대학생 연합 빅데이터 동아리
BITAmin

# 매개변수 갱신 방법과 가중치 초기화

## 여름방학세션 발표 6조

13기 강나영 김제민 배성윤 이형석 한진솔

2024-08-03

대학생 연합 빅데이터 동아리
BITAmin

# CONTENTS

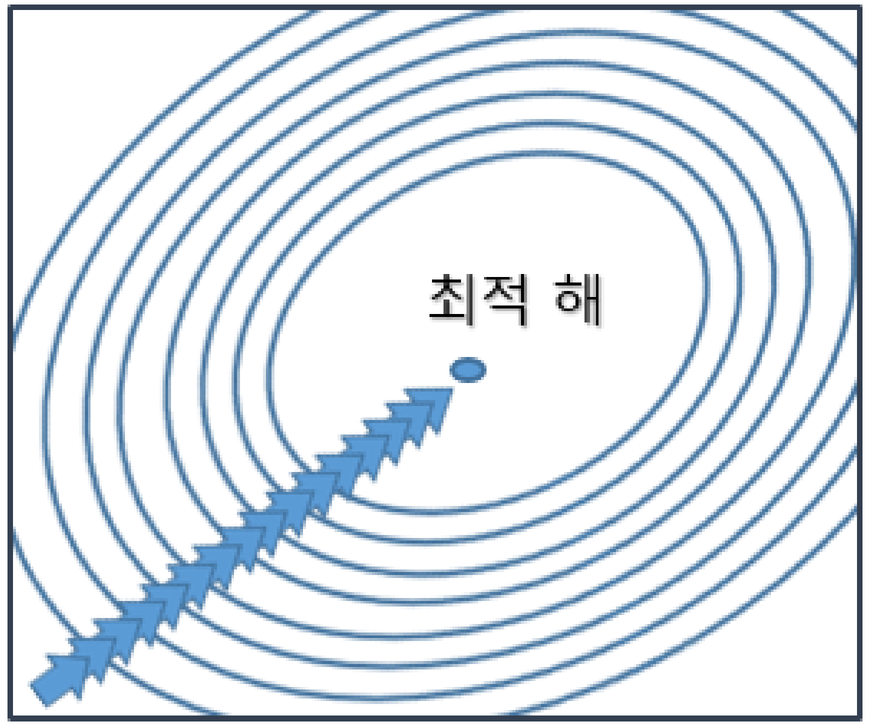
01

# 매개변수 갱신 방법

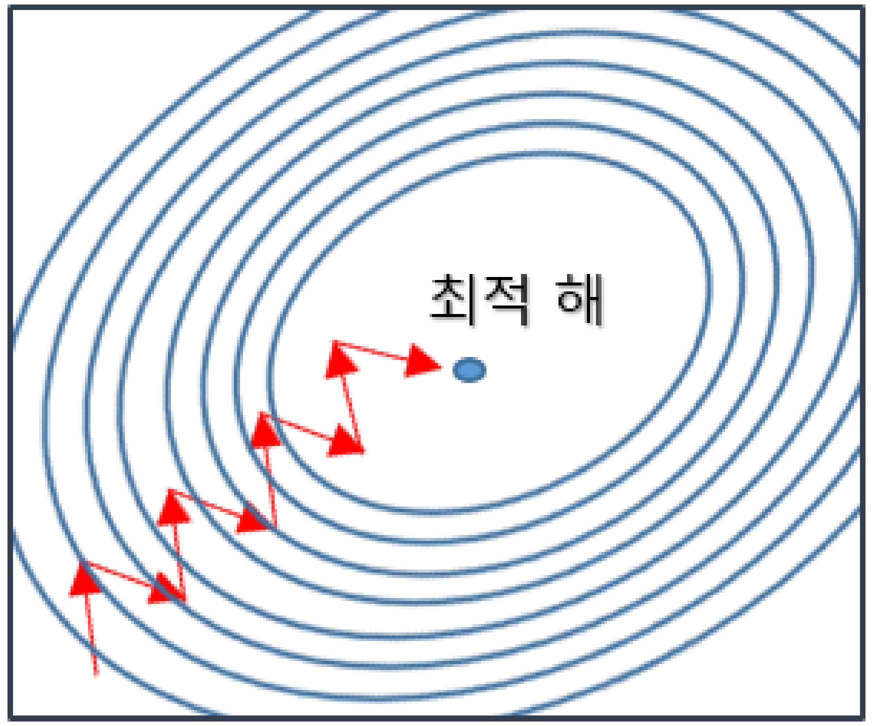
- 확률적 경사 하강법 (Stochastic Gradient Descent)
- 모멘텀 (Momentum)
- AdaGrad (Adaptive Gradient Algorithm)
- Adam (Adaptive Moment Estimation)

# 01) 확률적 경사 하강법(SGD)

-개요




경사 하강법




확률적 경사 하강법

전체 데이터 셋 대신

데이터 셋의 일부인 Quiz 1 를 사용하여 매개변수를 갱신

# 01) 확률적 경사 하강법(SGD)

-수식 및 코드

$$W = W - \eta \frac{\partial L}{\partial W}$$

```
import numpy as np

# SGD 함수 정의
def sgd(w, learning_rate, gradient):
    w_new = w - learning_rate * gradient
    return w_new

# SGD 알고리즘 사용 예시
num_epochs = 100
learning_rate = 0.01
w = np.random.randn(2)  # 가중치 초기화

for epoch in range(num_epochs):
    for x, y in zip(inputs, labels):
        gradient = compute_gradient(x, y, w)  # 기울기 계산
        w = sgd(w, learning_rate, gradient)  # 가중치 업데이트

# 최종 가중치 출력
print(f"Final weights: {w}")
```

