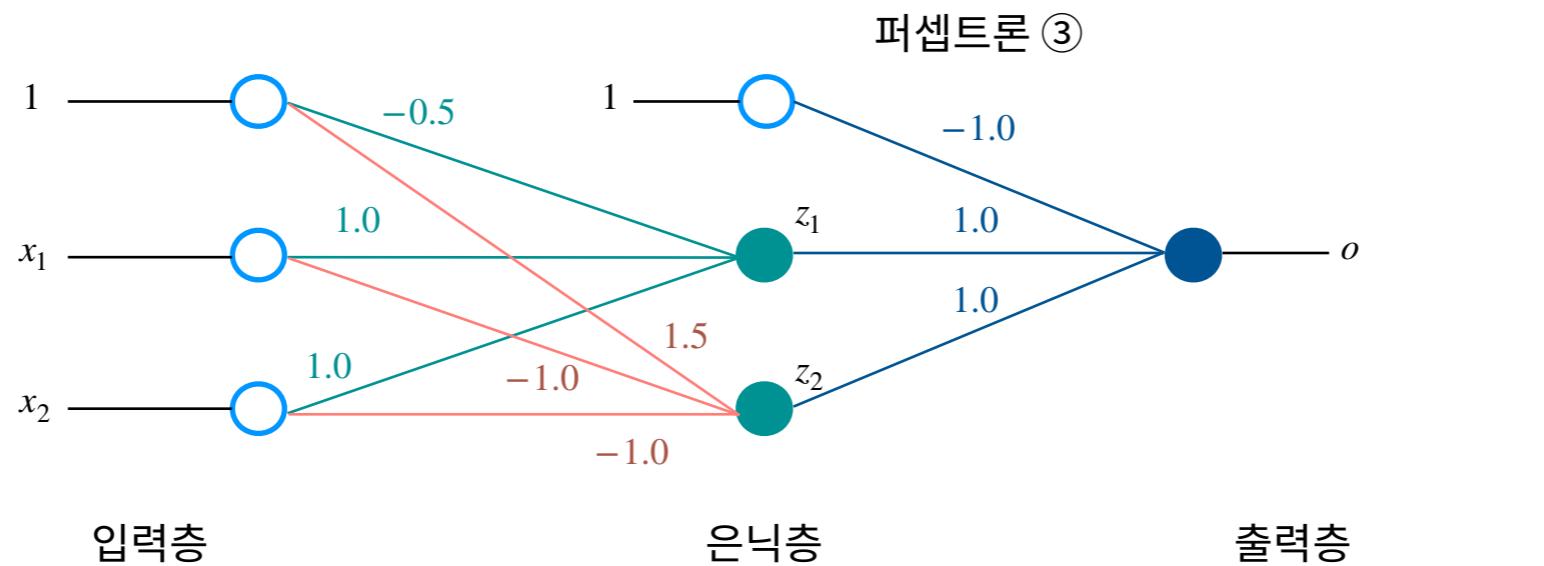


# 4. 퍼셉트론의 한계

## ○ 특징 공간의 변환

- 3번째 퍼셉트론을 순차적으로 덧붙이기 → 다층 퍼셉트론(MLP)

- 은닉층(hidden layer) : 입력층과 출력층 사이 숨어 있는 은닉 공간(잠복 공간)



