

# 신경망 학습

손실 함수, 수치 미분(편미분), 기울기(경사 하강법)

---

3조

김서윤, 김아진, 서윤, 이철민

# 목차

01 손실함수

---

02 수치 미분 - 편미분

---

03 기울기 - 경사 하강법

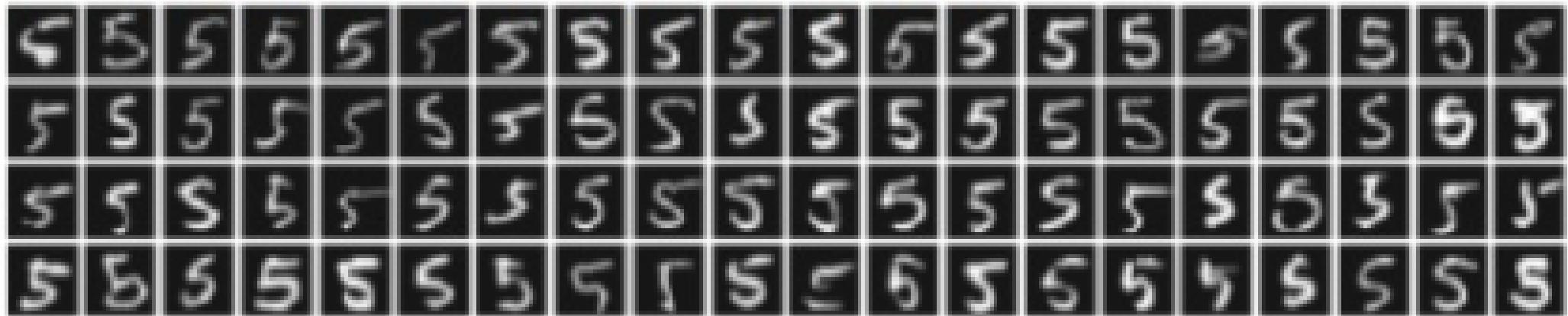
---

## 01 손실함수

# 신경망 학습

: 데이터에서 학습하는 신경망

예시 ) 자유분방한 손글씨 이미지를 보고 5인지 아닌지 알아보는 프로그램 구현



사람



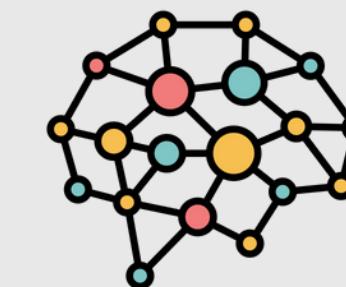
경험과 직관을 단서로 어렵지 않게 인식.  
그러나 그 안에 숨은 규칙성을 설명하기 어려움.

기계학습



이미지의 특징을 '사람'이 추출하고  
그 특징의 규칙성을 '기계'가 찾아냄.

신경망 (딥러닝)



사람의 개입없이 데이터로부터 학습.  
규칙성 뿐만 아니라 특징까지 스스로 찾아냄.

# 손실함수

## 손실함수

실제값과 예측값의 차이를 표현하도록 계산하는 함수

손실함수가 작을수록 정확하다는 것을 나타낸다.

해당 값을 가장 좋게 만들어주는 가중치 매개변수의 값을 탐색하는 것이 신경망 학습의 목표다.

## 대표적인 손실함수

※ 다양한 손실함수가 있지만 주로 많이 다루는 문제에 필요한 손실함수가 정의되어 있다.

### 평균 제곱 오차 (MSE)

직관적이고 쓰기 쉬움

회귀 문제(Regression)에서 주로 사용

### 교차 엔트로피 오차 (CEE)

손실을 더 잘 보여줌

분류 문제(Classification)에서 주로 사용

# 평균 제곱 오차

## 평균 제곱 오차(MSE, mean squared error)

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2$$

- $y_k$ : 예측값 (신경망의 출력)
- $t_k$ : 실제값 (정답 레이블)
- $k$ : 데이터 차원의 수

※ 딥러닝의 MSE가 원래 MSE와 달리 2로 나누어준 것은 경사하강법 과정에서 발생하는 오류를 최소화하기 위함

	6월	9월	예측값( $y_k$ )	실제값( $t_k$ )	오차	오차제곱
철민	60점	80점	90점	70점	20	400
서윤	50점	60점	70점	55점	15	225
윤	60점	60점	50점	40점	-10	100

```
import numpy as np

def mean_squared_error(y, t):
    return 0.5 * np.sum((y - t)**2)

# 정답 레이블, 클래스 2가 정답
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

# 모델이 클래스 2일 확률을 가장 높게 예측한 경우
y1 = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]

# 모델이 클래스 7일 확률을 가장 높게 예측한 경우
y2 = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

# MSE 계산
mse_y1 = mean_squared_error(np.array(y1), np.array(t))
mse_y2 = mean_squared_error(np.array(y2), np.array(t))

print(mse_y1)
print(mse_y2)

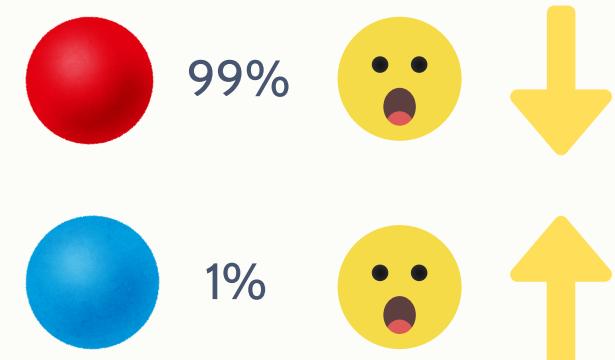
0.09750000000000003
0.5975
```

퀴즈1.  $y1$ 과  $y2$  중 더 정확한 예측을 한 것은?

