Toward Optimal Optimizer

DMQA Open Seminar

2022. 06. 17

Data Mining & Quality Analytics Lab.

임새린

KOREA UNIVERSITY Data Mining Quality Analytics

발표자 소개

❖ 임새린 (Saerin Lim)

- 고려대학교 산업경영공학과 Data Mining & Quality Analytics Lab.
- M.S. Student (2021.03 ~ Present)
- 지도 교수 : 김성범 교수님

❖ Research Interest

- Multivariate time-series data analysis
- Self-supervised learning

❖ Contact

- E-mail : momo_om@korea.ac.kr

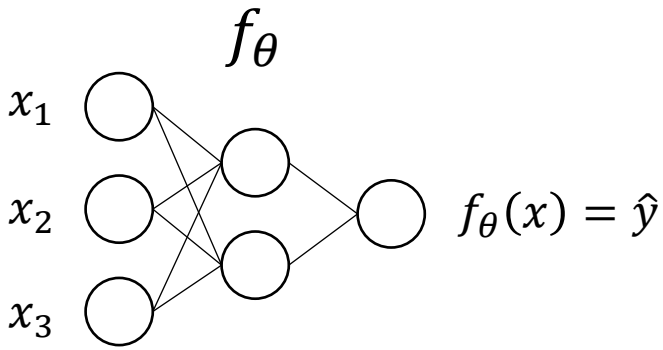
Contents

Introduction

❖ 모델을 학습하기 위한 주요 요소

obs	X1	X2	X3	Y
1	3	2.7	33	0
2	2	2.1	45	1
3	5	1.6	43	1
		⋮		
n	4	2.5	39	0

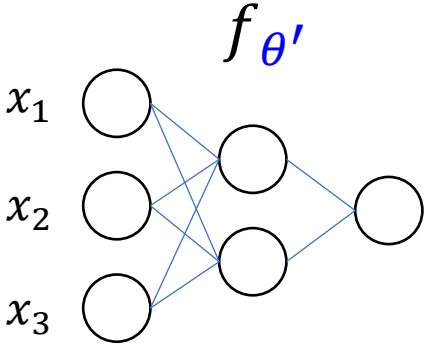
Data



Model

$$L(\hat{y}, y)$$

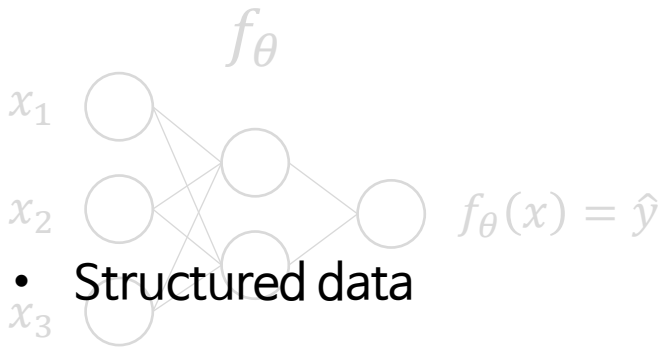
Loss function



Optimizer

Introduction

❖ 모델을 학습하기 위한 주요 요소: Data



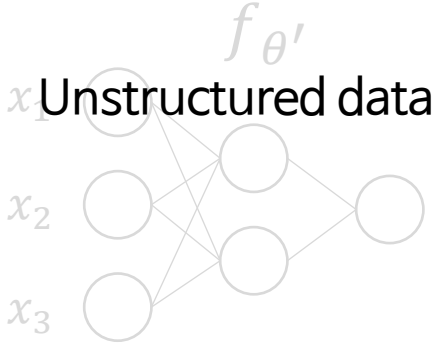
Model

obs	X1	X2	X3	Y
1	3	2.7	33	0
2	2	2.1	45	1
3	5	1.6	43	1
		⋮		
n	4	2.5	39	0

Data

- Structured data
- Unstructured data

Loss function

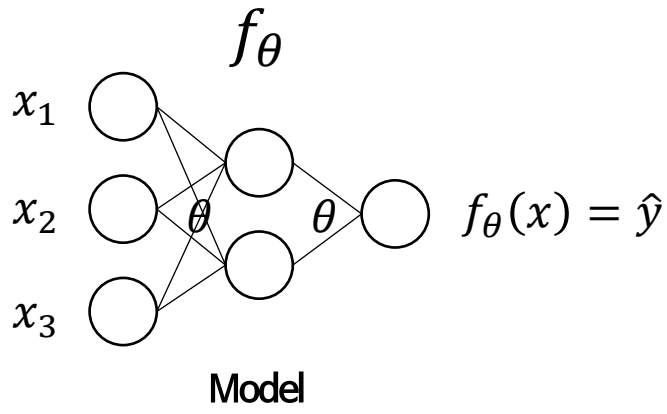


Optimizer

"Hello, deep learning"

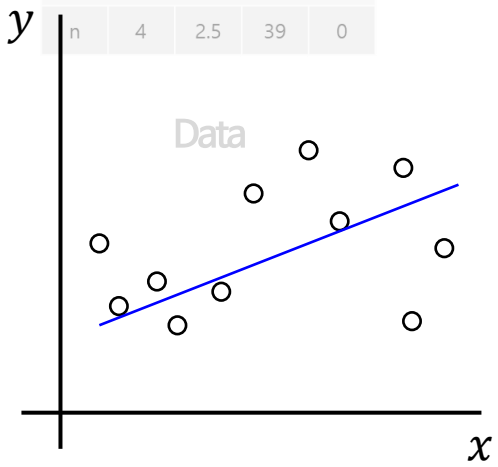
Introduction

❖ 모델을 학습하기 위한 주요 요소: Model

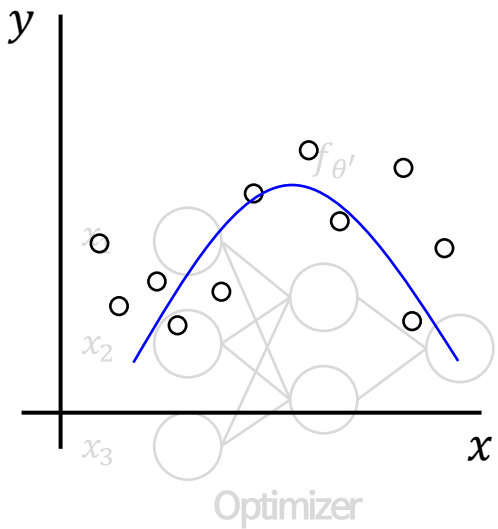


obs	X1	X2	X3	Y
1	3	2.7	33	0
2	2	2.1	45	1
3	5	1.6	43	1
n	4	2.5	39	0

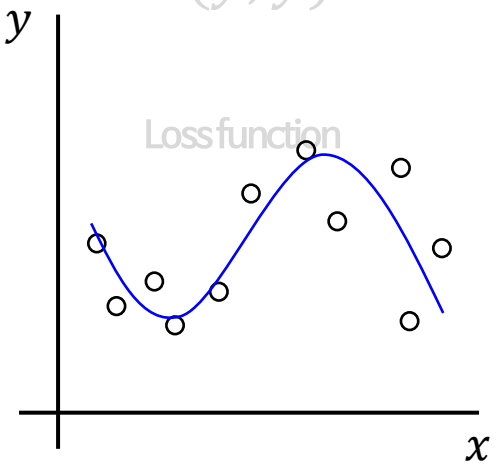
$y = \theta_1 x + b$



$y = \theta_1 x^2 + \theta_2 x + b$

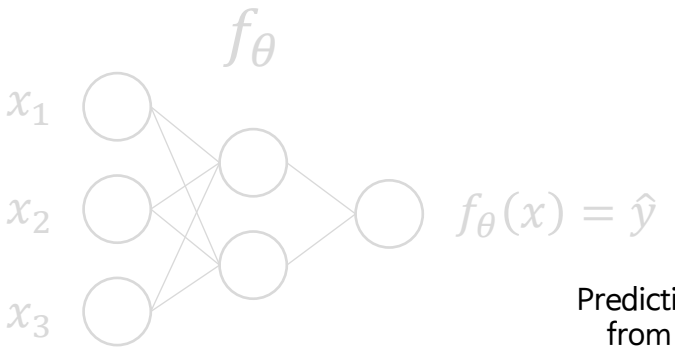


$y = \theta_1 x^3 + \theta_2 x^2 + \theta_3 x + b$



Introduction

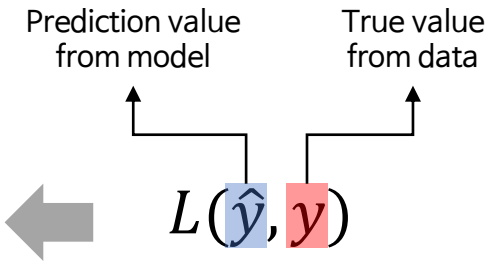
❖ 모델을 학습하기 위한 주요 요소: Loss Function



obs	X1	X2	X3	Y
1	3	2.7	33	0
2	2	2.1	45	1
3	5	1.6	43	1
		⋮		
n	4	2.5	39	0

Data

Penalty for failing to achieve a desired value



Loss function

Mean Squred Error

Categorical Cross Entropy Loss

Binary Cross Entropy Loss

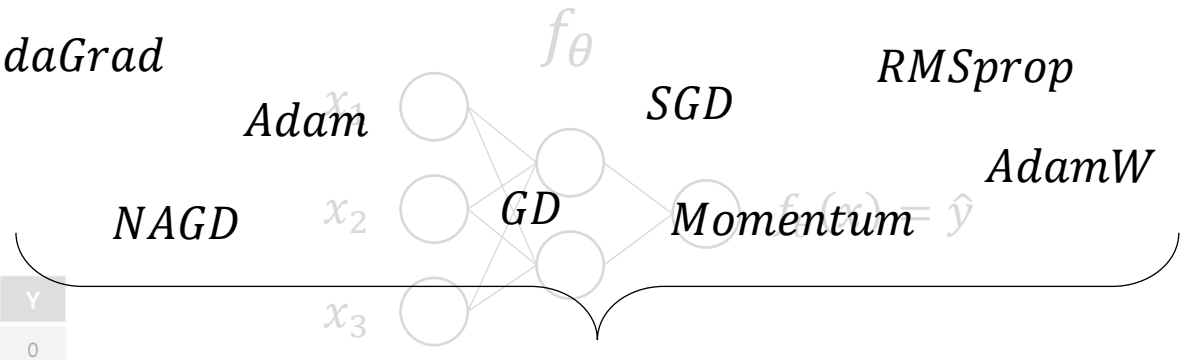
Contrastive Loss

Focal Loss

Triplet Loss

Introduction

❖ 모델을 학습하기 위한 주요 요소: Optimizer

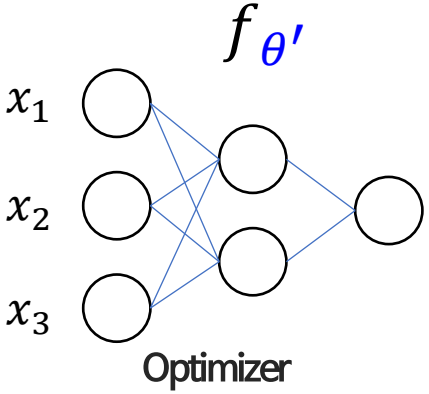


Update current parameter θ to θ'
to minimize penalty

obs	X1	X2	X3	Y
1	3	2.7	33	0
2	2	2.1	45	1
3	5	1.6	43	1
		⋮		
n	4	2.5	39	0

Data

Loss function

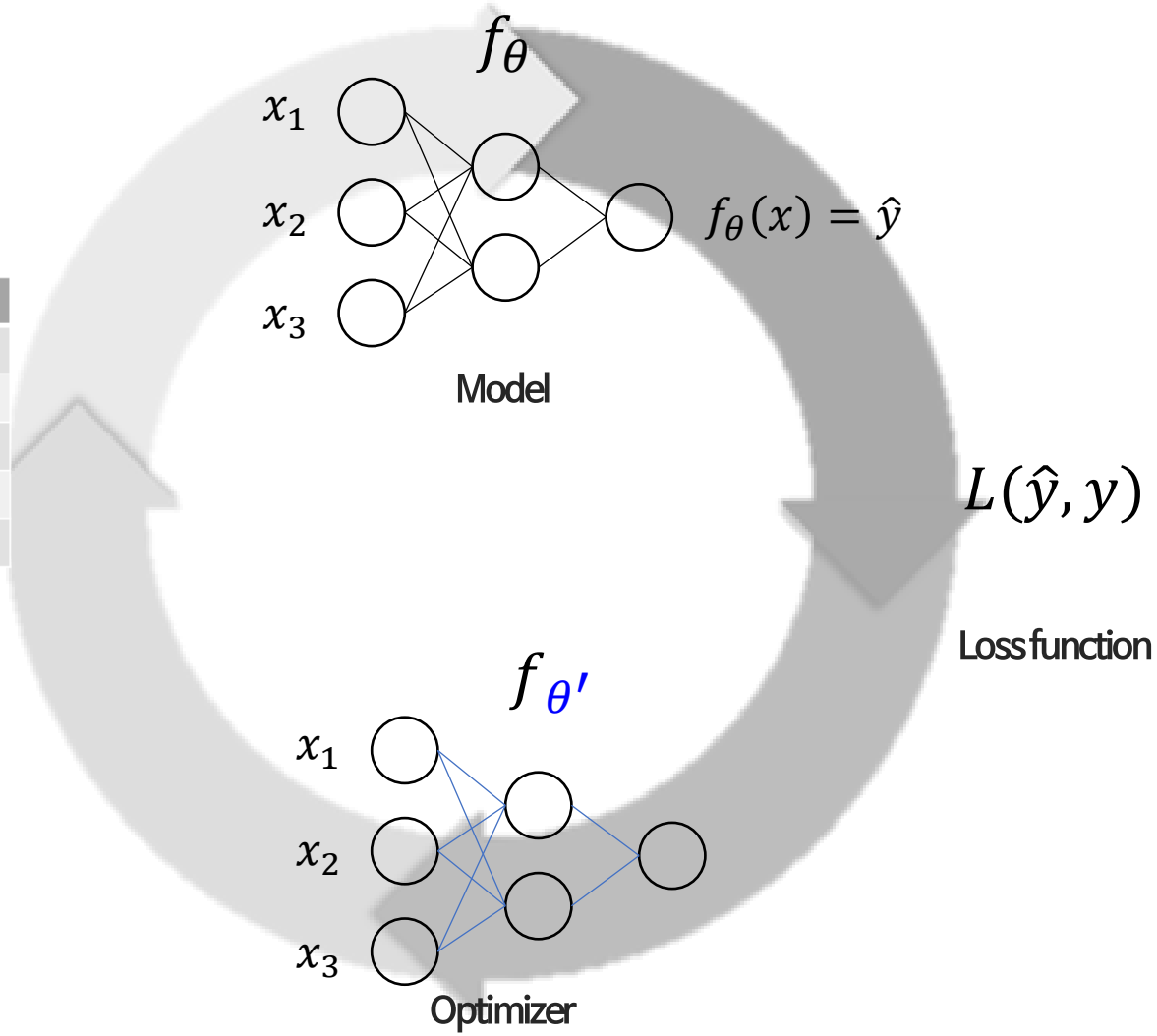


Introduction

❖ 모델 학습

obs	X1	X2	X3	Y
1	3	2.7	33	0
2	2	2.1	45	1
3	5	1.6	43	1
		⋮		
n	4	2.5	39	0

Data



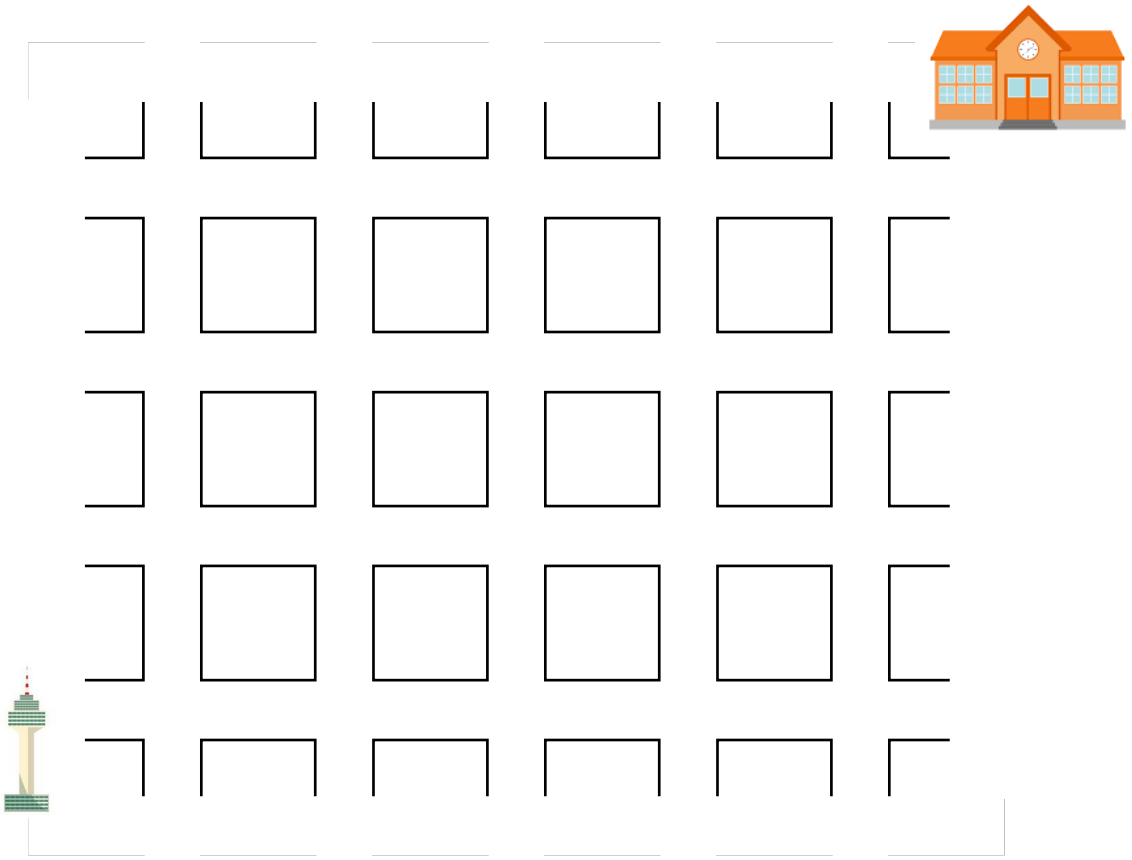
Introduction

❖ 모델 학습

- 학습이란 실제값과 예측값 사이의 차이를 줄일 수 있도록 **점진적으로** 파라미터를 업데이트하면서 최적의 파라미터를 찾아가는 과정
- Optimizer는 파라미터의 업데이트를 담당하는 중요한 요소로 성능에 직접적인 영향을 줌
- 다양한 Optimizer를 이해하고 상황에 알맞게 사용할 수 있어야 함