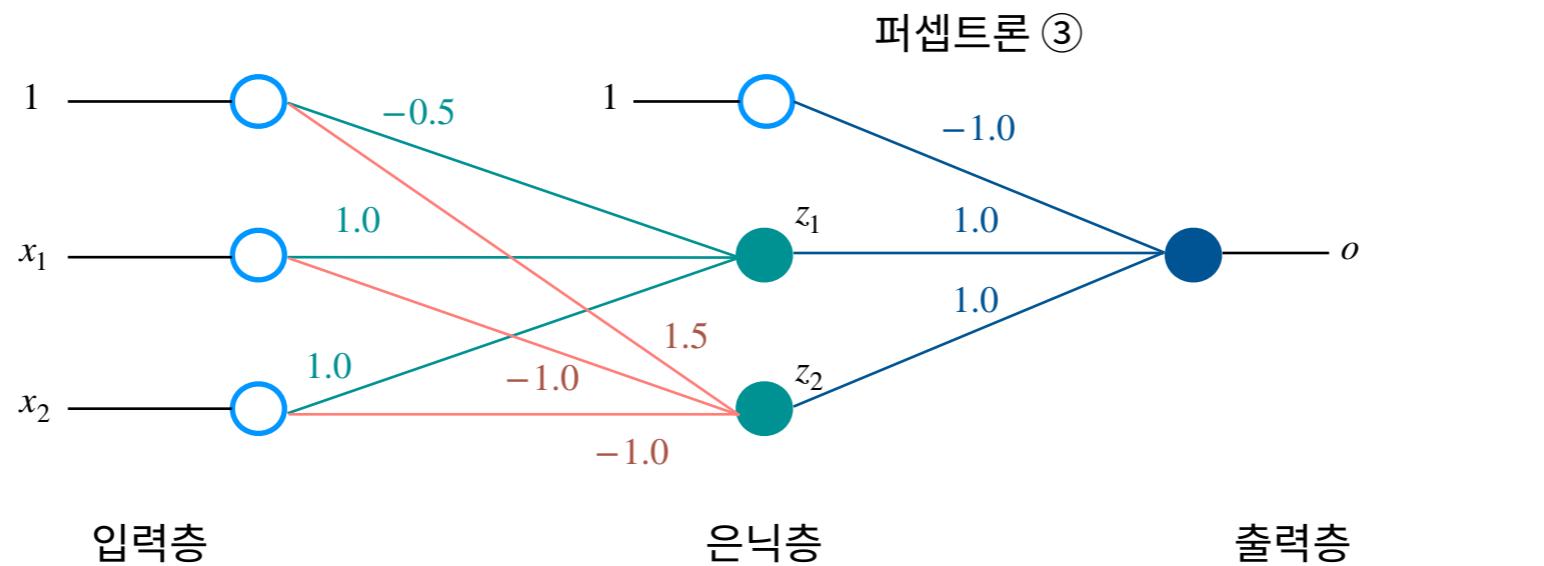
| 원래 특징 공간( $x_1, x_2$ ) | 은닉 특징 공간( $z_1, z_2$ ) | 출력 $o$ | 레이블 $y$ |
|------------------------|------------------------|--------|---------|
| (0,0)                  | (-1,1)                 | -1     | -1      |
| (0,1)                  | (1,1)                  | 1      | 1       |
| (1,0)                  | (1,1)                  | 1      | 1       |
| (1,1)                  | (1,-1)                 | -1     | -1      |

# 4. 퍼셉트론의 한계

## ○ 특징 공간의 변환

- 3번째 퍼셉트론을 순차적으로 덧붙이기 → 다층 퍼셉트론(MLP)

- 은닉층(hidden layer) : 입력층과 출력층 사이 숨어 있는 은닉 공간(잠복 공간)



| 원래 특징 공간( $x_1, x_2$ ) | 은닉 특징 공간( $z_1, z_2$ ) | 출력 $o$ | 레이블 $y$ |
|------------------------|------------------------|--------|---------|
| (0,0)                  | (-1,1)                 | -1     | -1      |
| (0,1)                  | (1,1)                  | 1      | 1       |
| (1,0)                  | (1,1)                  | 1      | 1       |
| (1,1)                  | (1,-1)                 | -1     | -1      |

