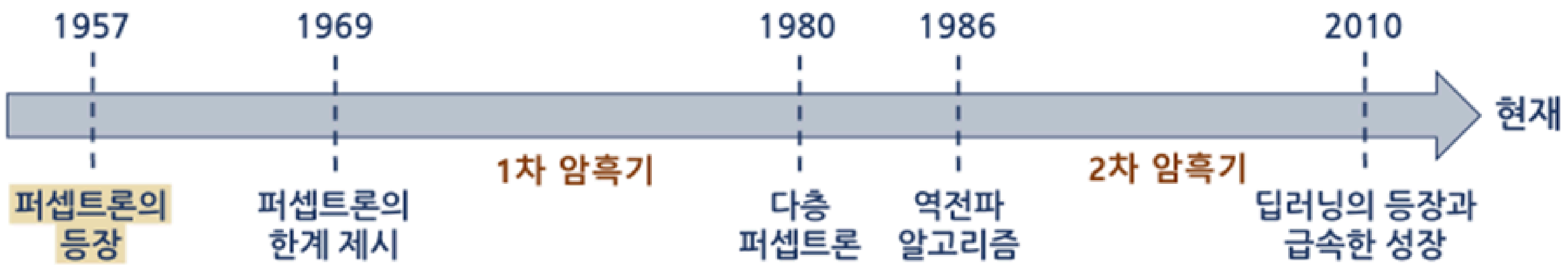
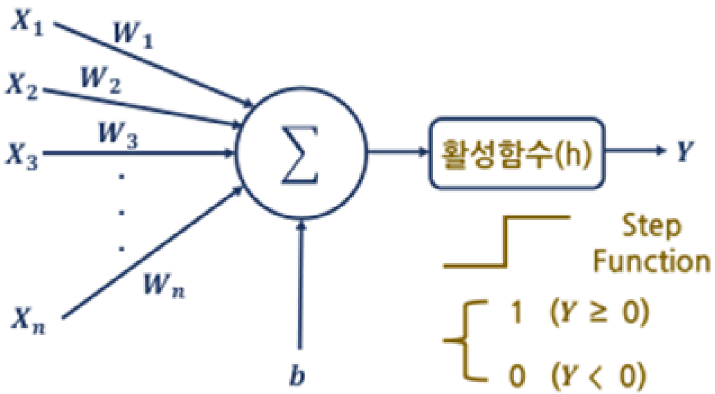
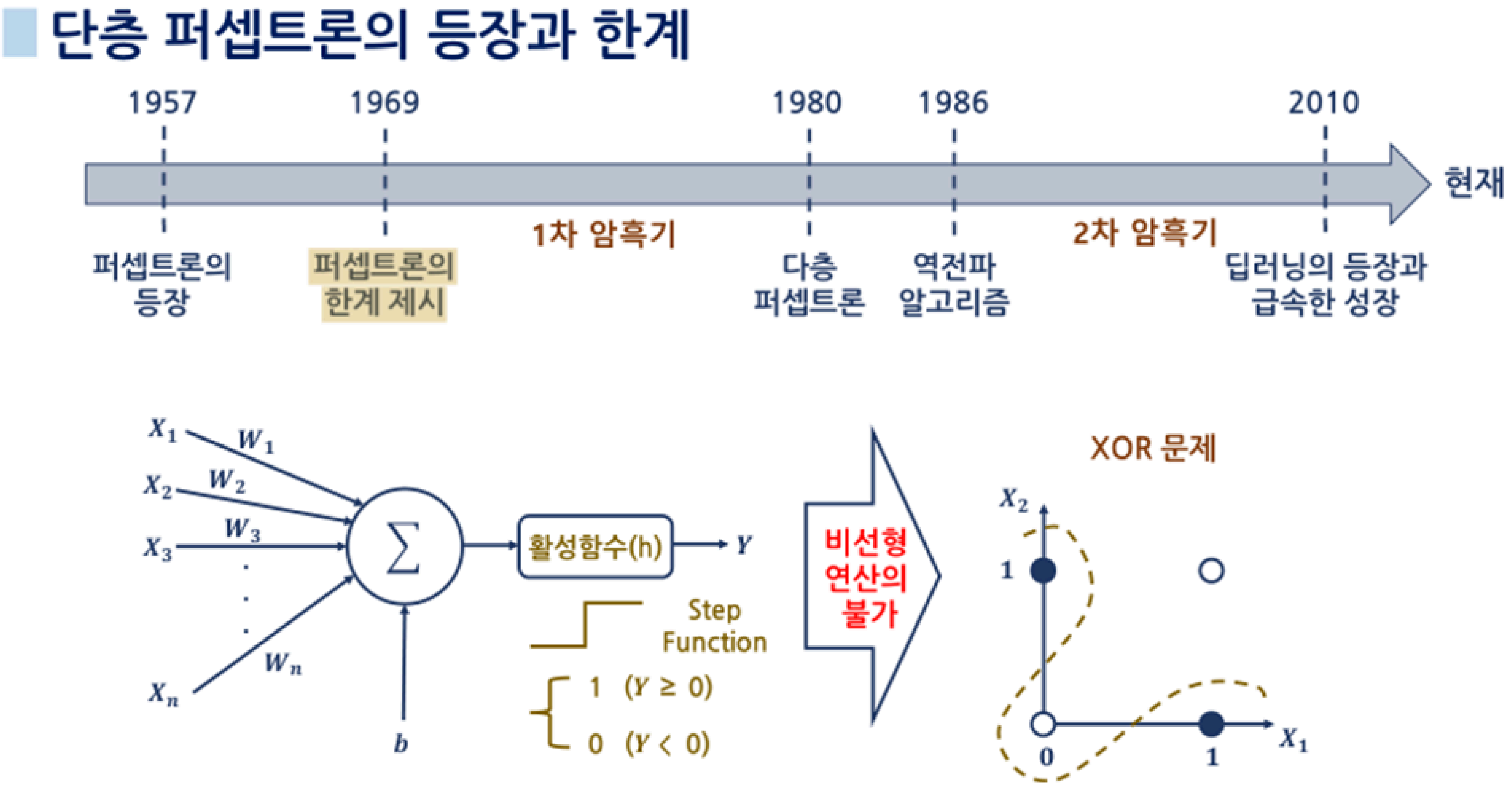
16강. 활성함수의 필요성과 오류역전파