# 01) 확률적 경사 하강법(SGD)

## -한계

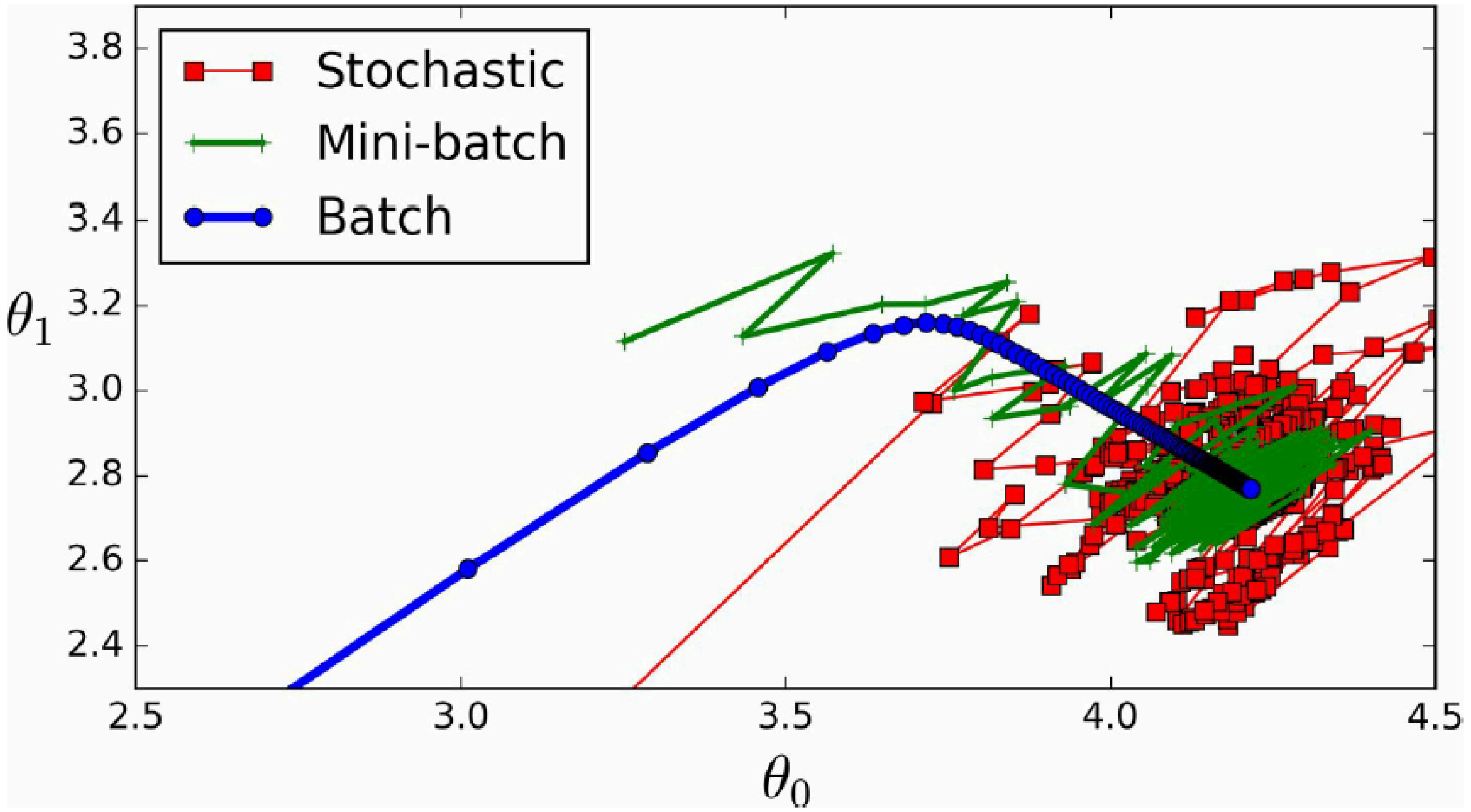
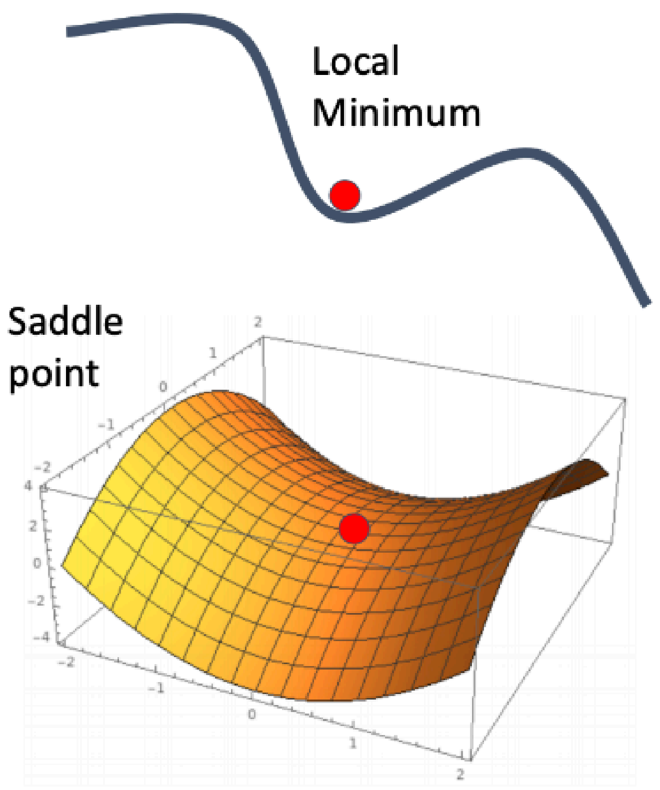