Definition of Optimizer

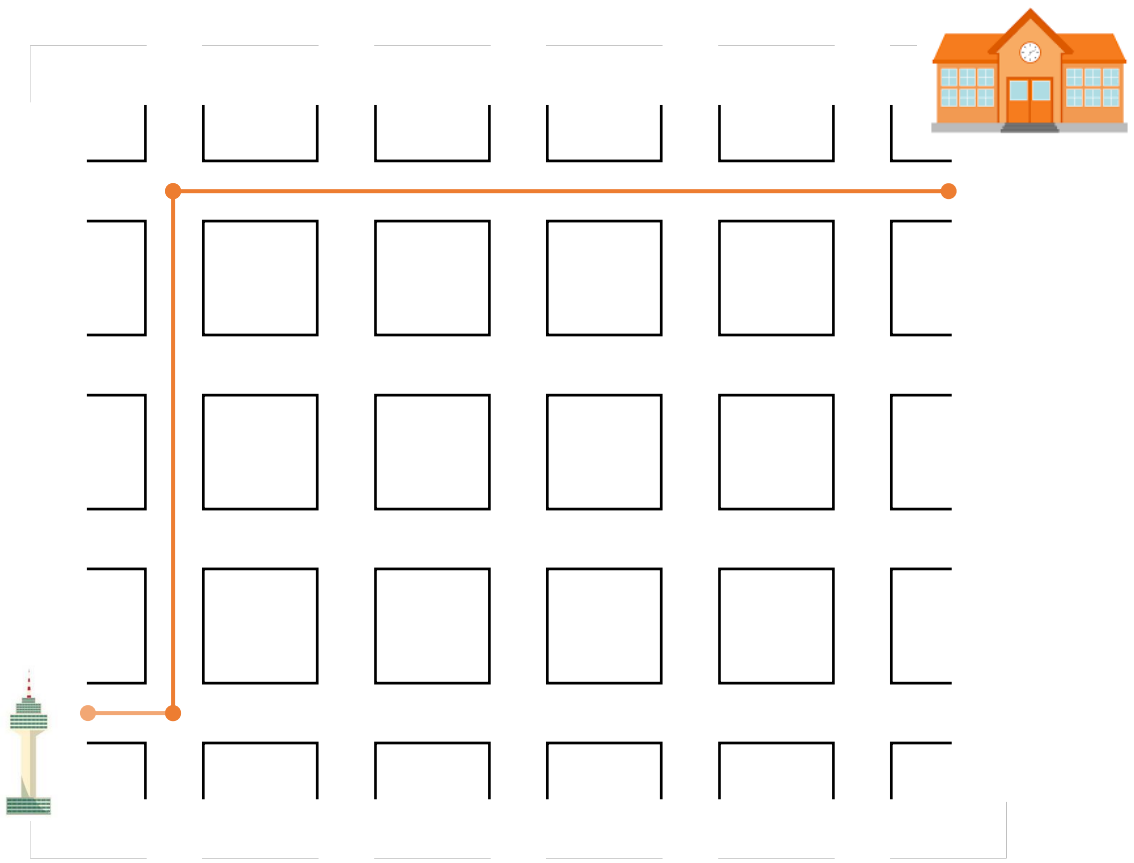
❖ 고려대학교에서 남산타워를 가는 방법



Definition of Optimizer

❖ 고려대학교에서 남산타워를 가는 방법

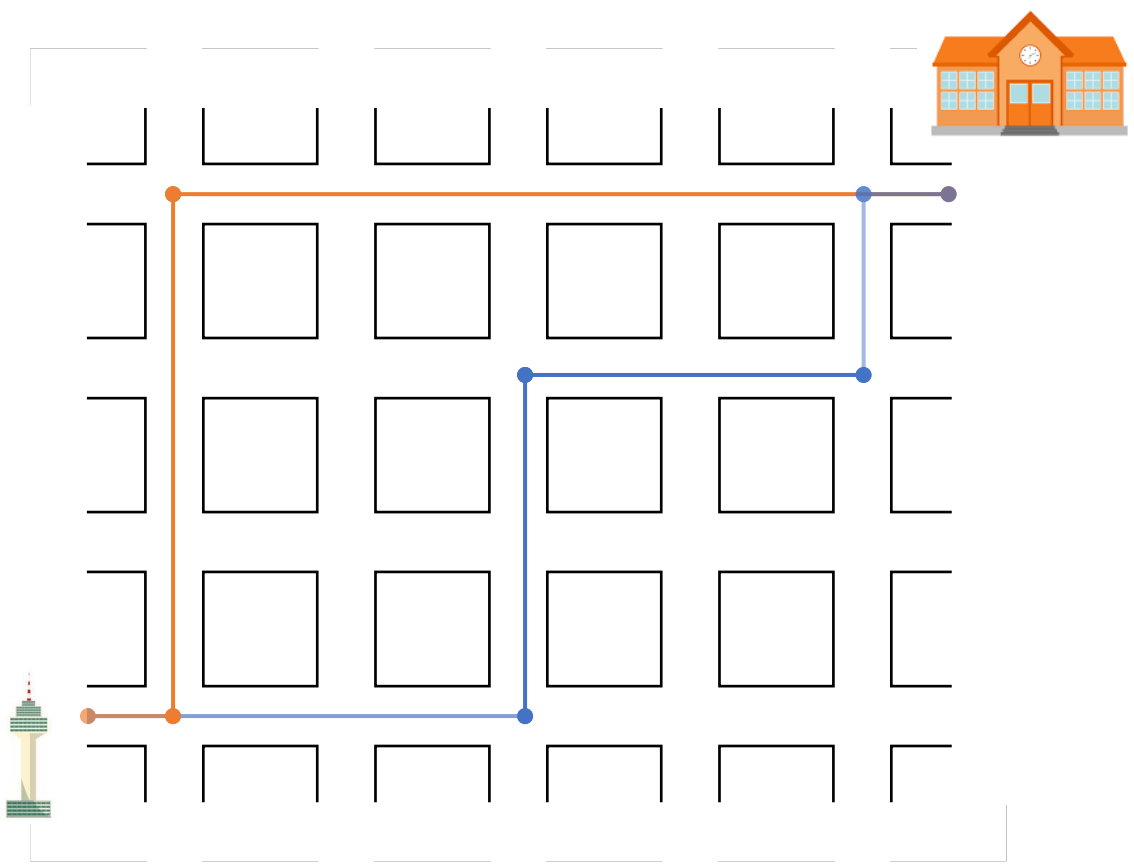
- 경로 1



Definition of Optimizer

❖ 고려대학교에서 남산타워를 가는 방법

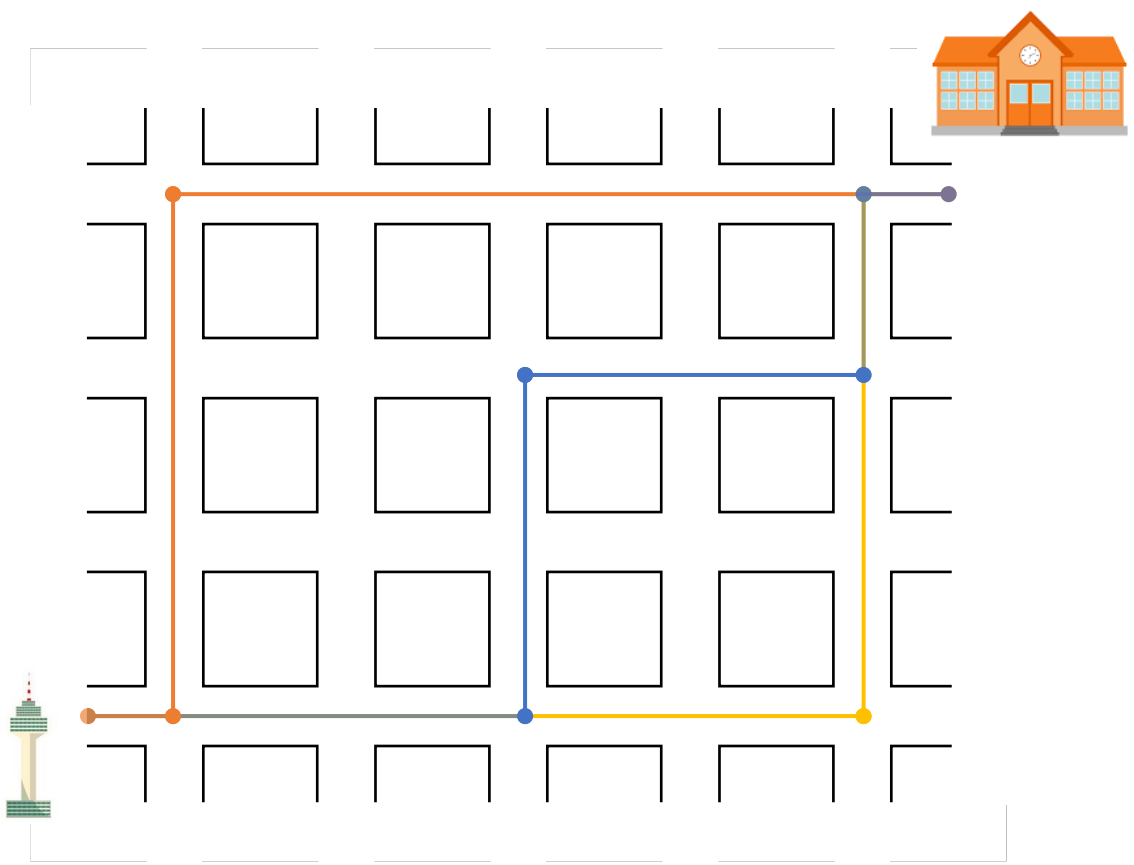
- 경로 1
- 경로 2



Definition of Optimizer

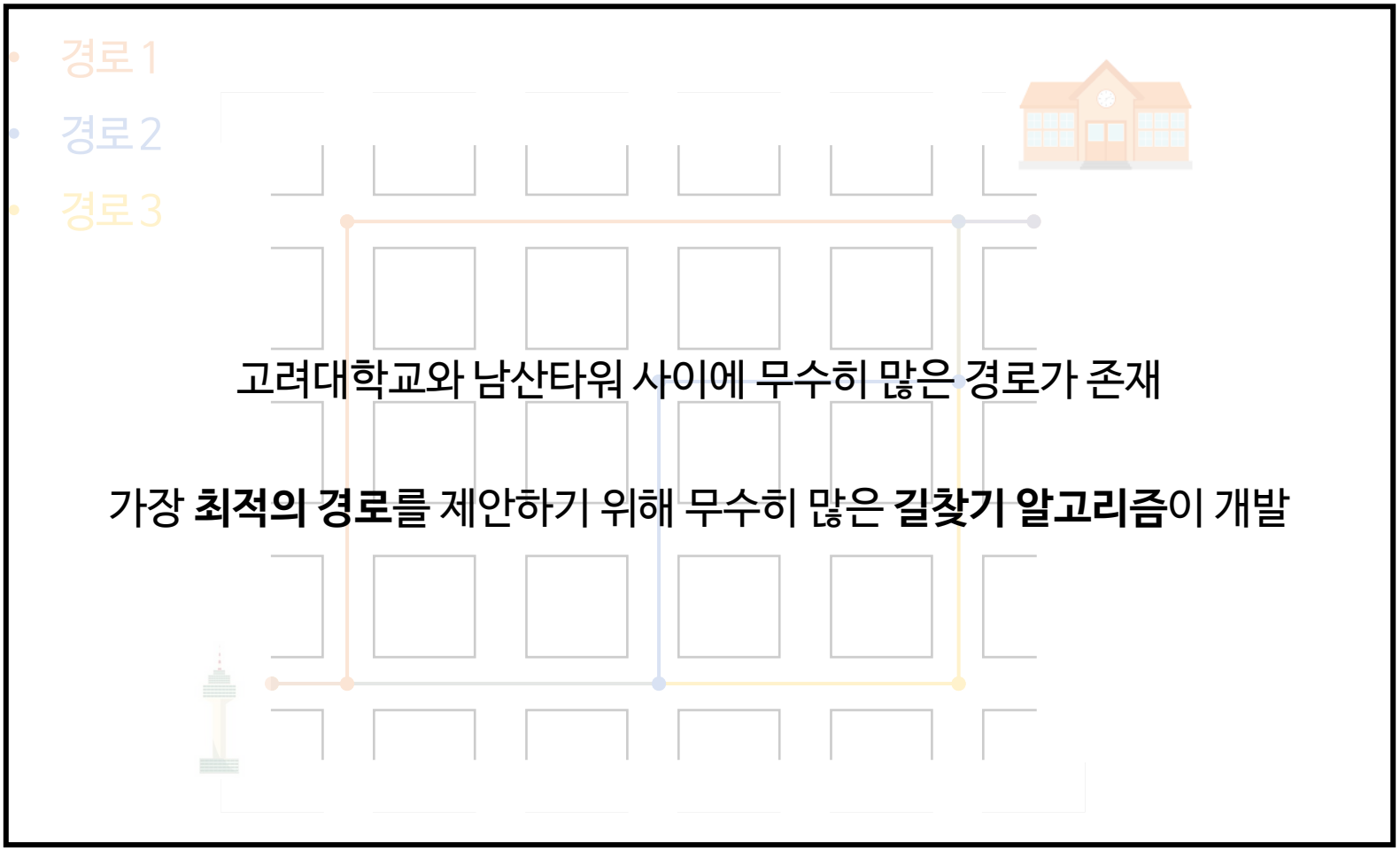
❖ 고려대학교에서 남산타워를 가는 방법

- 경로 1
- 경로 2
- 경로 3



Definition of Optimizer

❖ 고려대학교에서 남산타워를 가는 방법



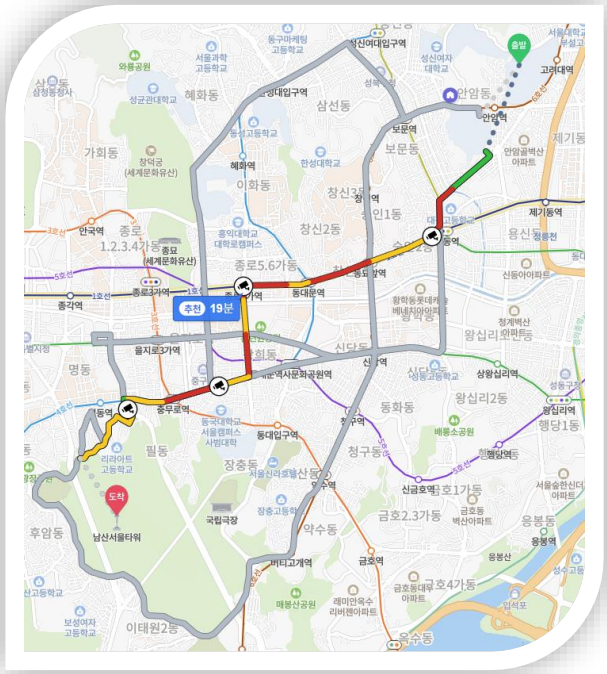
고려대학교와 남산타워 사이에 무수히 많은 경로가 존재

가장 **최적의 경로**를 제안하기 위해 무수히 많은 **길찾기 알고리즘**이 개발

Definition of Optimizer

❖ 고려대학교에서 남산타워를 가는 방법

- 고려대학교에서 남산타워를 가는 방법을 서로 다른 지도 어플로 검색
- 두 지도 어플의 **길찾기 알고리즘**이 다르기 때문에 서로 다른 길을 추천



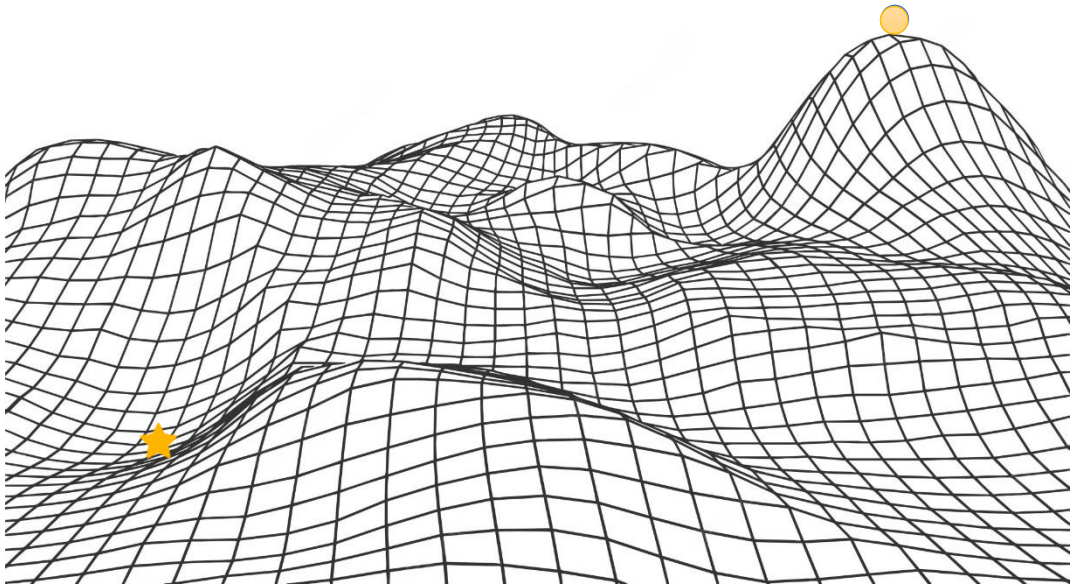
N사 지도 어플

K사 지도 어플

Definition of Optimizer

❖ Optimizer란?

- Optimizer는 비용함수를 최소화 시키는 최적의 모델 파라미터를 찾아주는 길찾기 알고리즘!!
- 최적의 모델 파라미터로 가는 경로 역시 다양하기 때문에 Optimizer도 여러 알고리즘으로 개발

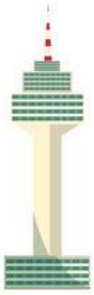


Definition of Optimizer

❖ Optimizer란?

= Optimizer

= Current parameters

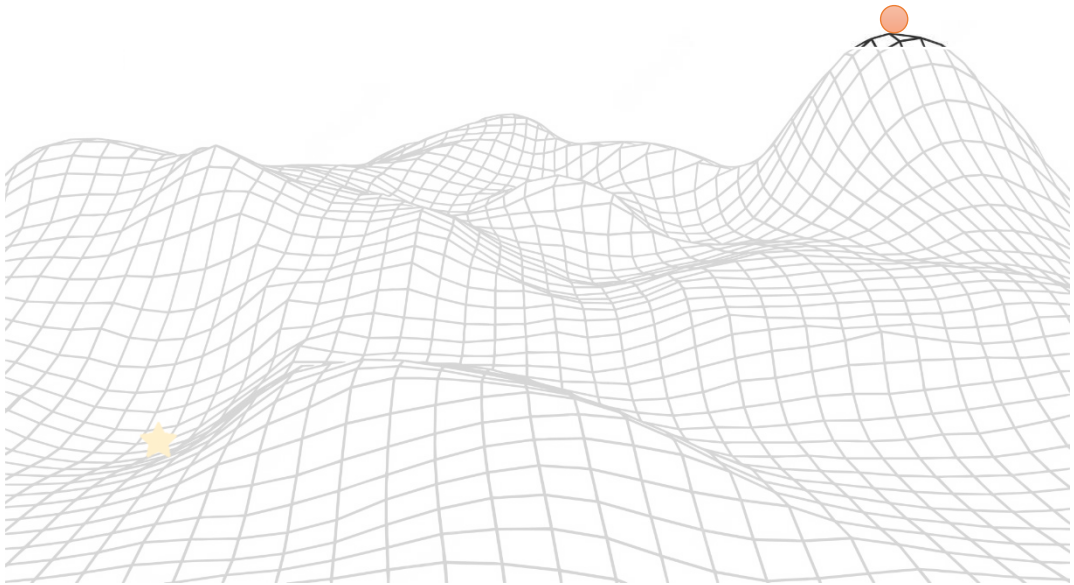


= Optimal parameters

Definition of Optimizer

❖ Optimizer란?

- 지도 어플은 목적지의 좌표를 알고 있지만 Optimizer는 최적해는 물론 비용함수 형태도 알 수 없음

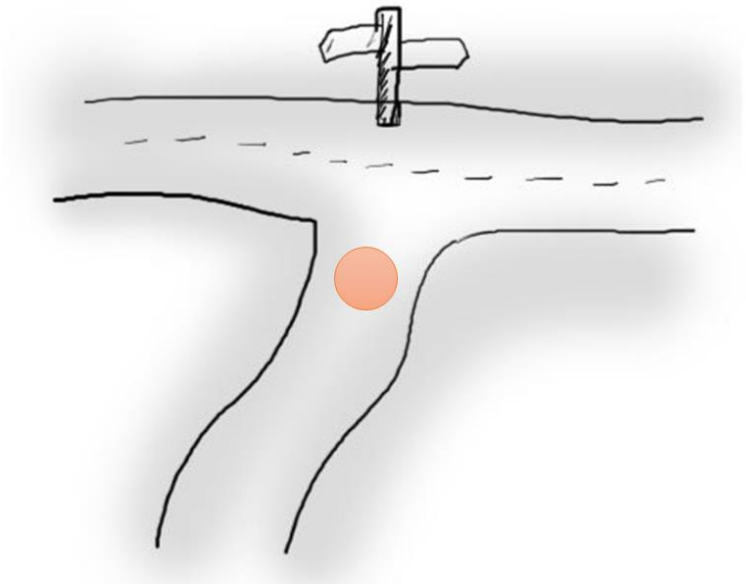


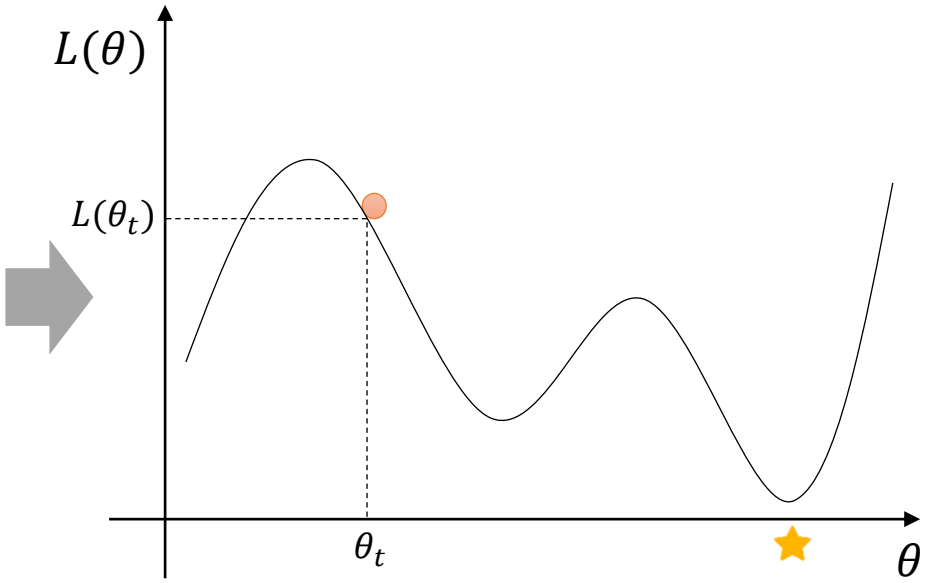
Definition of Optimizer

❖ Optimizer란?