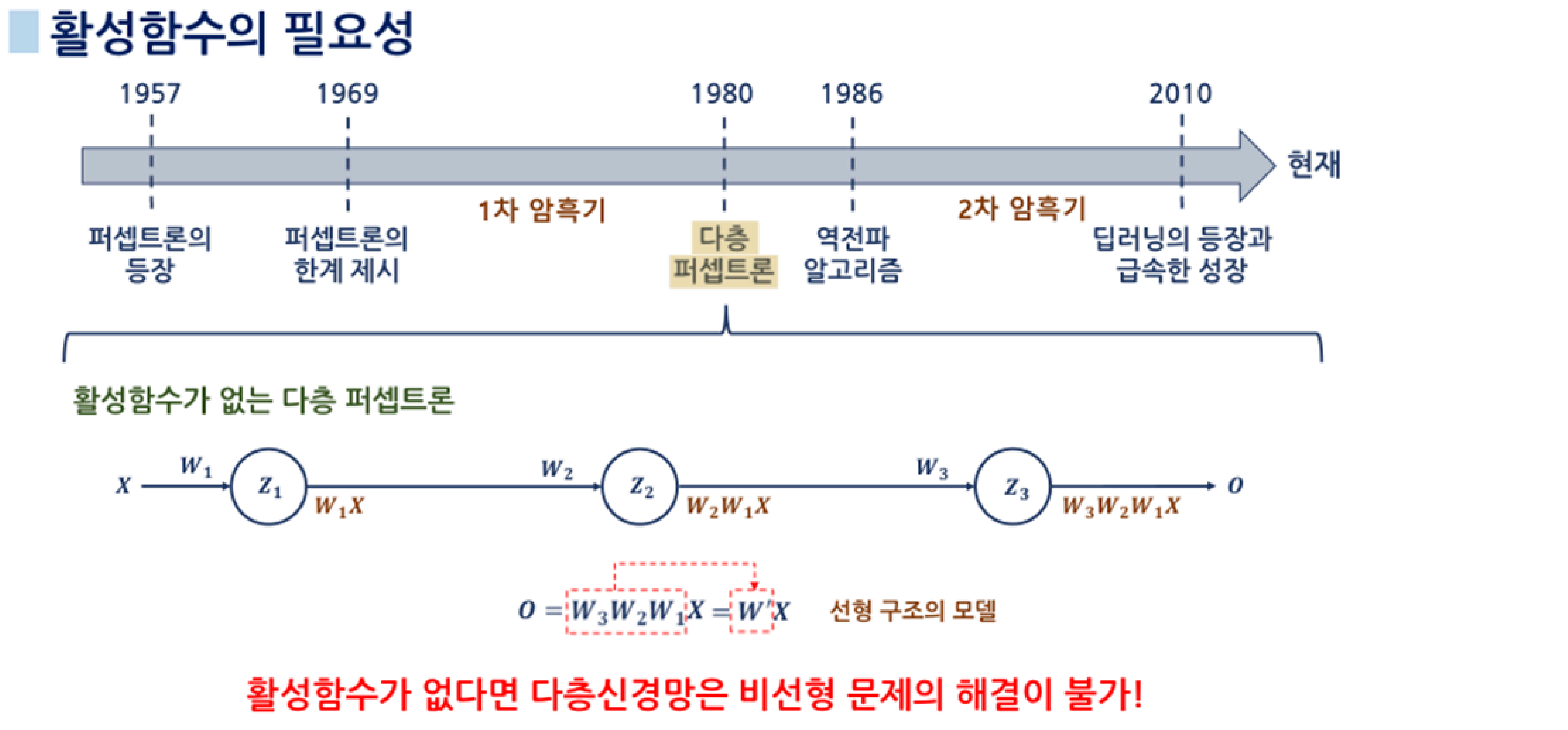
- 활성함수의 필요성
- 시그모이드 함수
- 오류역전파 알고리즘

단층 퍼셉트론의 등장과 한계

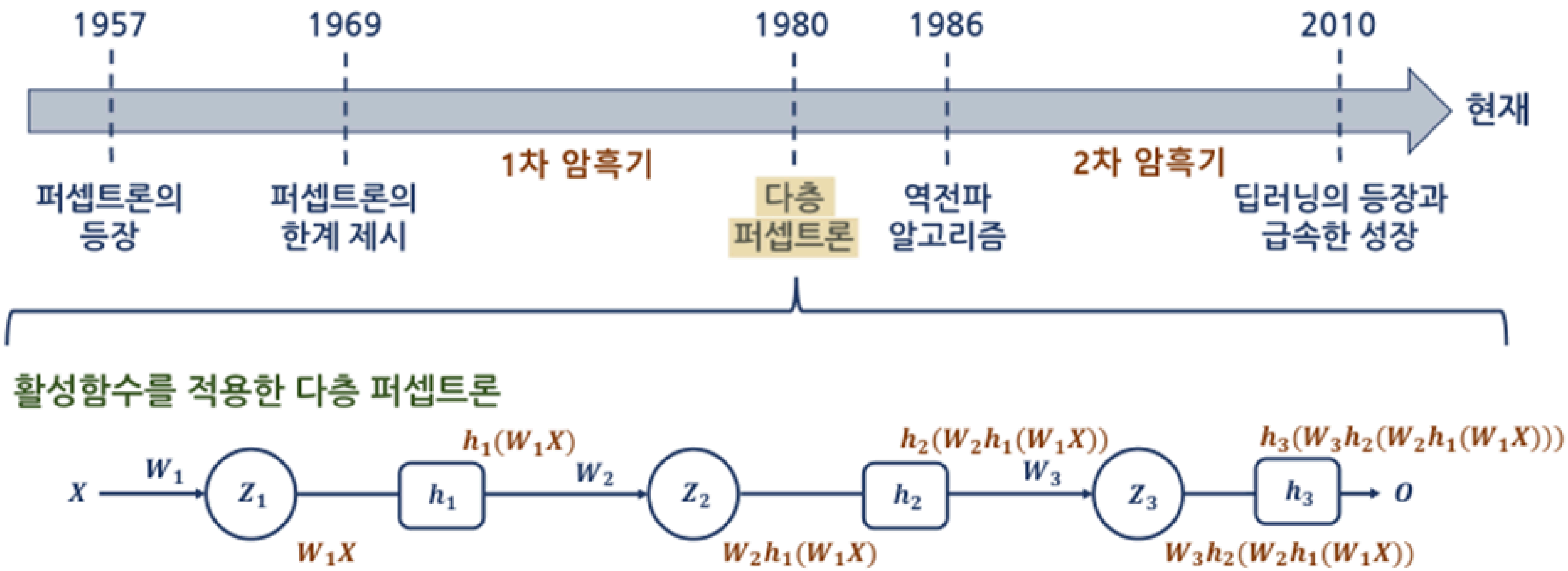





단층 퍼셉트론의 등장과 한계
1957
1969
1980
1986
2010
현재
퍼셉트론의 등장
퍼셉트론의 한계 제시
1차 암흑기
다층 퍼셉트론
역전파 알고리즘
2차 암흑기
딥러닝의 등장과 급속한 성장
X1
X2
X3
Xn
W1
W2
W3
Wn
b
Σ
활성함수(h)
Y
Step Function
1 (Y ≥ 0)
0 (Y < 0)
비선형 연산의 불가
XOR 문제
X2
X1
1
0
1

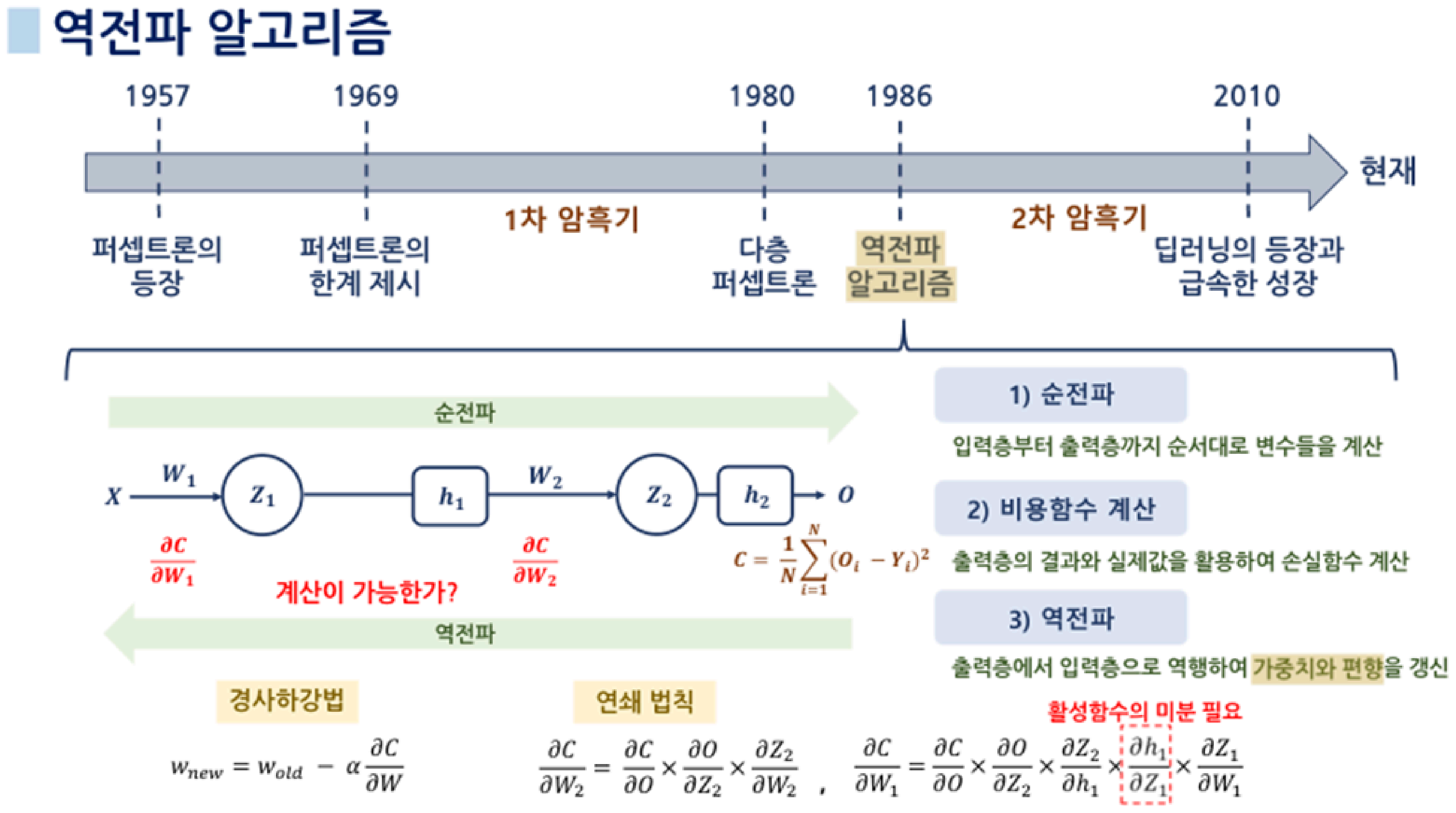

활성함수의 필요성
1957
1969
1980
1986
2010
현재
퍼셉트론의 등장
퍼셉트론의 한계 제시
1차 암흑기
다층 퍼셉트론
역전파 알고리즘
2차 암흑기
딥러닝의 등장과 급속한 성장
활성함수가 없는 다층 퍼셉트론
X
W_1
Z_1
W_1X
W_2
Z_2
W_2W_1X
W_3
Z_3
$W_3W_2W_1X$
O
$O = W_3W_2W_1X = W'X$
선형 구조의 모델
활성함수가 없다면 다층신경망은 비선형 문제의 해결이 불가!

