KAIG

딥러닝 스터디

모델분석	모두의 딥러닝	복 습
SVM	5~8장	학습 / 훈련

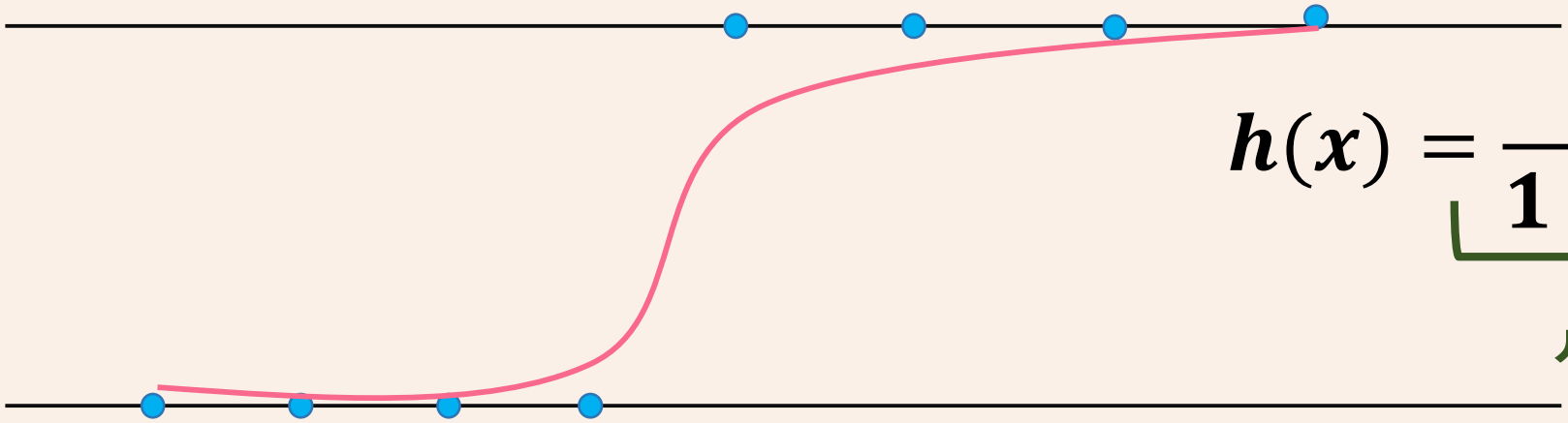
Index

01 로지스틱 회귀

02 다층 퍼셉트론

03 오차 역전파

로지스틱 회귀



$$h(x) = \frac{1}{1 + e^{(-ax+b)}}$$

시그모이드 함수

$$\text{Loss} = -\{y \log h(x) + (1-y) \log(1 - h(x))\}$$

A
실측값이 1인
그래프에서
생기는 오차

B
실측값이 0인
그래프에서
생기는 오차

```
# x,y의 데이터 값
data = [[2, 0], [4, 0], [6, 0], [8, 1], [10, 1], [12, 1], [14, 1]]
x_data = [x_row[0] for x_row in data]
y_data = [y_row[1] for y_row in data]

# a와 b의 값을 임의로 정함
a = tf.Variable(tf.random_normal([1], dtype=tf.float64, seed=0))
b = tf.Variable(tf.random_normal([1], dtype=tf.float64, seed=0))
```

준비물:

Numpy / tesorflow
x_data, y_data,
random a, b

```
# y 시그모이드 함수의 방정식을 세움
y = 1/(1 + np.e**(a * x_data + b))

# loss를 구하는 함수
loss = -tf.reduce_mean(np.array(y_data) * tf.log(y) + (1 - np.array(y_data)) * tf.log(1 - y))

# 학습률 값
learning_rate=0.5
```

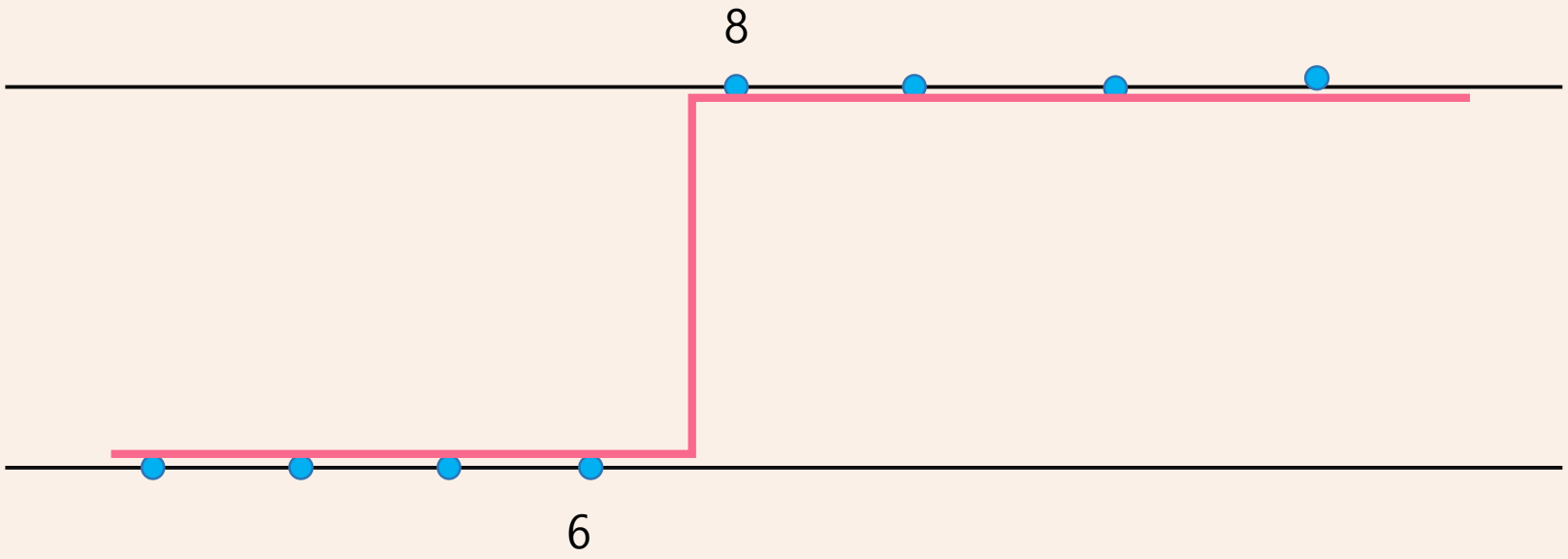
시그모이드 함수 (예측값)
오차함수 (실측값과 비교)

```python
# loss를 최소로 하는 값 찾기
gradient_decent = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

# 학습
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(60001):
        sess.run(gradient_decent)
        if i % 6000 == 0:
            print("Epoch: %.f, loss = %.4f, 기울기 a = %.4f, 바이어스 b = %.4f" % (i, sess.run(loss), sess.run(a), sess.run(b)))
```

```
Epoch: 0, loss = 1.2676, 기울기 a = 0.1849, 바이어스 b = -0.4334
Epoch: 6000, loss = 0.0152, 기울기 a = -2.9211, 바이어스 b = 20.2982
Epoch: 12000, loss = 0.0081, 기울기 a = -3.5637, 바이어스 b = 24.8016
Epoch: 18000, loss = 0.0055, 기울기 a = -3.9557, 바이어스 b = 27.5463
Epoch: 24000, loss = 0.0041, 기울기 a = -4.2380, 바이어스 b = 29.5231
Epoch: 30000, loss = 0.0033, 기울기 a = -4.4586, 바이어스 b = 31.0675
Epoch: 36000, loss = 0.0028, 기울기 a = -4.6396, 바이어스 b = 32.3346
Epoch: 42000, loss = 0.0024, 기울기 a = -4.7930, 바이어스 b = 33.4086
Epoch: 48000, loss = 0.0021, 기울기 a = -4.9261, 바이어스 b = 34.3406
Epoch: 54000, loss = 0.0019, 기울기 a = -5.0436, 바이어스 b = 35.1636
Epoch: 60000, loss = 0.0017, 기울기 a = -5.1489, 바이어스 b = 35.9005
```

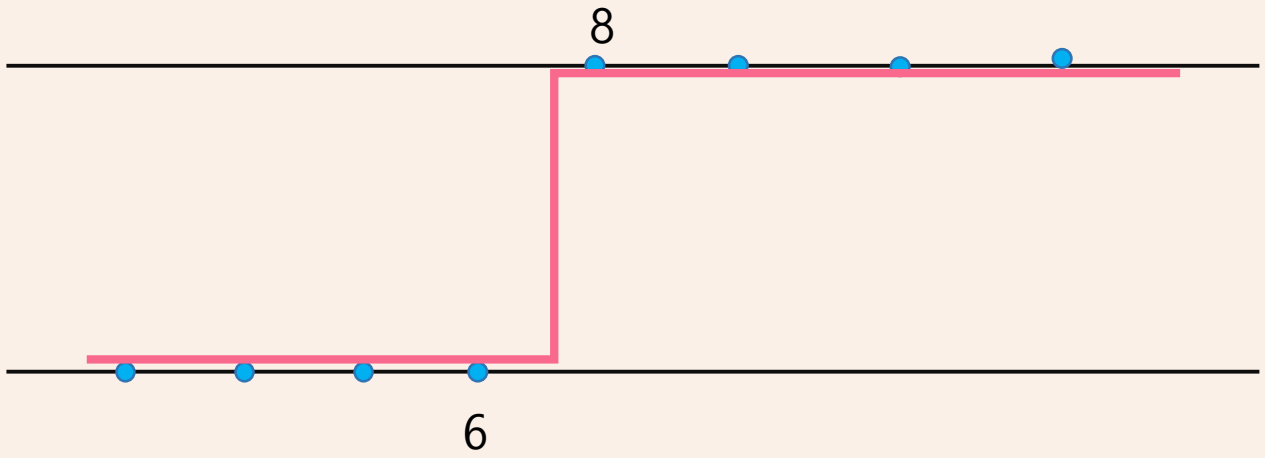
Loss값이 감소하며
최적해 a, b를 찾아나가는
과정을 볼 수 있다.