- 지도 어플은 목적지의 좌표를 알고 있지만 Optimizer는 최적해는 물론 비용함수 형태도 알 수 없음
- 하지만 현재 위치와 관련된 **표지판**은 존재!!

표지판



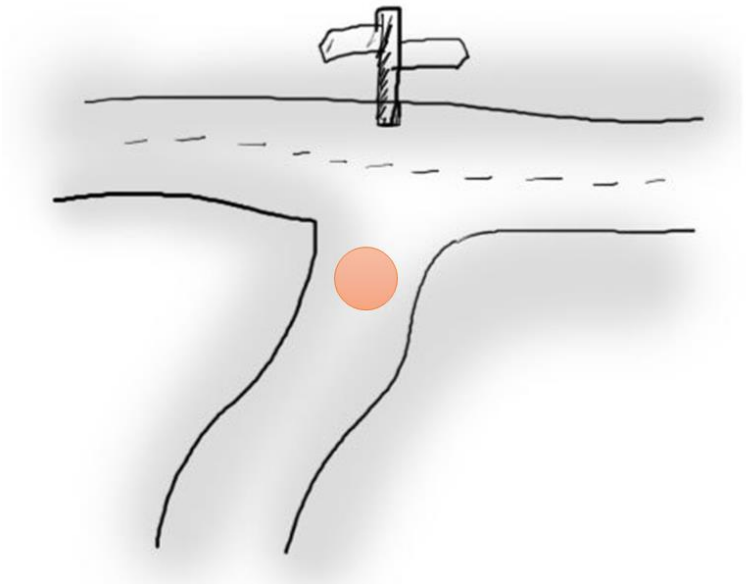


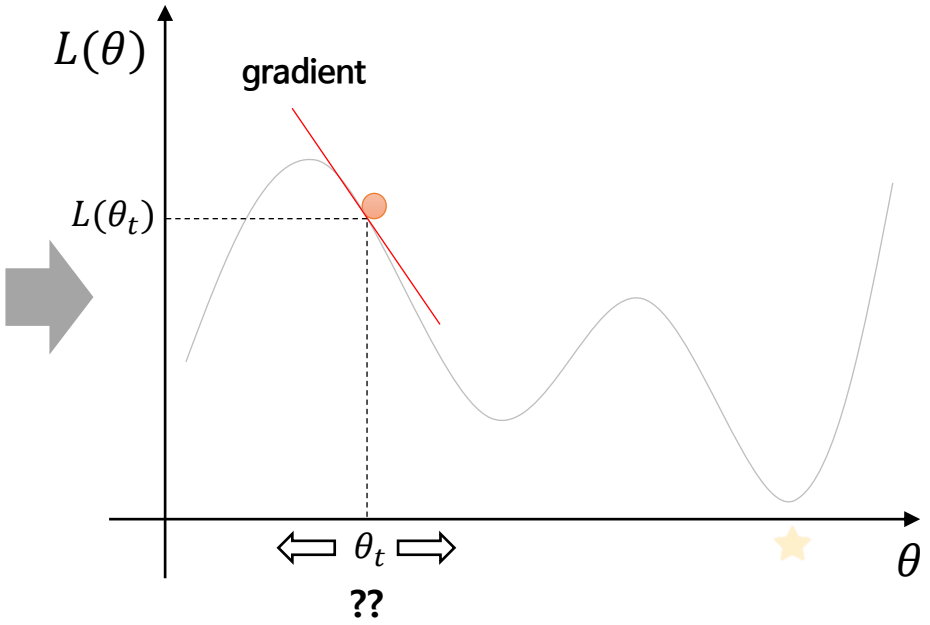
Definition of Optimizer

❖ Optimizer란?

- 지도 어플은 목적지의 좌표를 알고 있지만 Optimizer는 최적해는 물론 비용함수 형태도 알 수 없음
- 하지만 현재 위치와 관련된 **표지판**은 존재!!

표지판



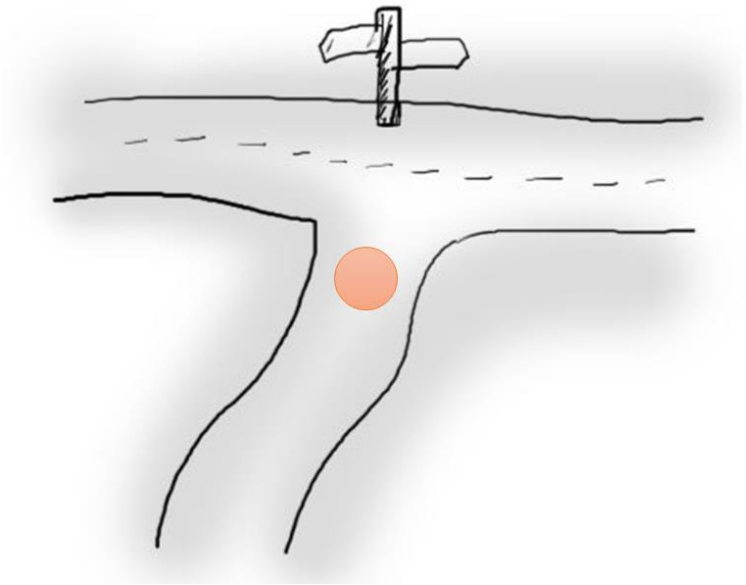


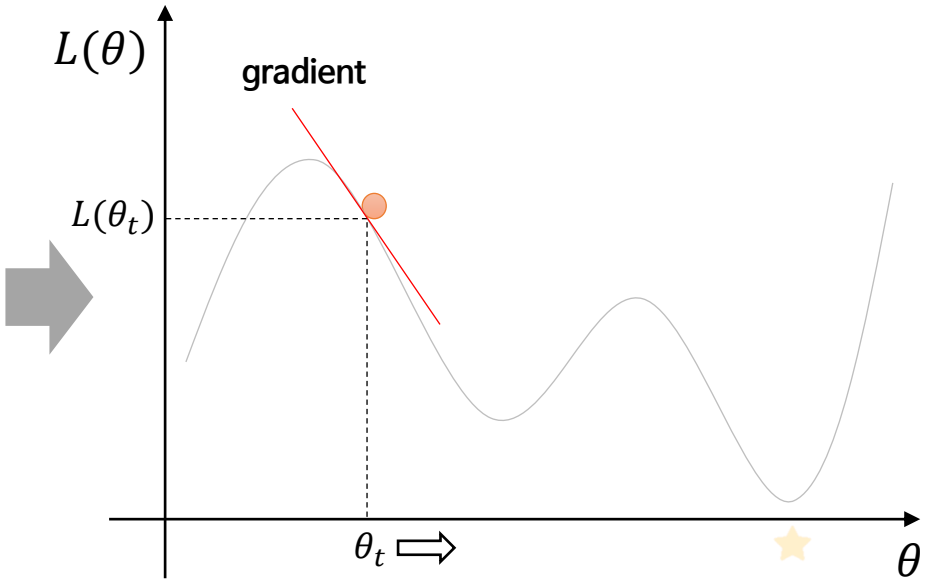
Definition of Optimizer

❖ Optimizer란?

- 지도 어플은 목적지의 좌표를 알고 있지만 Optimizer는 최적해는 물론 비용함수 형태도 알 수 없음
- 하지만 현재 위치와 관련된 **표지판**은 존재!!

표지판

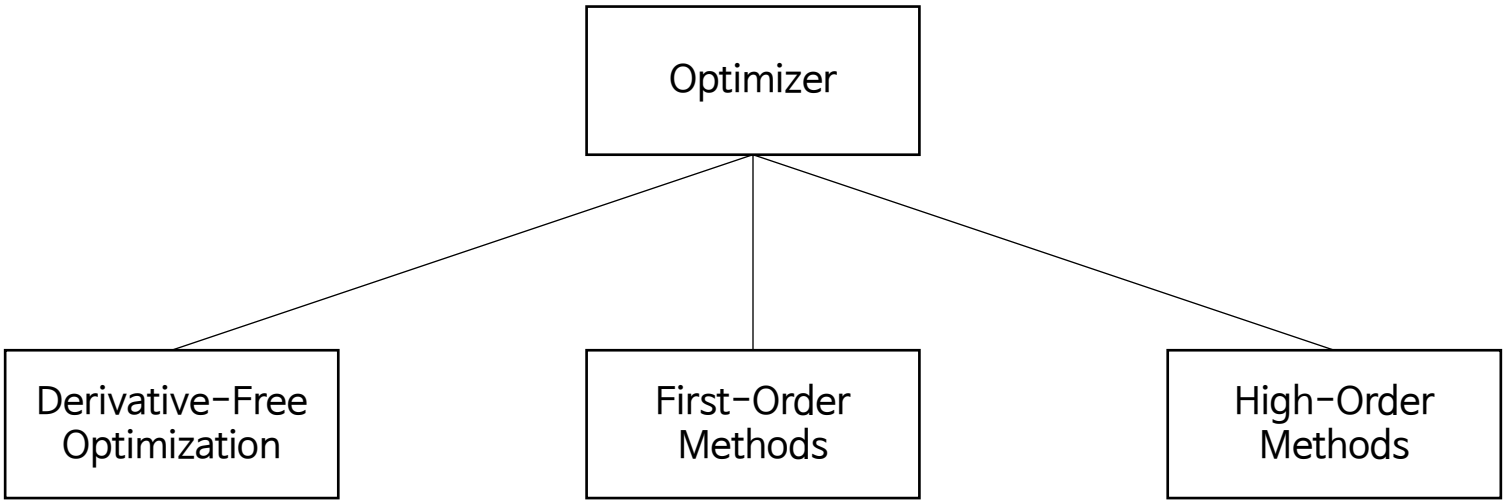




Definition of Optimizer

❖ Types of Optimizer

- 미분 정보를 어디까지 어떻게 사용하는지에 따라 분류



- 목적함수의 미분이 불가능한 상황에서 사용하는 최적화 알고리즘

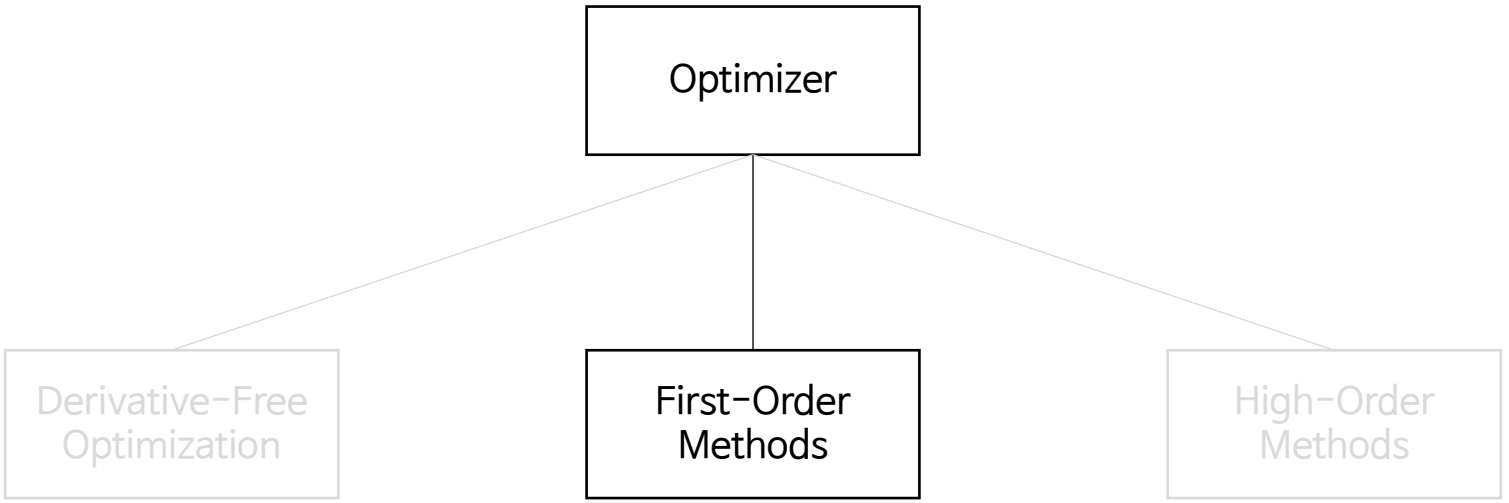
- 1차 미분 정보인 gradient 활용
- 가장 일반적인 형태

- 고차원 미분 정보(Hessian 등) 활용
- 목적함수가 highly non-linear and ill-conditioned 일 때 사용
- 계산 비용이 큼

Definition of Optimizer

❖ Types of Optimizer

- 미분 정보를 어디까지 어떻게 사용하는지에 따라 분류



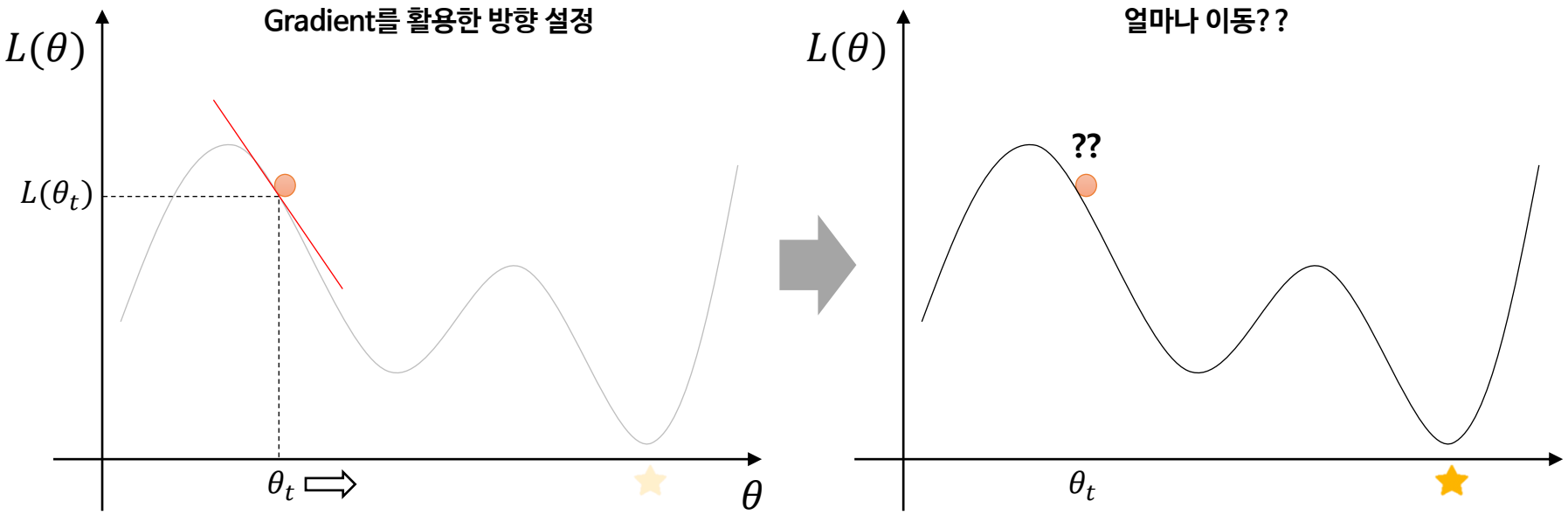
- 목적함수의 미분이 불가능한 상황에서 사용하는 최적화 알고리즘

- 1차 미분 정보인 gradient 활용
- 가장 일반적인 형태

- 고차원 미분 정보(Hessian 등) 활용
- 목적함수가 highly non-linear and ill-conditioned 일 때 사용
- 계산 비용이 큼