활성함수의 필요성

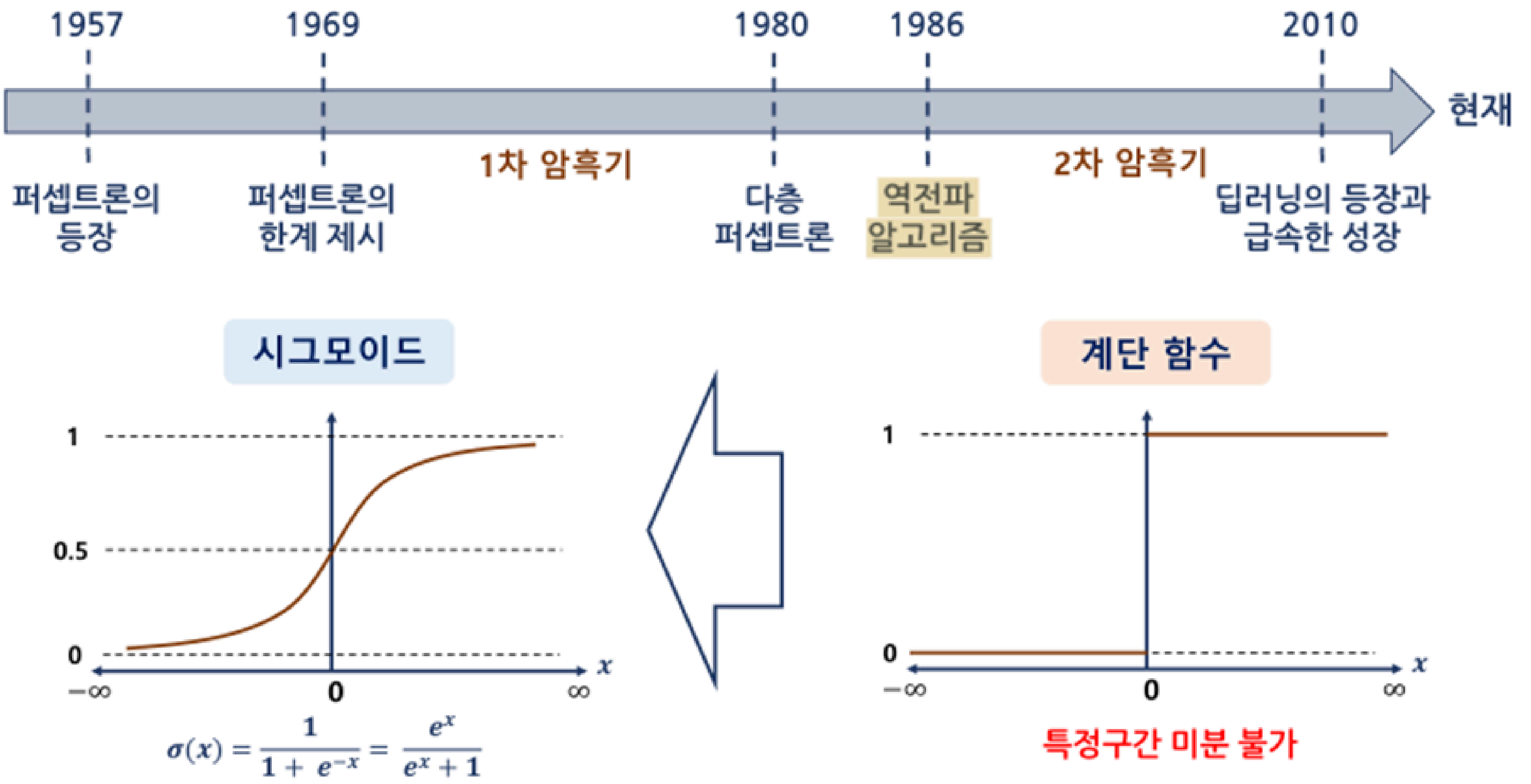


$$O = h_3(W_3\, h_2(W_2\, h_1(W_1 X)))$$

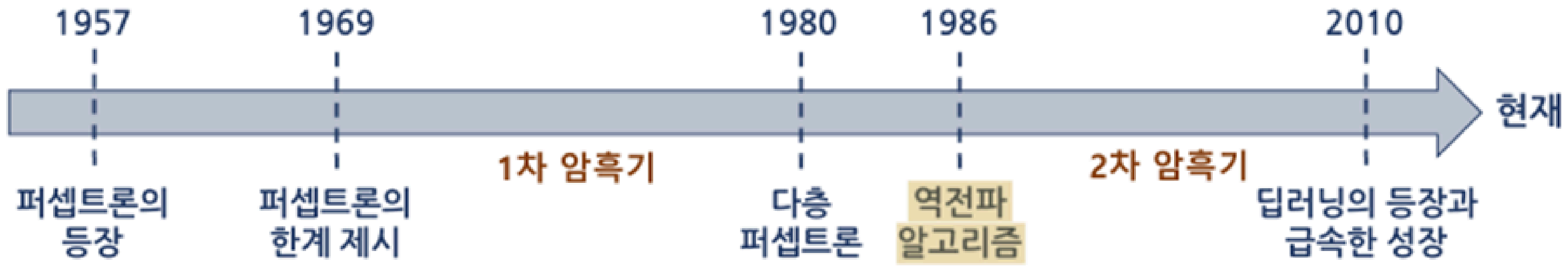
활성함수로 모델의 복잡도를 높여 비선형 문제를 해결


역전파 알고리즘
1957
1969
1980
1986
2010
현재
1차 암흑기
2차 암흑기
퍼셉트론의 등장
퍼셉트론의 한계 제시
다층 퍼셉트론
역전파 알고리즘
딥러닝의 등장과 급속한 성장
순전파
1) 순전파
입력층부터 출력층까지 순서대로 변수들을 계산
2) 비용함수 계산
출력층의 결과와 실제값을 활용하여 손실함수 계산
3) 역전파
출력층에서 입력층으로 역행하여 가중치와 편향을 갱신
계산이 가능한가?
역전파
경사하강법
연쇄 법칙
활성화함수의 미분 필요
$C = \frac{1}{N}\sum_{i=1}^{N}(O_i - Y_i)^2$
$w_{new} = w_{old} - \alpha\frac{\partial C}{\partial W}$
$\frac{\partial C}{\partial W_2} = \frac{\partial C}{\partial O} \times \frac{\partial O}{\partial Z_2} \times \frac{\partial Z_2}{\partial W_2}$, $\frac{\partial C}{\partial W_1} = \frac{\partial C}{\partial O} \times \frac{\partial O}{\partial Z_2} \times \frac{\partial Z_2}{\partial h_1} \times \frac{\partial h_1}{\partial Z_1} \times \frac{\partial Z_1}{\partial W_1}$

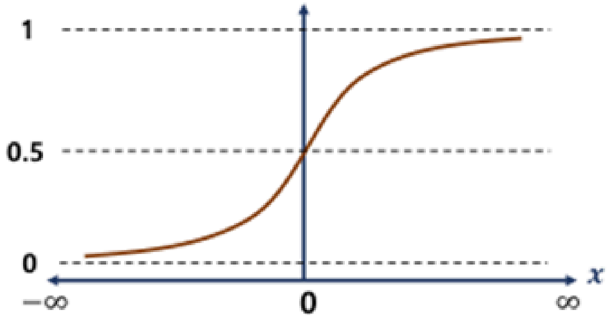
역전파 알고리즘


1957
1969
1980
1986
2010
현재
퍼셉트론의 등장
퍼셉트론의 한계 제시
1차 암흑기
다층 퍼셉트론
역전파 알고리즘
2차 암흑기
딥러닝의 등장과 급속한 성장
시그모이드
1
0.5
0
x
−∞
0
∞
$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$
계단 함수
1
0
x
−∞
0
∞
특정구간 미분 불가