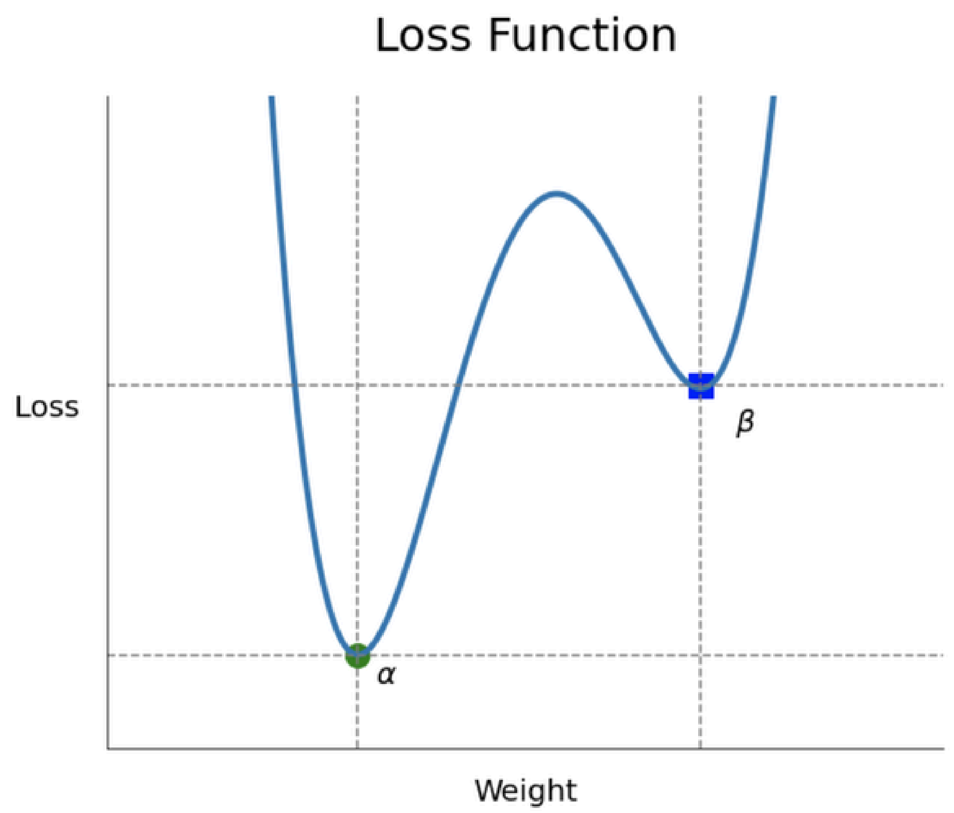

*Figure 4-11. Gradient Descent paths in parameter space*

전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용
따라서 수렴 속도는 빠르지만 **GLOBAL MINIMUM**을 찾지 못할 가능성 존재
데이터 하나씩 처리하기 때문에 오차율이 크고 **GPU**의 성능을 모두 활용하지 못함

# 01) 확률적 경사 하강법(SGD)

## -한계



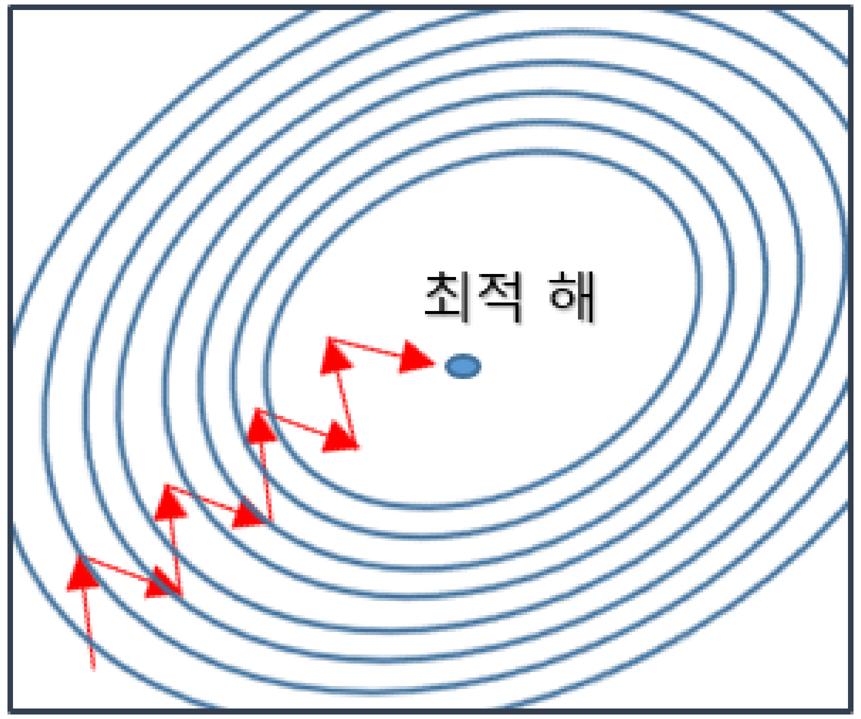




LOCAL MINIMUM에 빠질 위험이 높고
SADDLE POINT의 경우 GRADIENT가 0으로써
UPDATE가 이루지지 않게 됨

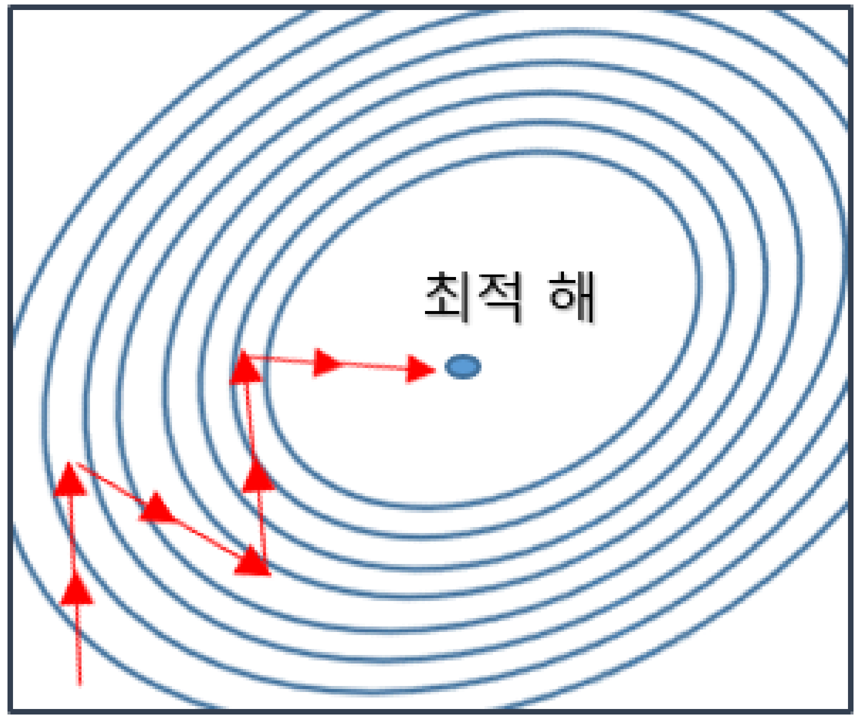
랜덤하게 선택된 가중치가 LOCAL MINIMUM에 가까이 있고
이에 수렴하게되면 실제 목표인 GLOBAL MINIMUM을 찾지 못하고
학습을 중단하는 문제가 발생할 수 있음