# 5. 다층 퍼셉트론

## ○ 다층 퍼셉트론의 구조

- 데이터 → 입력층의 노드 개수와 출력 층의 노드 개수를 결정

- 예) 붓꽃(Iris) 데이터 : 입력층 노드 = 5개, 출력층 노드 = 3개

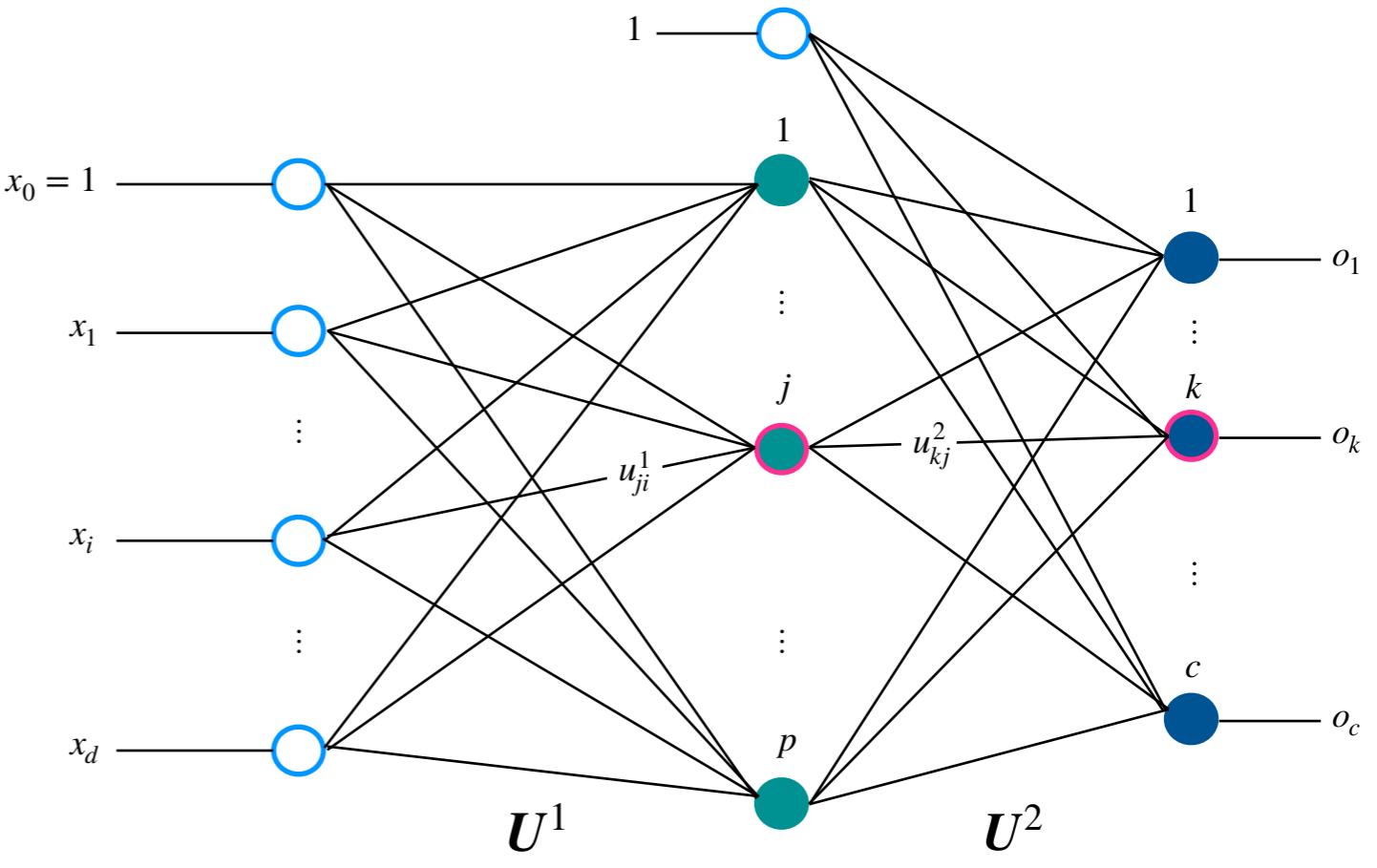
- 가중치 집합

- $U^1$  : 입력층과 은닉층을 연결

- $U^2$  : 은닉층과 출력층을 연결

$$U^1 = \begin{bmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{bmatrix}$$

$$U^2 = \begin{bmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^2 & u_{p1}^2 & \cdots & u_{cp}^2 \end{bmatrix}$$



# 5. 활성 함수

---

## ○ 신경망에서 사용하는 여러가지 활성함수(Activation Function)

- 계단 함수(Step Function)

- 입력이 0을 넘으면 1을 출력, 그외에는 0을 출력하는 함수

- 시그모이드(Sigmoid)

- 입력에 따라 출력이 연속적으로 변화, 출력 값은 실수

- 하이퍼볼릭 탄젠트(tanh)

- 시그모이드 편향이동 문제를 보완한 쌍곡선 함수, 1차 미분값이 양수/음수가 가능

- 렉티파이어(ReLU: Rectified Linear Unit)

- 입력이 0을 넘으면 그대로의 값을 출력

# 5. 활성 함수

---

## ○ 신경망에서 사용하는 여러가지 활성함수(Activation Function)

- 소프트 플러스

- ReLU가 0이 되는 순간을 완화

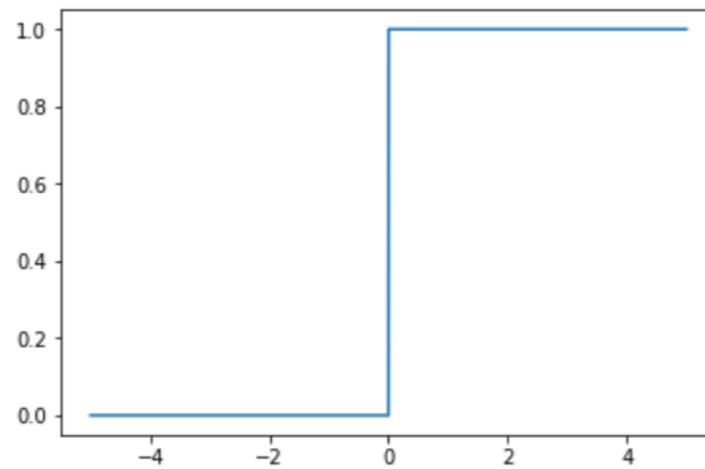
- 소프트맥스 함수(Softmax Function)