어느 정도에서 멈출까?



현 상황에서의 가장 이상적인 모델

어느 정도에서 멈출까?



$$h(x) = \frac{1}{1 + e^{(-ax+b)}}$$

기울기와 편중값

$$-ax + b$$

↑ y = 0에 가까워짐

↓ y = 1/2에 가까워짐

$$-a = large\ num$$

$$7 \approx \frac{b}{a}$$

```
In [7]: learning_rate = 0.4
        gradient_descent = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
```

```
# 학습
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(120001):
        sess.run(gradient_decent)
        if i % 6000 == 0:
            print("Epoch: %.f, loss = %.4f, 기울기 a = %.4f, 바이어스 b = %.4f" % (i, sess.
```

```
WARNING:tensorflow:From C:₩Users₩dm705₩Anaconda3₩envs₩tutorial₩lib₩site-packages₩tensorflow₩
e_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future
Instructions for updating:
Colocations handled automatically by placer.
Epoch: 0, loss = 0.7199, 기울기 a = 0.0210, 바이어스 b = -0.4736
Epoch: 6000, loss = 0.0186, 기울기 a = -2.7171, 바이어스 b = 18.8678
Epoch: 12000, loss = 0.0100, 기울기 a = -3.3477, 바이어스 b = 23.2877
Epoch: 18000, loss = 0.0068, 기울기 a = -3.7359, 바이어스 b = 26.0065
Epoch: 24000, loss = 0.0052, 기울기 a = -4.0165, 바이어스 b = 27.9716
Epoch: 30000, loss = 0.0041, 기울기 a = -4.2361, 바이어스 b = 29.5098
Epoch: 36000, loss = 0.0035, 기울기 a = -4.4166, 바이어스 b = 30.7733
Epoch: 42000, loss = 0.0030, 기울기 a = -4.5696, 바이어스 b = 31.8451
Epoch: 48000, loss = 0.0026, 기울기 a = -4.7025, 바이어스 b = 32.7756
Epoch: 54000, loss = 0.0023, 기울기 a = -4.8200, 바이어스 b = 33.5977
Epoch: 60000, loss = 0.0021, 기울기 a = -4.9251, 바이어스 b = 34.3339
Epoch: 66000, loss = 0.0019, 기울기 a = -5.0203, 바이어스 b = 35.0004
Epoch: 72000, loss = 0.0017, 기울기 a = -5.1073, 바이어스 b = 35.6093
Epoch: 78000, loss = 0.0016, 기울기 a = -5.1873, 바이어스 b = 36.1697
Epoch: 84000, loss = nan, 기울기 a = nan, 바이어스 b = nan
Epoch: 90000, loss = nan, 기울기 a = nan, 바이어스 b = nan
Epoch: 96000, loss = nan, 기울기 a = nan, 바이어스 b = nan
```

Nan ➔ 숫자가 아님
Inf ➔ 무한대를 의미

Nan ➔ Not a Number의 약자로, 0/0을 의미

Inf ➔ 무한대를 의미

```
In [5]: d = float('NaN')

In [7]: e = float('NaN')

In [9]: d == e
Out[9]: False

In [10]: f = d

In [11]: f == d
Out[11]: False
```

이유를 모르겠습니다.

다중 로지스틱 회귀

```
# 실행할 때마다 같은 결과를 출력하기 위한 seed 값 설정
seed = 0
np.random.seed(seed)
tf.set_random_seed(seed)

# x,y의 데이터 값
x_data = np.array([[2, 3],[4, 3],[6, 4],[8, 6],[10, 7],[12, 8],[14, 9]])
y_data = np.array([0, 0, 0, 1, 1, 1,1]).reshape(7, 1)

# 입력 값을 플레이스 홀더에 저장
X = tf.placeholder(tf.float64, shape=[None, 2])
Y = tf.placeholder(tf.float64, shape=[None, 1])

# 기울기 a와 bias b의 값을 임의로 정함.
a = tf.Variable(tf.random_uniform([2,1], dtype=tf.float64)) # [2,1] 의미: 들어오는 값은 2개, 나가는 값은 1개
b = tf.Variable(tf.random_uniform([1], dtype=tf.float64))
```

x_data [n개]
y_data
random a, b

Placeholder(저장 그릇)

X[0] : 공부 시간
X[1] : 과외 횟수
Y : 시험 합격 여부

```
# y 시그모이드 함수의 방정식을 세움
y = tf.sigmoid(tf.matmul(X, a) + b)

# 오차를 구하는 함수
loss = -tf.reduce_mean(Y * tf.log(y) + (1 - Y) * tf.log(1 - y))

# 학습률 값
learning_rate=0.1
```

시그모이드 함수 (예측값)
&
오차함수

```python
# 오차를 최소로 하는 값 찾기
gradient_descent = tf.train.GradintDescentOptimzer(learning_rate).minimize(loss)

predicted = tf.cast(y > 0.5, dtype = tf.float64)
accuracy = tf.reduce_mean (tf.cast(tf.equal(predicted, Y), dtype = tf.float64))

# 학습
with tf.Session as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(3001):
        a_, b_, loss_, _ = sess.run([a, b, loss, gradient_descent], feed_dict = {X: x_data, Y: y_data})

        if i + 1 % 300 == 0:
            print("step: %d, a1 = %.4f, a2 = %.4f, b = %.4f, loss = %.4f" %(i+1, a_[0], a_[1], b_, loss_))
```

```
step: 300, a1 = 0.8252, a2 = -0.5844, b = -2.3398, loss = 0.2725
step: 600, a1 = 0.8287, a2 = -0.3137, b = -3.8323, loss = 0.1945
step: 900, a1 = 0.7396, a2 = 0.0149, b = -4.9088, loss = 0.1517
step: 1200, a1 = 0.6360, a2 = 0.3228, b = -5.7589, loss = 0.1240
step: 1500, a1 = 0.5369, a2 = 0.5975, b = -6.4630, loss = 0.1045
step: 1800, a1 = 0.4470, a2 = 0.8398, b = -7.0645, loss = 0.0902
step: 2100, a1 = 0.3672, a2 = 1.0537, b = -7.5896, loss = 0.0793
step: 2400, a1 = 0.2966, a2 = 1.2435, b = -8.0558, loss = 0.0706
step: 2700, a1 = 0.2340, a2 = 1.4130, b = -8.4750, loss = 0.0637
step: 3000, a1 = 0.1783, a2 = 1.5654, b = -8.8558, loss = 0.0580
```

Loss값이 감소하며
최적해 a1, a2, b를
찾아가는 과정을 볼 수 있다!

```python
# 학습
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(3001):
        a_, b_, loss_, _ = sess.run([a, b, loss, gradient_descent], feed_dict = {X: x_data, Y: y_data})

        if (i + 1) % 300 == 0:
            print("step: %d, a1 = %.4f, a2 = %.4f, b = %.4f, loss = %.4f" %(i+1, a_[0], a_[1], b_, loss_))

    ## 어떻게 활용하는가
    new_x = np.array([7, 6.]).reshape(1,2) ## [공부한 시간, 과외 수업 횟수]
    new_y = sess.run(y, feed_dict={X: new_x})

    print("공부 시간: %d, 개인 과외 수: %d" % (new_x[:,0], new_x[:,1]))
    print("합격 가능성: %6.2f %%" % (new_y*100))
```

①

학습단계

활용단계

```
step: 300, a1 = 0.8252, a2 = -0.5844, b = -2.3398, loss = 0.2725
step: 600, a1 = 0.8287, a2 = -0.3137, b = -3.8323, loss = 0.1945
step: 900, a1 = 0.7396, a2 = 0.0149, b = -4.9088, loss = 0.1517
step: 1200, a1 = 0.6360, a2 = 0.3228, b = -5.7589, loss = 0.1240
step: 1500, a1 = 0.5369, a2 = 0.5975, b = -6.4630, loss = 0.1045
step: 1800, a1 = 0.4470, a2 = 0.8398, b = -7.0645, loss = 0.0902
step: 2100, a1 = 0.3672, a2 = 1.0537, b = -7.5896, loss = 0.0793
step: 2400, a1 = 0.2966, a2 = 1.2435, b = -8.0558, loss = 0.0706
step: 2700, a1 = 0.2340, a2 = 1.4130, b = -8.4750, loss = 0.0637
step: 3000, a1 = 0.1783, a2 = 1.5654, b = -8.8558, loss = 0.0580
```

```
공부 시간: 7, 개인 과외 수: 6
합격 가능성:  85.64 %
```

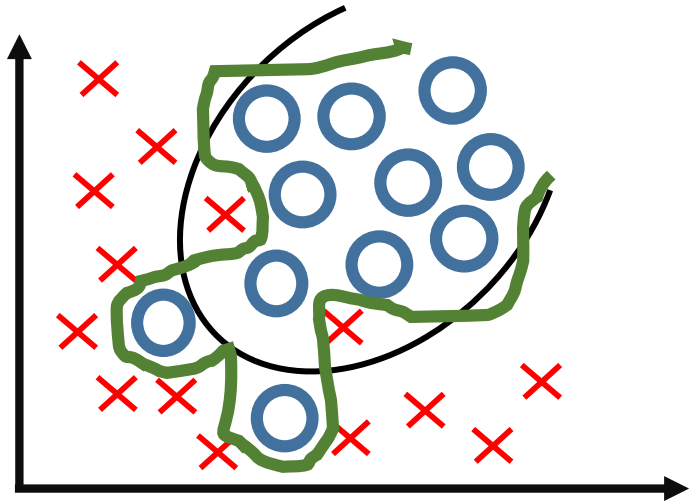
②

```
RuntimeError: Attempted to use a closed Session.
```

얼마나 진행해야 할까?