# 02) 모멘텀(Momentum)

-개요




확률적 경사 하강법




모멘텀

경사 하강법에 관성을 더해 지그재그 현상을 줄이고,
이전 이동 값을 고려해여 일정 비율만큼 다음 값을 결정

# 02) 모멘텀(Momentum)

-수식

**Momentum**
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자

Velocity term

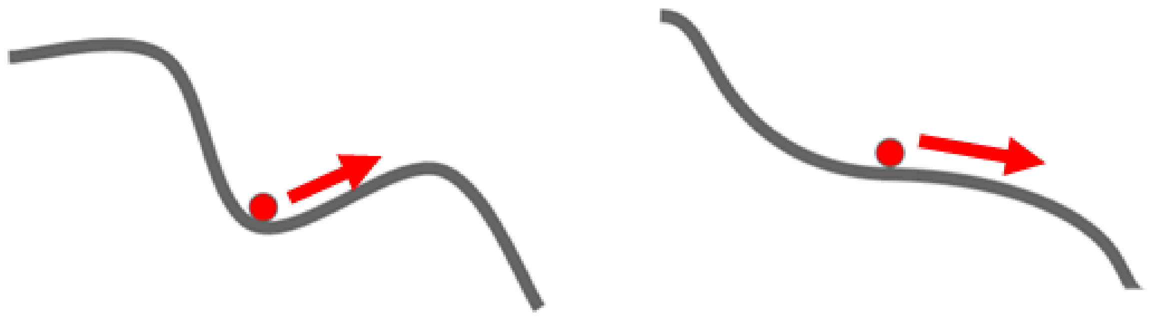
$$\underline{v_{t+1}} = \rho v_t + \boxed{\nabla f(x_t)}$$

$$x_{t+1} = x_t - \underline{\alpha v_{t+1}}$$

Gradient가 계속 동일한 부호를 가지면 v의 절대값은 Quiz 2

-> x의 변화폭이 커짐 -> 같은 방향으로 이동할수록 가속화

| weight의 업데이트 = | 에러 낮추는 방향 (decent) | x | 한발자국 크기 (learning rate) | x | 현 지점의 기울기 (gradient) |
|---|---|---|---|---|---|
| $-\gamma\nabla F(\mathbf{a}^n)$ | $-$ | | $\gamma$ | | $\nabla F(\mathbf{a}^n)$ |
| | | | 보폭 | | 방향 |