역전파 알고리즘



시그모이드

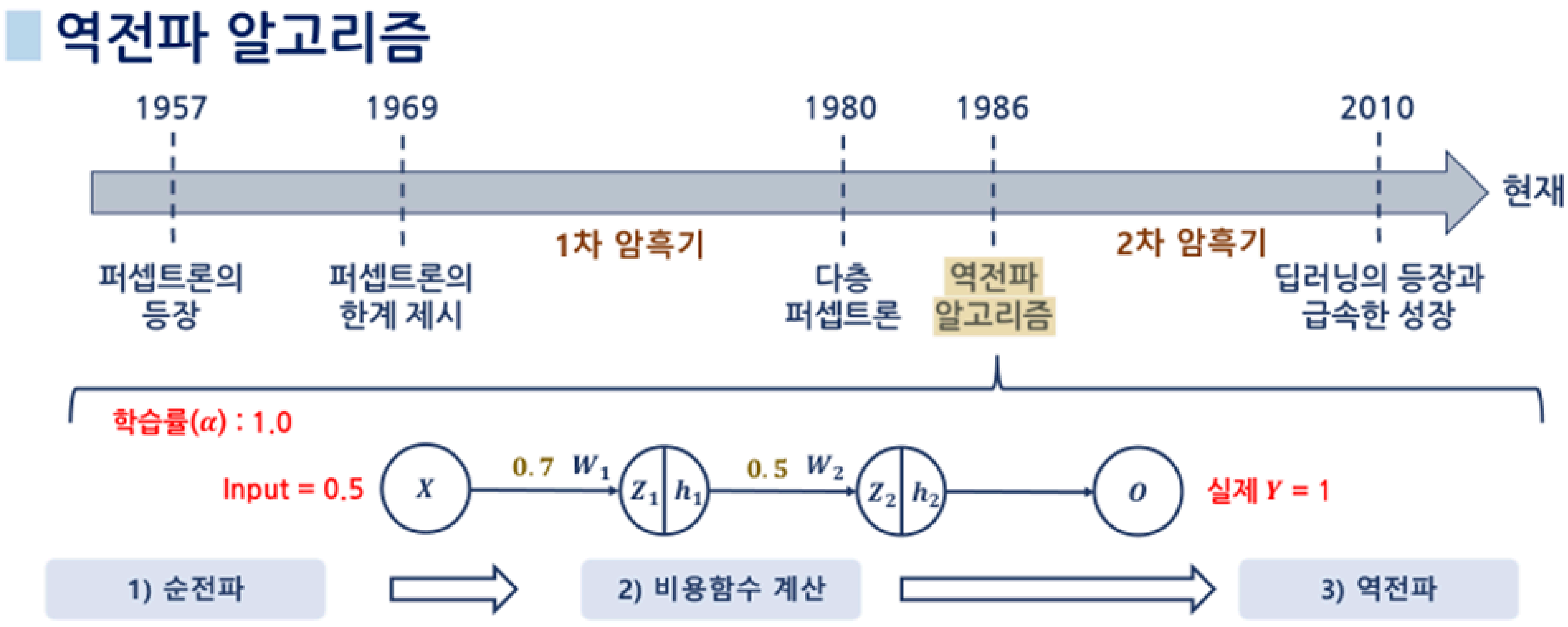


$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{e^{x} + 1}$$

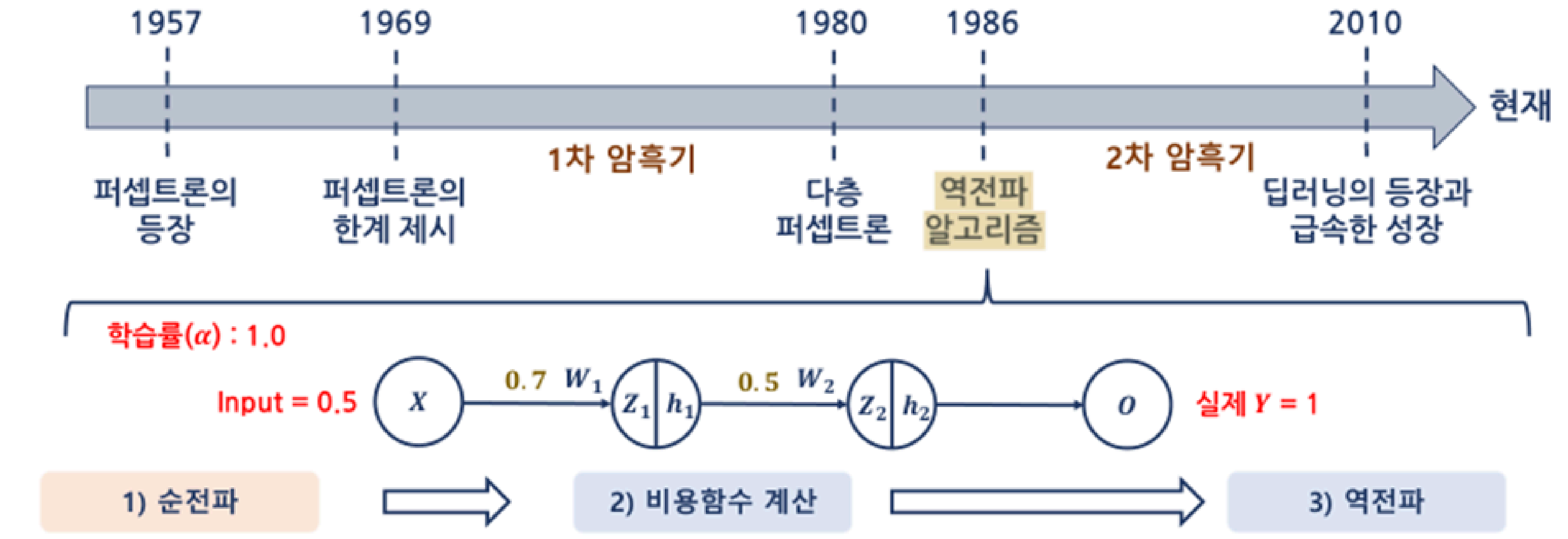
- 0 ~ 1 사이의 신호를 전달
- 전 구간에서의 미분이 가능
- 미분 계산이 단순(컴퓨팅 연산 효율)

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

시그모이드 함수를 활성함수로 사용

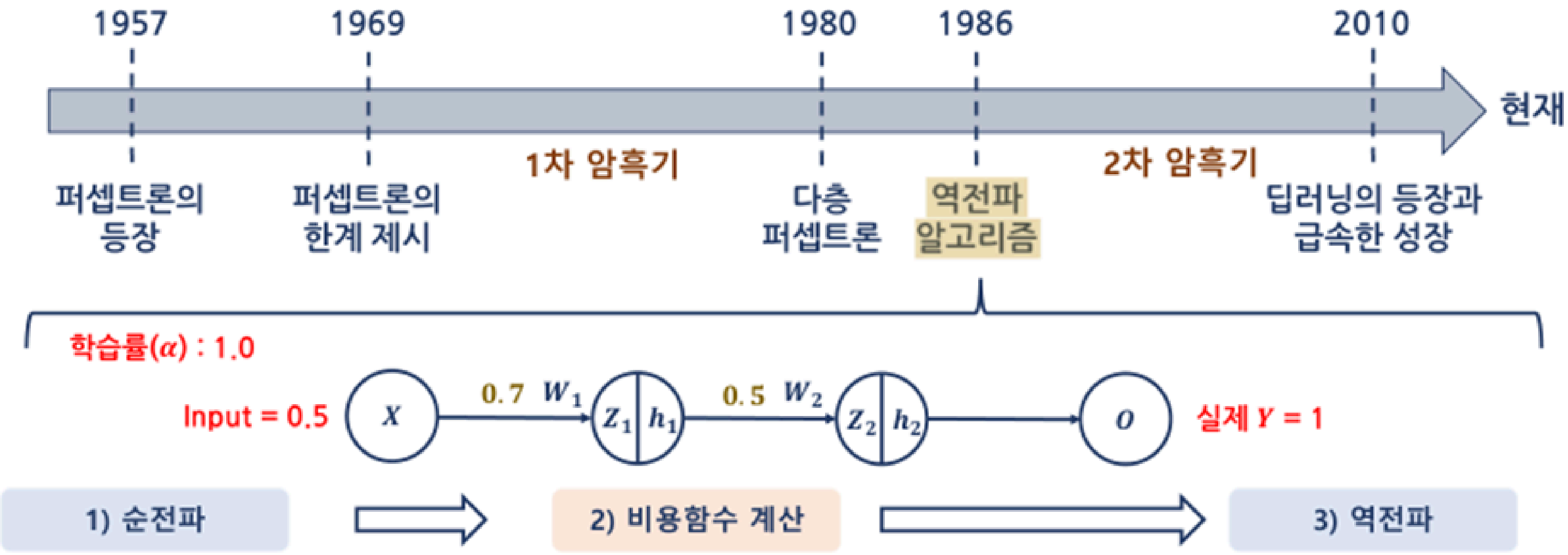

역전파 알고리즘
1957
1969
1980
1986
2010
현재
퍼셉트론의 등장
퍼셉트론의 한계 제시
1차 암흑기
다층 퍼셉트론
역전파 알고리즘
2차 암흑기
딥러닝의 등장과 급속한 성장
학습률(α) : 1.0
Input = 0.5
X
0.7 W1
Z1 h1
0.5 W2
Z2 h2
O
실제 Y = 1
1) 순전파
2) 비용함수 계산
3) 역전파

역전파 알고리즘



- $Z_1 = W_1X = 0.7 \times 0.5 = 0.35$
- $h_1 = \sigma(Z_1) = \sigma(0.35) = 0.587$
- $Z_2 = W_2h_1 = 0.5 \times 0.587 = 0.294$
- $O = h_2 = \sigma(Z_2) = \sigma(0.294) = 0.573$