# 교차 엔트로피 오차

## 정보량

: 정보란 놀람을 객관적인 수치로 표현한 것



## 기대값

: 확률변수에 평균적으로 기대하는 값

한 선수의 오늘의 예상 실력  
= 객관적인 기량 x 오늘 발휘될 확률



## 엔트로피

: 정보량의 기대값, 예측가능성에 대한 확률 판단 가능

$$E(x) = 5 * 0.9$$

$$E(x) = 9 * 0.1$$

안정적인 실력을 가진 선수 불안정한 실력을 가진 선수

정보는 확률과 반비례 개념으로  
 $\log(1/p(x)) = -\log(p(x))$

$$E(x) = \sum x * p(x)$$

기대값의 x에 정보량을 넣은 것  
 $E(x) = \sum \log(1/p(x)) * p(x)$

## 교차 엔트로피 오차 (CEE, Cross Entropy Error)

$$\sum \log\left(\frac{1}{q(x)}\right) \cdot p(x)$$

:  $p(x)$ 의 확률에  $q(x)$ 의 놀람도를 곱한 것

## 01 손실함수

# 교차 엔트로피 오차

```
import numpy as np

def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y + delta))

# 정답 레이블, 클래스 2가 정답
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

# 신경망이 클래스 2로 추정
y1 = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]

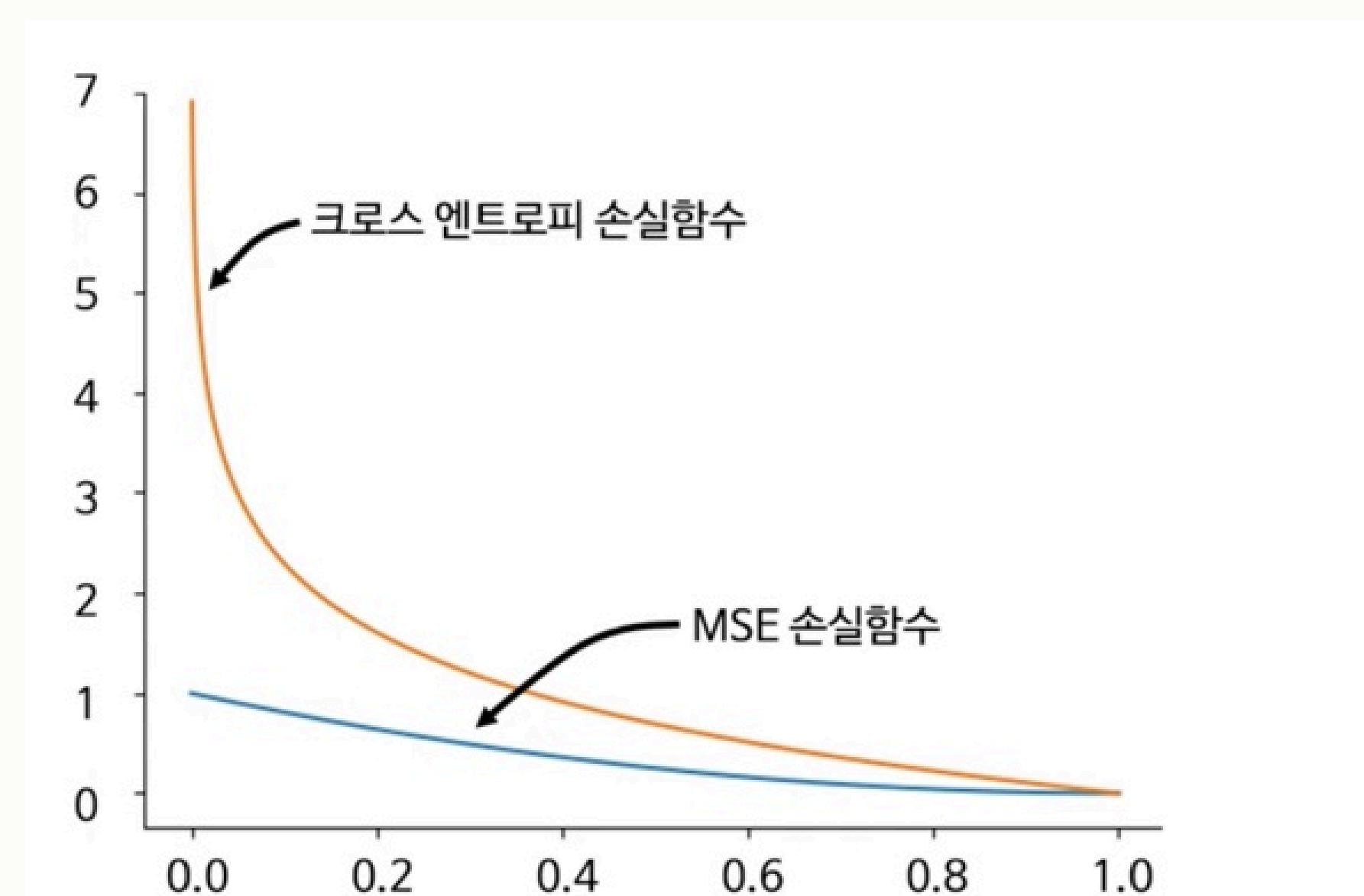
# 신경망이 클래스 7로 추정
y2 = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]

# 교차 엔트로피 오차 계산
ce_y1 = cross_entropy_error(np.array(y1), np.array(t))
ce_y2 = cross_entropy_error(np.array(y2), np.array(t))

print(ce_y1)
print(ce_y2)
```

0.510825457099338  
2.302584092994546

MSE와 마찬가지로  $y_1$ 이 더 정확함

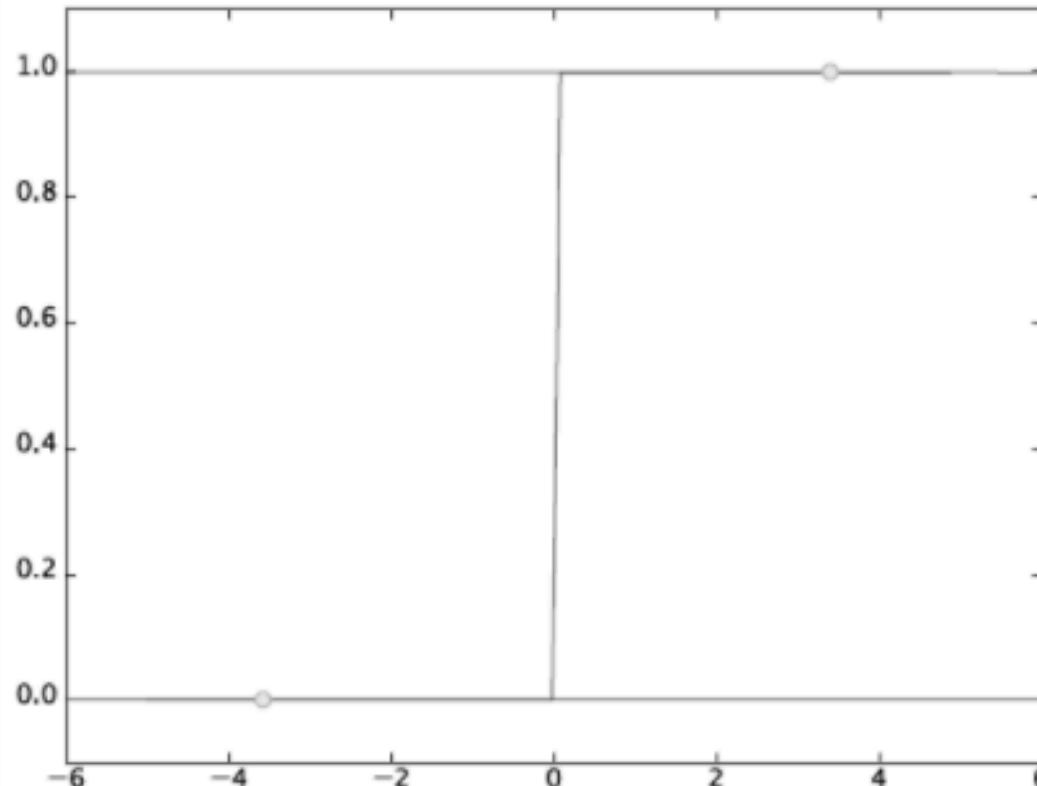


## 손실함수와 미분

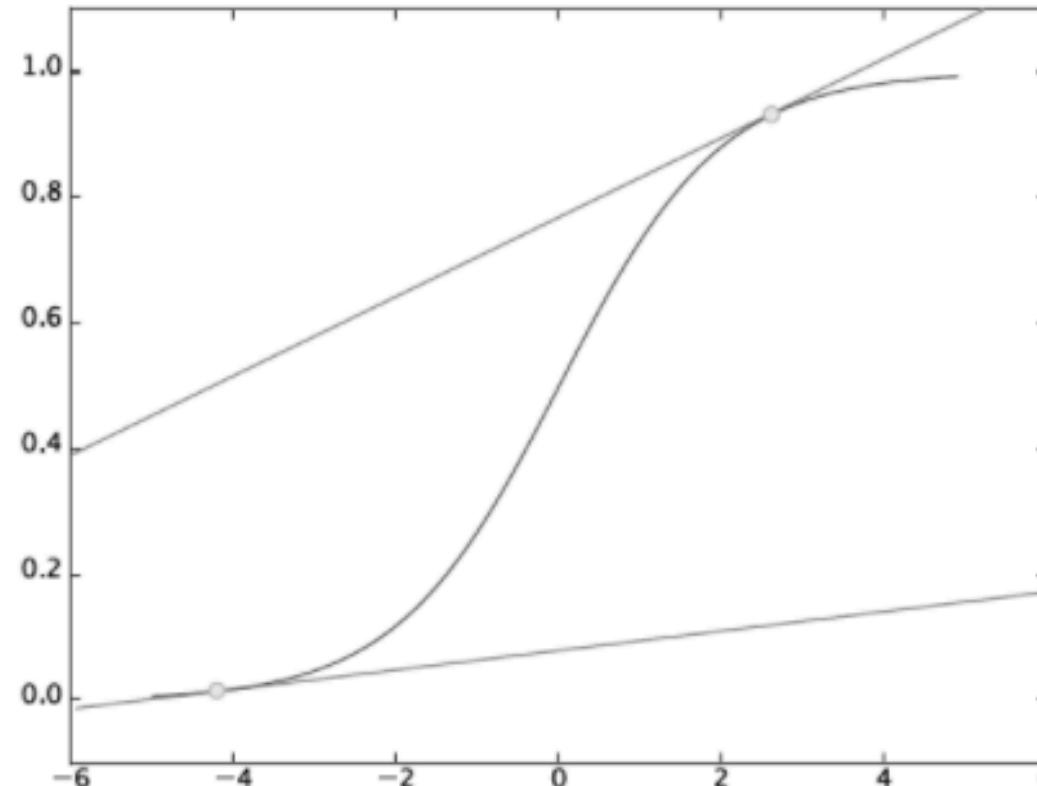
왜 정확도라는 지표를 놔두고 손실 함수의 값이라는 우회적인 방법을 사용할까?