- 신경망의 출력층에서 주로 사용하는 활성함수
  - 3개 이상으로 분류하는 다중 클래스 분류에 사용
  - 모든 노드를 고려하여 각 클래스에 속할 확률을 구함

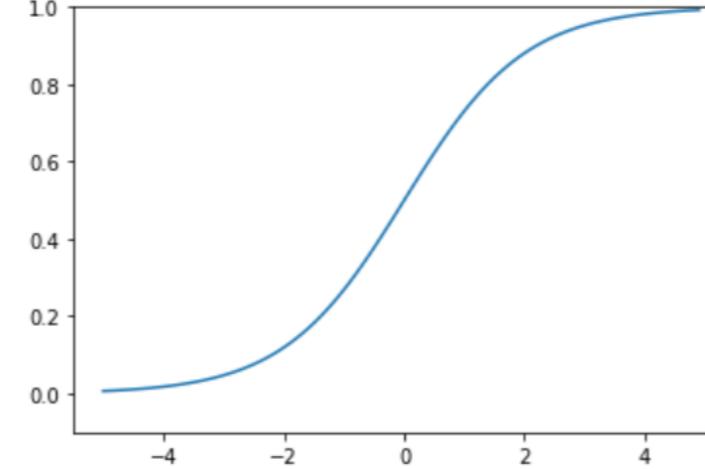
# 5. 활성 함수

## ○ 신경망에서 사용하는 여러가지 활성함수(Activation Function)