역전파 알고리즘



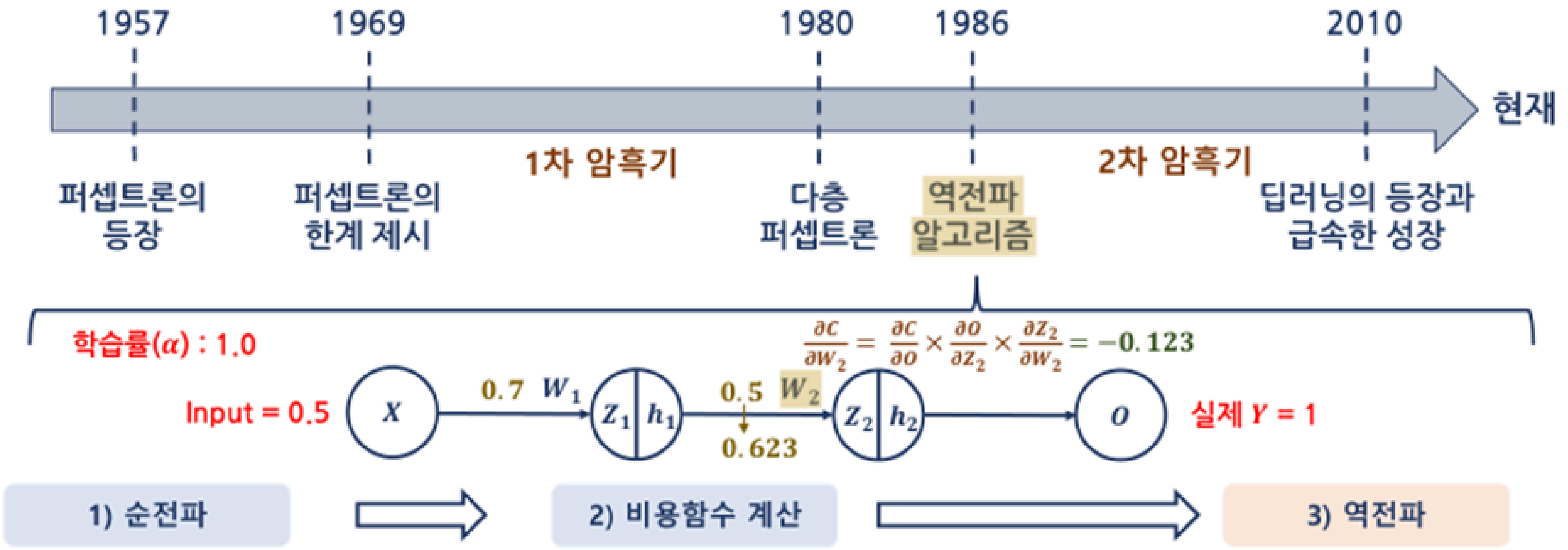
- $Z_1 = W_1 X = 0.7 \times 0.5 = 0.35$
- $h_1 = \sigma(Z_1) = \sigma(0.35) = 0.587$
- $Z_2 = W_2 h_1 = 0.5 \times 0.587 = 0.294$
- $O = h_2 = \sigma(Z_2) = \sigma(0.294) = 0.573$

MSE 사용

- $C = \frac{1}{N}\sum_{i=1}^{N}(O_i - Y_i)^2 = (O - Y)^2$

$= (0.573 - 1)^2 = 0.1823$

역전파 알고리즘



- $Z_1 = W_1X = 0.7 \times 0.5 = 0.35$
- $h_1 = \sigma(Z_1) = \sigma(0.35) = 0.587$
- $Z_2 = W_2h_1 = 0.5 \times 0.587 = 0.294$
- $O = h_2 = \sigma(Z_2) = \sigma(0.294) = 0.573$

MSE 사용

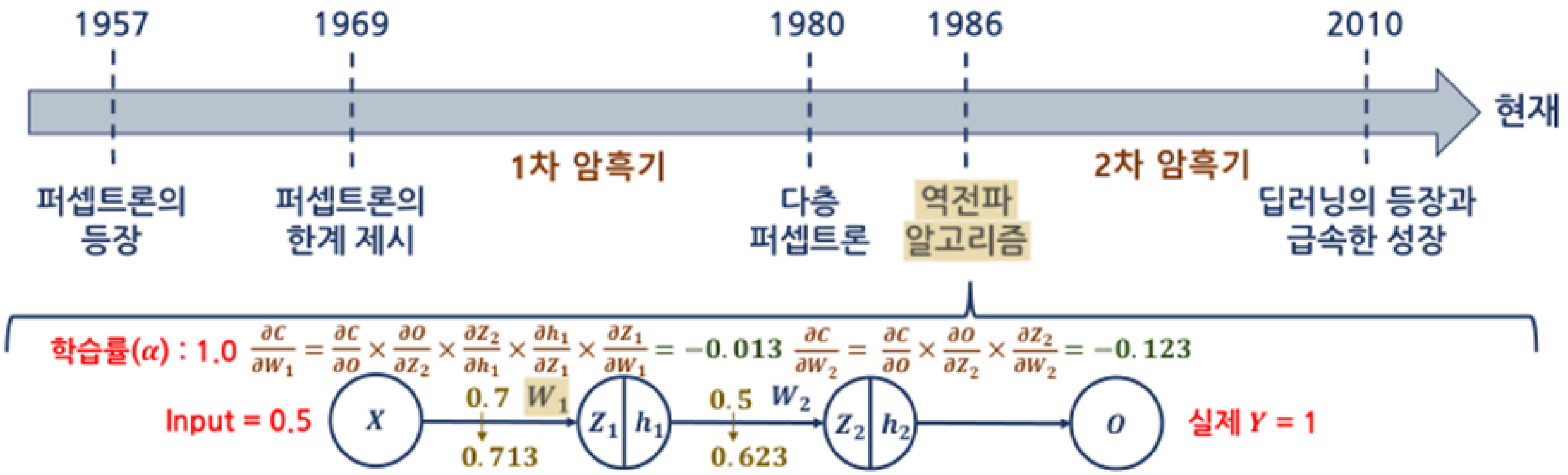
- $C = \frac{1}{N}\sum_{i=1}^{N}(O_i - Y_i)^2 = (O - Y)^2$

  $= (0.573 - 1)^2 = 0.1823$

- $\frac{\partial C}{\partial O} = 2(O - Y) = 2(0.573 - 1) = -0.854$
- $\frac{\partial O}{\partial Z_2} = \frac{\partial h_2}{\partial Z_2} = \frac{\partial}{\partial Z_2}\sigma(Z_2) = \sigma(Z_2)(1 - \sigma(Z_2)) = 0.245$
- $\frac{\partial Z_2}{\partial W_2} = \frac{\partial}{\partial W_2}(W_2h_1) = h_1 = 0.587$

$\rightarrow W_2 = W_2 - \alpha\frac{\partial C}{\partial W_2} = 0.5 - 1.0 \times (-0.123) = 0.623$

역전파 알고리즘



1) 순전파 ⇒ **2) 비용함수 계산** ⇒ **3) 역전파**

1) 순전파

- $Z_1 = W_1X = 0.7 \times 0.5 = 0.35$
- $h_1 = \sigma(Z_1) = \sigma(0.35) = 0.587$
- $Z_2 = W_2h_1 = 0.5 \times 0.587 = 0.294$
- $O = h_2 = \sigma(Z_2) = \sigma(0.294) = 0.573$