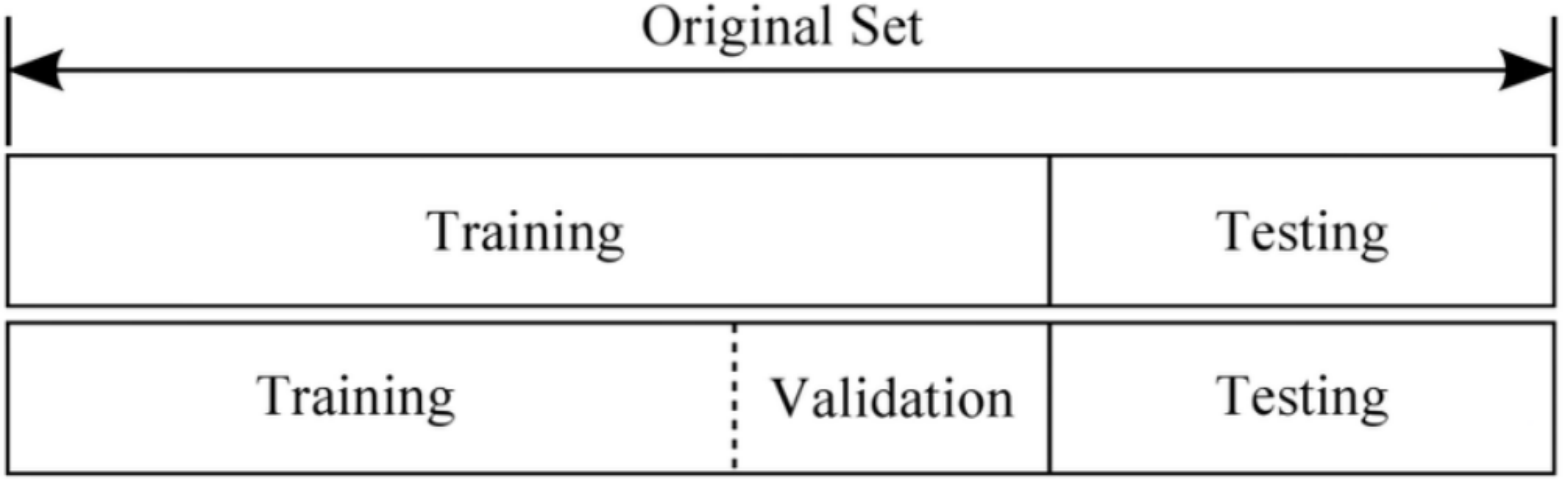
```
step: 175800, a1 = -1.3840, a2 = 7.0383, b = -25.3884, loss = 0.0011
step: 176100, a1 = -1.3845, a2 = 7.0406, b = -25.3957, loss = 0.0011
step: 176400, a1 = -1.3851, a2 = 7.0428, b = -25.4029, loss = 0.0011
step: 176700, a1 = -1.3857, a2 = 7.0451, b = -25.4101, loss = 0.0011
step: 177000, a1 = -1.3862, a2 = 7.0473, b = -25.4173, loss = 0.0011
step: 177300, a1 = -1.3868, a2 = 7.0495, b = -25.4245, loss = 0.0011
step: 177600, a1 = -1.3874, a2 = 7.0517, b = -25.4317, loss = 0.0011
step: 177900, a1 = -1.3879, a2 = 7.0540, b = -25.4388, loss = 0.0011
step: 178200, a1 = -1.3885, a2 = 7.0562, b = -25.4460, loss = 0.0011
step: 178500, a1 = -1.3891, a2 = 7.0584, b = -25.4531, loss = 0.0011
step: 178800, a1 = -1.3896, a2 = 7.0606, b = -25.4602, loss = 0.0011
step: 179100, a1 = -1.3902, a2 = 7.0628, b = -25.4674, loss = 0.0011
step: 179400, a1 = -1.3908, a2 = 7.0650, b = -25.4745, loss = 0.0011
step: 179700, a1 = -1.3913, a2 = 7.0672, b = -25.4816, loss = 0.0011
step: 180000, a1 = -1.3919, a2 = 7.0694, b = -25.4886, loss = 0.0011

공부 시간: 7, 개인 과외 수: 6
합격 가능성:  99.92 %
```

→ **overfitting**



```
step: 3000, a1 = 0.1783, a2 = 1.5654, b = -8.8558, loss = 0.0580

공부 시간: 7, 개인 과외 수: 6
합격 가능성:  85.64 %
```

K-Fold Cross Validation

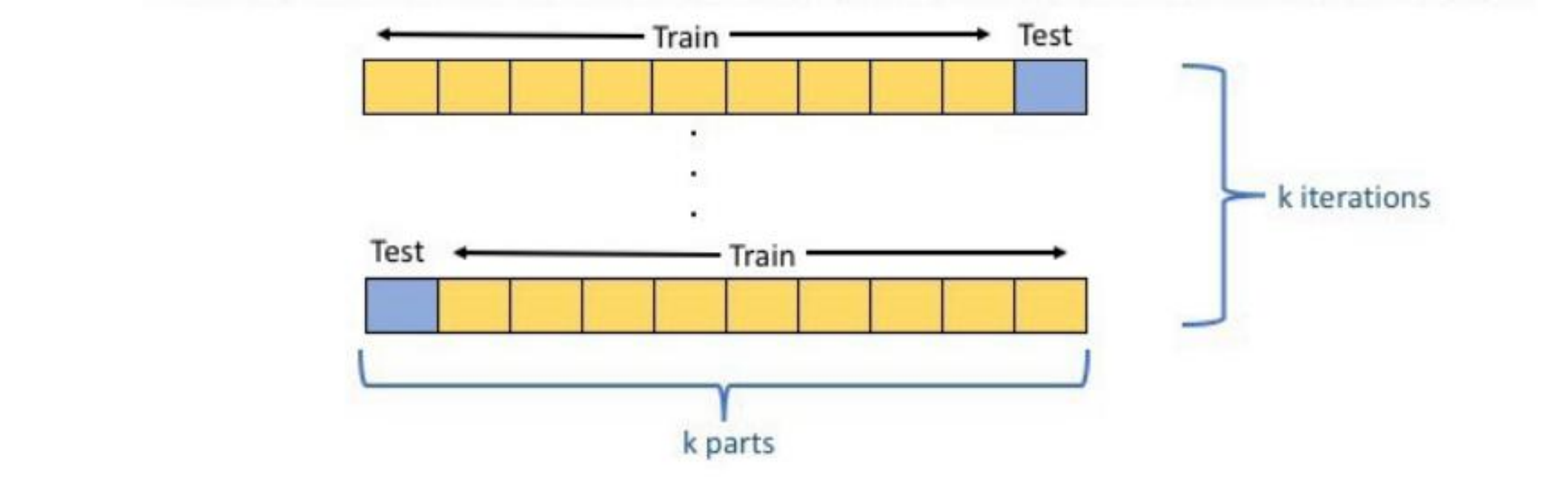


machine learning - What is the difference between test set and ...
https://stats.stackexchange.com/.../what-is-the-difference-between-... ▾ 이 페이지 번역하기
답변 11개
2016. 4. 7. - **In the** first part you just look at your models and select the best .... The concept of '

Validation Set: 모델을 개선하기 위해 사용 (여러 모델 중 최종 모델 선정에 기여)

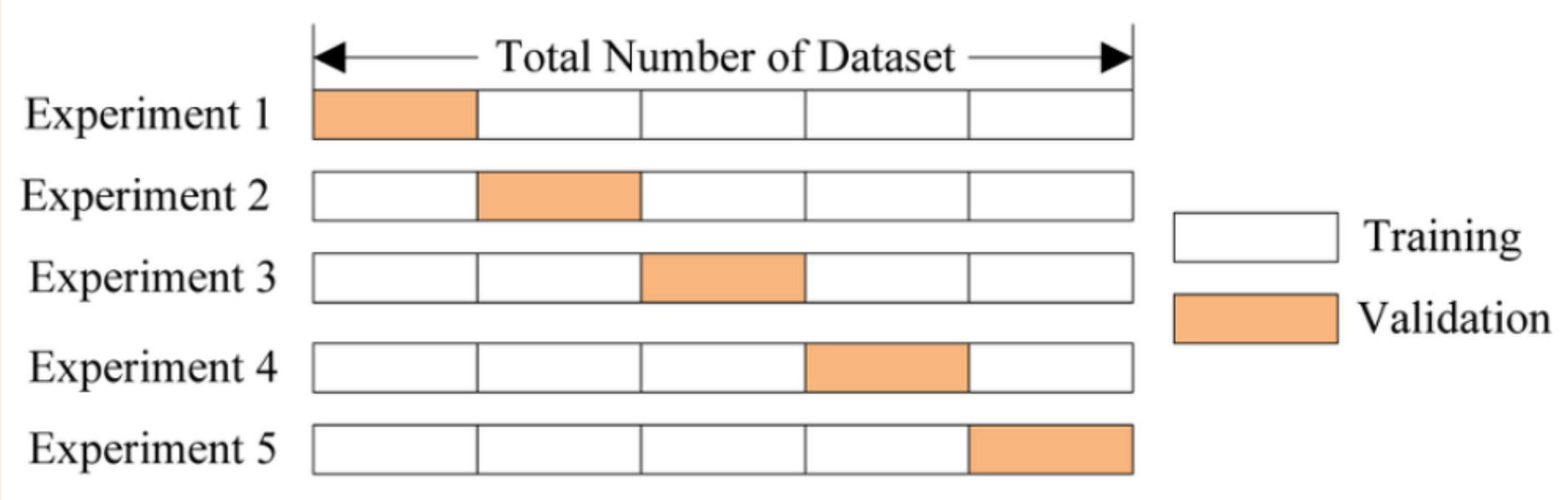
Test Set:모델 Training 이후에 최종적으로 모델의 성능을 평가하기 위해 사용

K-Fold Cross Validation



1. 총 데이터 개수가 적은 데이터셋에 대하여 정확도 향상
2. 기존에 Training / Validation / Test 세 개의 집단으로 나누는 것보다 Training, Test 로만 분류할 때 학습 데이터 셋이 더 많기 때문
3. 데이터 수가 적은데 검증과 테스트에 데이터를 더 뺏기면 underfitting 등 성능이 미달되는 모델이 학습됨