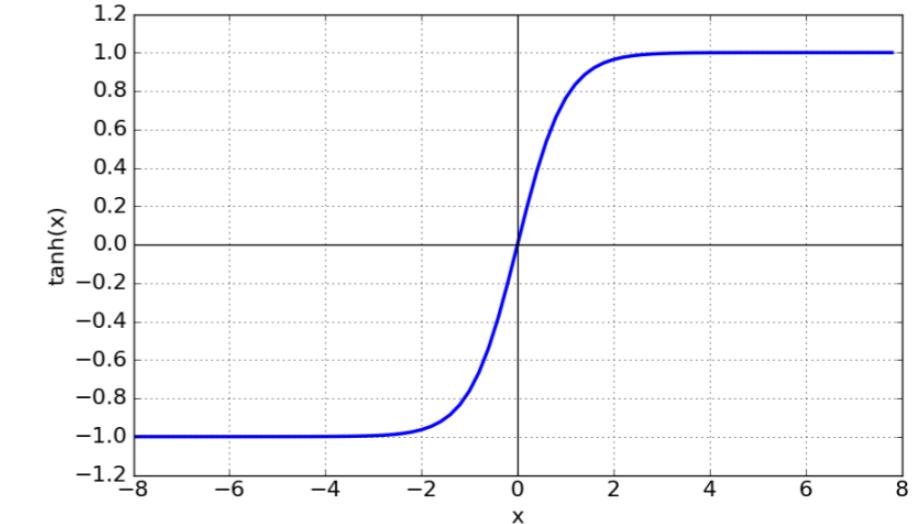
계단함수



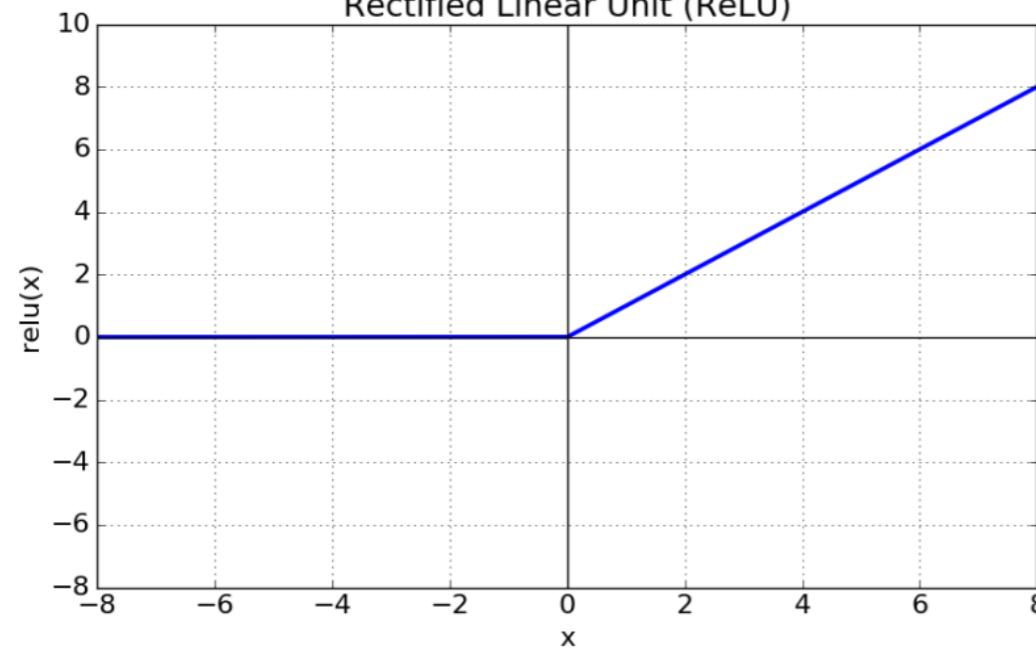
시그모이드



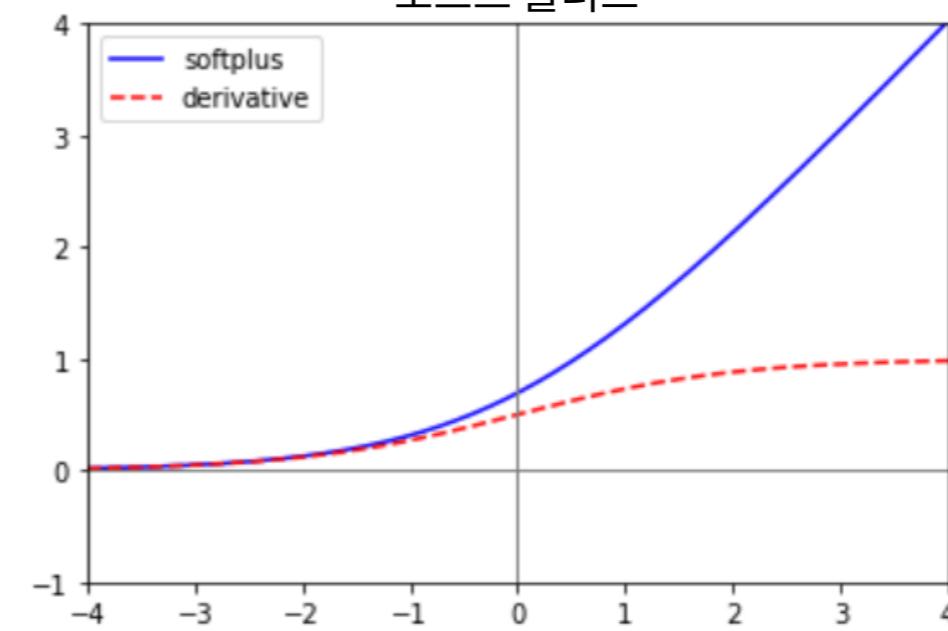
tanh 함수



Rectified Linear Unit (ReLU)