2) 비용함수 계산

MSE 사용

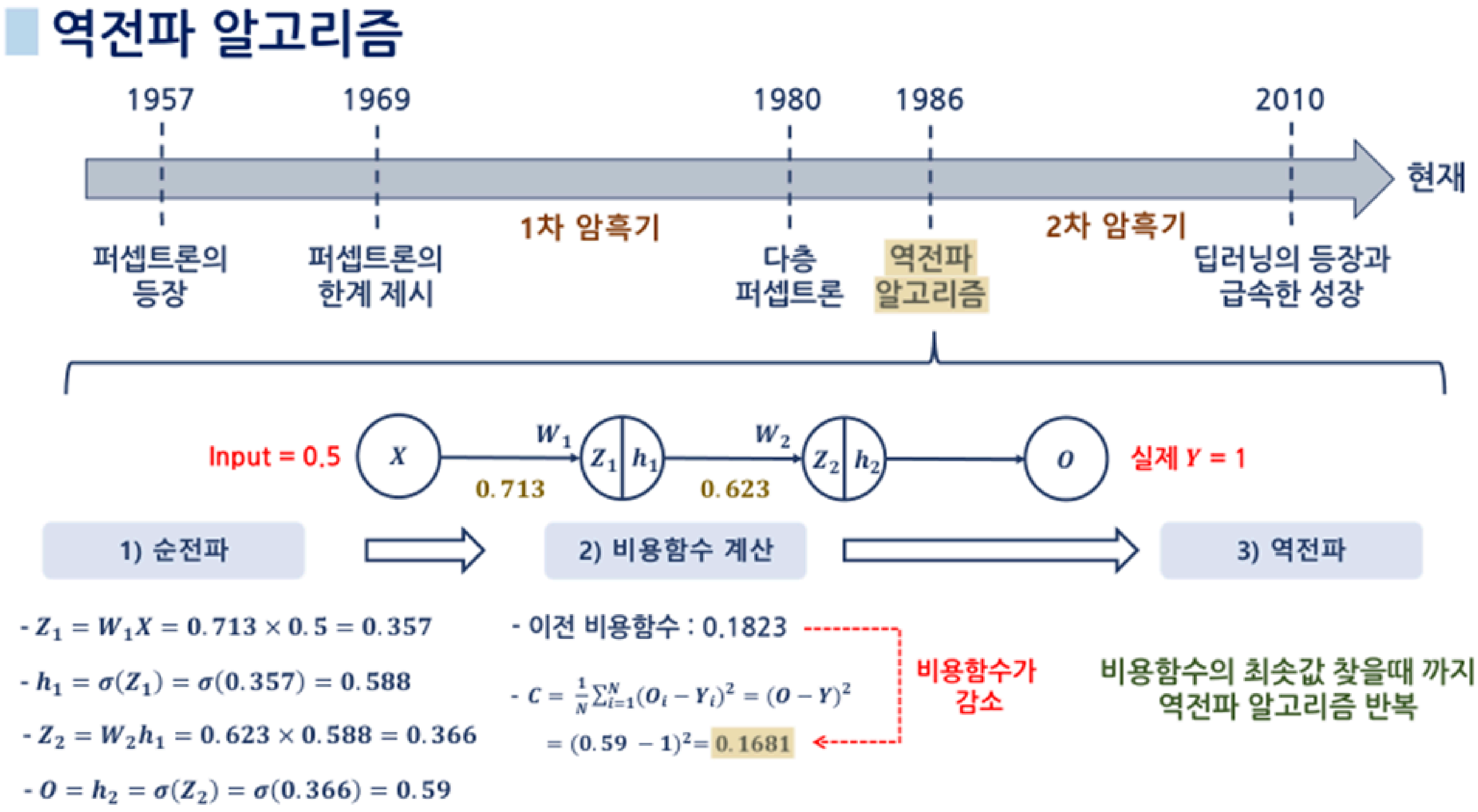
- $C = \frac{1}{N}\sum_{i=1}^{N}(O_i - Y_i)^2 = (O - Y)^2$

$= (0.573 - 1)^2 = 0.1823$

3) 역전파

- $\frac{\partial Z_2}{\partial h_1} = \frac{\partial}{\partial h_1}(W_2h_1) = W_2 = 0.5$
- $\frac{\partial h_1}{\partial Z_1} = \frac{\partial}{\partial Z_1}\sigma(Z_1) = \sigma(Z_1)(1 - \sigma(Z_1)) = 0.242$
- $\frac{\partial Z_1}{\partial W_1} = \frac{\partial}{\partial W_1}(W_1X) = X = 0.5$

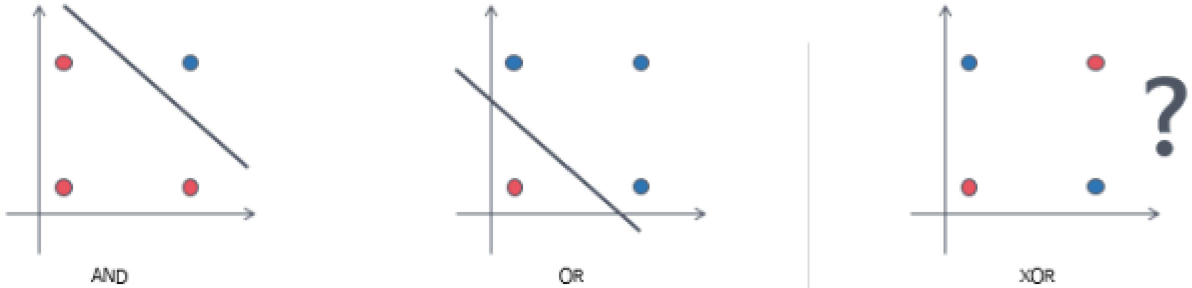
$\rightarrow W_2 = W_2 - \alpha\frac{\partial C}{\partial W_2} = 0.7 - 1.0 \times (-0.013) = 0.713$



활성화 함수(activation function)가 필요한 이유

- 신경망 모델의 각 layer에서는 input 값과 W, b를 곱, 합연산을 통해 a=WX+b를 계산하고 마지막에 활성화 함수를 거쳐 h(a)를 출력함
- 이렇게 각 layer마다 sigmoid, softmax, relu 등.. 여러 활성화 함수를 이용하는데 그 이유가 뭘까?

  - 기존의 퍼셉트론은 AND와 OR문제는 해결할 수 있었지만 선형 분류기라는 한계에 의해 XOR과 같은 non-linear한 문제는 해결할 수 없었음



AND		
Input_A	Input_B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR		
Input_A	Input_B	Output
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
Input_A	Input_B	Output
0	0	0
0	1	1
1	0	1
1	1	0

- 이를 해결하기 위해 나온 개념이 hidden layer이지만 이 hidden layer도 무작정 쌓기만 한다고 해서 퍼셉트론을 선형분류기에서 비선형분류기로 바꿀 수 있는 것은 아니었음

  - 왜냐하면 선형 시스템이 아무리 깊어지더라도 f(ax+by)=af(x) + bf(y)의 성질 때문에 결국 하나의 layer로 깊은 layer를 구현할 수 있기 때문임
  - 즉, linear한 연산을 갖는 layer를 수십개 쌓아도 결국 이는 하나의 linear 연산으로 나타낼 수 있음

활성화함수

- 활성화 함수를 사용하면 입력값에 대한 출력값이 linear하게 나오지 않으므로 선형분류기를 비선형 시스템으로 만들 수 있음
- 따라서 MLP(Multiple layer perceptron)는 단지 linear layer를 여러개 쌓는 개념이 아닌 활성화 함수를 이용한 non-linear 시스템을 여러 layer로 쌓는 개념임

- 이렇게 활성화 함수를 이용하여 비선형 시스템인 MLP를 이용하여 XOR는 해결될 수 있지만, MLP의 파라미터 개수가 점점 많아지면서 각각의 weight와 bias를 학습시키는 것이 매우 어려워 다시 한 번 침체기를 겪게되었음 -> 이를 해결한 알고리즘이 바로 역전파(Back Propagation)임

경사하강법

- 딥러닝에서 모델을 학습한다는 것은?

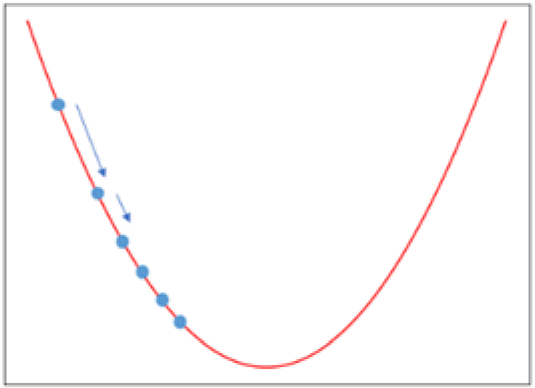
> - 실제 값과 예측 값의 오차를 최소화하는 가중치를 찾는 과정

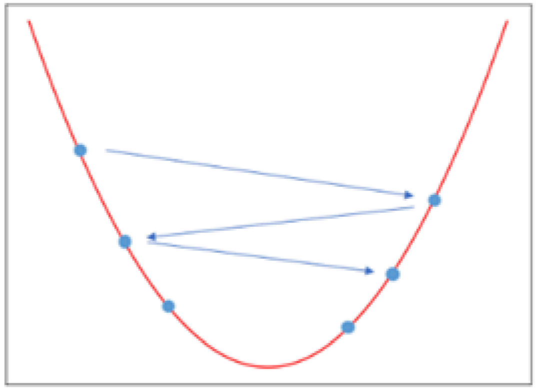
- 비용 함수(Cost function)

> - '오차'를 정의하는 함수
> - 비용 함수가 최솟값을 갖는 방향으로 가중치를 업데이트하는 작업이 필요

- 경사 하강법(Gradient Descent)

> - 최솟값을 찾을 때 사용할 수 있는 최적화 알고리즘
> - 먼저, 최솟값을 찾을 함수를 설정한 후, 임의의 값으로 초기화하고
>   해당 값의 기울기를 빼면서 최솟값에 가까워질 때까지 반복하는 방법