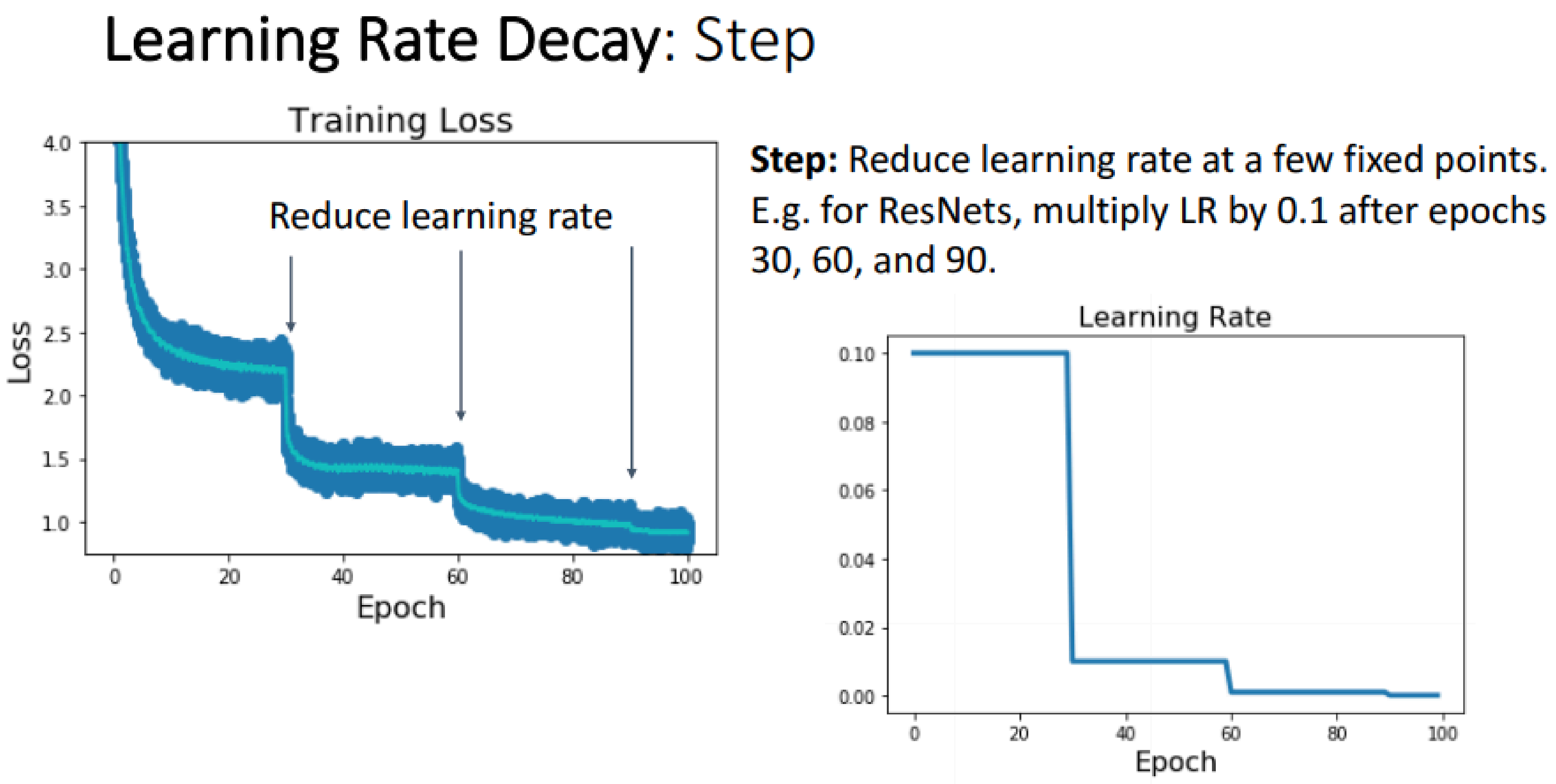
Local Minima　　Saddle point

Gradient가 0이라 하더라도,

v값이 더해지면서 멈추지 않고 옆으로 이동할 수 있다!

# 03) AdaGrad

-개요




개별 매개변수에 적응적으로 학습률을 조정하면서 학습을 진행

# 03) AdaGrad

-수식

**Adagrad**
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색

weight의 업데이트 = 에러 낮추는 방향 (decent) x 한발자국 크기 (learning rate) x 현 지점의 기울기 (gradient)

$-\gamma\nabla F(\mathbf{a}^n)$ = $-$ x $\gamma$ x $\nabla F(\mathbf{a}^n)$

보폭 방향

$$h \leftarrow h + \underbrace{\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}}_{}$$

Squared gradient

$$W \leftarrow W - \eta \boxed{\frac{1}{\sqrt{h}}} \frac{\partial L}{\partial W}$$

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

기울기 제곱에 반비례하도록 학습률 조정

-> 기울기가 가파를수록 조금만 이동 (각 가중치마다 다른 학습률 적용)

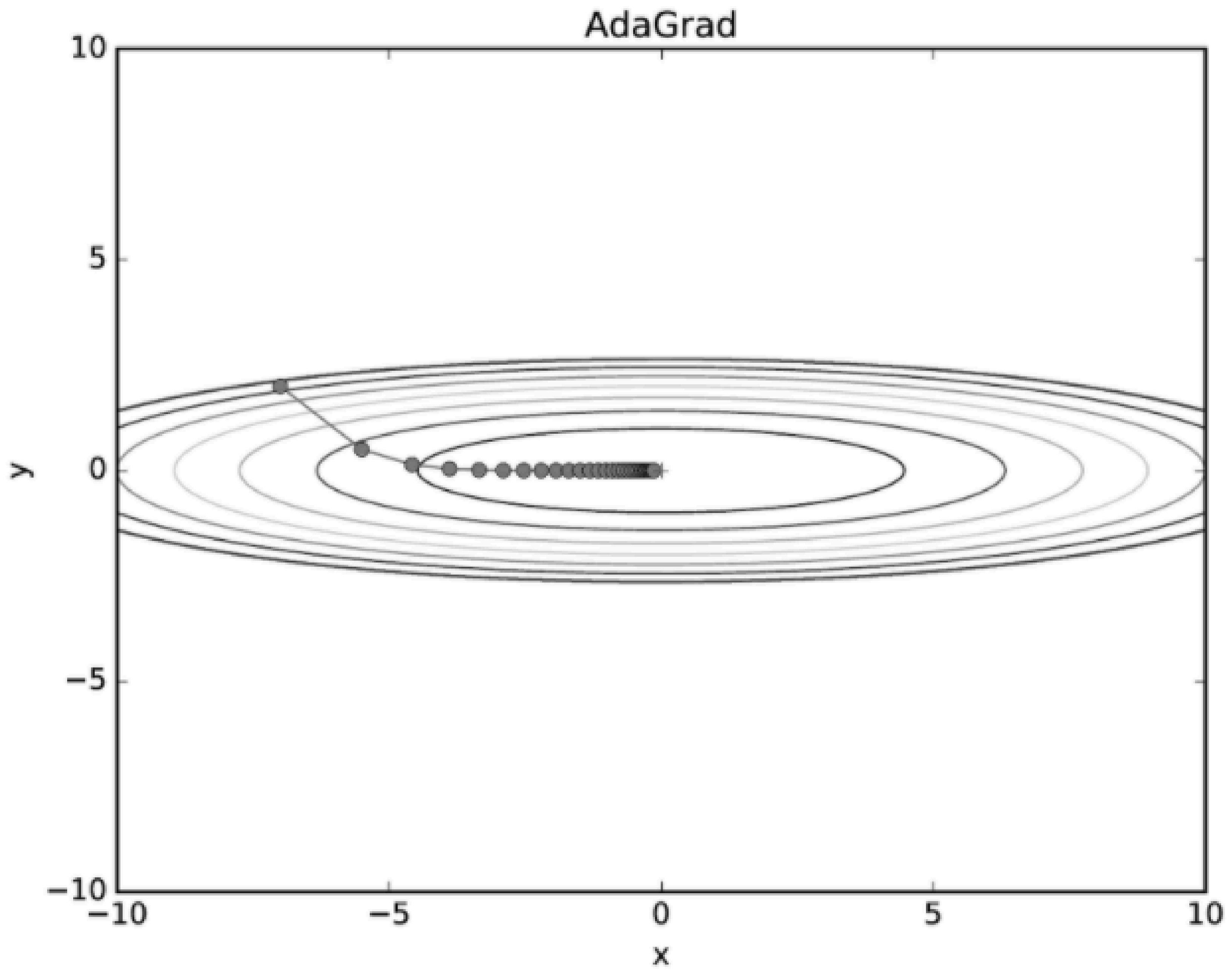
-> 변동을 줄이는 효과

Quiz 3 : h의 값이 무한대까지 커지면
학습률에는 어떤 변화가 있나요?

(-) step이 진행될수록 작아지는 learning rate!