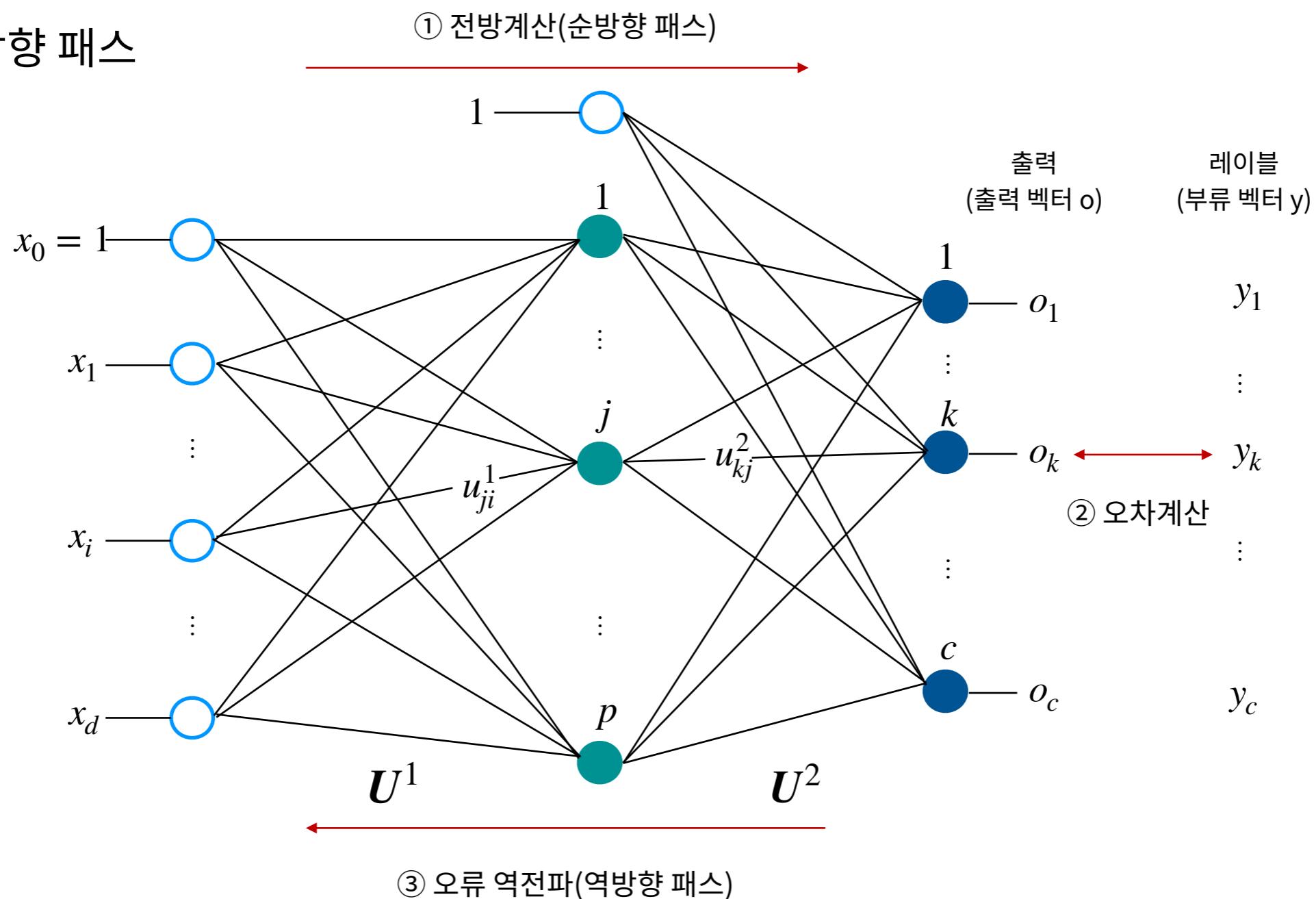
소프트 플러스



# 6. 학습 알고리즘

## ○ 학습 알고리즘

### ■ 순방향 패스



# 6. 손실함수의 설계

---

## ○ 손실함수

### ■ 오차(error)

- 결과값으로 얻기를 바라는 값(target)과 실제 얻은 결과값(output)의 차이
- 예) 평균 제곱 오차(MSE: Mean Squared Error)

$$E = \frac{1}{m} ((y_1 - o_1)^2 + (y_2 - o_2)^2 + \cdots + (y_c - o_c)^2) = \frac{1}{m} \sum_i (y_i - o_i)^2$$

### ■ 최적화(optimization)

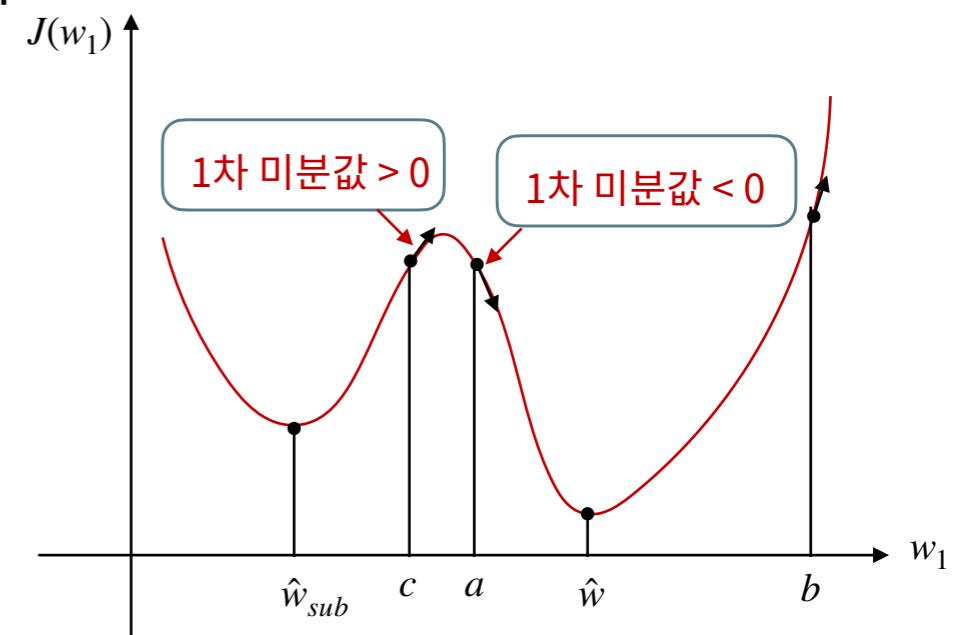
- 변수 간의 차이(거리)를 줄이는 방향
- 가중치  $w_{ij}$ 에 대해 손실함수  $E$ 의 그래디언트( $\nabla$ )가 음수 방향으로 조정