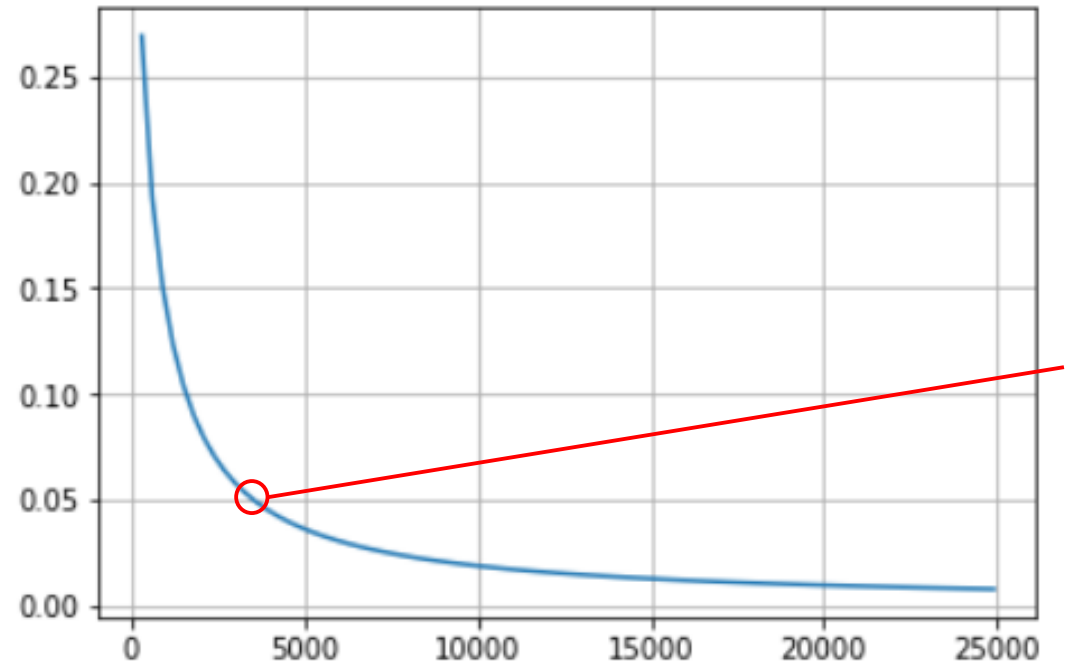
K-Fold Cross Validation



1. Training Data를 K개로 쪼갠다. (Labelling 된 것들을 사용)
2. K-1개를 Training set, 1개를 Test set으로 정의한다.
3. 2번의 작업을 K번 반복한다.
4. 예상 performance metrics(분류성능 척도)를 출력한다.
   ➔ ex) 평균 제곱근 오차, 신뢰 구간, Error rate 등

Elbow Method

```
In [21]: plt.plot(step_list, loss_list)
         plt.grid()
         plt.show()
```



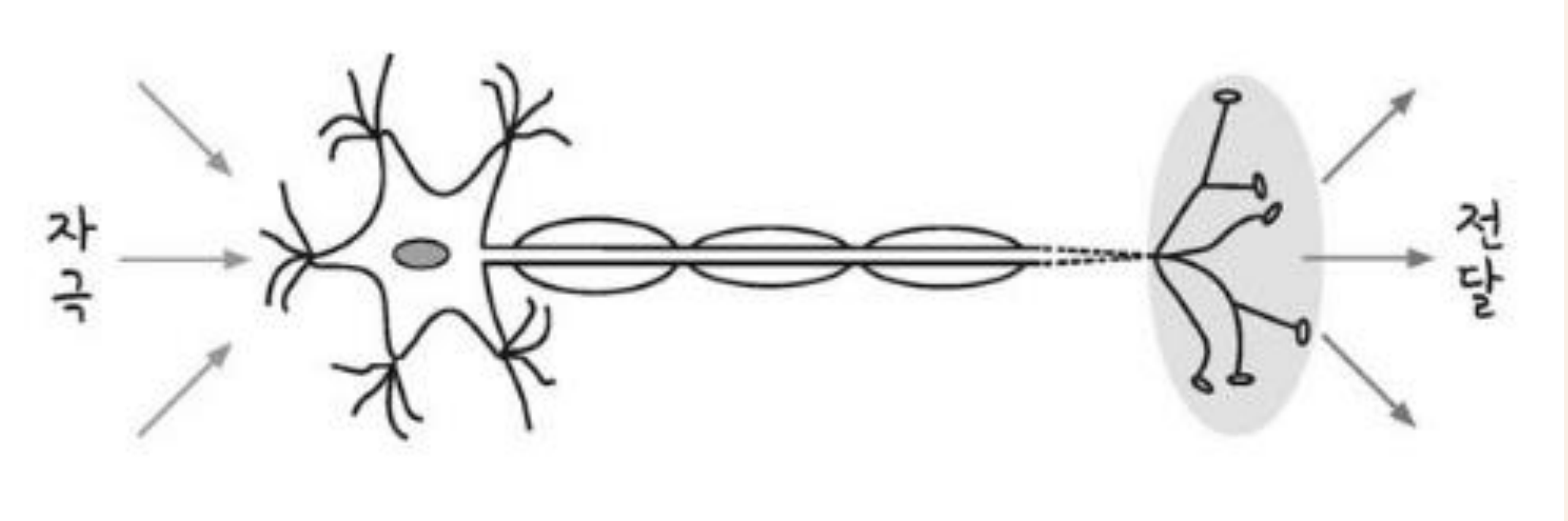
가장 효율적인 step

(약 3,000번)