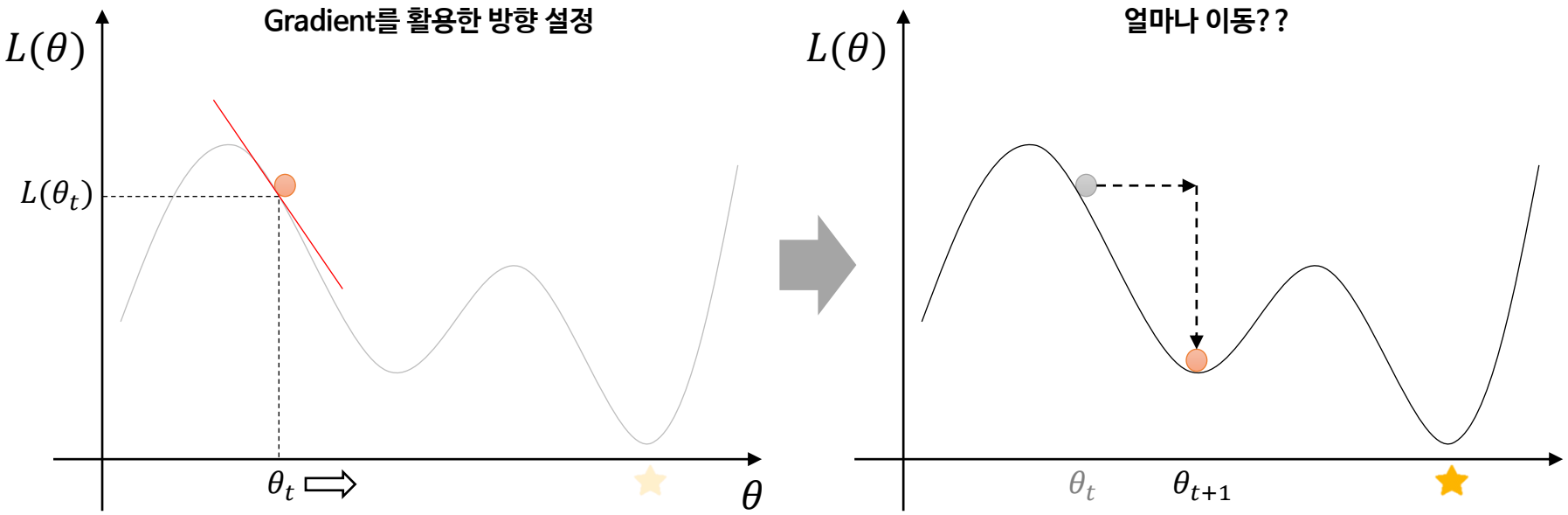
Definition of Optimizer

❖ Types of Optimizer



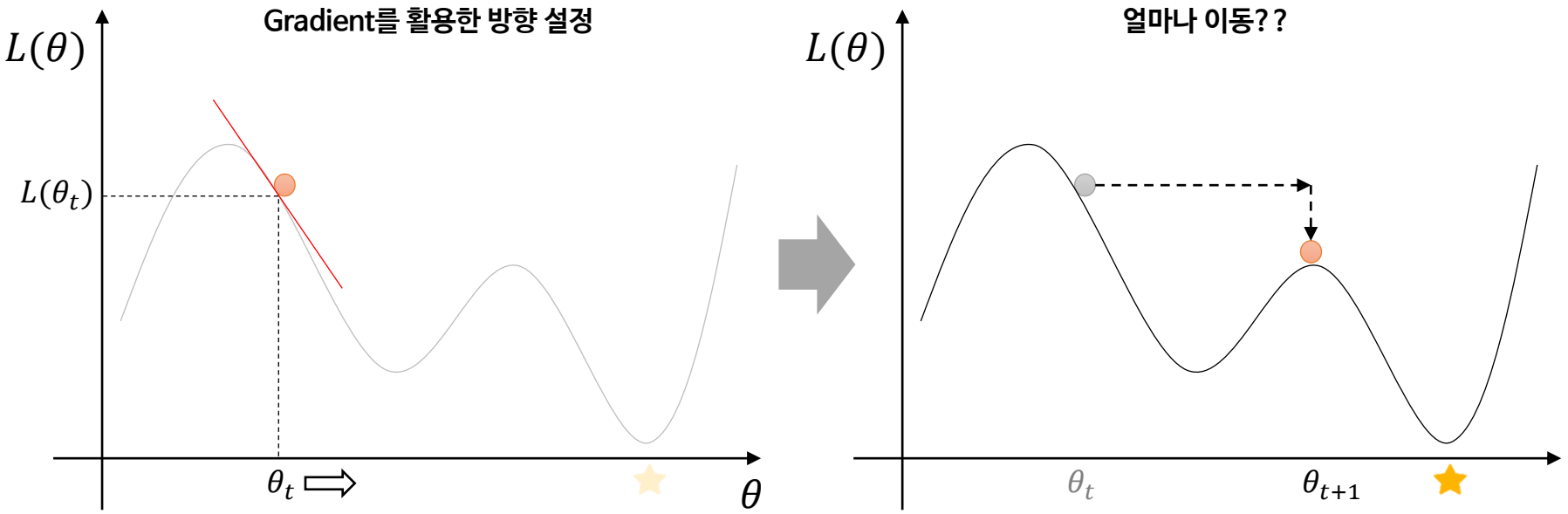
Definition of Optimizer

❖ Types of Optimizer



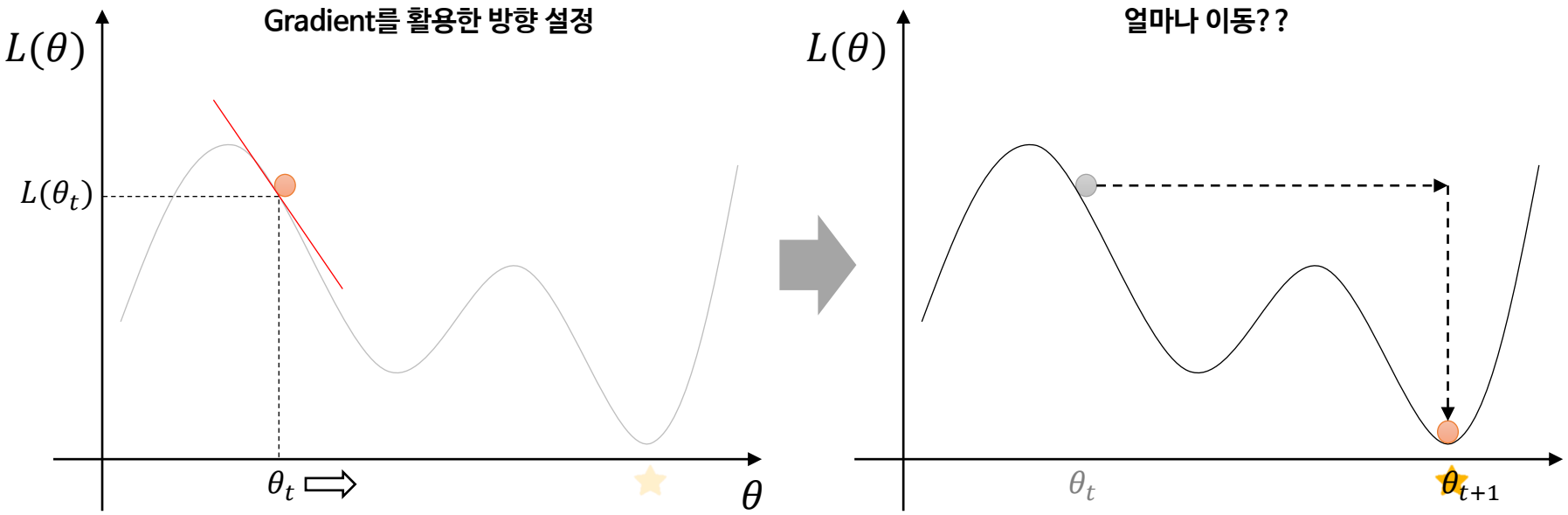
Definition of Optimizer

❖ Types of Optimizer



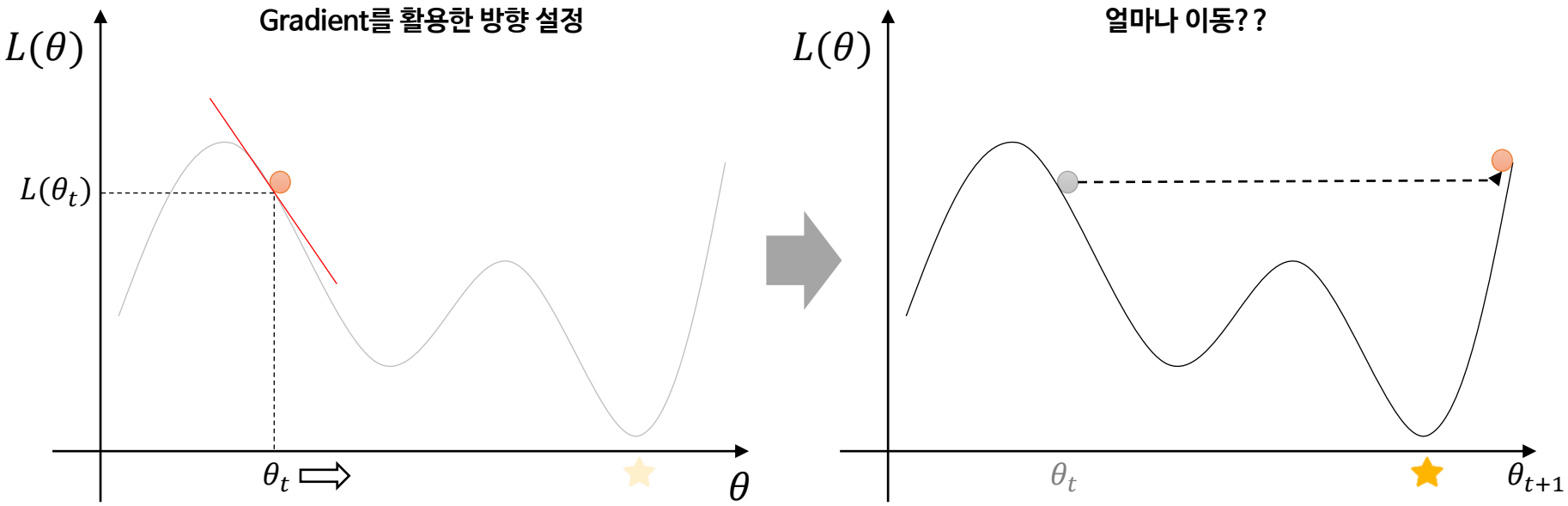
Definition of Optimizer

❖ Types of Optimizer



Definition of Optimizer

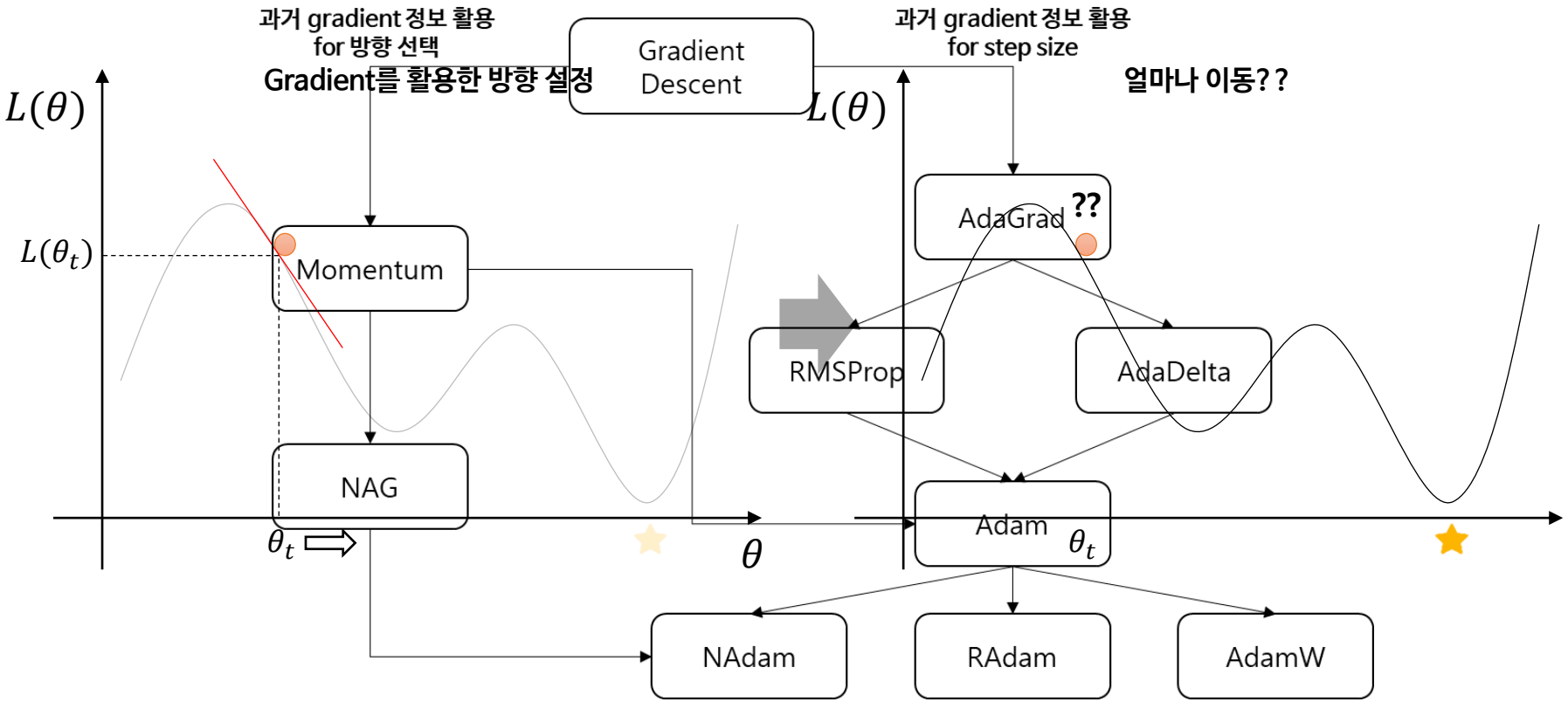
❖ Types of Optimizer



Definition of Optimizer

❖ Types of Optimizer

- First-order method의 두 주요 요소는 gradient를 활용한 방향 설정과 얼마나 이동할지 결정하는 step size
- Gradient를 활용해 이동방향과 step size를 더 효율적으로 설정하는 방향으로 연구가 진행



I. Introduction

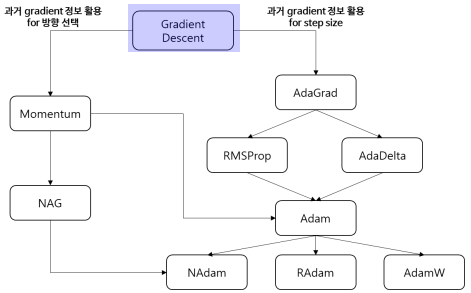
II. Definition of Optimizer

III. Gradient Decent Algorithm

IV. Variants of Gradient Descent

V. Conclusions

Gradient Decent Algorithm

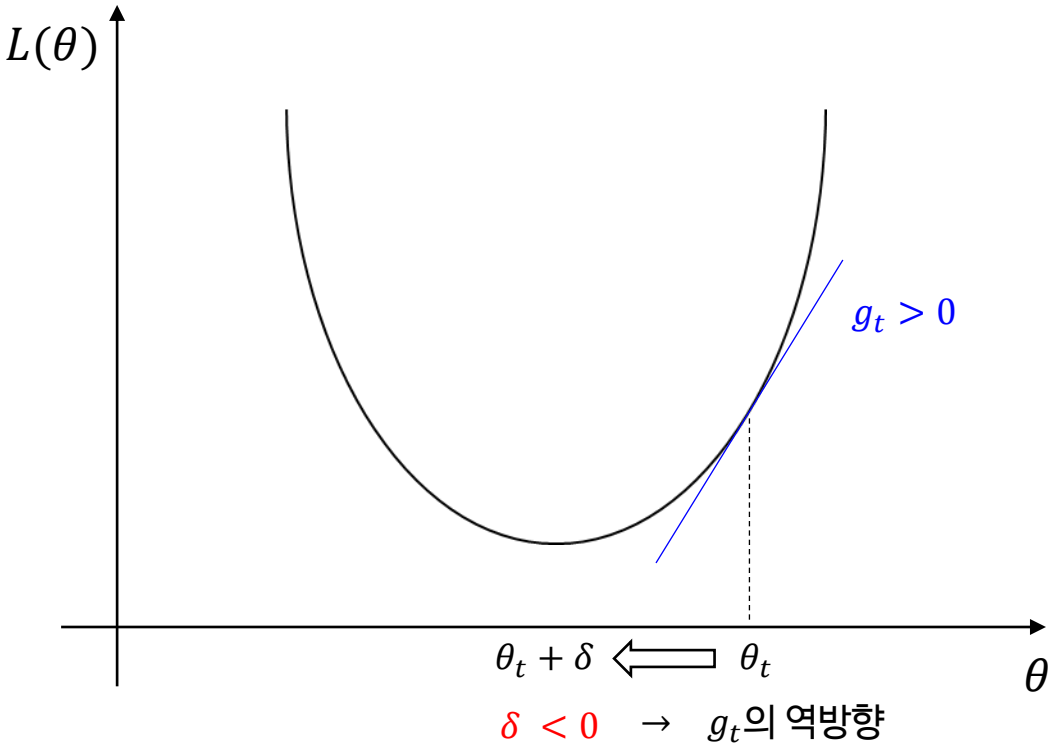


❖ Gradient Decent Algorithm

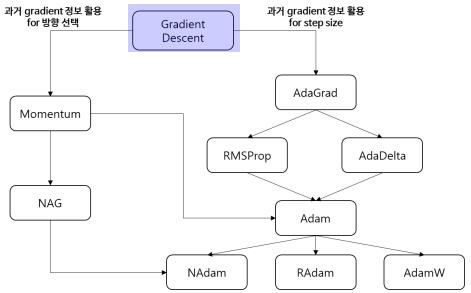
- 현재의 gradient 정보를 이용해 방향과 step size를 결정하는 가장 기본적인 알고리즘

1. 어느 방향으로

2. 얼마나 움직일지



Gradient Decent Algorithm



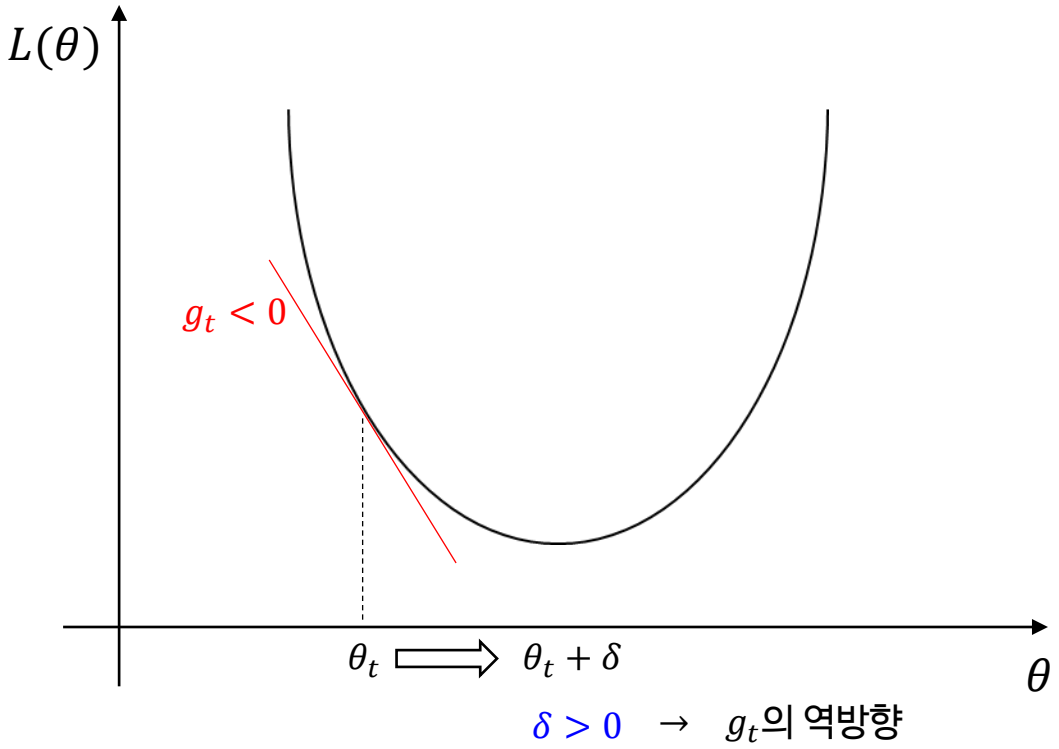
❖ Gradient Decent Algorithm

- 현재의 gradient 정보를 이용해 방향과 step size를 결정하는 가장 기본적인 알고리즘

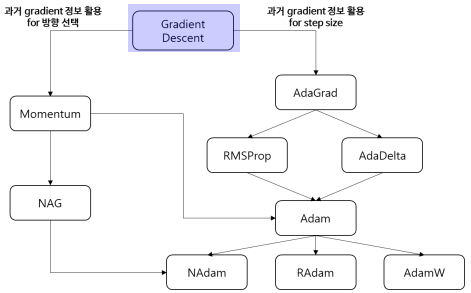
1. 어느 방향으로

→ gradient의 역방향

2. 얼마나 움직일지



Gradient Decent Algorithm



❖ Gradient Decent Algorithm

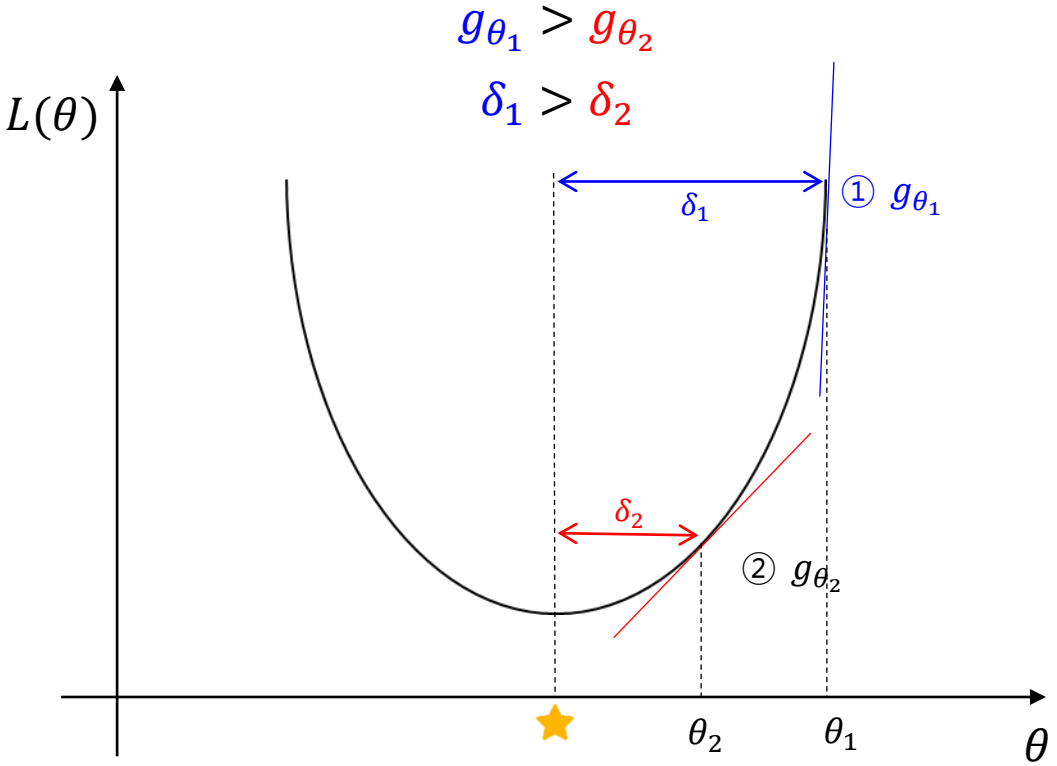
- 현재의 gradient 정보를 이용해 방향과 step size를 결정하는 가장 기본적인 알고리즘