$$\nabla E = \frac{\partial E}{\partial w_{ij}} \quad \mathbf{U}^2 = \mathbf{U}^2 + \rho (-\nabla \mathbf{U}^2) \quad \mathbf{U}^1 = \mathbf{U}^1 + \rho (-\nabla \mathbf{U}^1)$$

# 7. 역방향 패스

## ○ 역전파 알고리즘

- 손실함수를 최소로 만드는 최적화 문제
  - 최적화 문제를 해결하는 Optimizer(Solver)가 필요
  - 예) 경사하강법 : 매개변수가 1개인 함수의 1차 미분값이 음수(-)인 방향으로 이동
  - 손실함수를 가중치로 미분한 값이 양수 → 가중치를 감소
  - 손실함수를 가중치로 미분한 값이 음수 → 가중치를 증가



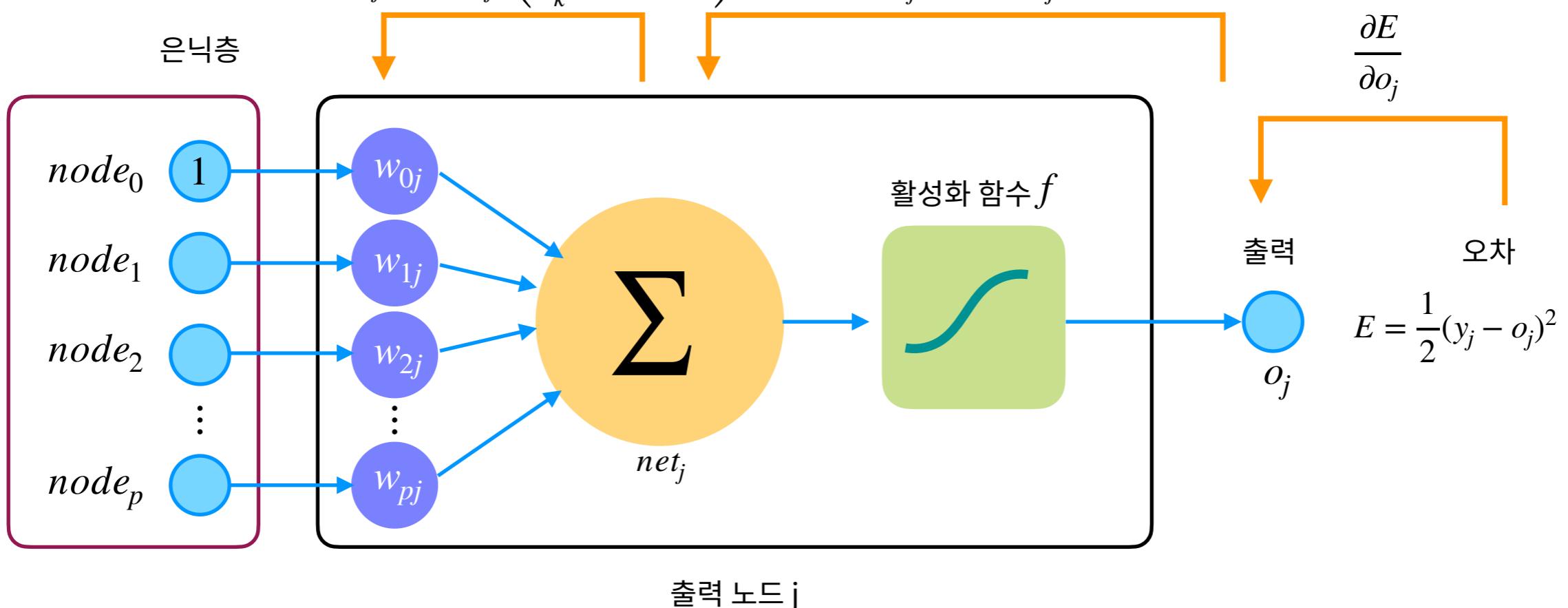
# 7. 역방향 패스

## ○ 역전파 알고리즘

- 가중치 업데이트 규칙

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left( \sum_k w_{kj} node_k \right) \quad \frac{\partial o_j}{\partial net_j} = \frac{\partial f(net_j)}{\partial net_j} = f'(0)$$