신경망 학습에서는 미분값을 단서로 매개변수의 값을 서서히 갱신해나가며 최적값을 탐색한다.  
만약 불연속적인 정확도( $33\% \rightarrow 32\%$ )를 지표로 삼게된다면 대부분의 미분값이 0이 되기에 갱신이 어려워진다.  
그렇기에 어느 장소라도 미분값이 0이 되지 않는 손실 함수를 사용하는 것이다.

계단 함수



시그모이드 함수

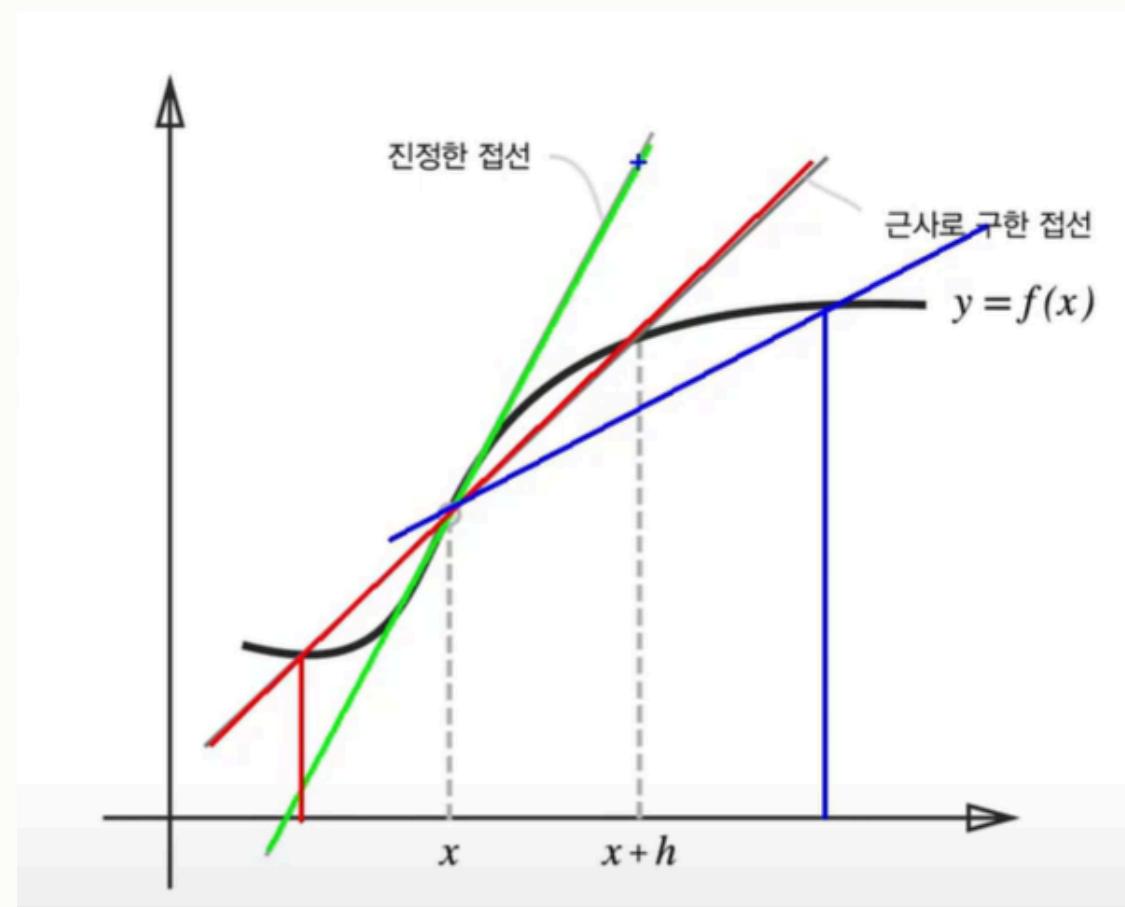


퀴즈2. 둘 중 손실함수에 더 적합한 함수는?

# 수치미분

## 수치미분

'해석적 미분'은 우리가 수학 시간에 배운 오차를 포함하지 않는 '진정한 미분' 값을 구해준다.  
 '수치 미분'은 충분히 작은  $h$ 를 잡아 이를 근사시키는 방법이다. 따라서 오차가 포함된다.



### 해석적 미분

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

컴퓨터 학습

### 수치 미분

$$\frac{f(x + h) - f(x)}{h}$$

$$\boxed{\lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}} = \boxed{\lim_{h \rightarrow 0} \frac{f(x + h) - f(x - h)}{2h}}$$

>>> 오차를 줄이기 위해 중심 차분 혹은 중앙 차분을 이용

※ 중앙차분 : ( $x_x + Hh$ )와 ( $x_x - Hh$ )일 때의 함수  $F_f$ 의 차분을 계산하는 방법

# 수치미분 예시

$$y = 0.01x^2 + 0.1x$$

```
import matplotlib.pyplot as plt

# 함수 정의
def function_1(x):
    return 0.01 * x ** 2 + 0.1 * x

# 수치적 미분 함수 정의
def numerical_diff(f, x):
    h = 1e-4 # 작은 값
    return (f(x + h) - f(x - h)) / (2 * h)

# 그래프 그리기
x = np.arange(0.0, 20.0, 0.1)

plt.xlabel('x')
plt.ylabel('f(x)')
plt.plot(x, y)
plt.show()
```

```
# x=5 일 때, 함수의 미분
diff_5 = numerical_diff(function_1, 5)
print("f'(5) =", diff_5)

# x=10 일 때, 함수의 미분
diff_10 = numerical_diff(function_1, 10)
print("f'(10) =", diff_10)
```