1. 어느 방향으로

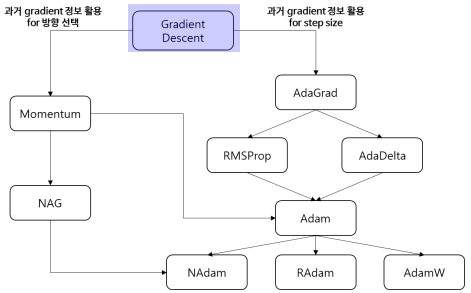
→ gradient의 역방향

2. 얼마나 움직일지

→ gradient의 크기



Gradient Decent Algorithm



❖ Gradient Decent Algorithm

- 현재의 gradient 정보를 이용해 방향과 step size를 결정하는 가장 기본적인 알고리즘

1. 어느 방향으로

→ gradient의 역방향

2. 얼마나 움직일지

→ gradient의 크기

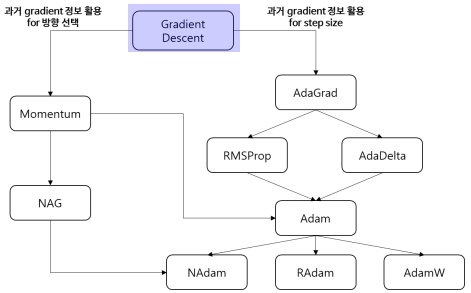
$$\theta_{t+1} = \theta_t + \delta$$

$$= \theta_t - g_t$$

$$= \theta_t - g_t \times \eta$$

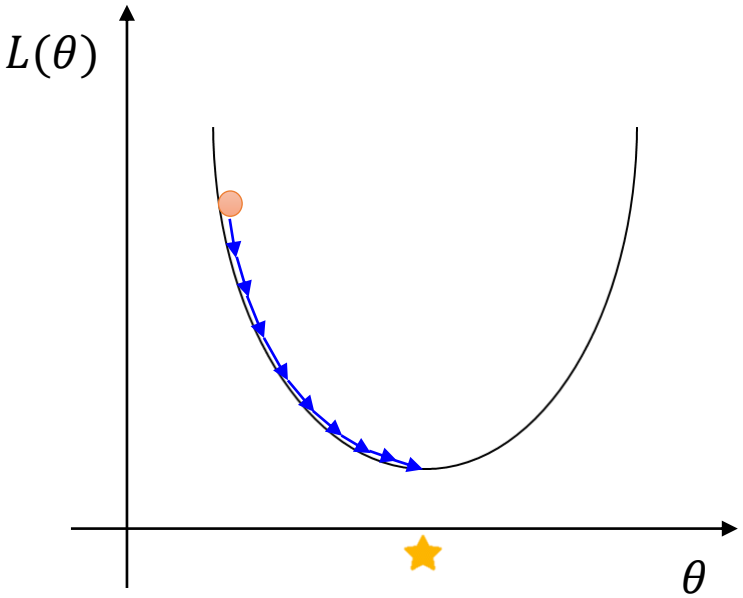
Learning rate

Gradient Decent Algorithm

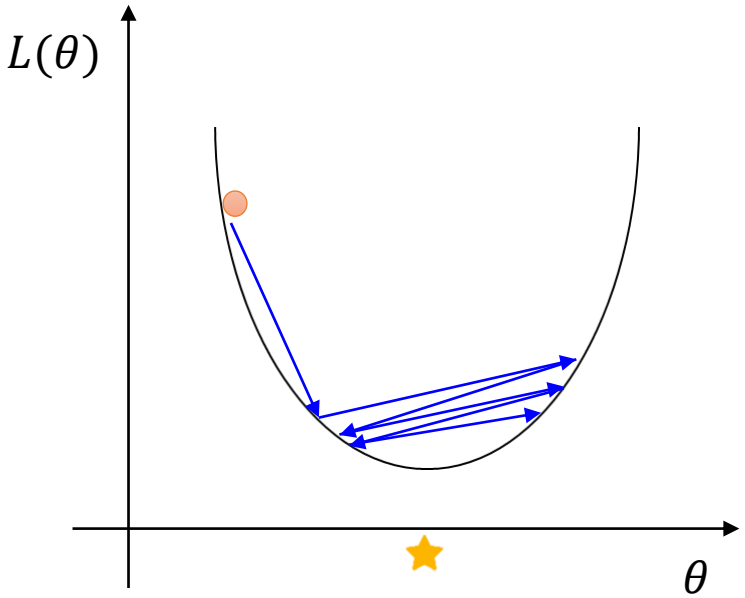


❖ The Role of Learning Rate

- 너무 작은 learning rate은 많은 iteration을 요구하고 수렴속도를 늦춤
- 너무 큰 learning rate은 빠르게 최적점 근처까지 갈 수 있지만 수렴을 못하고 발산할 가능성이 있음
- 적절한 learning rate을 찾는 것이 중요하며 일반적으로 0.01~0.0001 사이를 많이 사용
- Learning rate decay를 활용해 학습 초반에는 크게, 후반에는 작게 가져가는 방법도 많이 사용

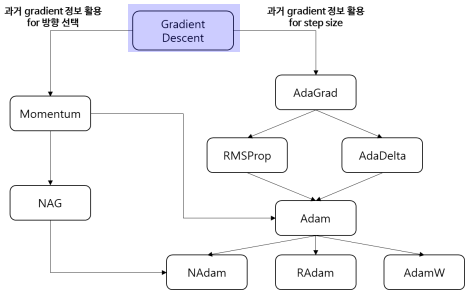


Small Learning Rate



Large Learning Rate

Gradient Decent Algorithm



❖ Example

Data

obs	X	Y
1	1	1
2	2	3
3	3	2

Model

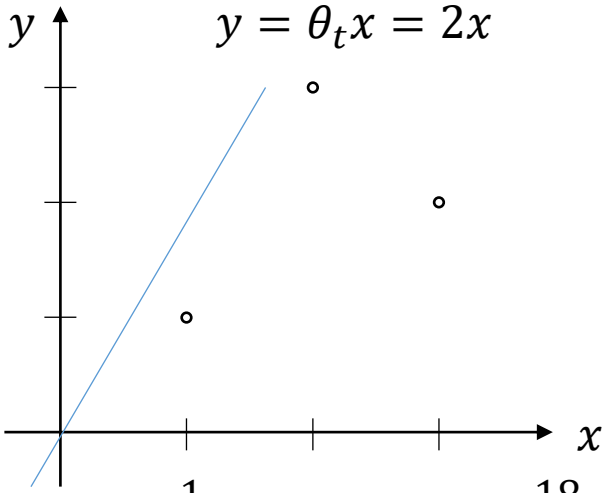
$$y = \theta x$$

Loss

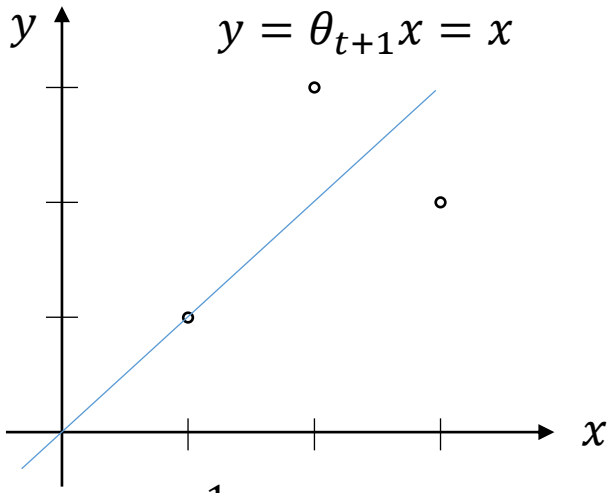
$$L(\theta) = \frac{1}{2N}\sum (y' - y)^2 = \frac{1}{2N}\sum (\theta x - y)^2$$

$$\frac{L(\theta_t)}{\partial \theta_t} = \frac{1}{N}\sum -(\theta_t x - xy) = \frac{1}{3}(-1 + 2 - 0) = 1$$

$$\theta_{t+1} = \theta_t - g_t \times \eta = 2 - \frac{1}{3} \times 3 = 1$$

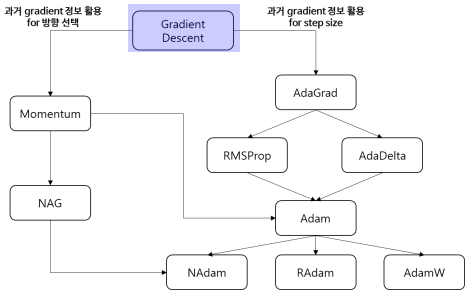


$$loss = \frac{1}{3}(1^2 + 1^2 + 4^2) = \frac{18}{3}$$



$$loss = \frac{1}{3}(0^2 + 1^2 + (-1)^2) = \frac{2}{3}$$

Gradient Decent Algorithm



❖ Stochastic Gradient Decent

- GD는 모든 관측치에 대한 gradient를 계산 → 관측치와 변수가 많아질수록 계산 복잡도 증가

obs	X1	X2	X3	⋯	Xm	Y
1	3	2.7	33	⋯	-3.1	0
2	2	2.1	45	⋯	-0.3	1
3	5	1.6	43	⋯	0.1	1
⋮			⋮			
n	4	2.5	39	⋯	-2.6	0

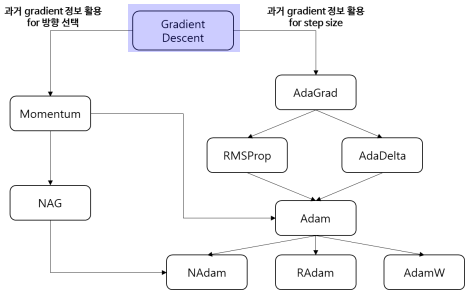
n (rows) / m (columns)

$$\begin{pmatrix} \frac{\partial L}{\partial x_{11}} & \frac{\partial L}{\partial x_{12}} & \frac{\partial L}{\partial x_{13}} & \cdots & \frac{\partial L}{\partial x_{1m}} \\ \frac{\partial L}{\partial x_{21}} & \frac{\partial L}{\partial x_{22}} & \frac{\partial L}{\partial x_{23}} & \cdots & \frac{\partial L}{\partial x_{2m}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial x_{n1}} & \frac{\partial L}{\partial x_{n2}} & \frac{\partial L}{\partial x_{n3}} & \cdots & \frac{\partial L}{\partial x_{nm}} \end{pmatrix}$$

average

$$g_t = \begin{pmatrix} \frac{\partial L}{\partial x_1} & \frac{\partial L}{\partial x_2} & \frac{\partial L}{\partial x_3} & \cdots & \frac{\partial L}{\partial x_m} \end{pmatrix}$$

Gradient Decent Algorithm



❖ Stochastic Gradient Decent

- GD는 모든 관측치에 대한 gradient를 계산 → 관측치와 변수가 많아질수록 계산 복잡도 증가
- 하나의 관측치를 샘플링하여 해당 관측치에 대한 gradient만 사용 ($\because E[g_{im}] = g_m$, 실제 $gradient$의 불편추정량)
- 계산 복잡도: $O(n * m) \rightarrow O(m)$

obs	X1	X2	X3	⋯	Xm	Y
1	3	2.7	33	⋯	-3.1	0
2	2	2.1	45	⋯	-0.3	1
3	5	1.6	43	⋯	0.1	1
⋮			⋮			
n	4	2.5	39	⋯	-2.6	0

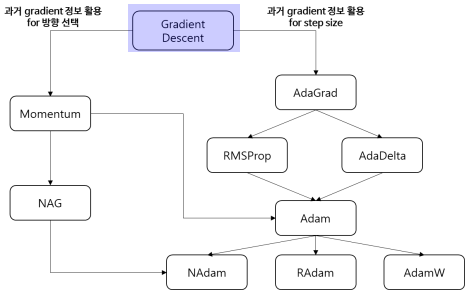
n (행)

m (열)

$$\begin{pmatrix} \frac{\partial L}{\partial x_{11}} & \frac{\partial L}{\partial x_{12}} & \frac{\partial L}{\partial x_{13}} & \cdots & \frac{\partial L}{\partial x_{1m}} \\ \frac{\partial L}{\partial x_{21}} & \frac{\partial L}{\partial x_{22}} & \frac{\partial L}{\partial x_{23}} & \cdots & \frac{\partial L}{\partial x_{2m}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial x_{n1}} & \frac{\partial L}{\partial x_{n2}} & \frac{\partial L}{\partial x_{n3}} & \cdots & \frac{\partial L}{\partial x_{nm}} \end{pmatrix}$$

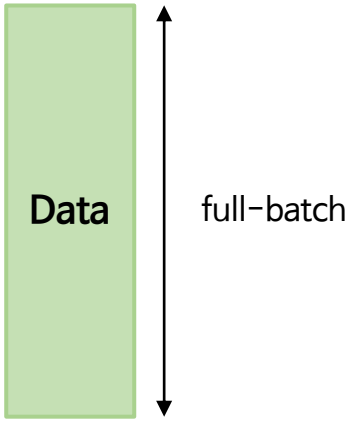
$$g_t = \begin{pmatrix} \frac{\partial L}{\partial x_{21}} & \frac{\partial L}{\partial x_{22}} & \frac{\partial L}{\partial x_{23}} & \cdots & \frac{\partial L}{\partial x_{2m}} \end{pmatrix}$$

Gradient Decent Algorithm



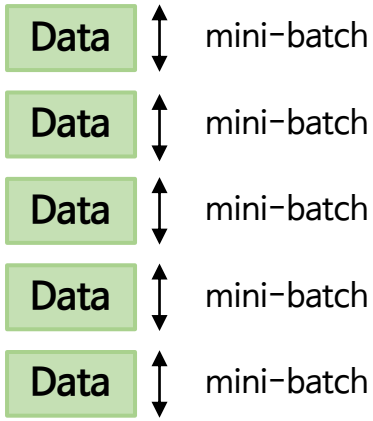
❖ Mini-Batch Stochastic Gradient Decent

GD



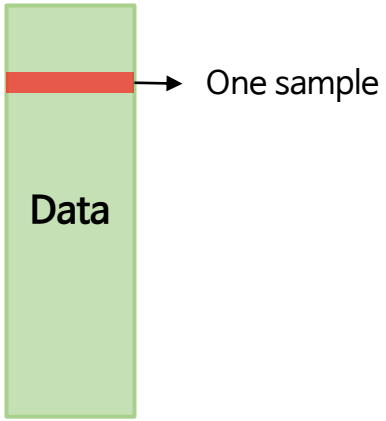
- **Pros**
  - 현 시점 최적의 gradient 사용
- **Cons**
  - 데이터셋이 큰 경우 계산 비용이 크고 학습이 오래 걸림
  - Online learning 불가능

MSGD



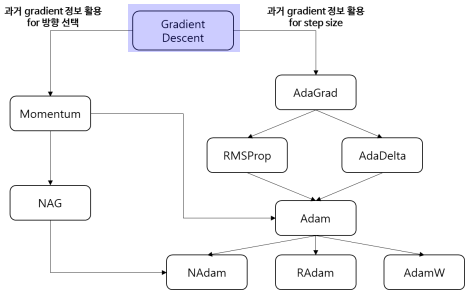
- **Pros**
  - GD와 SGD를 적절히 섞어 적당한 연산으로 빠른 수렴 가능
- **Cons**
  - 여전히 변동성이 큼
  - batch size가 hyperparameter

SGD

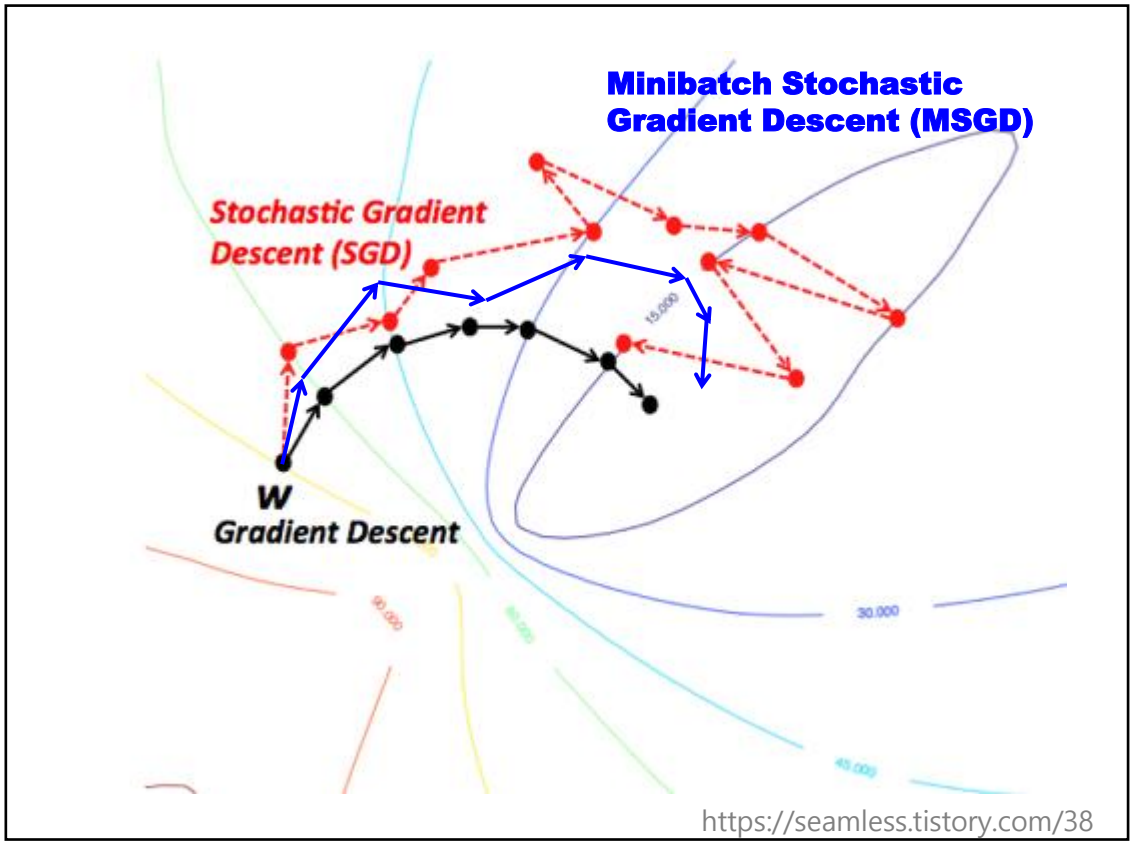


- **Pros**
  - 빠른 연산을 통한 빠른 수렴
  - Online learning 가능
- **Cons**
  - 최적방향이 아니라 변동성이 큼
  - 모든 데이터를 고려하지 않을 수도 있음