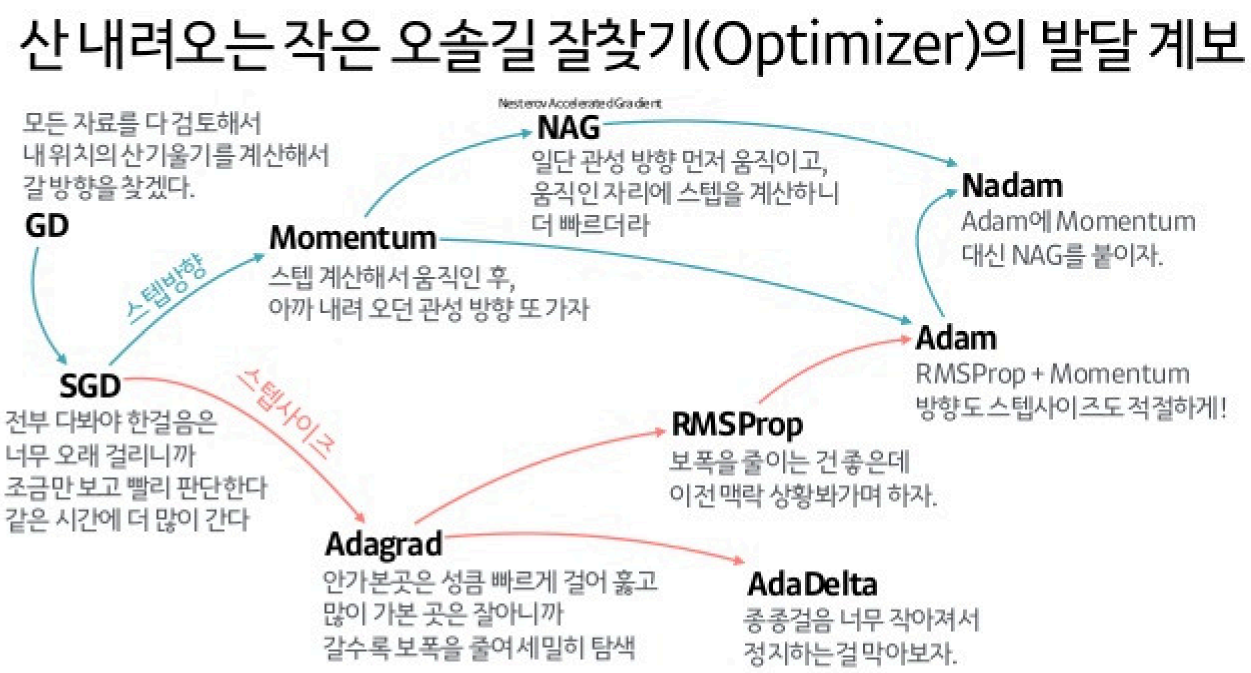


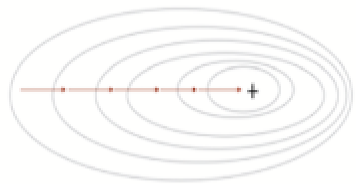
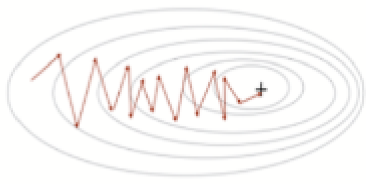
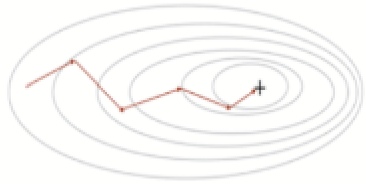
t 시점의 w에 대해 편미분한 값을 빼서 t+1 시점의 w를 구하는 과정을 식으로 나타낸 것

$$w^{t+1} = w^t - \mu \frac{dE(w)}{dw}$$

- E(w)는 오차를 계산하는 비용 함수(목적 함수)
- w는 오차를 구하는 과정에서 사용된 가중치
- 미분 값 앞에 곱해져 있는 상수는 학습률(learning rate)
- 학습률이 작으면 왼쪽 그래프처럼 값이 천천히 변하고, 학습률이 크면 오른쪽 그래프처럼 큰 보폭으로 움직임

경사 하강법 종류



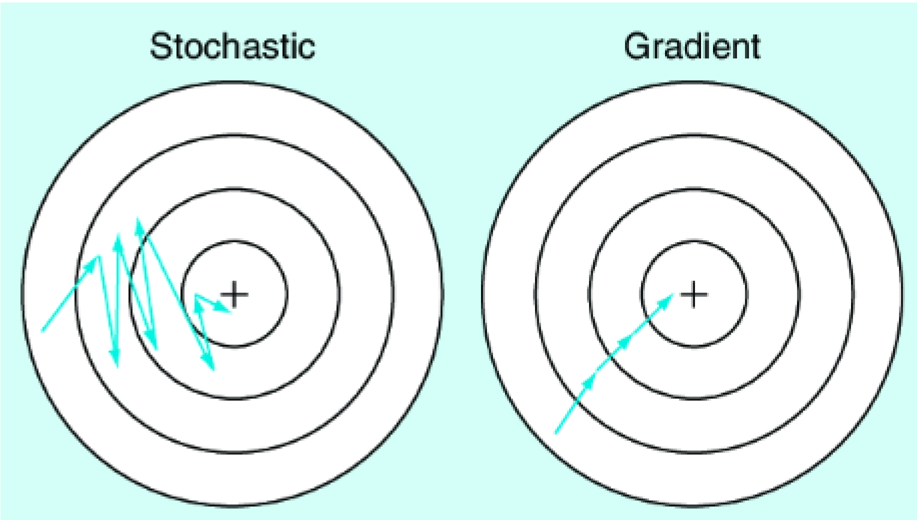
	경사하강법	확률적 경사하강법	미니배치 경사하강법
해를 찾는 과정	Gradient Descent 	Stochastic Gradient Descent 	Mini-Batch Gradient Descent 
1 Iteration에 사용되는 데이터 크기	모든 데이터	임의의 하나의 데이터	설정한 미니배치 사이즈만큼

배치 경사 하강법(Batch Gradient Descent)

- 가장 기본적인 경사 하강법으로 Vanilla Gradient Descent라고 부르기도 함
- 배치 경사 하강법은 데이터셋 전체를 고려하여 손실함수를 계산함
- 배치 경사 하강법은 한 번의 Epoch에 모든 파라미터 업데이트를 단 한 번만 수행함 -> Batch의 개수와 Iteration은 1이고 Batch size는 전체 데이터의 개수임
- 파라미터 업데이트할 때 한 번에 전체 데이터셋을 고려하기 때문에 모델 학습 시 많은 시간과 메모리가 필요하다는 단점이 있음

확률적 경사 하강법(Stochastic Gradient Descent)

- 배치 경사 하강법이 모델 학습 시 많은 시간과 메모리가 필요하다는 단점을 개선하기 위해 제안된 기법
- 확률적 경사 하강법은 Batch size를 1로 설정하여 파라미터를 업데이트하기 때문에 배치 경사 하강법보다 훨씬 빠르고 적은 메모리로 학습이 진행됨
- 확률적 경사 하강법은 파라미터 값의 업데이트 폭이 불안정하기 때문에 배치 경사 하강법보다 정확도가 낮은 경우가 생길 수도 있음
- 단점을 개선하는 모멘텀, AdaGrad, Adam 방법이 있음



미니 배치 경사 하강법(Mini-Batch Gradient Descent)

- Batch size가 1도 전체 데이터 개수도 아닌 경우를 말함
- 미니 배치 경사 하강법은 배치 경사 하강법보다 모델 학습 속도가 빠르고, 확률적 경사 하강법보다 안정적인 장점이 있음
- 딥러닝 분야에서 가장 많이 활용하는 경사 하강법
- Batch size는 어떻게 정하면 좋을까요? 일반적으로 32, 64, 128과 같이 2의 n제곱에 해당하는 값으로 사용하는 게 보편적임

```
# coding: utf-8
import numpy as np
# 경사법
# 현 위치에서 기울어진 방향으로 일정 거리만큼 이동
# 그런다음 다시 기울기를 구하고 기울어진 방향으로 이동
# 점차 함수의 값을 줄이는 것이다.
# 기계 학습을 최적화 하는데 흔히 쓰는 방법.
#### 경사 하강법
# init_x : 초기값
# lr  : learning rate 학습률
# step_num 경사법에 따른 반복 횟수
# 함수의 기울기에 학습률을 곱한값으로 갱신하는 처리는 step_num번 반복
# 학습률과 같은 매개변수를 하이퍼파라미터 라고한다.
# 이는 가중치와 편향같은 신경망의 매개변수와는 성질이 다른 매개변수.
def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x = init_x
    for i in range(step_num):
        grad = numerucal_gradient(f, x)
        x -= lr * grad
    return x

def numerucal_gradient(f, x):
    h = 1e-4  # 0.0001
```