f'(5) = 0.1999999999990898  
f'(10) = 0.29999999999986347

**해석적 미분 결과 (0.2, 0.3) vs 수치 미분 결과 (0.199, 0.299)**  
**>>> 매우 적은 오차!**

## O2. 수치 미분 - 편미분

# 편미분

## 편미분

다변수 함수에서 특정 변수에 대해 다른 변수들은 고정시키고 해당 변수만 변화시킬 때의 변화율을 구하는 것. 다른 변수는 상수취급을 한다.

```
def numerical_diff(f, x):
    h = 1e-4 # 작은 값
    return (f(x + h) - f(x - h)) / (2 * h)

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

# x0 = 3, x1 = 4에서의 편미분을 구하는 함수 정의
def function_tmp1(x0): # x0에 대한 편미분
    return x0 ** 2 + 4.0 ** 2

def function_tmp2(x1): # x1에 대한 편미분
    return 3.0 ** 2 + x1 ** 2

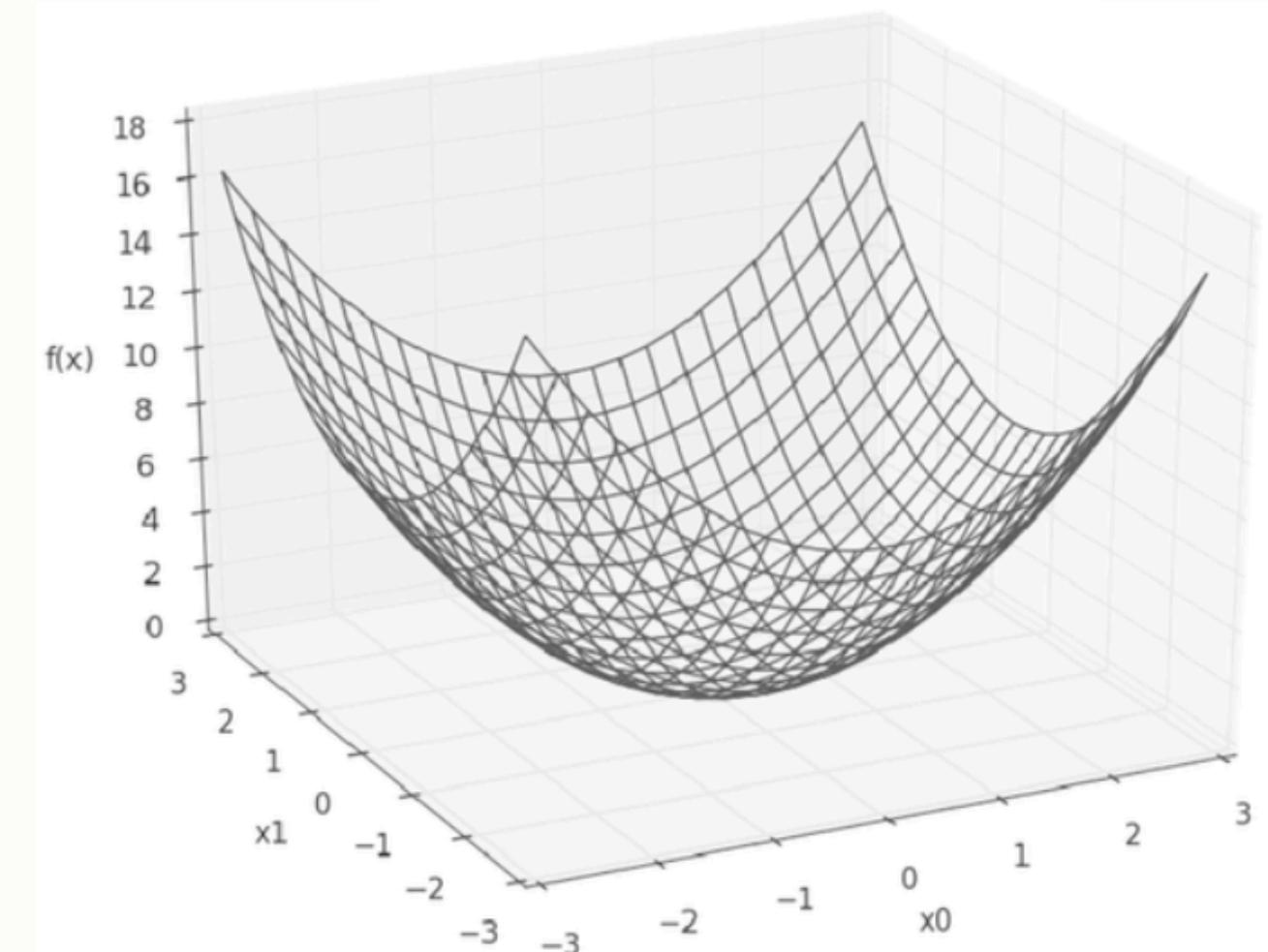
# x0 = 3에서의 편미분 계산
diff_x0 = numerical_diff(function_tmp1, 3.0)
print("df/dx0 at x0=3:", diff_x0)

# x1 = 4에서의 편미분 계산
diff_x1 = numerical_diff(function_tmp2, 4.0)
print("df/dx1 at x1=4:", diff_x1)

df/dx0 at x0=3: 6.0000000000378
df/dx1 at x1=4: 7.9999999999119
```

예시

$$f(x_0, x_1) = x_0^2 + x_1^2$$



# 경사하강법 정의

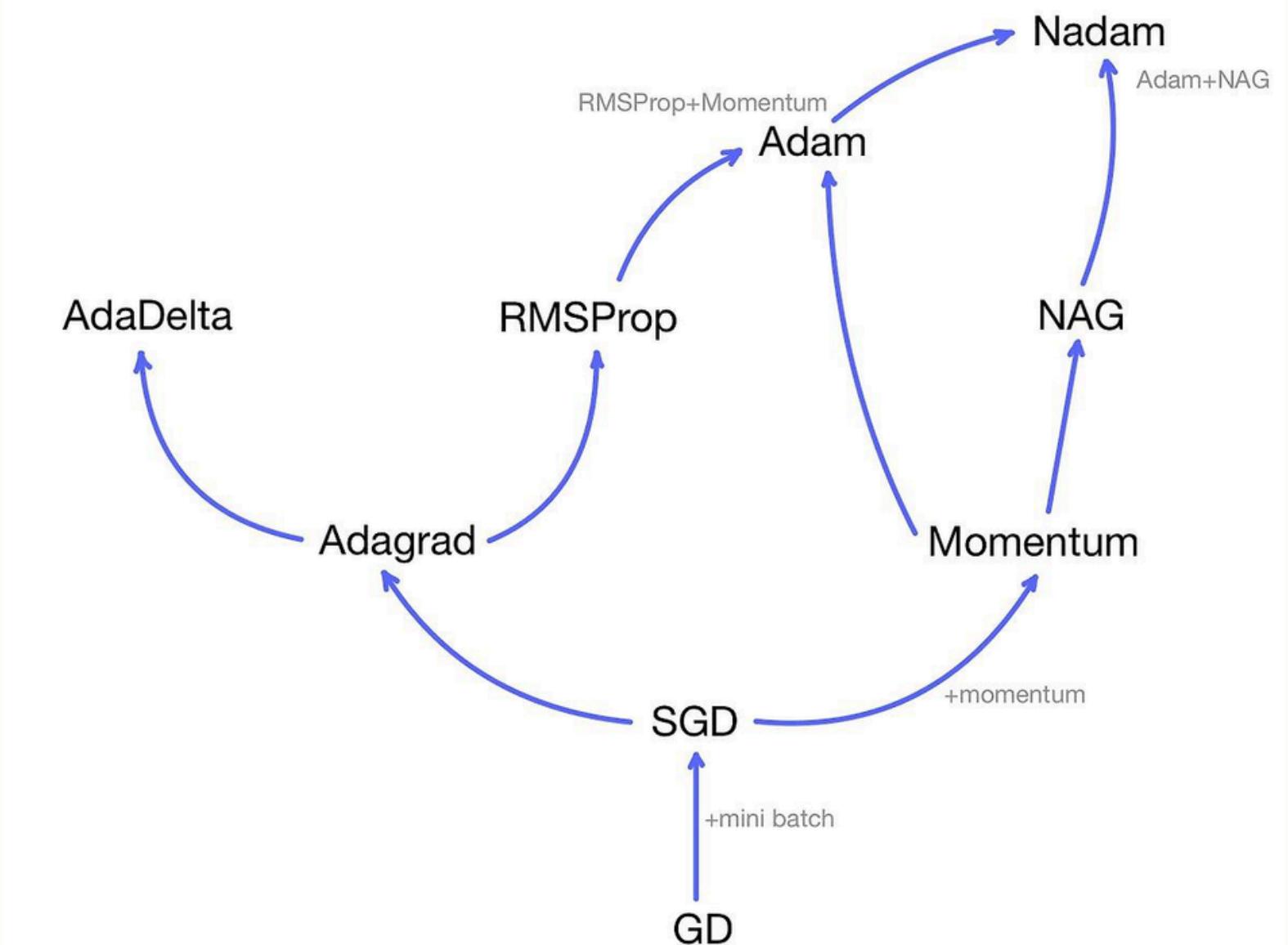
## 경사하강법

머신러닝 모델의 옵티마이저(OPTIMIZER)의 한 종류

옵티마이저는 주어진 데이터에 맞게 모델 파라미터들을 최적화시켜주는 역할  
손실 함수의 값이 최대한 작아지도록 모델 파라미터를 조정

## 경사하강법의 필요성

머신러닝에서 옵티마이저들은  
**경사하강법(GRADIENT DESCENT, GD)**에서  
새로운 개념을 보완해가며 만들어진 것이기 때문에  
경사하강법 개념 정립 필요