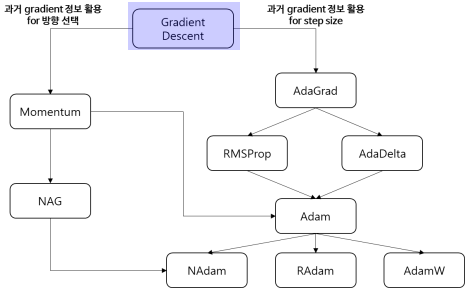
Gradient Decent Algorithm



❖ Mini-Batch Stochastic Gradient Decent

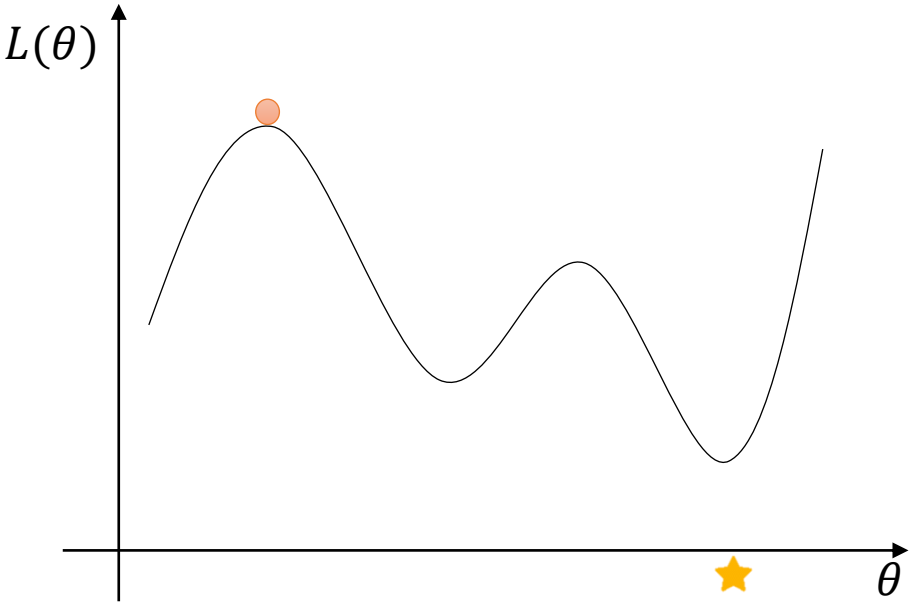


Gradient Decent Algorithm

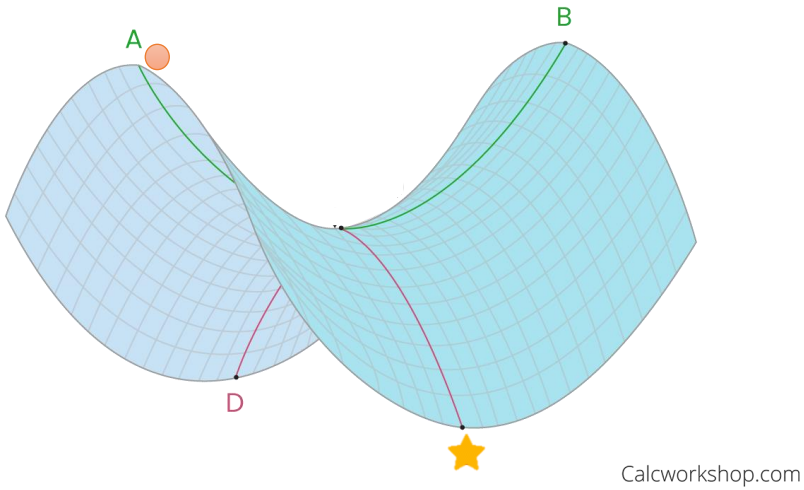


❖ Local Minima & Saddle Point

- SGD는 local minima와 saddle point에 갇히면 쉽게 빠져나오지 못한다는 문제가 있음

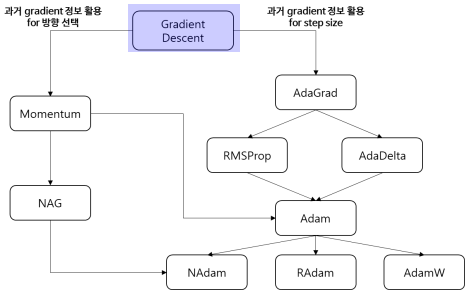


Local minima



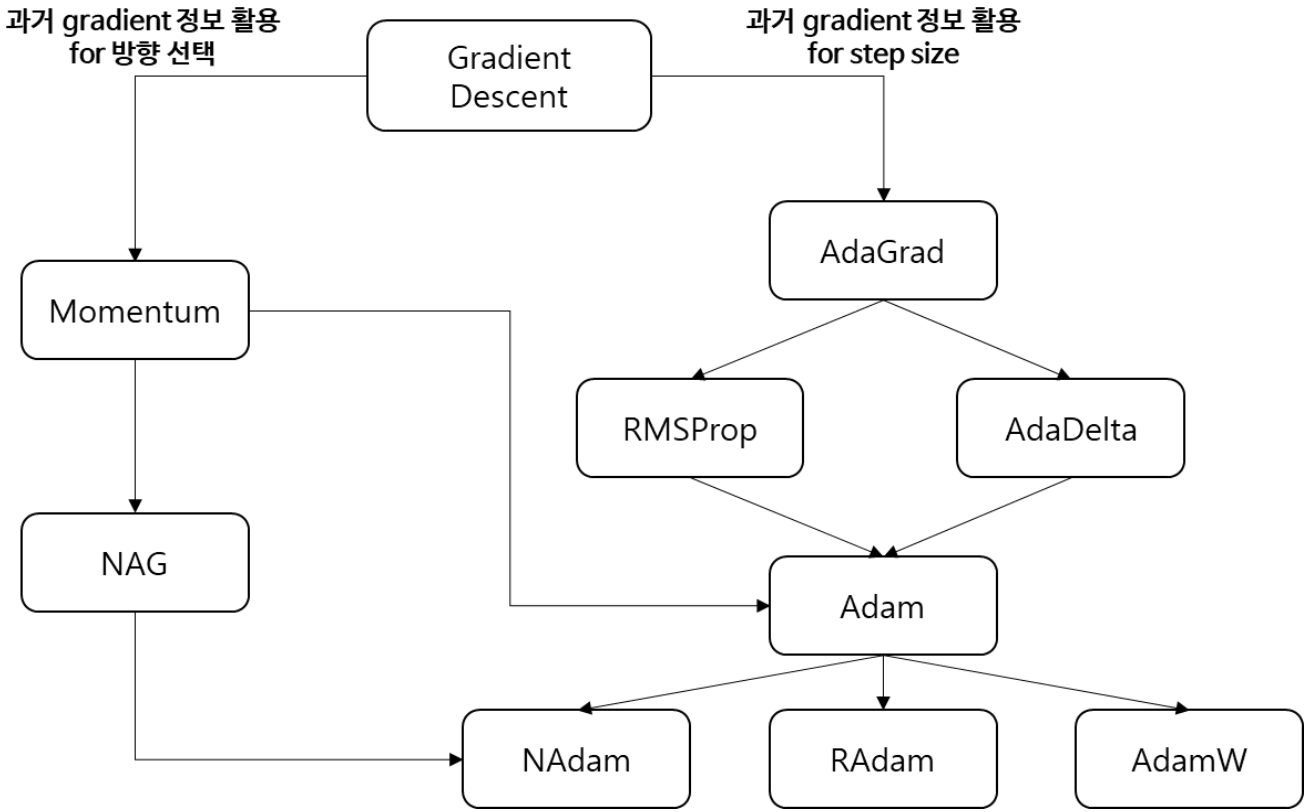
Saddle Point

Gradient Decent Algorithm



❖ Local Minima & Saddle Point

- SGD는 local minima와 saddle point에 갇히면 쉽게 빠져나오지 못한다는 문제가 있음
- 과거의 gradient 정보를 활용해서 이 문제를 해결하기 위한 연구가 진행



I. Introduction

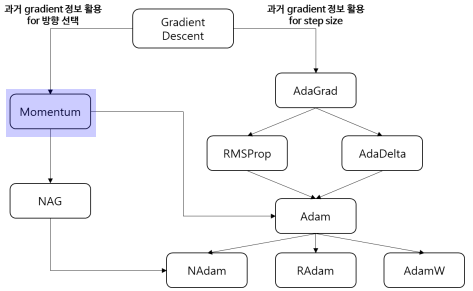
II. Definition of Optimizer

III. Gradient Decent Algorithm

IV. Variants of Gradient Descent

V. Conclusions

Gradient Decent Algorithm



❖ Momentum

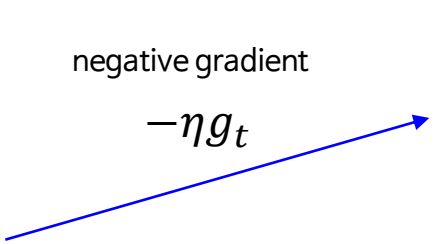
- 관성, 가속도를 의미하는 momentum 개념을 SGD에 추가하여 한 방향으로 일정하게 이동할 수 있게 유도

$$\theta_{t+1} = \theta_t - g_t \times \eta \quad \Rightarrow \quad \theta_{t+1} = \theta_t + v_{t+1}$$

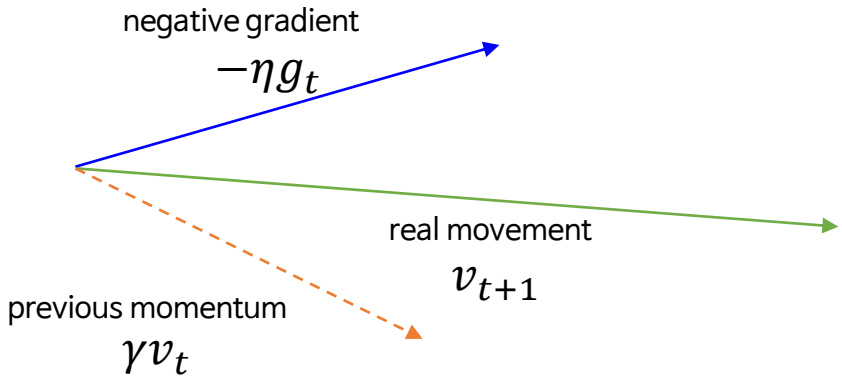
$$v_{t+1} = \gamma v_t - \eta \times g_t$$

과거 방향 정보 현재 방향 정보

$$v_t = \gamma v_{t-1} - \eta g_{t-1} = \gamma(\gamma v_{t-2} - \eta g_{t-2}) - \eta g_{t-1}$$
$$= \gamma(\gamma(v_{t-3} - \eta g_{t-3}) - \eta g_{t-2}) - \eta g_{t-1} = \cdots$$

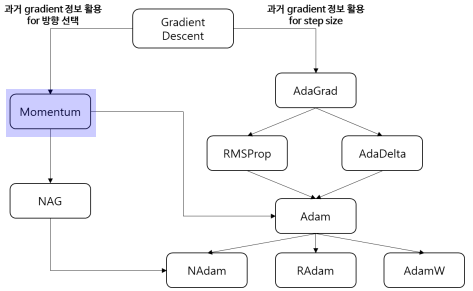


SGD



Momentum

Gradient Decent Algorithm



❖ Momentum

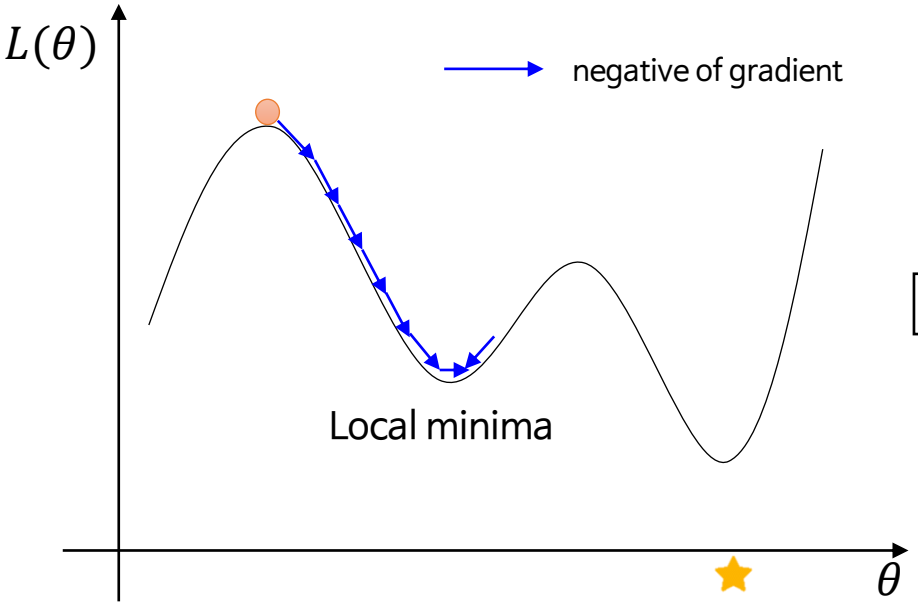
- 관성, 가속도를 의미하는 momentum 개념을 SGD에 추가하여 한 방향으로 일정하게 이동할 수 있게 유도

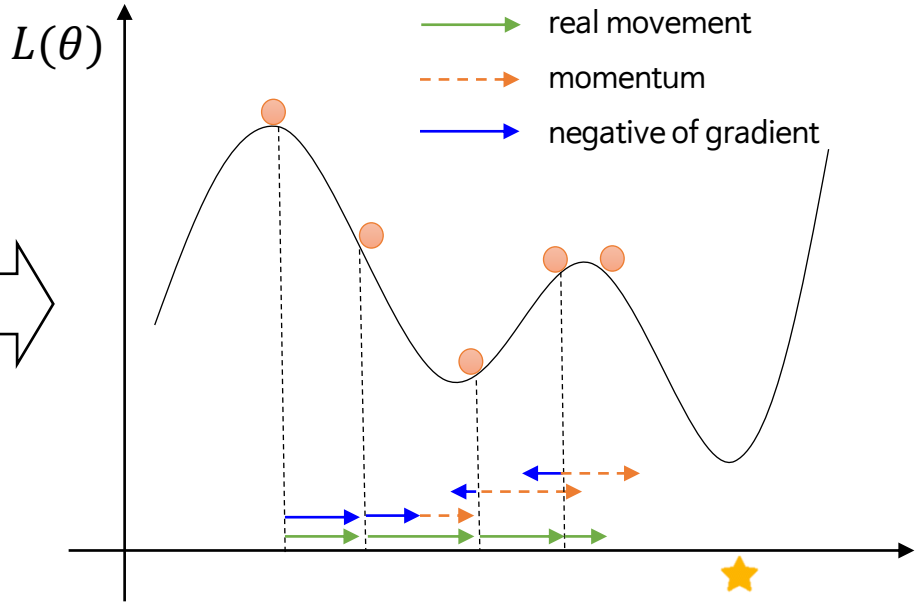
$$\theta_{t+1} = \theta_t - g_t \times \eta \quad \Rightarrow$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

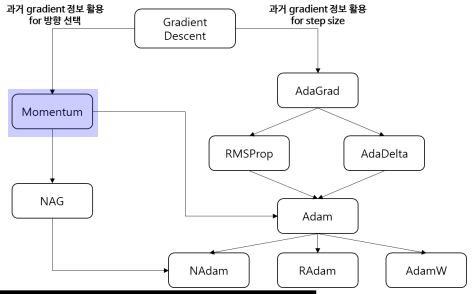
$$v_{t+1} = \gamma v_t - \eta \times g_t$$

과거 방향 정보 현재 방향 정보





Gradient Decent Algorithm



❖ Momentum

- 관성, 가속도를 의미하는 momentum 개념을 SGD에 추가하여 한 방향으로 일정하게 이동할 수 있게 유도

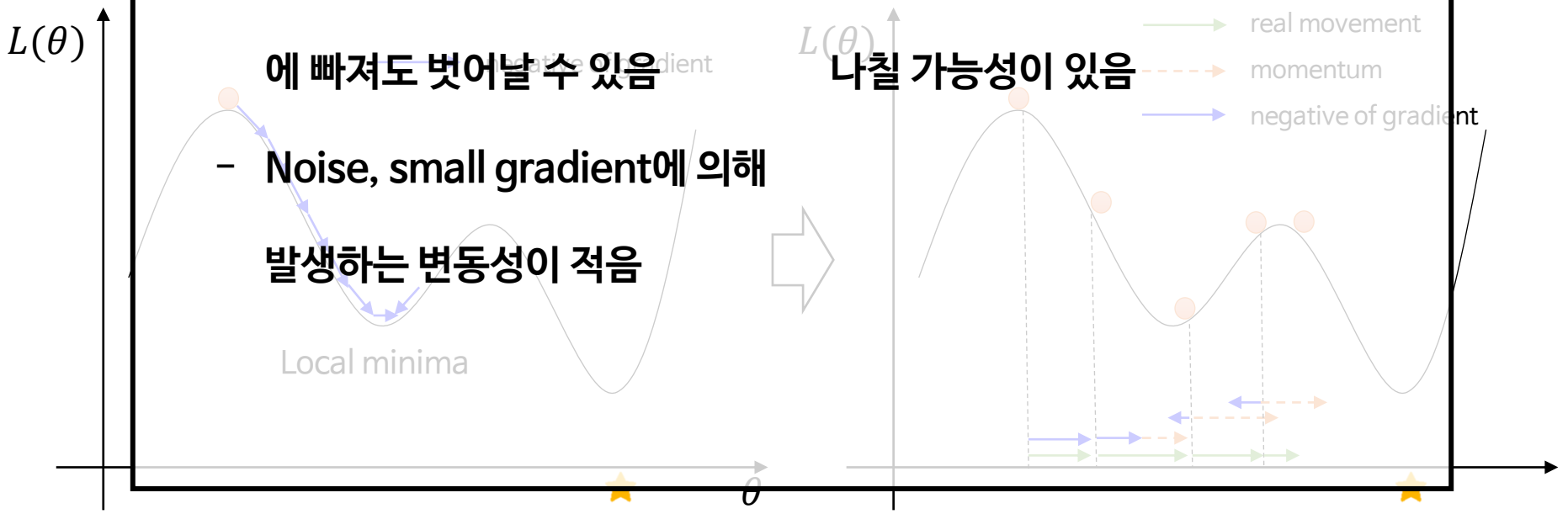
$$\theta_{t+1} = \theta_t - g_t \times \eta$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

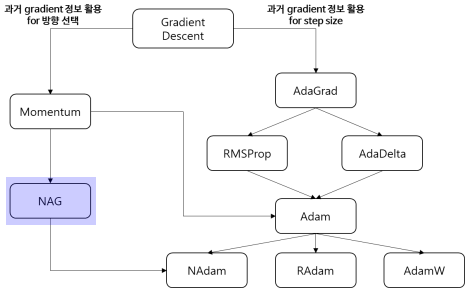
$$v_{t+1} = \gamma v_t - \eta \times g_t$$

과거 방향 정보 | 현재 방향 정보

- **Pros**
  - Local minima나 saddle point에 빠져도 벗어날 수 있음
  - Noise, small gradient에 의해 발생하는 변동성이 적음
- **Cons**
  - 관성에 의해 Global minima를 지나칠 가능성이 있음



Gradient Decent Algorithm



❖ Nesterov Accelerated Gradient Descent(NAG)

- Momentum 방식을 베이스로 **현재의 gradient**가 아닌 momentum step 이후 gradient를 통해 업데이트
- 현재의 가속도를 고려하지 않는 momentum과 달리 현재의 가속도를 고려함

$$\theta_{t+1} = \theta_t + v_{t+1}$$
$$v_{t+1} = \gamma v_t - \eta \times g_t$$

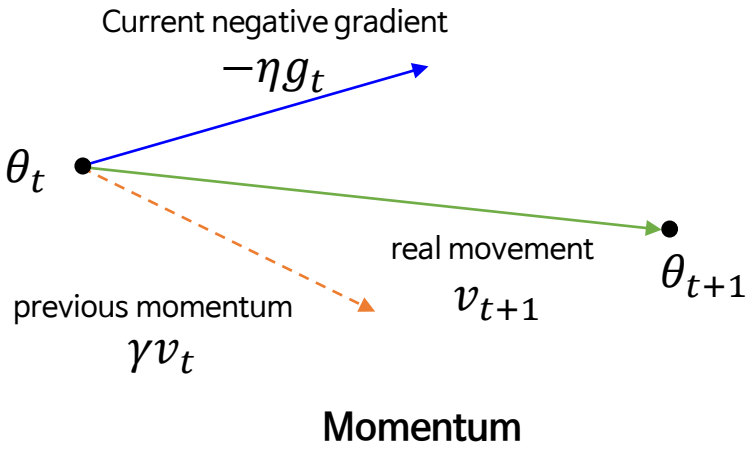
현재 gradient

⇨

$$\theta_{t+1} = \theta_t + v_{t+1}$$
$$v_{t+1} = \gamma v_t - \eta \times g_{\tilde{\theta}_t}$$

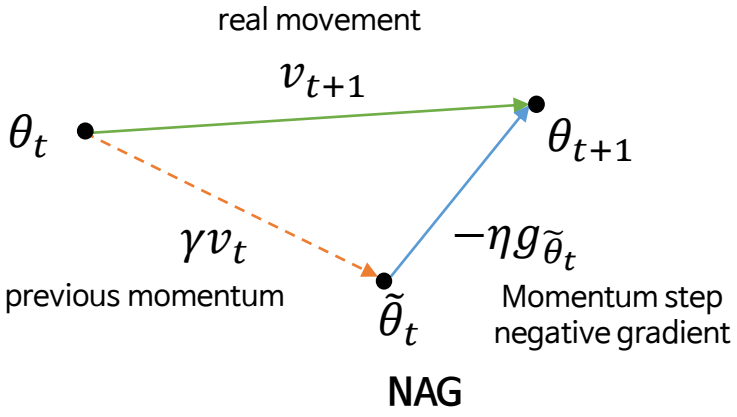
momentum step 이후 gradient

$$\tilde{\theta}_t = \theta_t + \gamma v_t$$



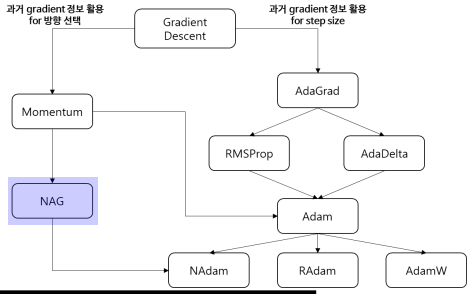
Momentum

⇨



NAG

Gradient Decent Algorithm



❖ Nesterov Accelerated Gradient Descent (NAG)

- Momentum 방식을 베이스로 현재의 gradient가 아닌 momentum step 이후 gradient를 통해 업데이트
- 현재의 가속도를 고려하지 않는 momentum과 달리 현재의 가속도를 고려함

- **Pros**
  - **현재의 momentum 정보를 반영함으로써 momentum의 과하게 이동한다는 단점을 완화**
  - **기존의 momentum보다 더 좋은 수렴속도를 실험적으로 증명**