Step에 따른 loss의 감소정도

6장) 퍼셉트론

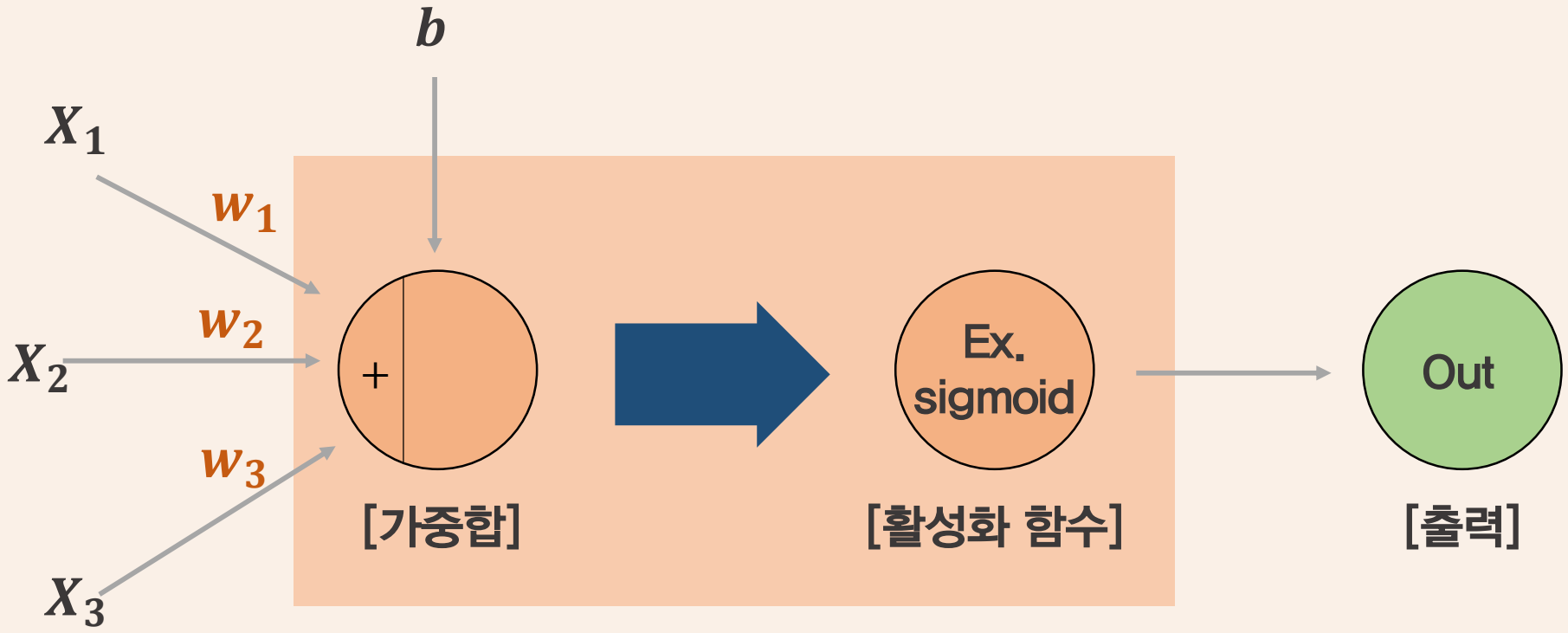
뉴런의 정보전달



$$y = a_1x_1 + a_2x_2 + b$$

6장) 퍼셉트론

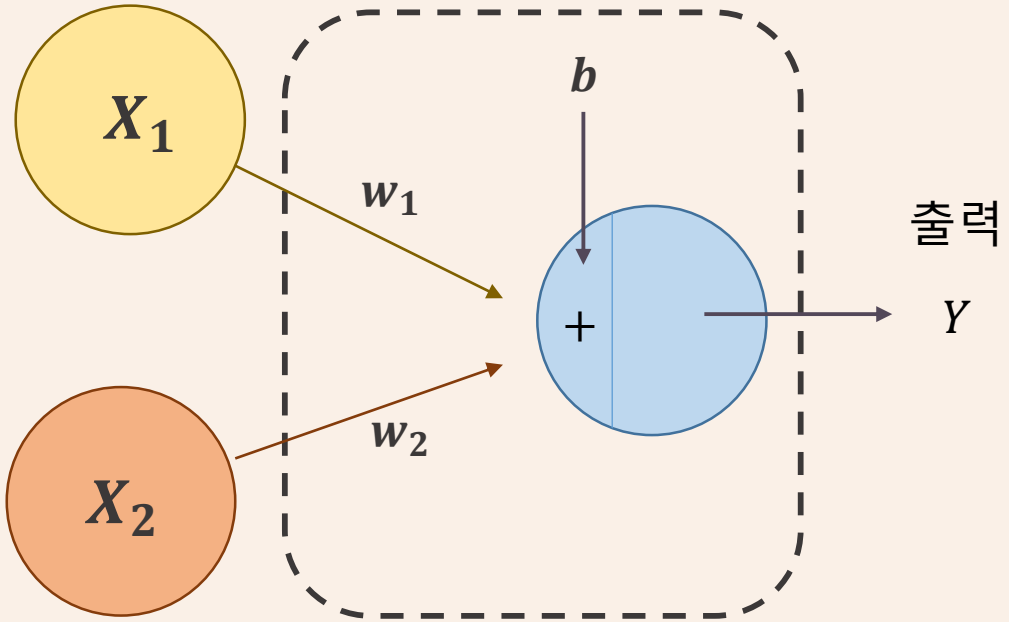
뉴런의 정보전달



$$y = w_1x_1 + w_2x_2 + b$$

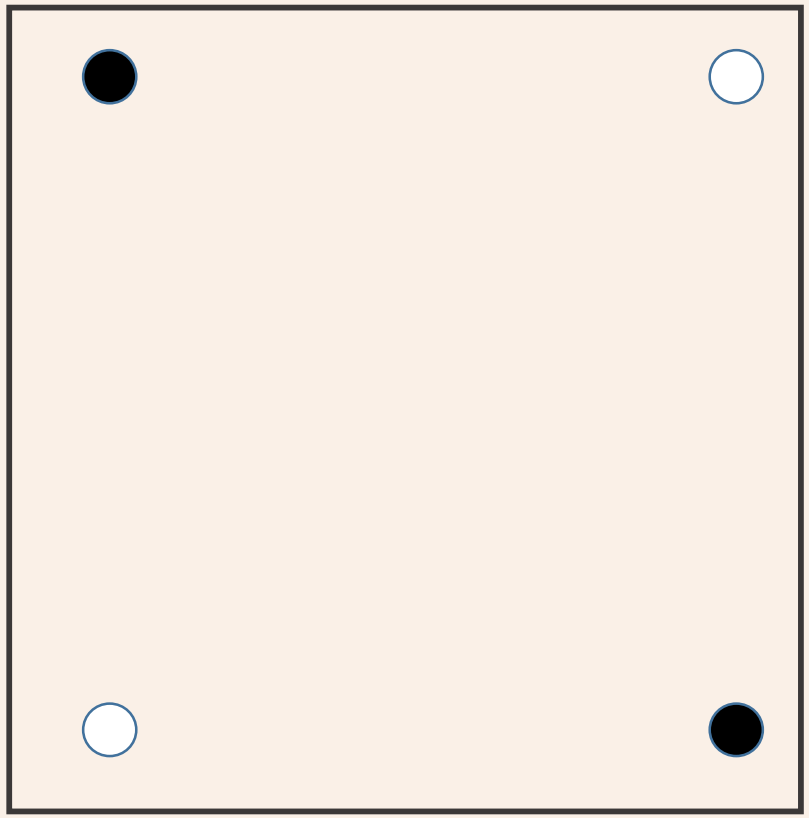
로지스틱 = 퍼셉트론

$$y = w_1x_1 + w_2x_2 + b$$

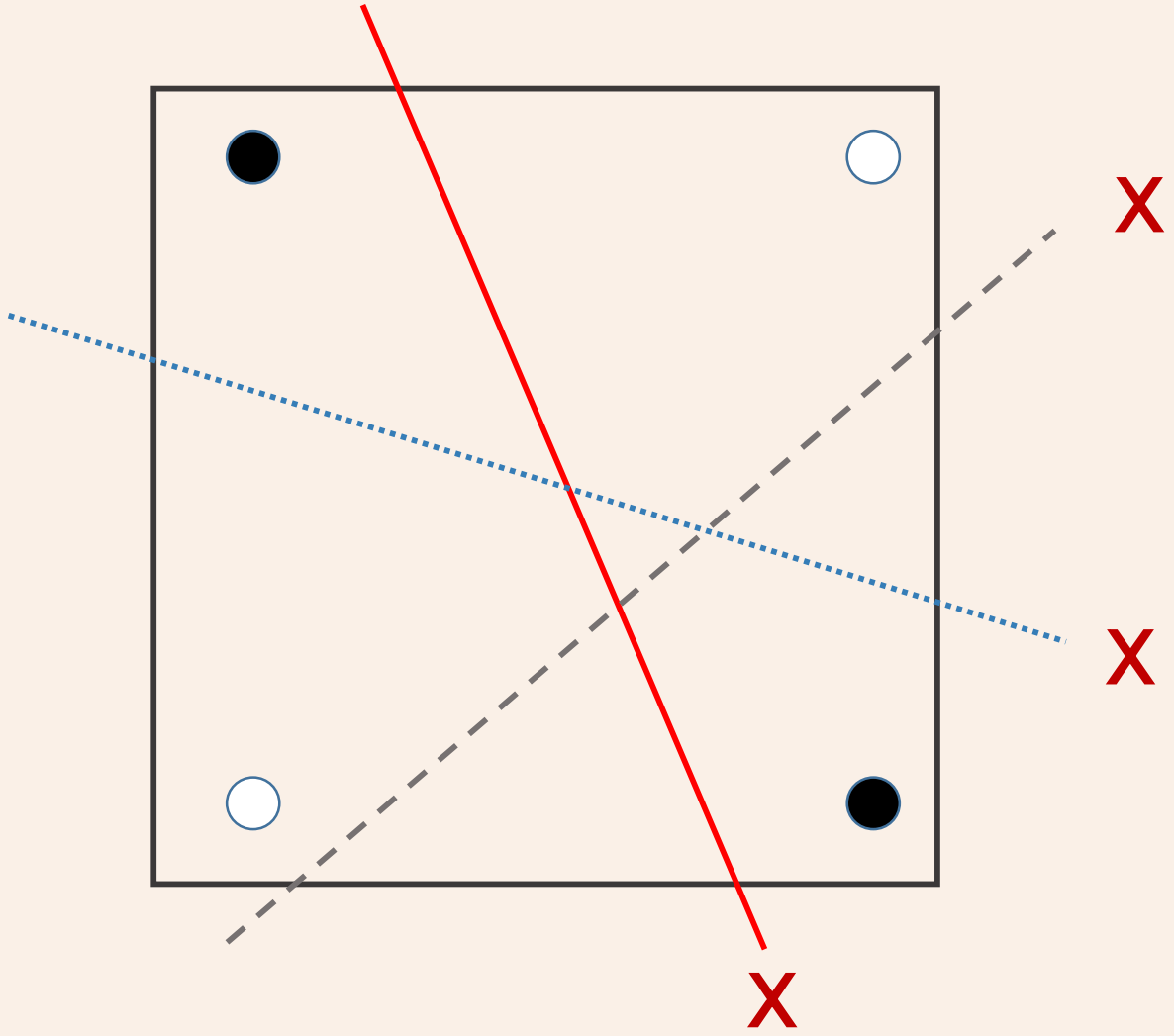


인공신경망의 기본

직선으로 검은 점 / 흰 점 나누기

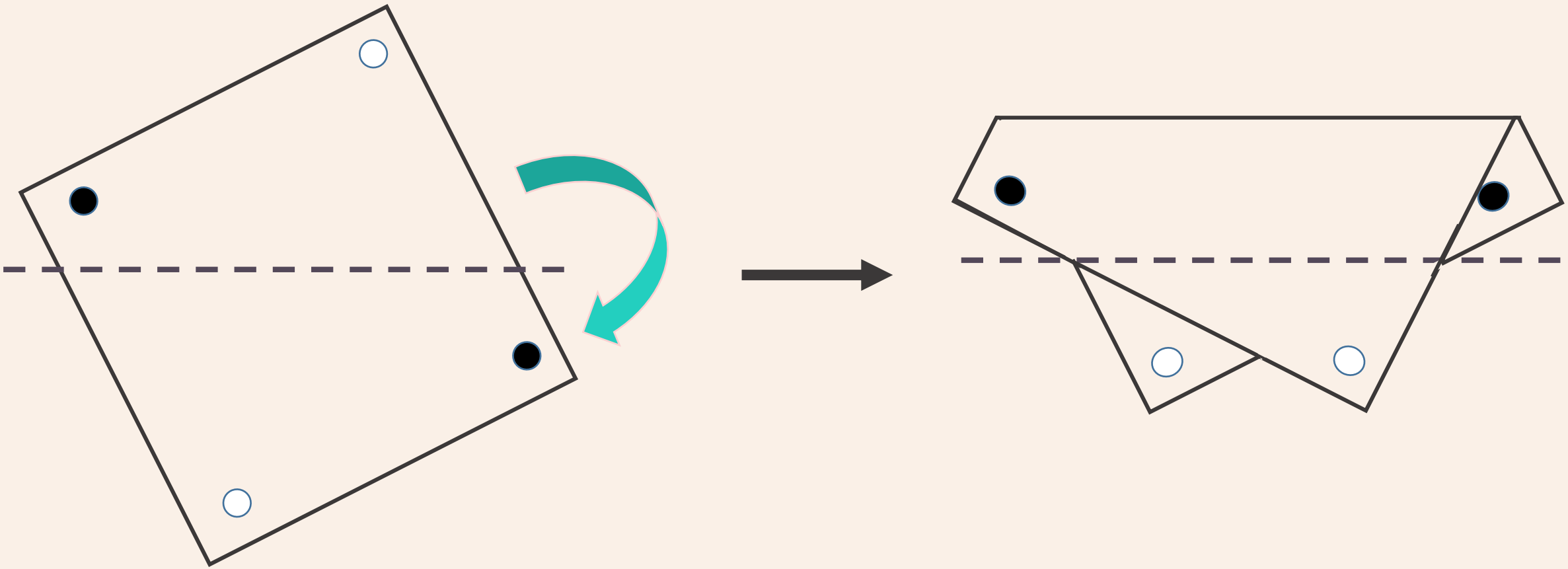


직선으로 검은 점 / 흰 점 나누기



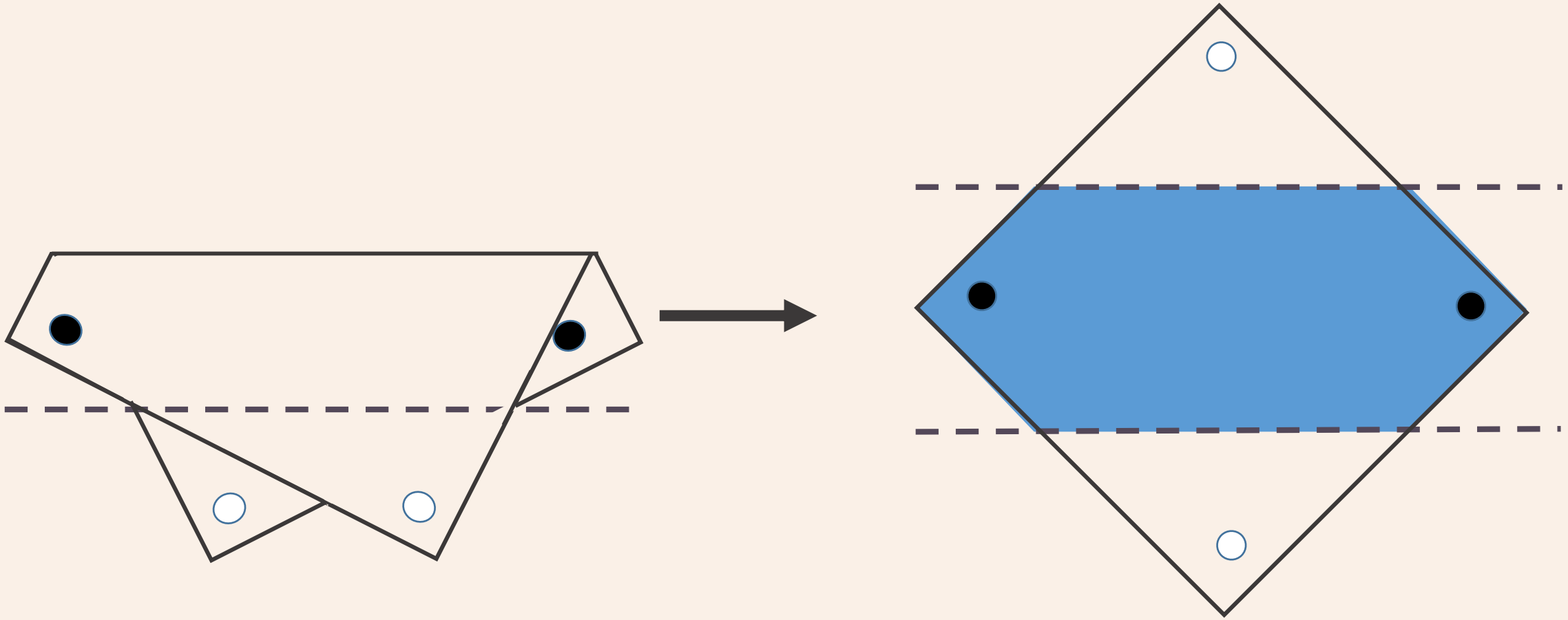
퍼셉트론의 한계 – XOR

7장) 다층 퍼셉트론 – XOR 문제 해결



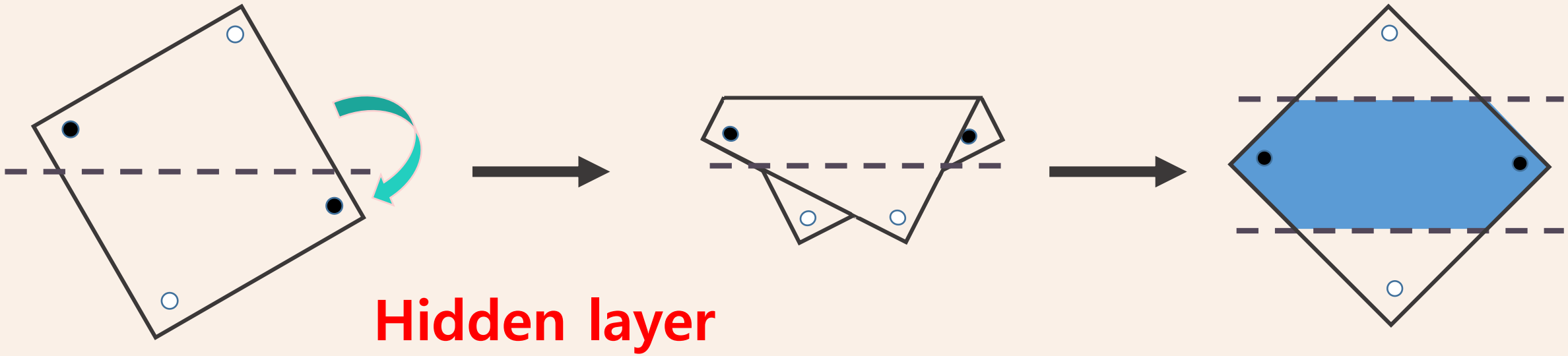
평면(공간)을 비틀고, 뒤집어서 문제를 해결

7장) 다층 퍼셉트론



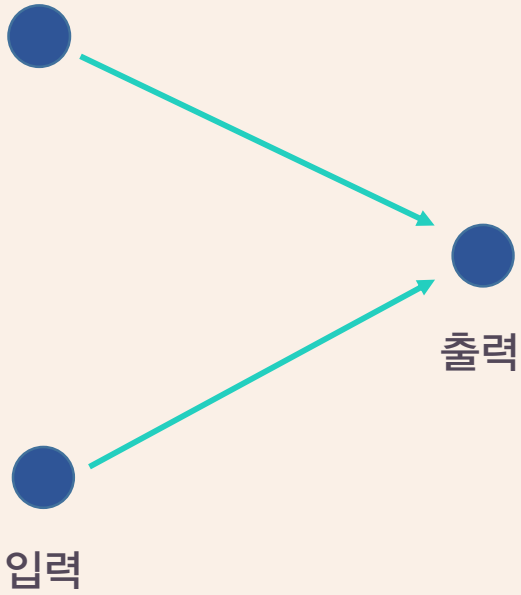
평면(공간)을 비틀고, 뒤집어서 문제를 해결

7장) 다층 퍼셉트론

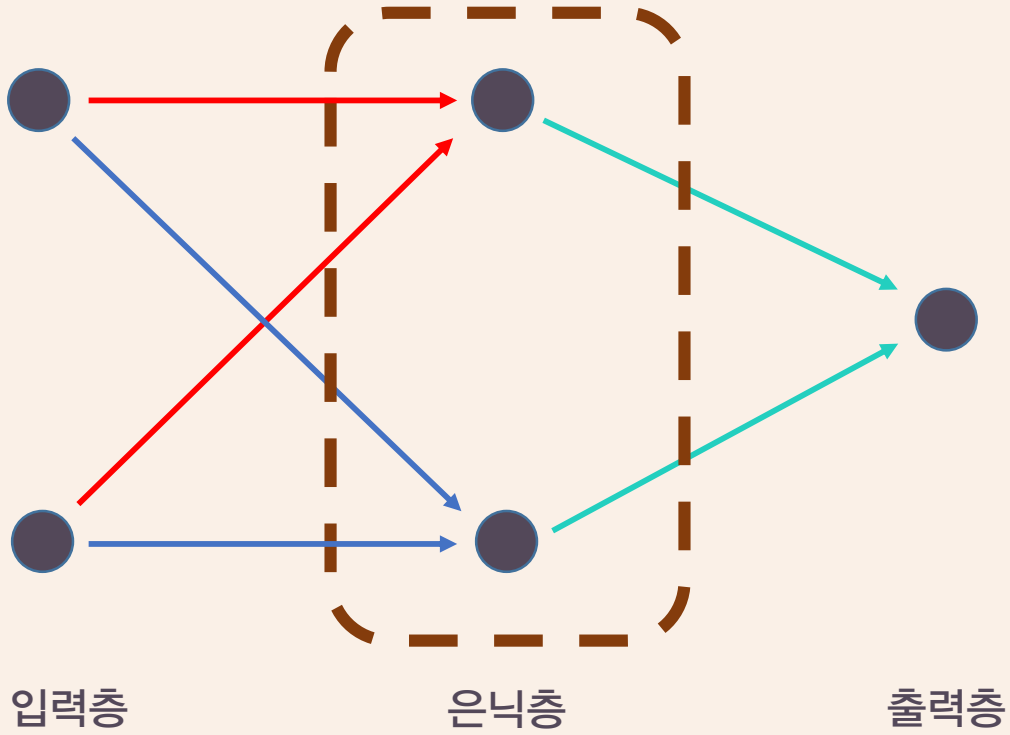


두 개의 퍼셉트론(다층 퍼셉트론)을 한 번에 계산

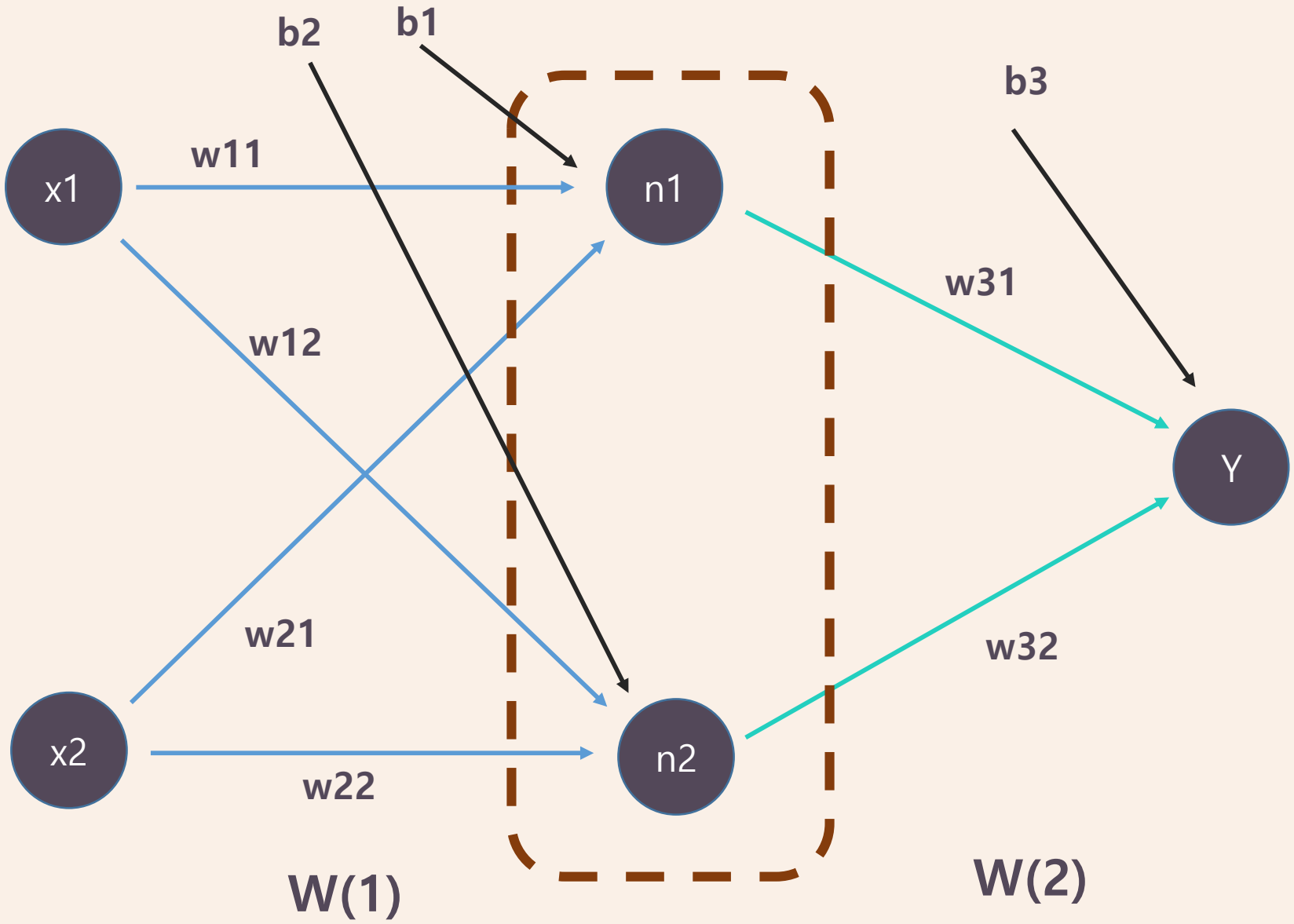
퍼셉트론



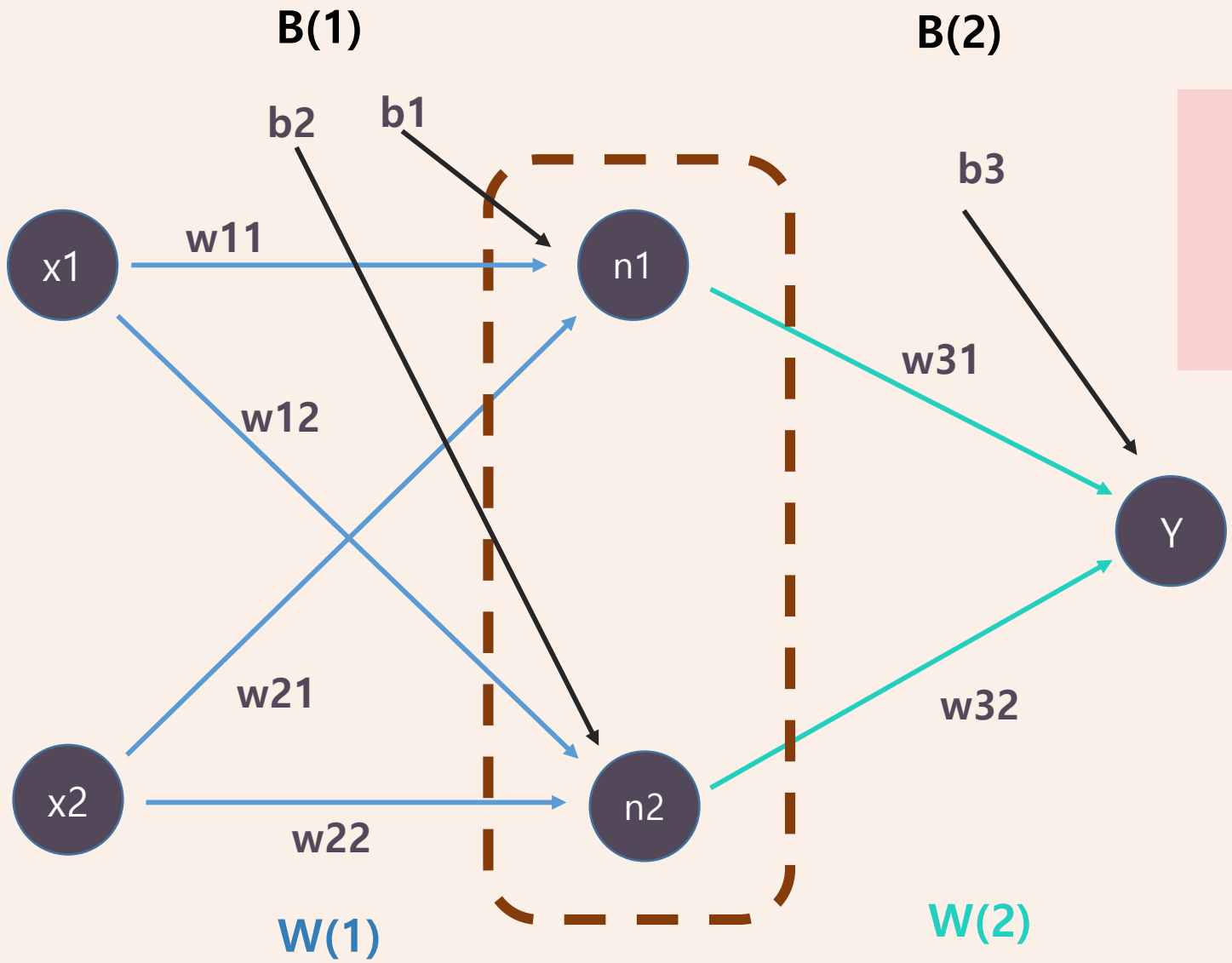
다층 퍼셉트론



다층 퍼셉트론



다층 퍼셉트론



$(\sigma: sigmoid)$

$$n_1 = \sigma(x_1 w_{11} + x_2 w_{21} + b_1)$$
$$n_2 = \sigma(x_1 w_{12} + x_2 w_{22} + b_2)$$