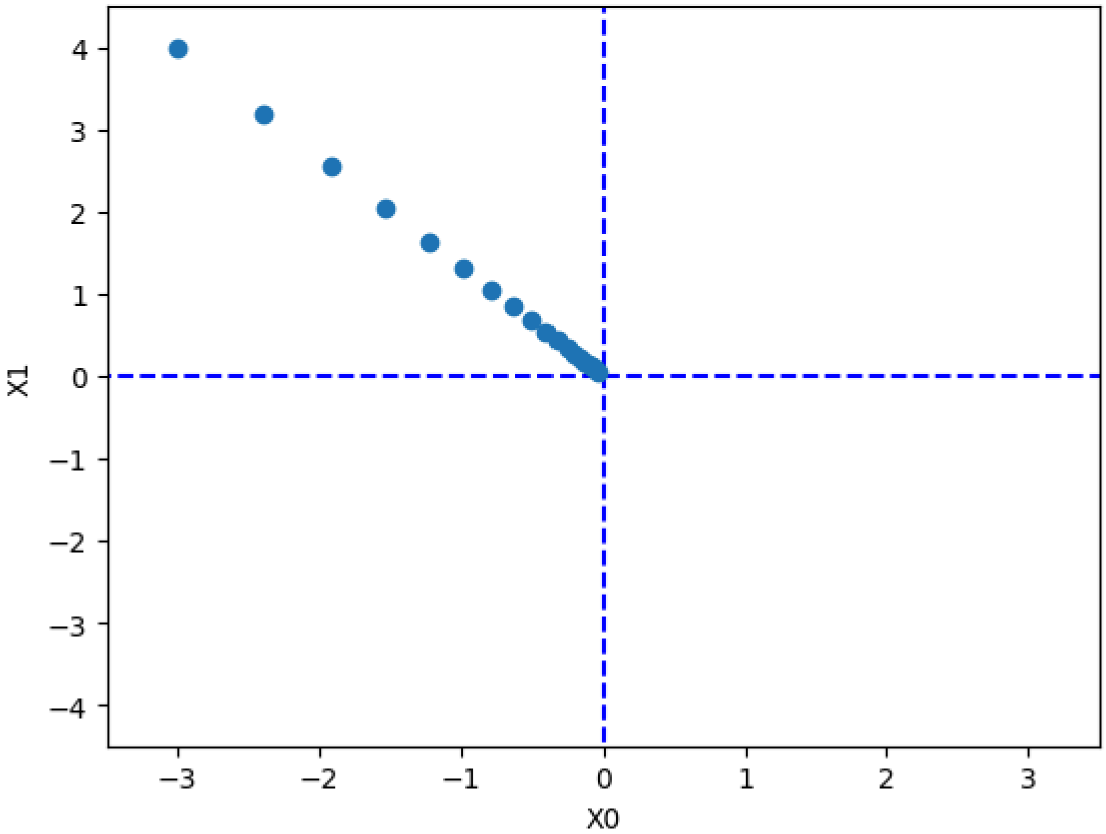
```
    grad = np.zeros_like(x)
    for idx in range(x.size):
        tmp_val = x[idx]
        # f(x+h) 계산
        x[idx] = tmp_val + h
        fxh1 = f(x)
        # f(x-h) 계산
        x[idx] = tmp_val - h
        fxh2 = f(x)
        grad[idx] = (fxh1 - fxh2) / (2 * h)
        x[idx] = tmp_val
    return grad
```

```
#경사법으로 f(x0,x1) = x0^2+x1^2의 최소값을 구하라. https://blog.naver.com/ssdyka/221300959357
def function_2(x):
# return x[0]**2 + x[1]**2
    return np.sum(x ** 2)

init_x = np.array([-3.0, 4.0])
res = gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)
print(res)
```

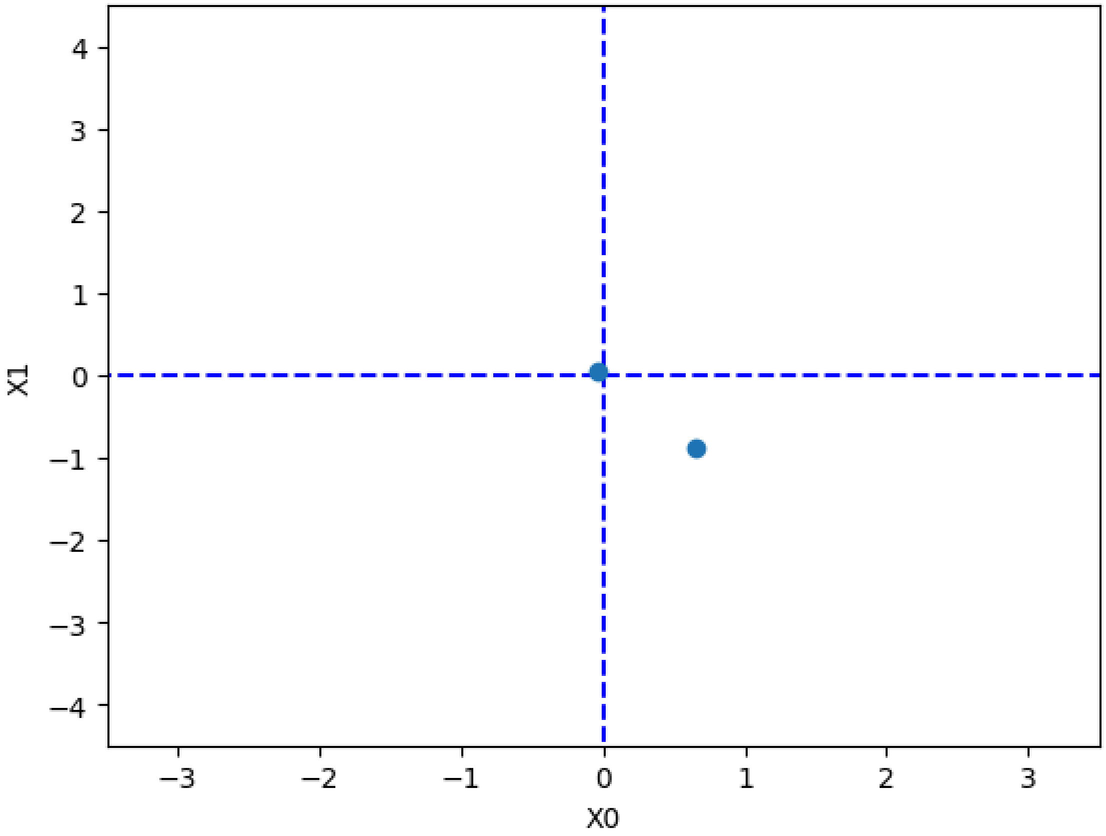
```
[-6.11110793e-10  8.14814391e-10]
```

```
#갱신 과정
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt
def _numerical_gradient_no_batch(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)
    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)
        x[idx] = tmp_val - h
```

```
            fxh2 = f(x) # f(x-h)
            grad[idx] = (fxh1 - fxh2) / (2*h)
            x[idx] = tmp_val
        return grad

def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x = init_x
    x_history = []
    for i in range(step_num):
        x_history.append( x.copy() )
        grad = _numerical_gradient_no_batch(f, x)
        x -= lr * grad
    return x, np.array(x_history)

def function_2(x):
    return x[0]**2 + x[1]**2

init_x = np.array([-3.0, 4.0])
lr = 0.1
step_num = 20
x, x_history = gradient_descent(function_2, init_x, lr=lr, step_num=step_num)
plt.plot( [-5, 5], [0,0], '--b')
plt.plot( [0,0], [-5, 5], '--b')
plt.plot(x_history[:,0], x_history[:,1], 'o')
plt.xlim(-3.5, 3.5)
plt.ylim(-4.5, 4.5)
plt.xlabel("X0")
plt.ylabel("X1")
plt.show()
```


4
3
2
1
0
−1
−2
−3
−4
X1
−3
−2
−1
0
1
2
3
X0

```
############ 학습률이 너무 큰 예
# 학습률이 크면 발산한다.
lr = 10
step_num = 20
x, x_history = gradient_descent(function_2, init_x, lr=lr, step_num=step_num)
plt.plot( [-5, 5], [0,0], '--b')
plt.plot( [0,0], [-5, 5], '--b')
plt.plot(x_history[:,0], x_history[:,1], 'o')
plt.xlim(-3.5, 3.5)
plt.ylim(-4.5, 4.5)
plt.xlabel("X0")
plt.ylabel("X1")
plt.show()
```



```
############# 학습률이 너무 작은 예
# 값이 너무 작으면 거의 갱신되지 않은채 끝난다.
lr =0.1
step_num = 1
x, x_history = gradient_descent(function_2, init_x, lr=lr, step_num=step_num)
plt.plot( [-5, 5], [0,0], '--b')
plt.plot( [0,0], [-5, 5], '--b')
plt.plot(x_history[:,0], x_history[:,1], 'o')
plt.xlim(-3.5, 3.5)
plt.ylim(-4.5, 4.5)
plt.xlabel("X0")
plt.ylabel("X1")
plt.show()
```