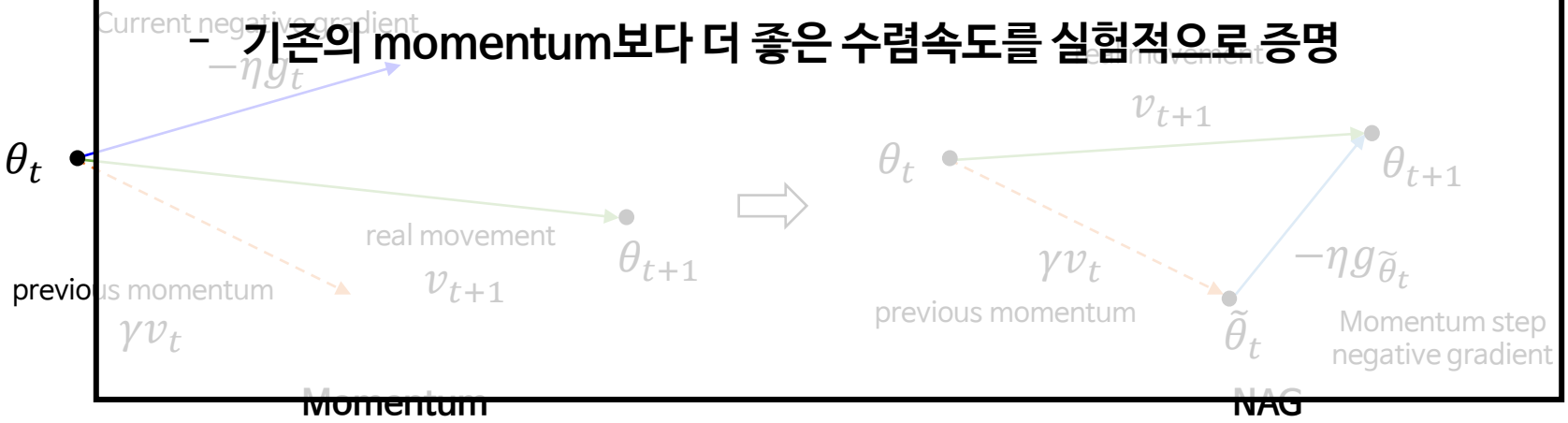
$$\theta_{t+1} = \theta_t + v_{t+1}$$

$$v_{t+1} = \gamma v_t - \eta \times g_t$$

$$\theta_{t+1} = \theta_t + \gamma v_{t+1} - \eta g_{\tilde{\theta}_t}$$

$$\tilde{\theta}_t = \theta_t + \gamma v_t$$

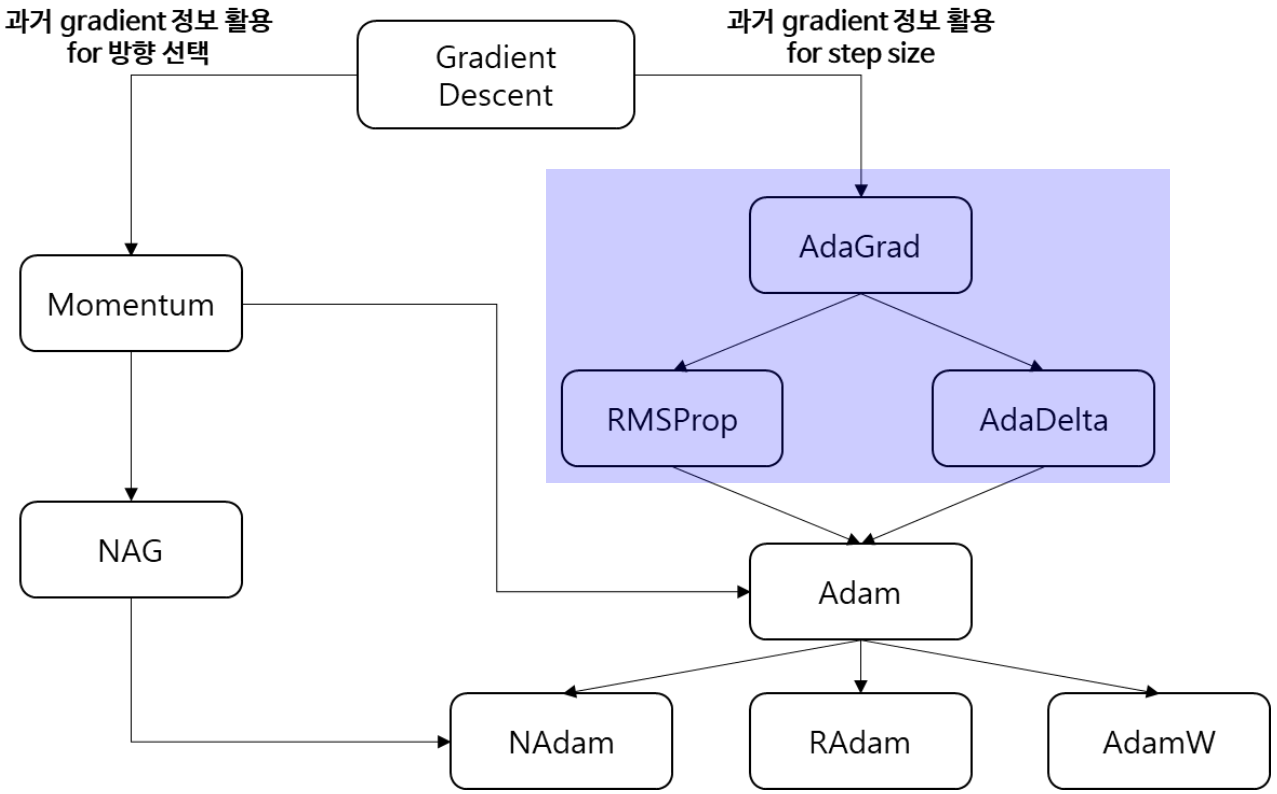
momentum step 이후 gradient



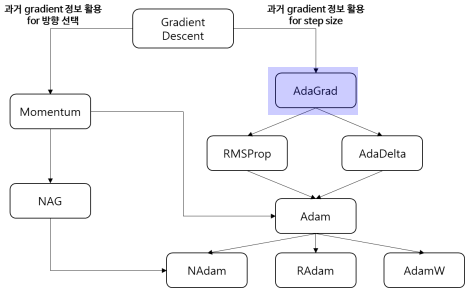
Gradient Decent Algorithm

❖ Adaptive Learning Rate Methods

- SGD는 learning rate에 매우 민감한 성능 변화를 보임 → 적절한 learning rate 설정이 중요하지만 어려움
- 상황에 따라 자동으로 적절한 learning rate을 설정하는 Adaptive learning rate method 연구가 활발히 진행



Gradient Decent Algorithm



❖ Adaptive Gradient (AdaGrid)

- AdaGrid는 과거 gradient 정보를 통해서 각각의 파라미터와 iteration마다 다른 learning rate을 적용

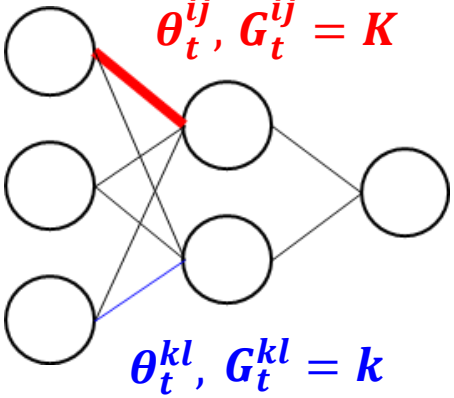
$$\theta_{t+1} = \theta_t - g_t \times \eta \quad \Rightarrow \quad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t$$

$$G_t = \sum_{i=0}^{t} g_i^2$$

Step size가 작아짐

Step size가 커짐

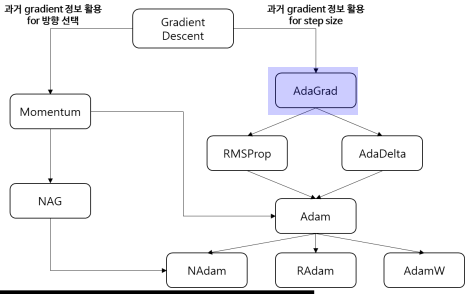
과거 gradient의 제곱합



$$\theta_{t+1}^{ij} = \theta_t^{ij} - \frac{\eta}{\sqrt{K + \varepsilon}} g_t^{ij}$$ → 이전에 업데이트가 **많이 진행**되었기 때문에 step size를 작게 해서 세밀하게 조정

$$\theta_{t+1}^{kl} = \theta_t^{kl} - \frac{\eta}{\sqrt{k + \varepsilon}} g_t^{kl}$$ → 이전에 업데이트가 **조금 진행**되었기 때문에 step size를 크게 해서 빠르게 이동

Gradient Decent Algorithm



❖ Adaptive Gradient (AdaGrid)

- AdaGrid는 과거 gradient 정보를 통해서 각각의 파라미터와 iteration마다 다른 learning rate을 적용

$$\theta_{t+1} = \theta_t - g_t \times \eta \quad \Rightarrow \quad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t$$

$$G_t = \sum_{i=0}^{t} g_i^2$$

- **Pros**
  - 자연어처리와 같이 sparse data에 대해서 업데이트할 때 step size를 유연하게 가져가기 때문에 효과적임
  - 학습과정에서 직접 learning rate을 조정할 필요가 없음

- **Cons**
  - 여전히 적절한 global learning rate η 을 필요로 함
  - 학습이 진행될수록 G_t의 값이 무한히 커져 learning rate이 0에 수렴하기 때문에 파라미터 업데이트가 안됨

Step size가 작아짐

Step size가 커짐

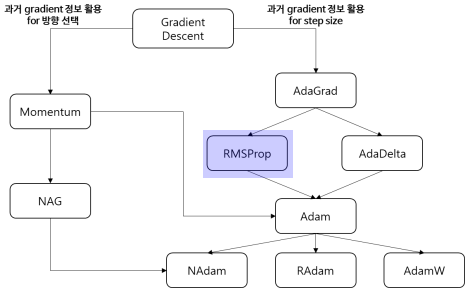
$\theta_t^{ij}, G_t^{ij} = K$

$$\theta_{t+1}^{ij} = \theta_t^{ij} - \frac{\eta}{\sqrt{K + \varepsilon}} g_t^{ij}$$ → 이전에 업데이트가 **많이 진행**되었기 때문에 step size를 크게 해서 빠르게 이동

$\theta_t^{kl}, G_t^{kl} = k$

$$\theta_{t+1}^{ij} = \theta_t^{ij} - \frac{\eta}{\sqrt{k + \varepsilon}} g_t^{kl}$$ → 이전에 업데이트가 **조금 진행**되었기 때문에 step size를 크게 해서 빠르게 이동

Gradient Decent Algorithm



❖ Root Mean Square Propagation (RMSProp)

- AdaGrid에서 learning rate이 0으로 수렴하는 것을 막기 위해 제안된 방법론
- 과거 gradient를 단순히 합하는 것이 아닌 제곱의 지수 이동 평균으로 정의
- $0 < \beta < 1$를 통해서 오래된 과거 gradient의 영향력을 0으로 수렴시킴

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t$$

$$G_t = \sum_{i=0}^{t} g_i^2$$

⇨

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t$$

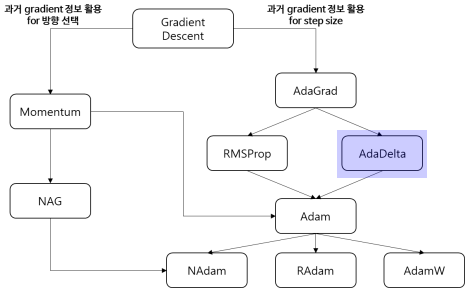
$$G_t = \beta G_{t-1} + (1-\beta) g_t^2$$

gradient 제곱의 지수 이동 평균

$$G_t = \beta(\beta G_{t-2} + (1-\beta) g_{t-1}^2) + (1-\beta) g_t^2$$

$$= \beta(\beta(\beta G_{t-3} + (1-\beta) g_{t-2}^2) + (1-\beta) g_{t-1}^2) + (1-\beta) g_t^2 = \cdots$$

Gradient Decent Algorithm



❖ Adaptive Delta (AdaDelta)

- RMSProp와 비슷하게 AdaGrid의 단점을 보완하기 위해 제안된 방법
- RMSProp과 동일하게 G를 계산할 때 지수 평균을 사용
- 다만, step size를 단순하게 η로 사용하는 대신 step size의 변화량의 제곱의 지수 평균 사용

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t$$

$$G_t = \beta G_{t-1} + (1-\beta) g_t^2$$

⇨

$$\theta_{t+1} = \theta_t - \frac{\sqrt{H_{t-1} + \varepsilon}}{\sqrt{G_t + \varepsilon}} g_t$$

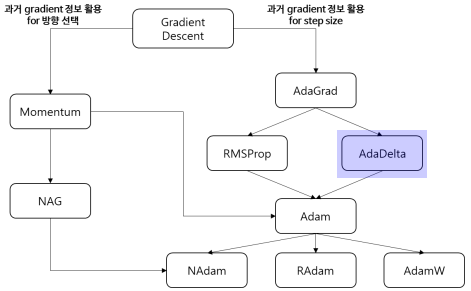
$$G_t = \beta G_{t-1} + (1-\beta) g_t^2$$

gradient 제곱의 지수 이동 평균

$$H_t = \beta H_{t-1} + (1-\beta) \Delta\theta_t^2$$

Step size의 변화량의 제곱의 지수 이동 평균

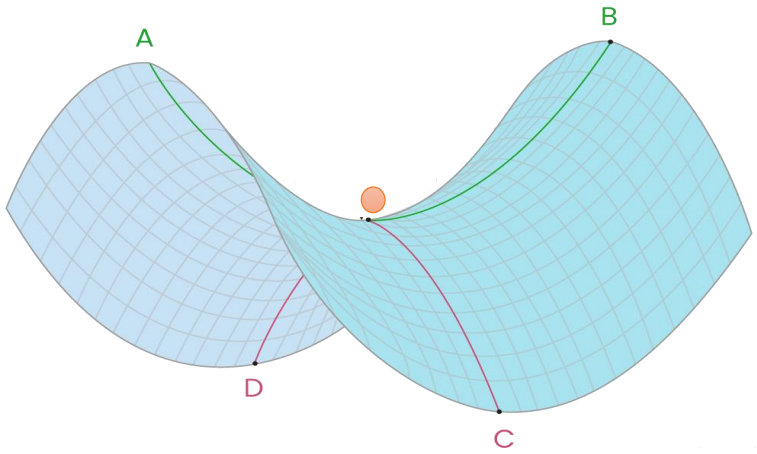
Gradient Decent Algorithm



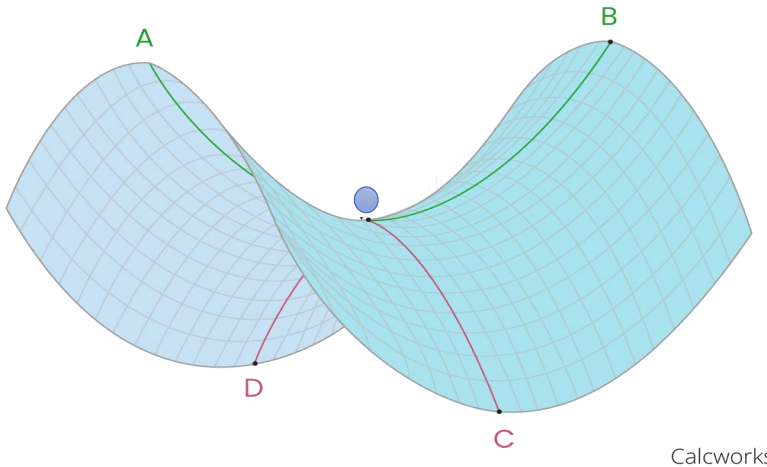
❖ Adaptive Delta (AdaDelta)

- 학습이 많이 진행되었더라도 파라미터의 변화량이 크면 step size가 커야한다는 전략을 통해 수렴 속도 향상

AdaGrid

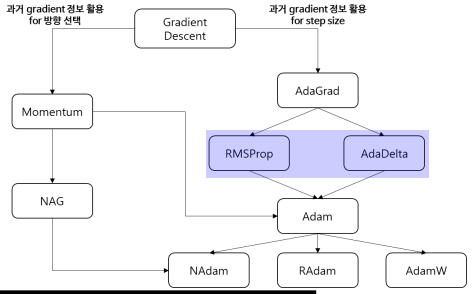


AdaDelta



Calcworkshop.com

Gradient Decent Algorithm

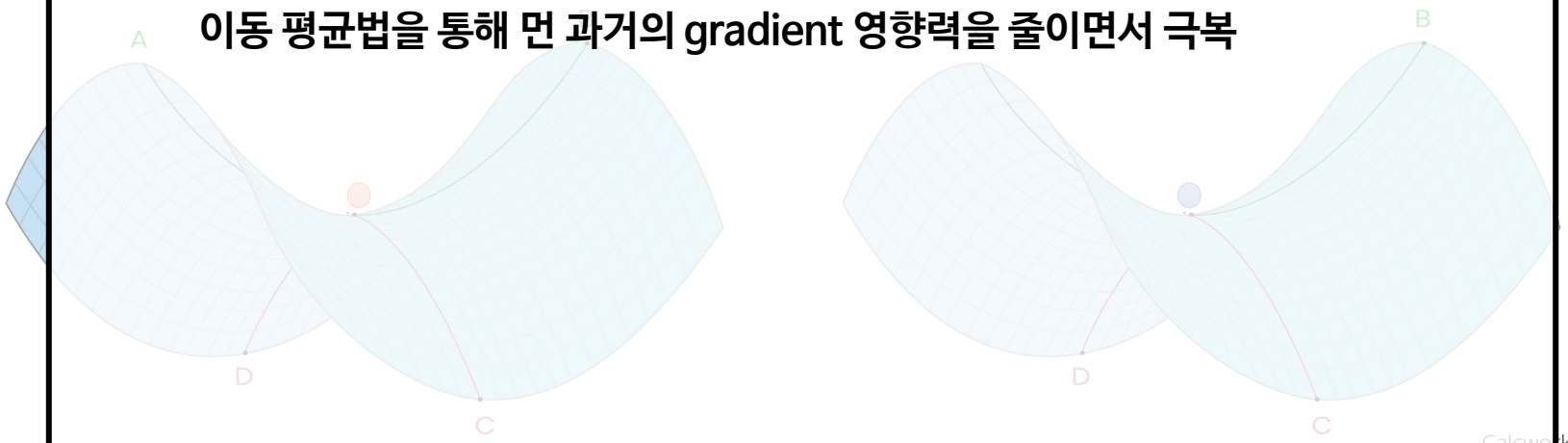


❖ RMSProp & AdaDelta

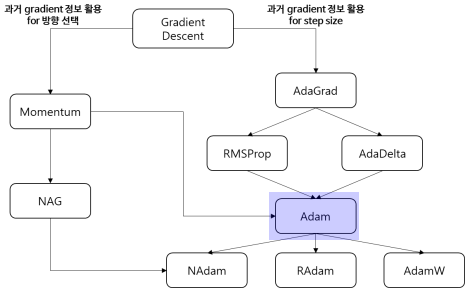
- 학습이 많이 진행되었더라도 파라미터의 변화량이 크면 step size가 커야한다는 전략을 통해 수렴 속도 향상

  - **Pros**

    - **학습이 진행될수록 learning rate이 0으로 수렴한다는 AdaGrid의 단점을 지수 이동 평균법을 통해 먼 과거의 gradient 영향력을 줄이면서 극복**



Gradient Decent Algorithm



❖ Adaptive Moment Estimation (Adam)

- Momentum과 RMSProp을 조합하여 만든 방법론
- 대부분의 상황에서 준수한 성능을 보이기 때문에 가장 많이 사용되는 optimizer

$$\theta_{t+1} = \theta_t + v_{t+1}$$

$$v_{t+1} = \gamma v_t - \eta g_t$$

Momentum

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t$$

$$G_t = \beta G_{t-1} + (1-\beta) g_t^2$$

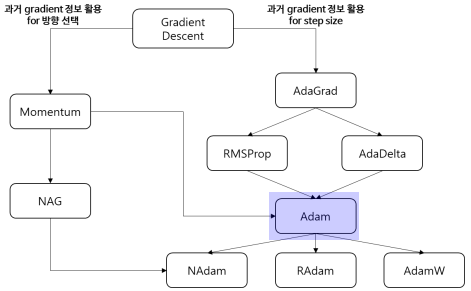
RMSProp

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{G}_{t+1} + \varepsilon}} \hat{v}_{t+1}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - (\beta_1)^{t+1}}$$

$$\hat{G}_{t+1} = \frac{G_{t+1}}{1 - (\beta_2)^{t+1}}$$

Gradient Decent Algorithm



❖ Adaptive Moment Estimation (Adam)

$$v_{t+1} = \beta_1 v_t - (1 - \beta_1) g_t$$

$$G_t = \beta_2 G_{t-1} + (1 - \beta_2) g_t^2$$

$$\Rightarrow$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - (\beta_1)^{t+1}}$$

$$\hat{G}_{t+1} = \frac{G_{t+1}}{1 - (\beta_2)^{t+1}}$$

학습 초기

$v_0 = 0$

$G_0 = 0$

$\Rightarrow$

$v_t, G_t \approx 0$

0에 가깝게 bias

학습 초기 이후

$(\beta_1)^{t+1} \approx 0$

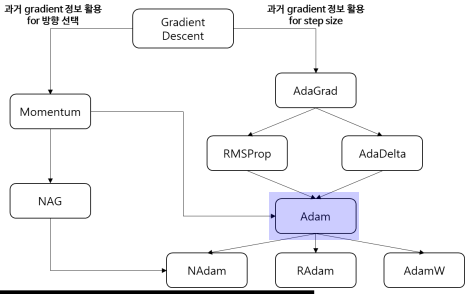
$(\beta_2)^{t+1} \approx 0$

$\Rightarrow$

$\hat{v}_{t+1} = v_{t+1}$

$\hat{G}_{t+1} = G_{t+1}$

Gradient Decent Algorithm

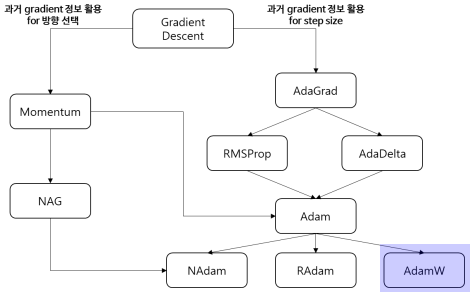


❖ Adaptive Moment Estimation (Adam)