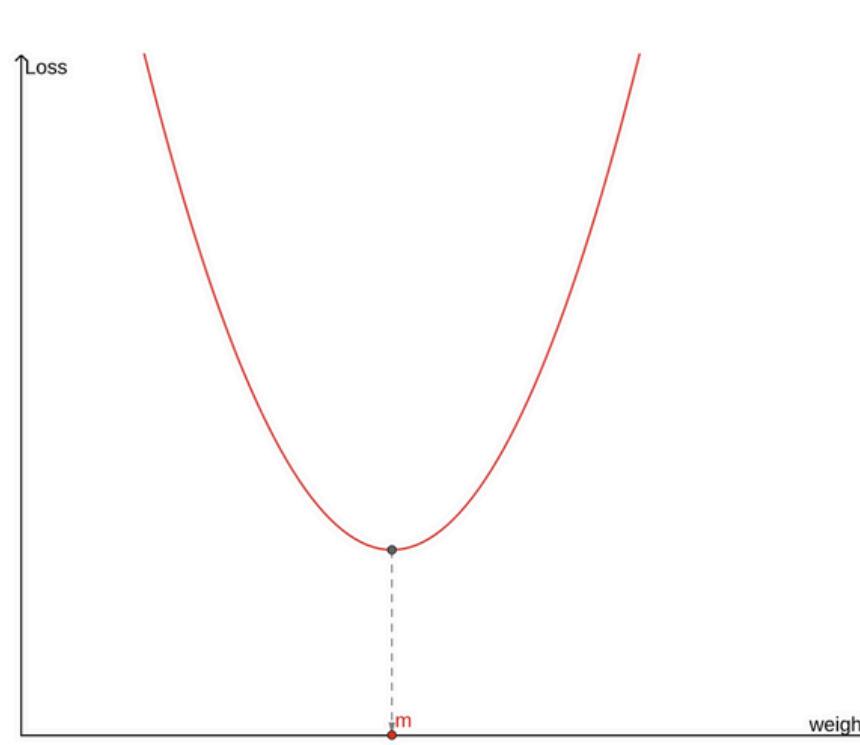


# 경사하강법 정의

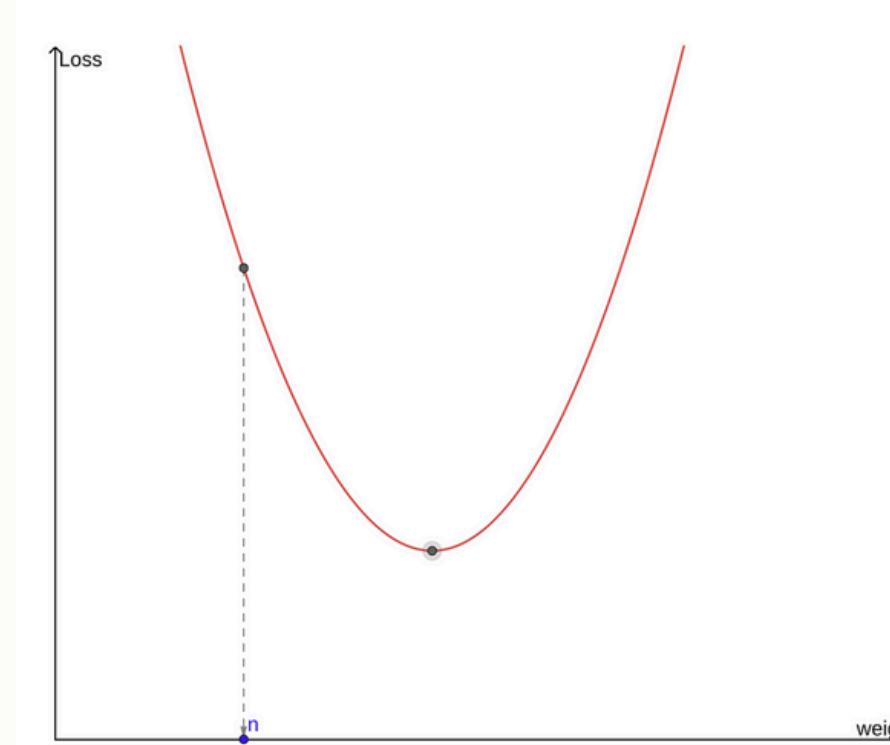
## 경사하강법

기울기(경사, Gradient)를 이용하여 손실 함수의 값을 최소화하는 방법  
조정하고자 하는 값(변수)은 가중치(weight)와 바이어스(bias)  
즉, **퀴즈3. 손실함수는 O와 O에 대한 함수일까요?**