$$Y_{out} = \sigma(n_1 w_{31} + n_2 w_{32} + b_3)$$

$$W_{(1)} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \qquad B_{(1)} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$W_{(2)} = \begin{bmatrix} w_{31} \\ w_{32} \end{bmatrix} \qquad B_{(2)} = \begin{bmatrix} b_3 \end{bmatrix}$$

다층 퍼셉트론

$$W_{(1)} = \begin{matrix} n_1 & n_2 \\ \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \end{matrix} \qquad B_{(1)} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

$$W_{(2)} = \begin{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ Y_{out} \end{matrix} \qquad B_{(2)} = \begin{bmatrix} -1 \end{bmatrix}$$

$$n_1 = \sigma(x_1 w_{11} + x_2 w_{21} + b_1)$$
$$n_2 = \sigma(x_1 w_{12} + x_2 w_{22} + b_2)$$

$$Y_{out} = \sigma(n_1 w_{31} + n_2 w_{32} + b_3)$$

x1	x2	n1	n2	Y(out)	원하는 값
0	0	$\sigma(0*(-2)+0*(-2)+3) \approx 1$	$\sigma(0*2+0*2-1) \approx 0$	$\sigma(1*1+0*1-1) \approx 0$	0
0	1	$\sigma(0*(-2)+1*(-2)+3) \approx 1$	$\sigma(0*2+1*2-1) \approx 1$	$\sigma(1*1+1*1-1) \approx 1$	1