# 03) AdaGrad

-한계



# + RMSProp

**RMSProp**
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.

| weight의 업데이트 = | 에러 낮추는 방향 (decent) | x | 한발자국 크기 (learning rate) | x | 현 지점의 기울기 (gradient) |
|---|---|---|---|---|---|
| $-\gamma \nabla F(\mathbf{a}^n)$ | $-$ | | $\gamma$ | | $\nabla F(\mathbf{a}^n)$ |
| | | | 보폭 | | 방향 |

Decay rate

$$h \leftarrow \boxed{\rho} h + \boxed{(1-\rho)} \underbrace{\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}}_{\text{Squared gradient}}$$

$$W \leftarrow W - \eta \boxed{\frac{1}{\sqrt{h}}} \frac{\partial L}{\partial W}$$

Adagrad에 decay rate (보통 0.9 or 0.99) 추가

-> 이전 기울기를 더 크게 반영하여

h가 단순 누적되어 증가하는 것을 방지

(지수 가중 이동 평균 Exponentially weighted moving average)

-> 무작정 slow down하지 않도록 조정된 것!

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

# 04) Adam

weight의 업데이트 = 에러 낮추는 방향 (decent) x 한발자국 크기 (learning rate) x 현 지점의 기울기 (gradient)

$-\gamma\nabla F(\mathbf{a}^n)$ $-$ $\gamma$ $\nabla F(\mathbf{a}^n)$

보폭 방향

방향

**Momentum**
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자

Nesterov Accelerated Gradient
**NAG** (Nesterov Momentum)
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라

보폭

**Adagrad**
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색

**RMSProp**
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.

**Adam**
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!

Quiz 4 : 방향과 보폭의 학습률을 조정하는 데 활용되는 주요 정보의 차이점은?
(hint : gradient)

Momentum과 Ada를 합쳐보자!

# 04) Adam

**Adam**
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!

| weight의 업데이트 = | 에러 낮추는 방향 (decent) | x | 한발자국 크기 (learning rate) | x | 현 지점의 기울기 (gradient) |
|---|---|---|---|---|---|
| $-\gamma\nabla F(\mathbf{a^n})$ | $-$ | | $\gamma$ | | $\nabla F(\mathbf{a^n})$ |
| | | | 보폭 | | 방향 |

Gradient의 1차 moment에 대한 추정치 $\boxed{m_t} = \beta_1 m_{t-1} + (1-\beta_1)\boxed{g_t}$ Gradient -> momentum

Gradient의 2차 moment에 대한 추정치 $\boxed{v_t} = \beta_2 v_{t-1} + (1-\beta_2)\boxed{g_t^2}$ gradient$^2$ -> Ada

# 04) Adam

**Adam**
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!

Adaptive Moment(적률) Estimation

확률변수 $X$의 $n$차 moment(적률) : $E[X^n]$

1차 moment(적률) : $E[\mathrm{X}]$ 모평균

2차 moment(적률) : $E[X^2]$

$$Var[\mathrm{X}] = E[X^2] - E[\mathrm{X}]^2$$

Gradient의 1차 moment

$$E[Gradient]$$

Gradient의 2차 moment

$$E[Gradient^2]$$

표본평균과 표본제곱평균을 통해 모수인 $E[\mathrm{X}]$와 $E[X^2]$를 추정

# 04) Adam

**Adam**
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!

weight의 업데이트 = 에러 낮추는 방향 (decent) x 한발자국 크기 (learning rate) x 현 지점의 기울기 (gradient)

$-\gamma\nabla F(\mathbf{a^n})$   $-$   $\gamma$   $\nabla F(\mathbf{a^n})$

보폭   방향

Gradient의 1차 moment에 대한 추정치 $\boxed{m_t} = \beta_1 m_{t-1} + (1-\beta_1)\boxed{g_t}$   Gradient -> momentum

Gradient의 2차 moment에 대한 추정치 $\boxed{v_t} = \beta_2 v_{t-1} + (1-\beta_2)\boxed{g_t^2}$   gradient$^2$ -> Ada

$$\boxed{\hat{m}_t} = \frac{m_t}{1-\beta_1^t} \quad ( E[\tilde{m}_t] = E[Gradient^2] )$$

$$\boxed{\hat{v}_t} = \frac{v_t}{1-\beta_2^t} \quad ( E[\tilde{v}_t] = E[Gradient^2] )$$

불편추정치 (unbiased estimate)

: 편향을 잡아주기 위함 (Bias-corrected)

$\beta_1 = 0.9, \beta_2 = 0,999, \epsilon = 10^{-8}$ (best는 1e-3 or 5e-4)

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t+\epsilon}}\hat{m}_t$$

02

# 가중치 초기화

- Lecun 초깃값
- Xavier 초깃값
- He 초깃값

# 가중치 초기화?

## 가중치 초기화의 목적 및 필요성

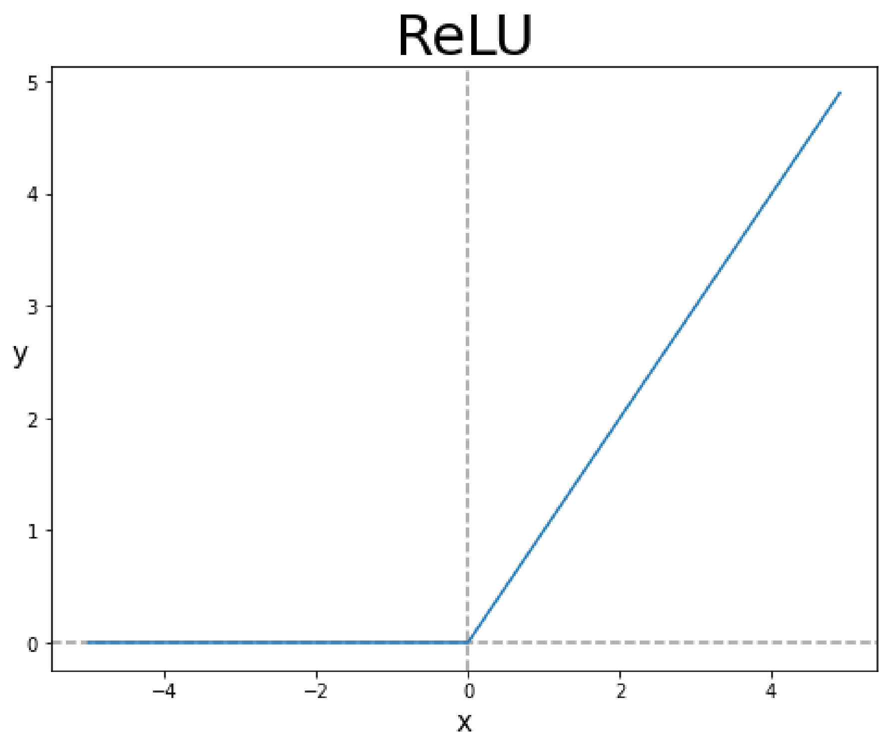
신경망의 가중치 초기화는 모델 학습 시작 전에
가중치를 적절히 설정하는 과정