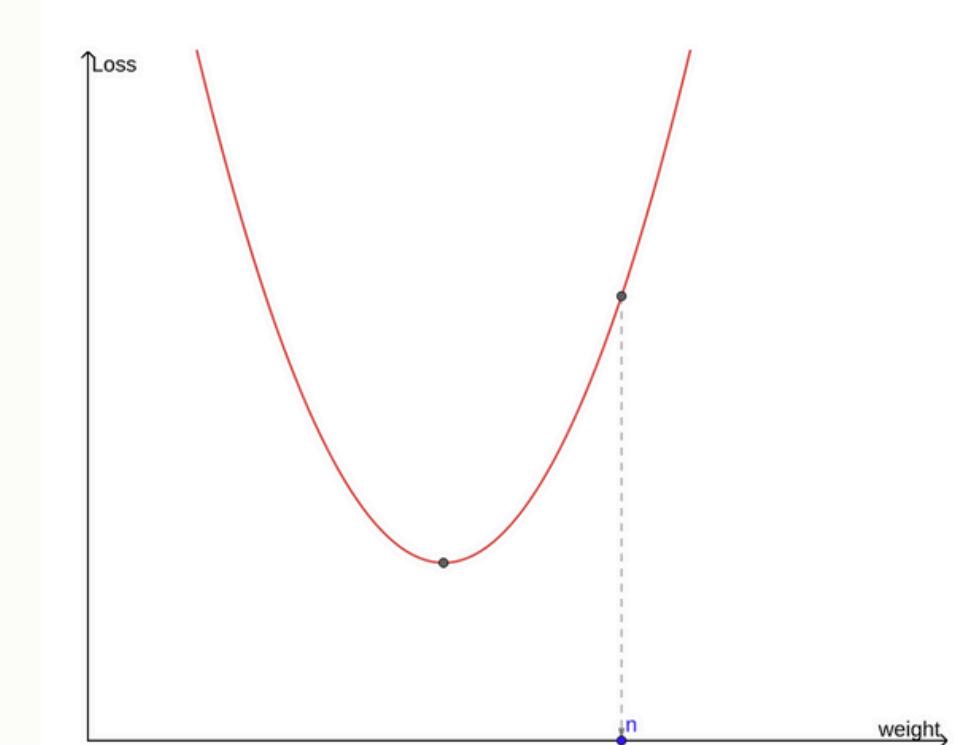
$w = m$ 에서 최소



$w = w +$  양수



$w = w +$  음수



# 경사하강법의 이해

## 경사하강법의 이해

만약,  $w$ 값에서 손실 함수의 미분계수가 음이라면,

$w$ 를 양의 방향으로 이동시켜주어야 함

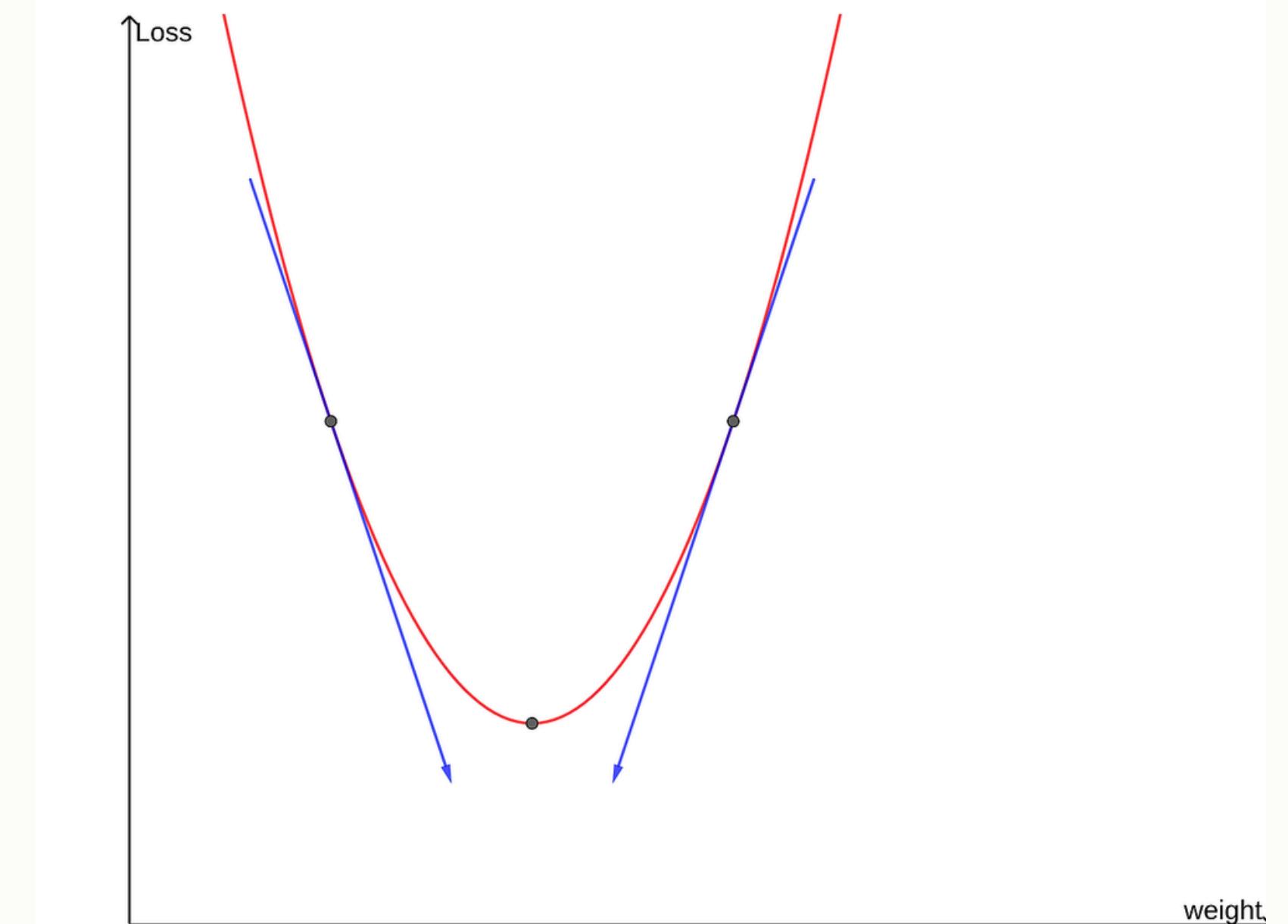
만약,  $w$ 값에서 손실 함수의 미분계수가 양이라면,

$w$ 를 음의 방향으로 이동시켜주어야 함

이 과정을 반복하는 것이 경사하강법임

$$w = w - \alpha \times \frac{\partial L}{\partial w}$$

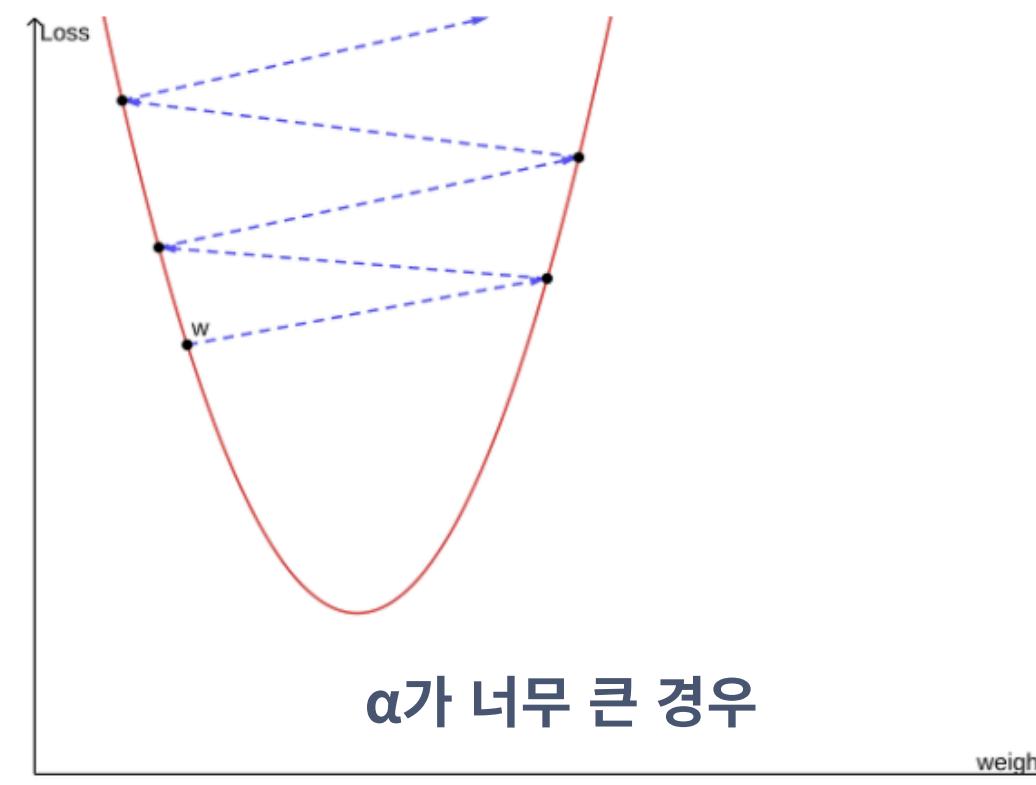
(  $w$  : weight,  $\alpha$  : learning-rate)