1	0	$\sigma(1*(-2)+0*(-2)+3) \approx 1$	$\sigma(1*2+0*2-1) \approx 1$	$\sigma(1*1+1*1-1) \approx 1$	1
1	1	$\sigma(1*(-2)+1*(-2)+3) \approx 0$	$\sigma(1*2+1*2-1) \approx 1$	$\sigma(0*1+0*2-1) \approx 0$	0

XOR

NAND OR AND

In [2]:
```
# 가중치와 바이어스
w11 = np.array([-2,-2])
w12 = np.array([2, 2])
w2 = np.array([1, 1])
b1 = 3
b2 = -1
b3 = -1
```

In [4]:
```
# 퍼셉트론
def MLP(x, w, b):
    y = np.sum(w * x) + b
    if y <= 0:
        return 0
    else:
        return 1
```

In [7]:
```
# NAND 게이트
def NAND(x1, x2) :
    return MLP(np.array([x1, x2]), w11, b1)

# OR 게이트
def OR(x1, x2) :
    return MLP(np.array([x1, x2]), w12, b2)

# AND 게이트
def AND(x1, x2) :
    return MLP(np.array([x1, x2]), w2, b3)

# AND 게이트
def XOR(x1, x2) :
    return AND(NAND(x1, x2), OR(x1, x2))
```

In [8]:
```
if __name__ == '__main__':
    for x in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = XOR(x[0], x[1])
        print("입력 값: " + str(x) + "출력 값: " + str(y))
```

```
입력 값: (0, 0)출력 값: 0
입력 값: (1, 0)출력 값: 1
입력 값: (0, 1)출력 값: 1
입력 값: (1, 1)출력 값: 0
```