| | |
|---|---|
| **과적화 감소** | 모델이 훈련 데이터에 과도하게 맞춰지는 것을 방지하여 일반화 성능 향상. |
| **최적화된 초기화** | 네트워크의 깊이와 구조를 고려한 초기화 과정을 통해 학습 안정화 및 성능 향상. |
| **학습 속도 향상** | 모델이 빠르게 수렴하도록 도와 훈련 시간 단축 가능. (Quiz 5 가 너무 크거나 작으면 학습이 느려지거나 수렴하지 않을 수 있기 때문.) |

# 01) Lecun 초깃값

얀 앙드레 르쿤
(Yann André LeCun)

CNN의 창시자이자 LeNet의 창시자




- 주로 ReLU 활성화 함수에 사용됨
- gradient vanishing 문제 개선
- Normal 분포와 Uniform 분포를 따르도록 weight 초기화

# 01) Lecun 초깃값

- 수식

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{1}{n_{in}}}$$

Xavier Normal Initialization

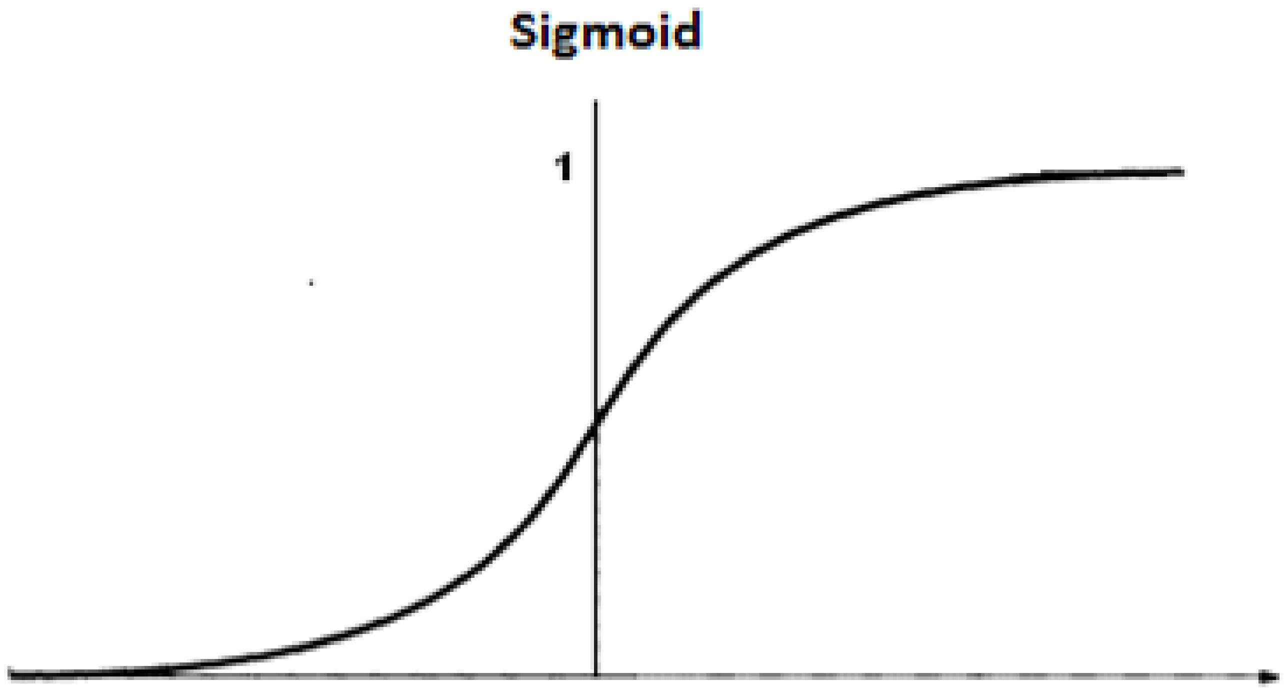
$$W \sim U(-\sqrt{\frac{1}{n_{in}}},\ +\sqrt{\frac{1}{n_{in}}})$$