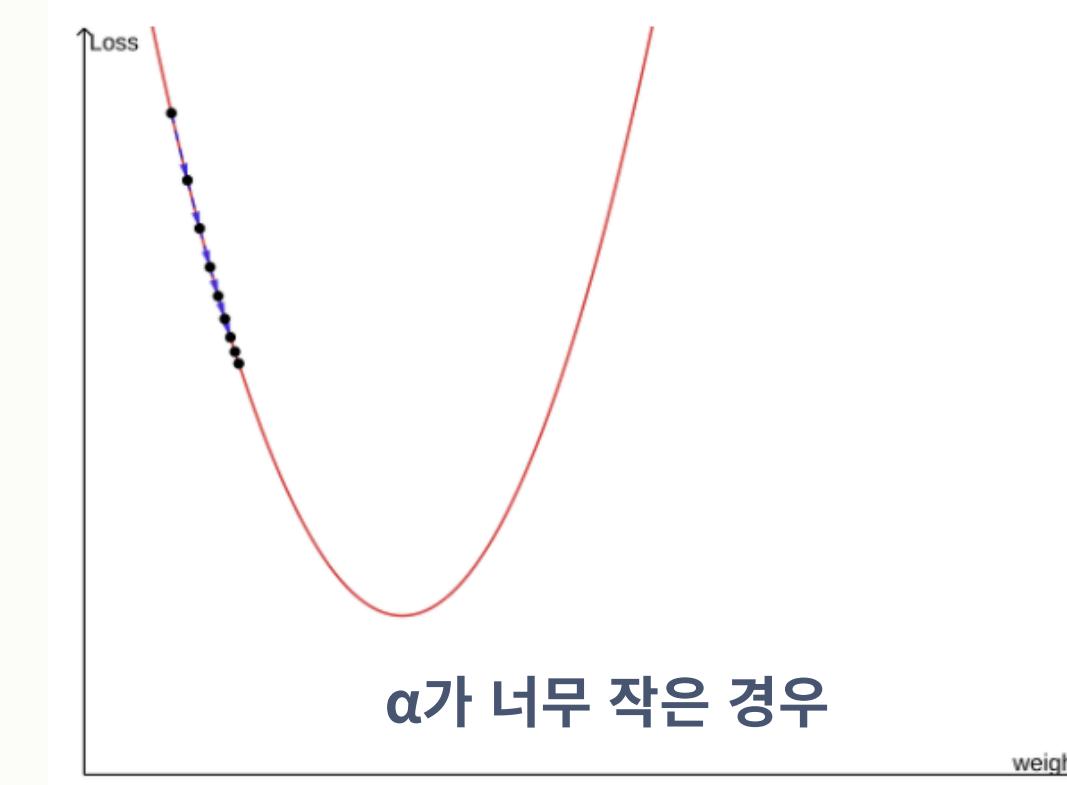
# 경사하강법의 이해

## 경사하강법 수식 - $\alpha$ 란?

$\alpha$ 란 [퀴즈 4.  $\alpha$ 의 명칭은 무엇일까요?]라는 파라미터로,  $w$ 값이 움직이는 거리를 조절해주기 위해 사용함  
만약,  $w$ 가 이동하는 거리가 너무 크다면 손실 값이 커지는 방향으로  $w$ 가 조정될 가능성이 있고,  
반대로,  $w$ 가 이동하는 거리가 너무 작다면  $w$ 가 적합한 값으로 수렴하는 데에 너무 오랜 시간이 걸림



$\alpha$ 가 너무 큰 경우



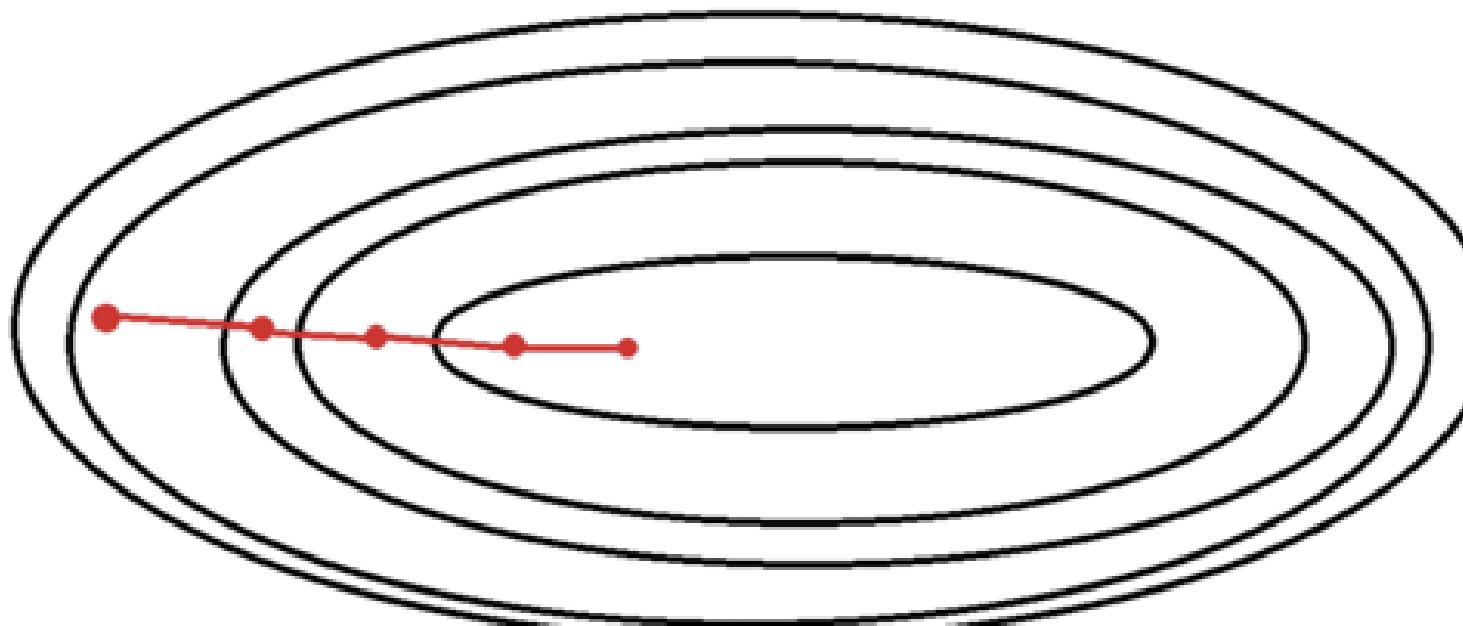
$\alpha$ 가 너무 작은 경우

# 경사하강법의 종류

## 배치 경사 하강법 (BGD, Batch Gradient Descent)

매개변수를 한 번 갱신하는데 전체 학습 데이터를 하나의 배치로 묶어 사용하는 방법

\* 배치 (Batch) : GPU가 한번에 처리하는 데이터의 묶음 >> 전체 학습 데이터



### 장점

적은 업데이트 횟수  
안정적인 수렴  
GPU를 활용한 병렬 처리에 유리

### 단점

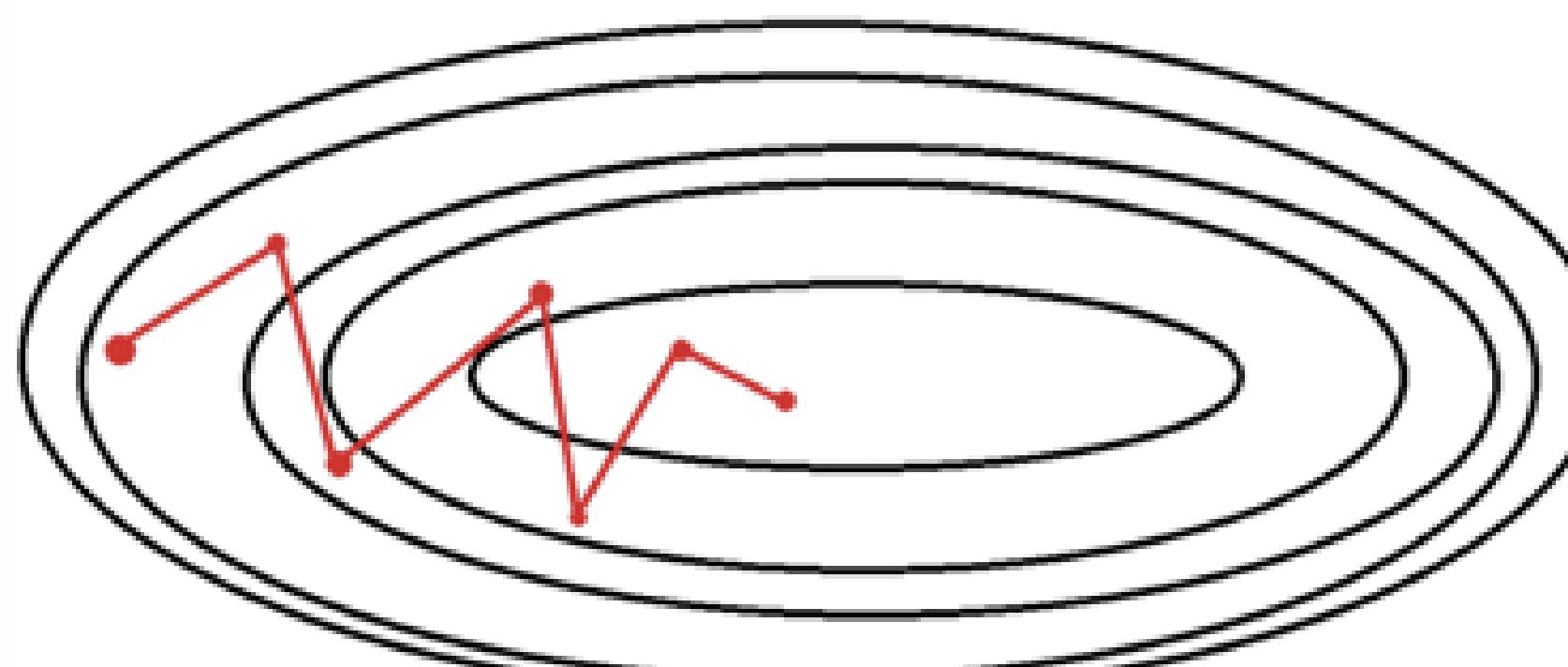
계산 시간이 오래 걸림  
지역 최소(local minimum)에 수렴될 경우 탈출이 어려움  
많은 메모리 필요