# 7. 역방향 패스

## ○ 최적화 알고리즘(Optimizer, Solver)의 종류

| Solver 이름                             | 특징  | 개선효과     |
|---------------------------------------|---|----------|
| 확률적 경사하강법<br>(SGD)                    | 미니배치(mini-batch)를 사용<br>랜덤하게 추출한 일부 데이터를 사용해 더 빠르게 더 자주 업데이트    | 속도       |
| 모멘텀(Momentum)                         | 이전의 batch 학습 결과(운동량)를 고려한 GD<br>관성의 방향을 고려해 진동과 폭을 줄임           | 정확도      |
| 네스테로프 모멘텀<br>(Nesterov Accelerated)   | 모멘텀이 이동시킬 방향으로 미리 이동해서 기울기를 계산(불필요한 이동을 줄임)                     | 정확도      |
| 아다그라드<br>(AdaGrad: Adaptive Gradient) | 학습을 통해 크게 변동이 있던 가중치에 대해서는 학습률을 감소<br>변동이 별로 없었던 가중치는 학습률을 증가시킴 | 보폭       |
| 알엠에스프롭(RMSProp)                       | 아다그라드의 학습률 소실 문제를 보완<br>가중치 기울기 중 최신 기울기들이 더 반영되도록 개선           | 보폭       |
| 아담(Adam)                              | 모멘텀 + 알엠에스프롬, 현재 가장 자주 사용<br>방향과 스텝 사이즈를 적절히 조절                 | 정확도 & 보폭 |

# 8. MLP 프로그래밍

## ○ MLP로 필기 숫자 인식하기

CO  MLP\_digits.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 오후 12:15에 마지막으로 저장됨

+ 코드 + 텍스트

[1] from sklearn import datasets  
from sklearn.neural\_network import MLPClassifier  
from sklearn.model\_selection import train\_test\_split  
import numpy as np

[2] # 데이터 셋을 읽고 훈련 집합과 테스트 집합으로 분할  
digit = datasets.load\_digits()  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 digit.data, digit.target, train\_size=0.6)

# 8. MLP 프로그래밍

## ○ MLP로 필기 숫자 인식하기

```
[ ] # MLP 분류기 모델 학습  
mlp = MLPClassifier(hidden_layer_sizes=(100),  
                     learning_rate_init = 0.001, batch_size=32,  
                     max_iter=300, solver='sgd', verbose=True)  
  
mlp.fit(X_train, y_train)
```

```
[ ] # 테스트 집합으로 예측  
res = mlp.predict(X_test)
```

▶ # 성능 평가

```
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score  
  
print(f'정확도는 {accuracy_score(y_test, res)*100:.3f}%')  
  
ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test)
```

# 8. MLP 프로그래밍

## ○ MNIST

- 미국 국립표준 기술 연구소(NIST)가 수집한 데이터

- 필기 숫자 데이터
  - 해상도 28x28의 숫자당 7천개씩 총 7만개 샘플
  - 픽셀당 [0,255]사이의 명암값을 가짐

sklearn의 digits

- 해상도 8x8의 1,797자
- 픽셀당 [0,16]사이의 명암값

- fashion MNIST

- 28x28 픽셀 맵으로 이루어진 패션 아이템 이미지 데이터셋
- 훈련집합 6,000개, 테스트집합 10,000개 샘플로 구성



# 8. MLP 프로그래밍

## ○ MLP로 MNIST 필기 숫자 인식하기

CO  MLP\_MNIST.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

[ ] `from sklearn.datasets import fetch_openml  
from sklearn.neural_network import MLPClassifier  
import numpy as np`

{x}

[ ] `# MNIST 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할  
mnist = fetch_openml('mnist_784')  
mnist.data = mnist.data/255.0 # [0, 255] 범위를 [0, 1]로 정규화  
X_train = mnist.data[:60000]  
X_test = mnist.data[60000:]  
y_train = np.int16(mnist.target[:60000]) # mnist.target은 str 타입  
y_test = np.int16(mnist.target[60000:])`

key

file

# 8. MLP 프로그래밍

## ○ MLP로 MNIST 필기 숫자 인식하기

```
▶ # MLP 분류기 모델 학습
mlp = MLPClassifier(hidden_layer_sizes=(100),
                     learning_rate_init=0.001, batch_size=512,
                     max_iter=300, solver='adam', verbose=True)
mlp.fit(X_train, y_train)

# 테스트 집합으로 예측
res = mlp.predict(X_test)

# 성능 평가
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score

print(f'정확도는 {accuracy_score(y_test, res)*100 : .3f}%')
ConfusionMatrixDisplay.from_estimator(mlp, X_test, y_test)
```