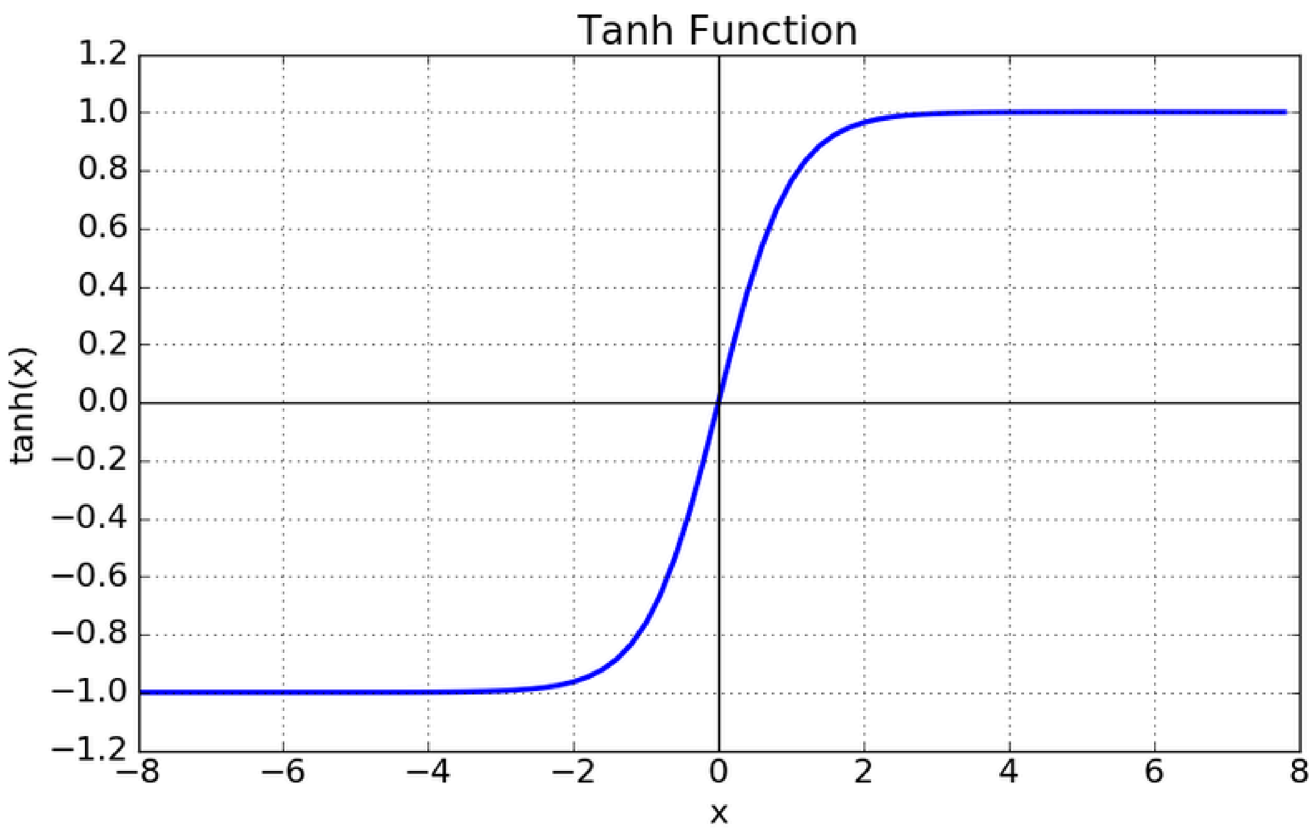
Xavier Uniform Initialization

$n_{in}$ : 이전 layer(input)의 노드 수

# 02) Xavier 초깃값

- 주로 시그모이드 계열의 Quiz 6 함수에 사용
- 가중치의 분산이 입력 데이터의 개수 n에 반비례하도록 초기화
- gradient 소실 문제를 개선하여 신경망의 학습 원활



# 02) Xavier 초깃값

- 가중치 분포는 가우시안 분포(Normal) or 균등분포(Uniform)로 정의
- Xavier Glorot와 Yoshua Bengio가 제안한 방법으로, 입력과 출력의 분산을 동일하게 유지하는 것이 목표

$$X \sim N\left(0, \frac{2}{fan_{in} + fan_{out}}\right)$$

Xavier Normal Initialization

$$X \sim U\left(-\sqrt{\frac{6}{fan_{in} + fan_{out}}}, \sqrt{\frac{6}{fan_{in} + fan_{out}}}\right)$$

Xavier Uniform Initialization

$n_{in}$은 입력 뉴런의 수, $n_{out}$은 출력 뉴런의 수

# 02) Xavier 초깃값

- 가중치 분포는 가우시안 분포(Normal) or 균등분포(Uniform)로 정의
- Xavier Glorot와 Yoshua Bengio가 제안한 방법으로, 입력과 출력의 분산을 동일하게 유지하는 것이 목표

Xaiver함수는 비선형 함수(ex. sigmoid, tanh)에서 효과적인 결과를 보이지만,

ReLU함수에서는 출력 값이 **0으로 수렴!!**

$$X \sim N(0, \frac{2}{fan_{in} + fan_{out}})$$

Xavier Normal Initialization

$$X \sim U(-\sqrt{\frac{6}{fan_{in} + fan_{out}}}, \sqrt{\frac{6}{fan_{in} + fan_{out}}})$$

Xavier Uniform Initialization

>> ReLU함수에는 **또 다른 초기화 방법** 적용

$n_{in}$은 입력 뉴런의 수, $n_{out}$은 출력 뉴런의 수

# 03) He 초깃값

- Xavier Initialization의 비효율성을 개선한 초기화방식
- ReLU 및 그 변형(예: Leaky ReLU, PReLU 등)에 적합하게 설계됨
- gradient 소실 문제 개선

- 코드 활용 예시

```
# 균등 분포
torch.nn.init.kaiming_uniform_(tensor, a=0, mode='fan_in', nonlinearity='leaky_relu', generator=None)
```

```
# 정규 분포
torch.nn.init.kaiming_normal_(tensor, a=0, mode='fan_in', nonlinearity='leaky_relu', generator=None)
```

** Kaiming He는 컴퓨터 비전과 딥러닝을 연구하는 중국 컴퓨터 과학자

# 참고 자료


- https://twinw.tistory.com/247
- https://huangdi.tistory.com/6 https://gr-st-dev.tistory.com/150
- https://nonmeyet.tistory.com/entry/Batch-MiniBatch-Stochastic-%EC%A0%95%EC%9D%98%EC%99%80-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EC%98%88%EC%8B%9C
- https://velog.io/@seoyeon/Lecture-4-Optimization-sgd-momentum
- https://twinw.tistory.com/247
- https://velog.io/@good159897/Learning-rate-Decay%EC%9D%98-%EC%A2%85%EB%A5%98
- https://amber-chaeeunk.tistory.com/23
- https://velog.io/@yookyungkho/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EC%98%B5%ED%8B%B0%EB%A7%88%EC%9D%B4%EC%A0%80-%EC%A0%95%EB%B3%B5%EA%B8%B0%EB%B6%80%EC%A0%9C-CS231n-Lecture7-Review

대학생 연합 빅데이터 동아리

# 감사합니다

강나영 김제민 배성윤 이형석 한진솔

2024-08-03