# 경사하강법의 종류

## 확률적 경사 하강법 (SGD, Stochastic Gradient Descent)

매개변수를 갱신하기 위해 **무작위로 샘플링된 한 개의 데이터**를 이용하여 경사하강법을 진행하는 방법

\*확률적(Stochastic) : 전체 학습 데이터 중 무작위로 선택된 하나의 데이터를 이용하는 것



### 장점

배치 경사 하강법에 비해 적은 데이터로 학습, 빠른 속도  
배치 경사 하강법보다 전역 최솟값을 찾을 가능성 높음

### 단점

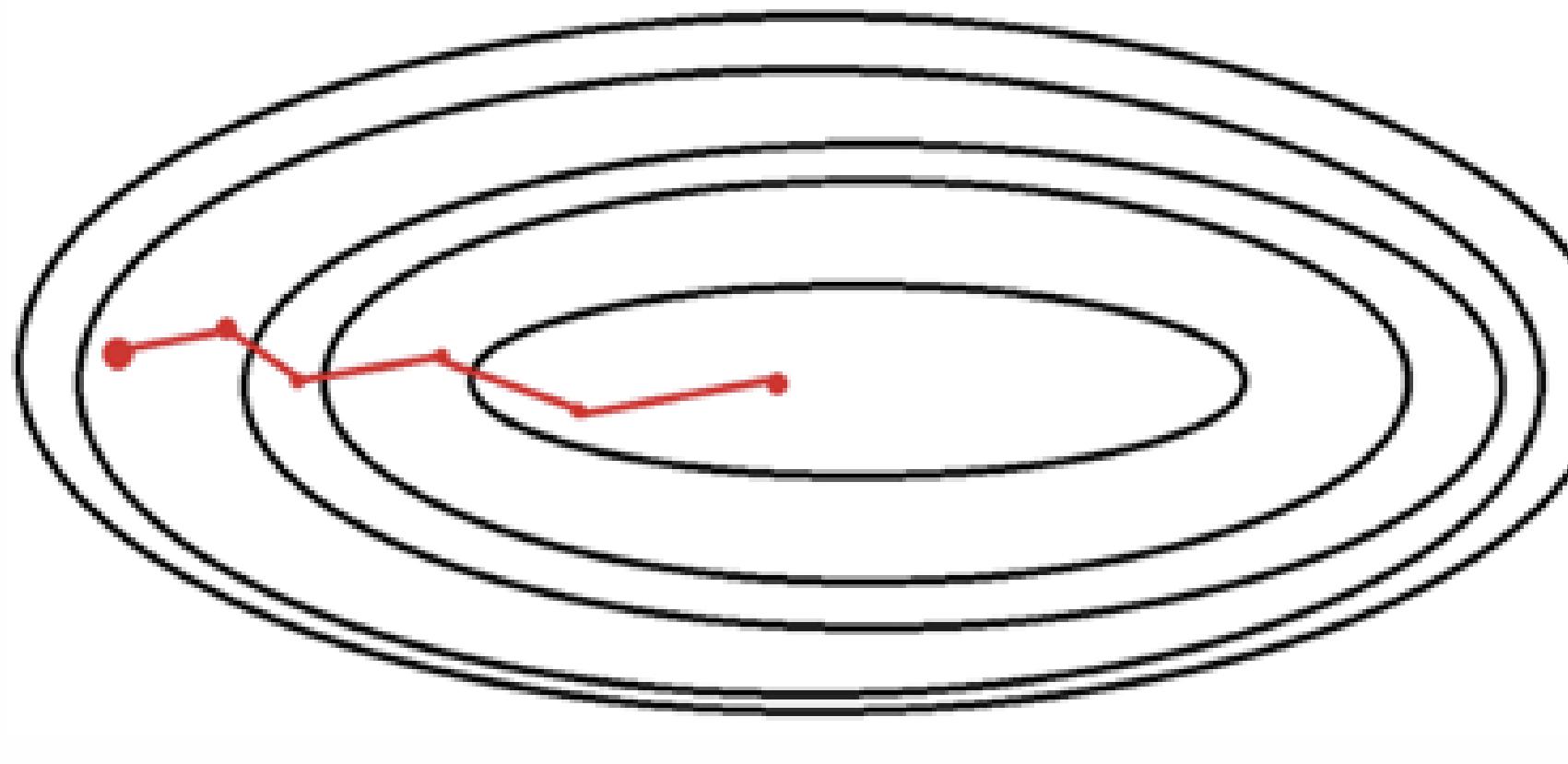
배치 경사 하강법보다 불안정 (무작위성)  
오차율이 크고, GPU의 성능을 전부 활용할 수 없음  
전역 최솟값에 다다르지 못할 수 있음

# 경사하강법의 종류

## 미니배치 경사 하강법 (MBGD, Mini-Batch Gradient Descent)

전체 경사 하강법과 확률적 경사 하강법의 절충안

\*미니배치: 훈련 데이터로부터 학습을 수행하기 위해 추출된 일부



### 장점

행렬 연산에 최적화된 하드웨어(GPU)를 사용해서 얻는 성능 향상  
미니배치를 어느 정도 크게 하면, SGD보다 덜 불규칙적  
SGD보다 최솟값에 더 가까이 도달할 수 있음

### 단점

SGD에 비해 지역 최솟값에서 빠져나오기 힘들 수도 있음

## 참고문헌

---

<https://yhyun225.tistory.com/5>

<https://bruders.tistory.com/91>

<https://www.youtube.com/watch?v=vAaReyHMfY8>

<https://deep-learning-study.tistory.com/12>

<https://velog.io/@kyj93790/%EB%B0%91%EB%B0%94%EB%8B%A5%EB%B6%80%ED%84%BO-%EC%8B%9C%EC%91%ED%95%98%EB%8A%94-%EB%94%A5%EB%9F%AC%EB%8B%9D-4.-%EC%8B%A0%EA%B2%BD%EB%A7%9D-%ED%95%99%EC%8A%B5-part2-%EC%88%98%EC%B9%98-%EB%AF%B8%EB%B6%84-%EA%B8%BO%EC%9A%B8%EA%B8%BO>

# 감사합니다!

---

3조

김서윤, 김아진, 서윤, 이철민