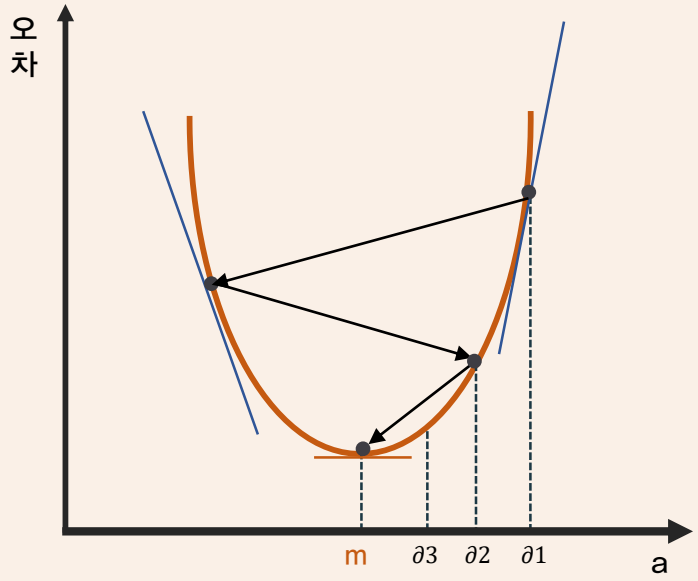
8장) 오차 역전파 (Back Propagation)

$$W_{(1)} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \qquad B_{(1)} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$W_{(2)} = \begin{bmatrix} w_{31} \\ w_{32} \end{bmatrix} \qquad B_{(2)} = \begin{bmatrix} b_3 \end{bmatrix}$$

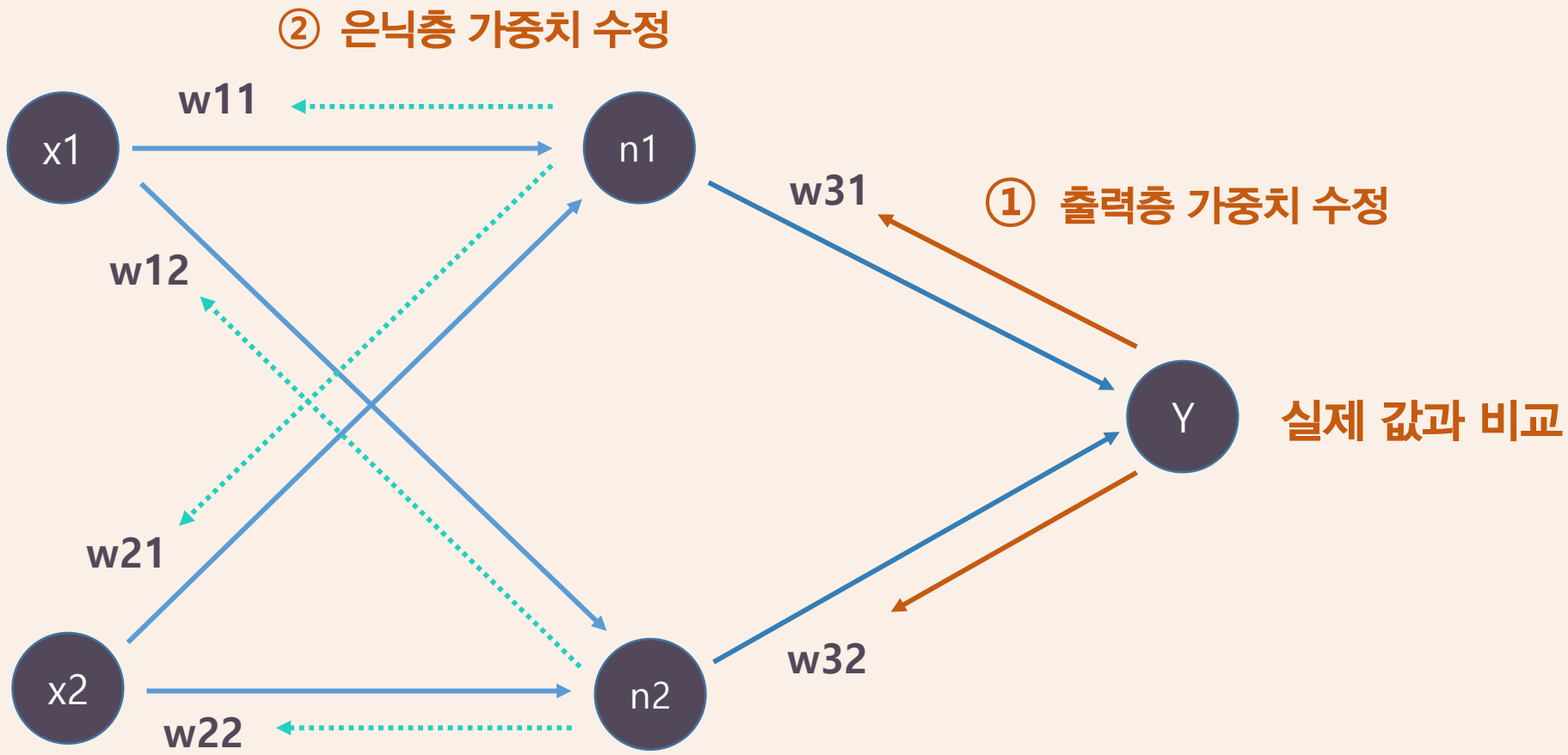
내부 가중치는 어떻게 구할까?

➔ 경사하강법



1. 임의의 가중치 w를 선언 ➔ 결과 확인
2. 계산 값과 원하는 값 사이의 오차 확인
3. 바로 앞 가중치를 오차가 작아지는 지점으로 이동
4. 오차가 최소가 되는 지점 ($\frac{d}{dx} f(x) = 0$)에서 멈춤

8장) 오차 역전파 (Back Propagation)



계속해서 오차를 줄여나간다

오차를 줄여나간다 == $\frac{d}{dx}f(x) = 0$에 가까워진다

➔ 오차(미분 값 = 기울기)가 0이 되는 방향으로 나아간다.

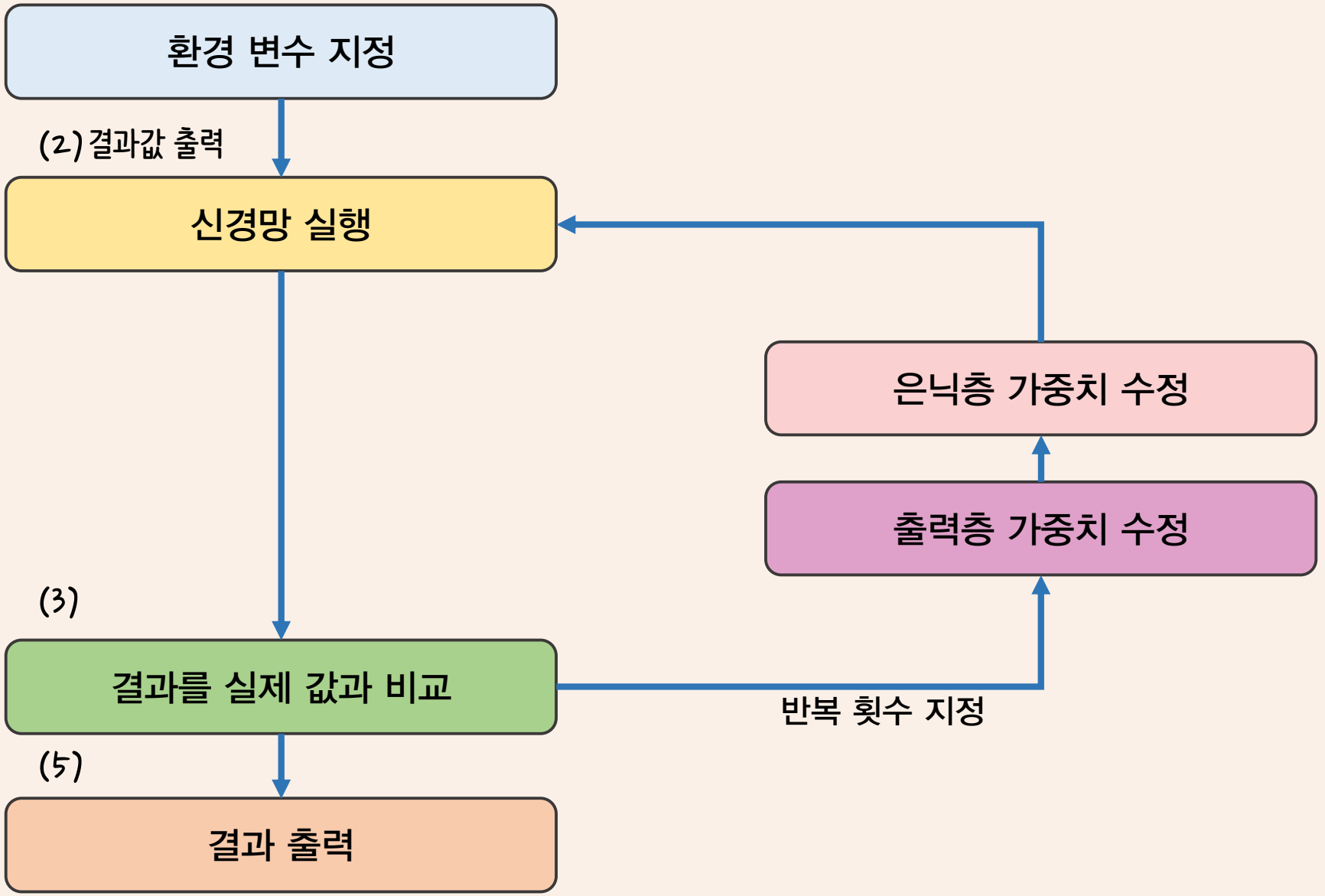
➔ 현 가중치에서 기울기를 뺐을 때, 가중치의 변화가 없는 상태

새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값

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

➔ 더 이상 가중치의 업데이트를 하지 않는 상태에서 종료

(1) 데이터셋(입력값, 타깃 결과값), 학습률, 활성화함수, 가중치 선언 등



다음시간에 계속..

Thank you