- **Pros**
  - Momentum과 adaptive learning rate을 알맞게 조합하여 좋은 성능을 보이는 방법론 개발
  - 대부분의 상황에서 좋은 성능을 보이기 때문에 가장 보편적으로 사용됨

- **Cons**
  - 학습 초기에 adaptive learning rate의 분산이 커져서 local minima에 빠르게 도달해 학습이 안되는 현상인 "Bad Local Optima Convergence"
  - 일반화 성능이 SGD보다 좋지 않을 때가 있음

Gradient Decent Algorithm



❖ Decoupled Weight Decay Regularization (AdamW)

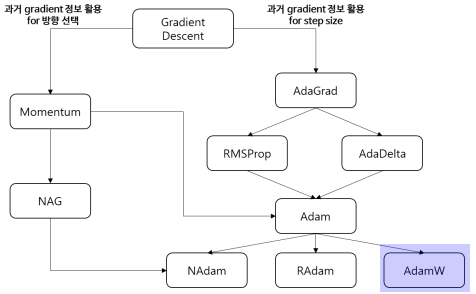
- 기본적인 SGD가 Adam보다 좋은 일반화 성능을 보여줄 때가 있음
- 저자들은 이 문제를 Adam이 파라미터마다 다른 leaning rate을 사용하기 때문에 weight decay를 l2-regularization으로 구현한다면 동일한 효과를 얻지 못해 성능이 하락한다고 가정

$$Loss = f(\theta) \quad \longrightarrow \quad Regularized\ Loss = f(\theta) + \frac{\lambda}{2}\sum_i \theta_i^2$$

L2-regularization term
=L2-panelty
=weight decay

$$\underset{\theta}{\text{minimize}} \; f(\theta) + \frac{\lambda}{2}\sum_i \theta_i^2$$

Gradient Decent Algorithm



❖ Decoupled Weight Decay Regularization (AdamW)

- SGD에서 $\lambda' = \frac{\lambda}{\eta}$ 라면 weight decay = l2-regularization이 성립됨

$$Regularized\ Loss = f^{reg}(\theta) = f(\theta) + \frac{\lambda'}{2}\|\theta\|_2^2, \qquad \lambda' = \frac{\lambda}{\eta}$$

$$\theta_{t+1} = \theta_t - \eta\frac{\partial f^{reg}(\theta_t)}{\partial \theta_t} = \theta_t - \eta\left(\frac{\partial f(\theta_t)}{\partial \theta_t} + \frac{\frac{\lambda'}{2}\|\theta_t\|_2^2}{\partial \theta_t}\right)$$

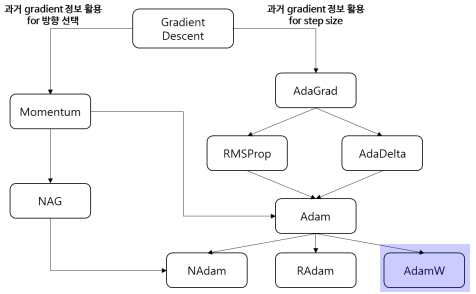
일반적인 SGD

$$= \theta_t - \eta g_t - \eta\lambda'\theta_t = \theta_t - \eta g_t - \lambda\theta_t$$

$$\theta_{t+1} = (1-\lambda)\theta_t - \eta g_t$$

Weight decay

Gradient Decent Algorithm



❖ Decoupled Weight Decay Regularization (AdamW)

- 하지만 Adam에서는 $\lambda' = \frac{\lambda}{\eta}$에서 weight decay = l2-regularization이 성립되지 않음
- 실제 weight decay rate보다 더 작은 decay rate으로 weight decay가 진행됨

$$Regularized\ Loss = f^{reg}(\theta) = f(\theta) + \frac{\lambda'}{2}\|\theta\|_2^2, \qquad \lambda' = \frac{\lambda}{\eta}$$

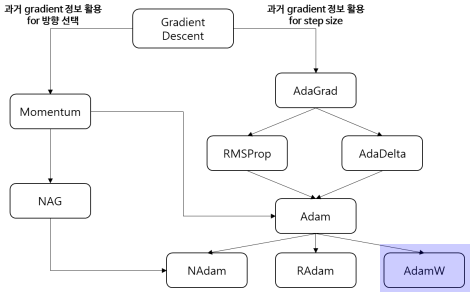
$$\theta_{t+1} = \theta_t - \eta M_t \frac{\partial f^{reg}(\theta_t)}{\partial \theta_t}$$

$$= \theta_t - \eta M_t (g_t - \lambda'\theta_t) = \theta_t - \eta M_t g_t - \eta M_t \lambda' \theta_t$$

$$\theta_{t+1} = (1 - M_t\lambda)\theta_t - \eta M_t g_t$$

$$only\ \boldsymbol{weight\ decay}\ when\ M_t = kI, and\ \boldsymbol{M_t\lambda < \lambda}$$

Gradient Decent Algorithm



❖ Decoupled Weight Decay Regularization (AdamW)

- 저자들은 regularization term과 weight decay term을 분리하여 이 문제를 해결

$$g_{t+1} = g_t + \lambda\theta_t$$ L2-regularization term

$$v_{t+1} = \beta_1 v_t - (1 - \beta_1) g_{t+1}$$

$$G_{t+1} = \beta_2 G_t + (1 - \beta_2) g_{t+1}^2$$

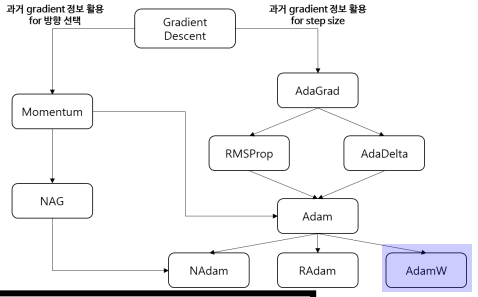
$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - (\beta_1)^{t+1}}$$

$$\hat{G}_{t+1} = \frac{G_{t+1}}{1 - (\beta_2)^{t+1}}$$

Weight decay term

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{G}_{t+1} + \varepsilon}} \hat{v}_{t+1} - \lambda\theta_t$$

Gradient Decent Algorithm



❖ Decoupled Weight Decay Regularization (AdamW)

- 저자들은 regularization term과 weight decay term을 분리하여 이 문제를 해결

$$g_{t+1} = g_t + \lambda\theta_t \quad \text{L2-regularization term}$$

$$v_{t+1} = \beta_1 v_t - (1-\beta_1) g_{t+1}$$

$$G_{t+1} = \beta_2 G_t + (1-\beta_2) g_{t+1}^2$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1-(\beta_1)^{t+1}}$$

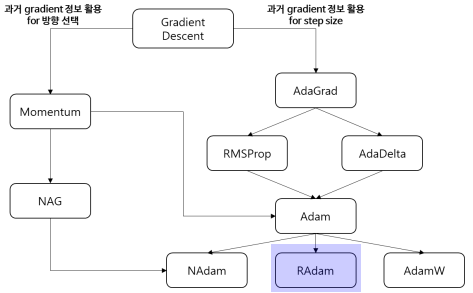
$$\hat{G}_{t+1} = \frac{G_{t+1}}{1-(\beta_2)^{t+1}}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{G}_{t+1}+\varepsilon}}\hat{v}_{t+1} - \lambda\theta_t$$

- **Pros**
  - Adam 에서 l2-regularization이 weight decay 효과를 낼 수 없다는 것을 확인하고 weight decay와 l2-regularization을 분리하여 일반화 성능 향상

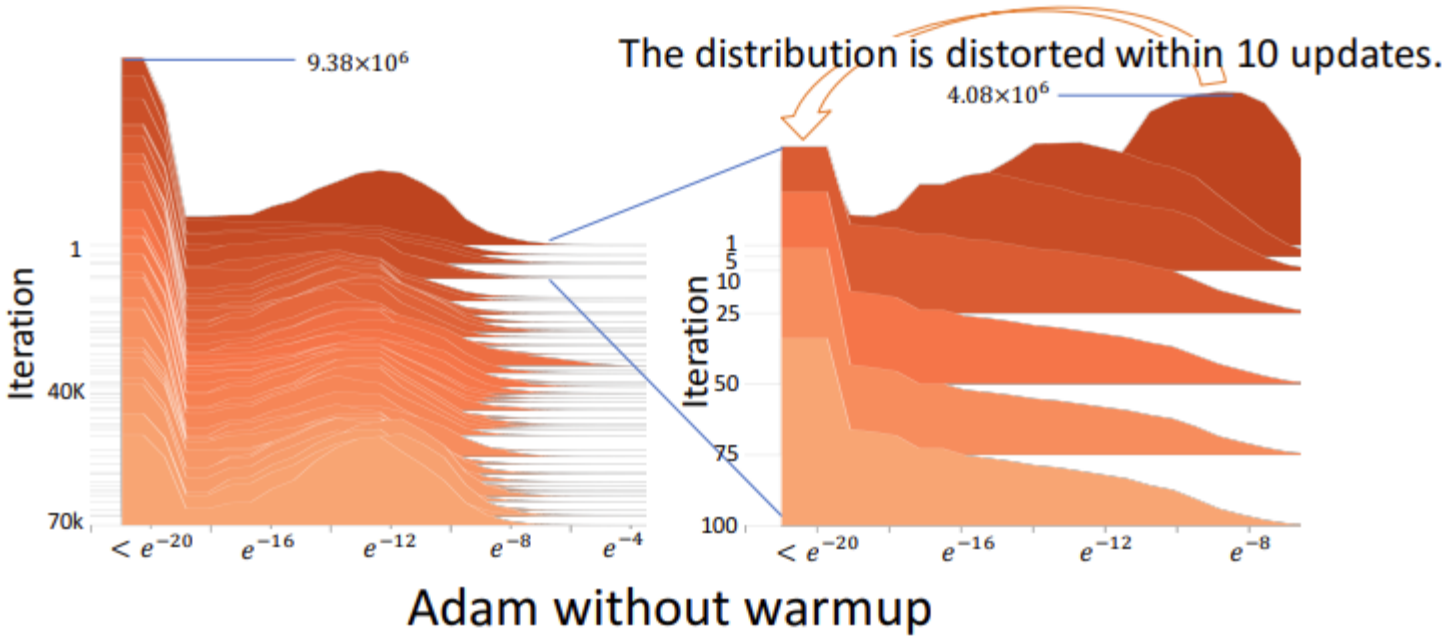
weight decay term

Gradient Decent Algorithm

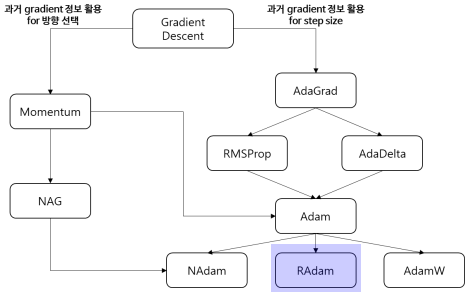


❖ Rectified Adam (RAdam)

- Adaptive learning rate method에서 학습 초기에 샘플이 부족하여 adaptive learning rate의 분산이 커짐
- 이에 따라 초기에 local minima에 빠지게 될 수 있고 너무 일찍 도달하여 학습이 거의 일어나지 않을 수 있음
- 이를 'Bad Local Optima Convergence Problem'이라 하며 이 문제를 해결하기 위해 RAdam 개발



Gradient Decent Algorithm



❖ Rectified Adam (RAdam)

- 저자들은 adaptive learning rate의 분산을 구하고 이를 일정하게 만들어 주기 위한 rectification term을 제안

1. t-step에서의 adaptive lr 분산 정의

$$Var[\psi(g_1, g_2, \dots, g_t)] = Var[\sqrt{x}] = E[x] - E[\sqrt{x}]^2$$

2. Rectification term 정의

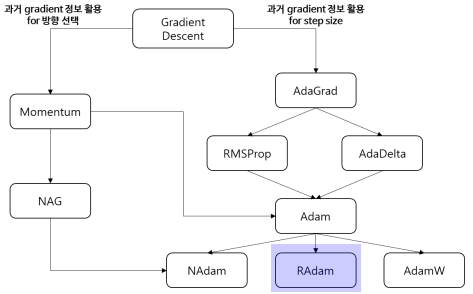
$$r_t = \sqrt{\frac{C_{var}}{Var[\psi(g_1,g_2,\dots,g_t)]}}, C_{var} : Var[\psi(\cdot)]\text{의 최소값}$$

$$\rightarrow \quad Var[r_t\psi(\cdot)] = r_t^2 Var[\psi(\cdot)] = \frac{C_{var}}{Var[\psi(\cdot)]} \cdot Var[\psi(\cdot)] = C_{var}$$

3. Rectification term 추정

$$r_t = \sqrt{\frac{(\rho_t - 4)((\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)((\rho_\infty - 2)\rho_t}} \quad , \qquad \rho_\infty = \frac{2}{1-\beta_2} - 1, \rho_t = \rho_\infty - 2t\beta_2^t/(1-\beta_2^t)$$

Gradient Decent Algorithm



❖ Rectified Adam (RAdam)

- 저자들은 adaptive learning rate의 분산을 구하고 이를 일정하게 만들어 주기 위한 rectification term을 제안

$$v_t = \beta_1 v_{t-1} - (1 - \beta_1) g_t$$

$$G_t = \beta_2 G_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{v}_t = \frac{v_t}{1 - (\beta_1)^t}$$

$$\hat{G}_t = \frac{G_t}{1 - (\beta_2)^t}$$

$$\rho_\infty = \frac{2}{1 - \beta_2} - 1$$

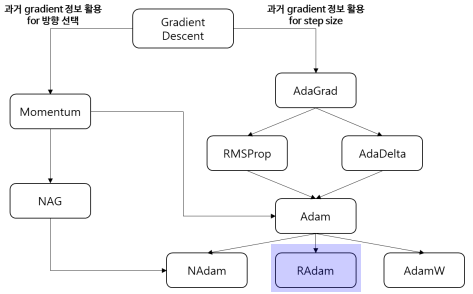
$$\rho_t = \rho_\infty - 2t\beta_2^t / (1 - \beta_2^t)$$

$$r_t = \sqrt{\frac{(\rho_t - 4)((\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)((\rho_\infty - 2)\rho_t}}$$

$$l_t = \sqrt{(1 - \beta_2^t)/v_t}$$

$$\theta_t = \theta_{t-1} - \eta r_t l_t \hat{v}_t$$

Gradient Decent Algorithm



❖ Rectified Adam (RAdam)

- 저자들은 adaptive learning rate의 분산을 구하고 이를 일정하게 만들어 주기 위한 rectification term을 제안

$$v_t = \beta_1 v_{t-1} - (1 - \beta_1) g_t$$

$$G_t = \beta_2 G_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{v}_t = \frac{v_t}{1 - (\beta_1)^t}$$

$$\hat{G}_t = \frac{G_t}{1 - (\beta_2)^t}$$

$$\rho_\infty = \frac{2}{1 - \beta_2} - 1$$

$$\rho_t = \rho_\infty - 2t\beta_2^t / (1 - \beta_2^t)$$

$$r_t = \sqrt{\frac{(\rho_t - 4)((\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)((\rho_\infty - 2)\rho_t}}$$

$$l_t = \sqrt{(1 - \beta_2^t)/v_t}$$

$$\theta_t = \theta_{t-1} - \eta r_t l_t \hat{v}_t$$

I. Introduction

II. Definition of Optimizer

III. Gradient Descent Algorithm

IV. Variants of Gradient Descent

V. Conclusions

- **Pros**
  - Adaptive learning rate method 전반에서 발생하는 문제인 'Bad Local Optima Convergence Problem'를 rectification term을 통해 해결

I. Introduction

II. Definition of Optimizer

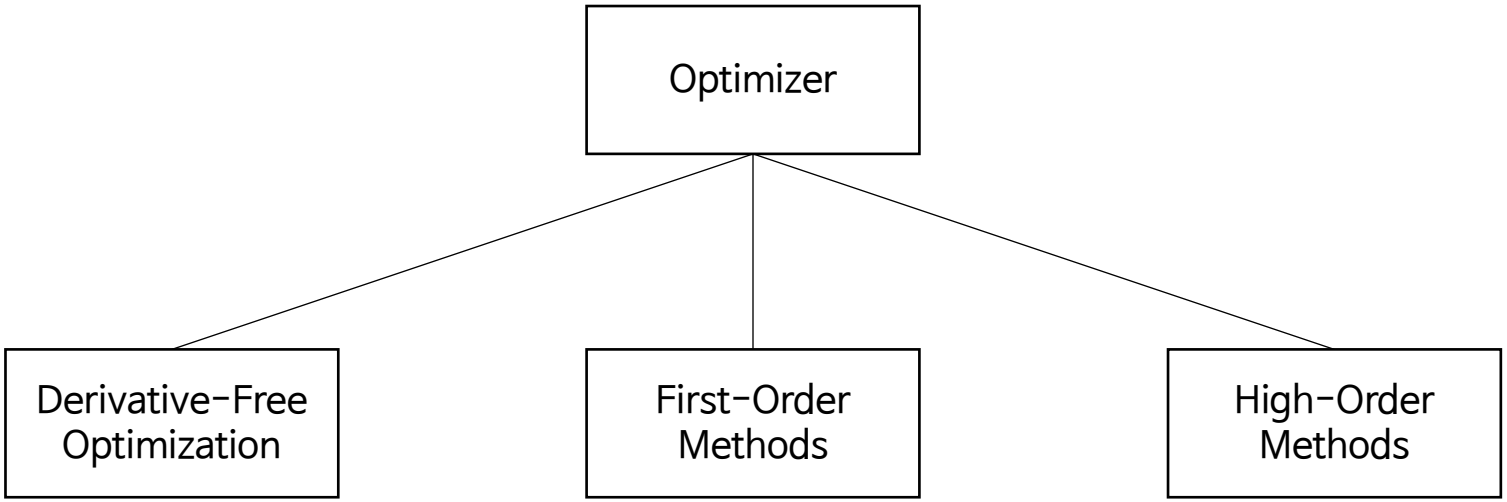
III. Gradient Decent Algorithm

IV. Variants of Gradient Descent

V. Conclusions

Conclusions

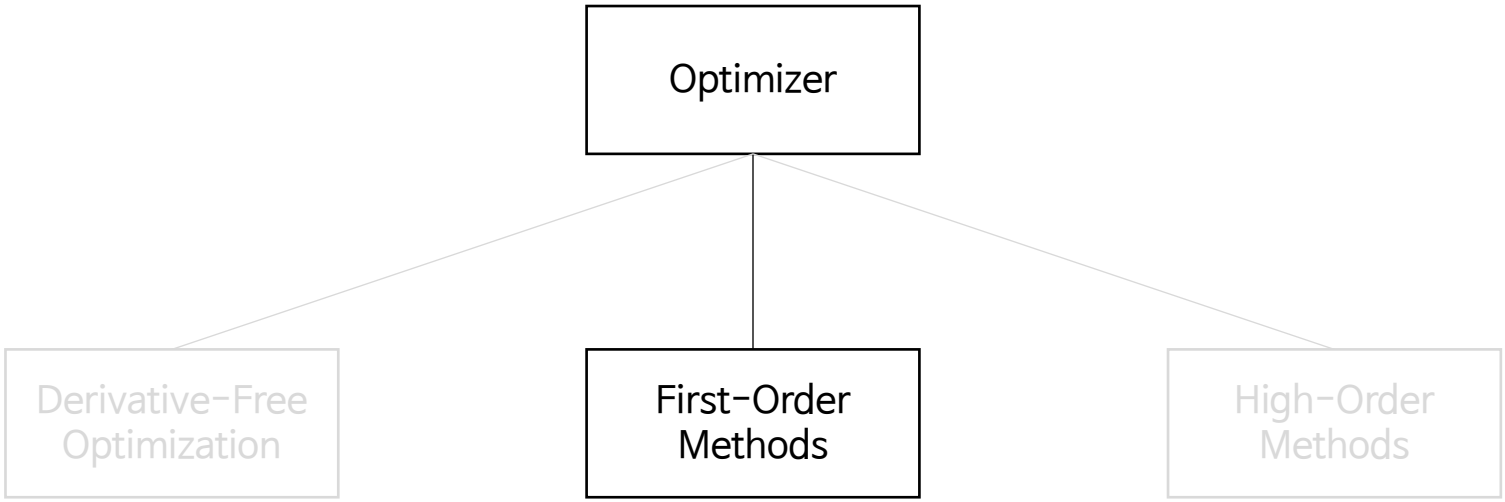
❖ Toward Optimal Optimizer



- 목적함수의 미분이 불가능한 상황에서 사용하는 최적화 알고리즘
- 1차 미분 정보인 gradient 활용
- 가장 일반적인 형태
- 고차원 미분 정보(Hessian 등) 활용
- 목적함수가 highly non-linear and ill-conditioned 일 때 사용
- 계산 비용이 큼

Conclusions

❖ Toward Optimal Optimizer



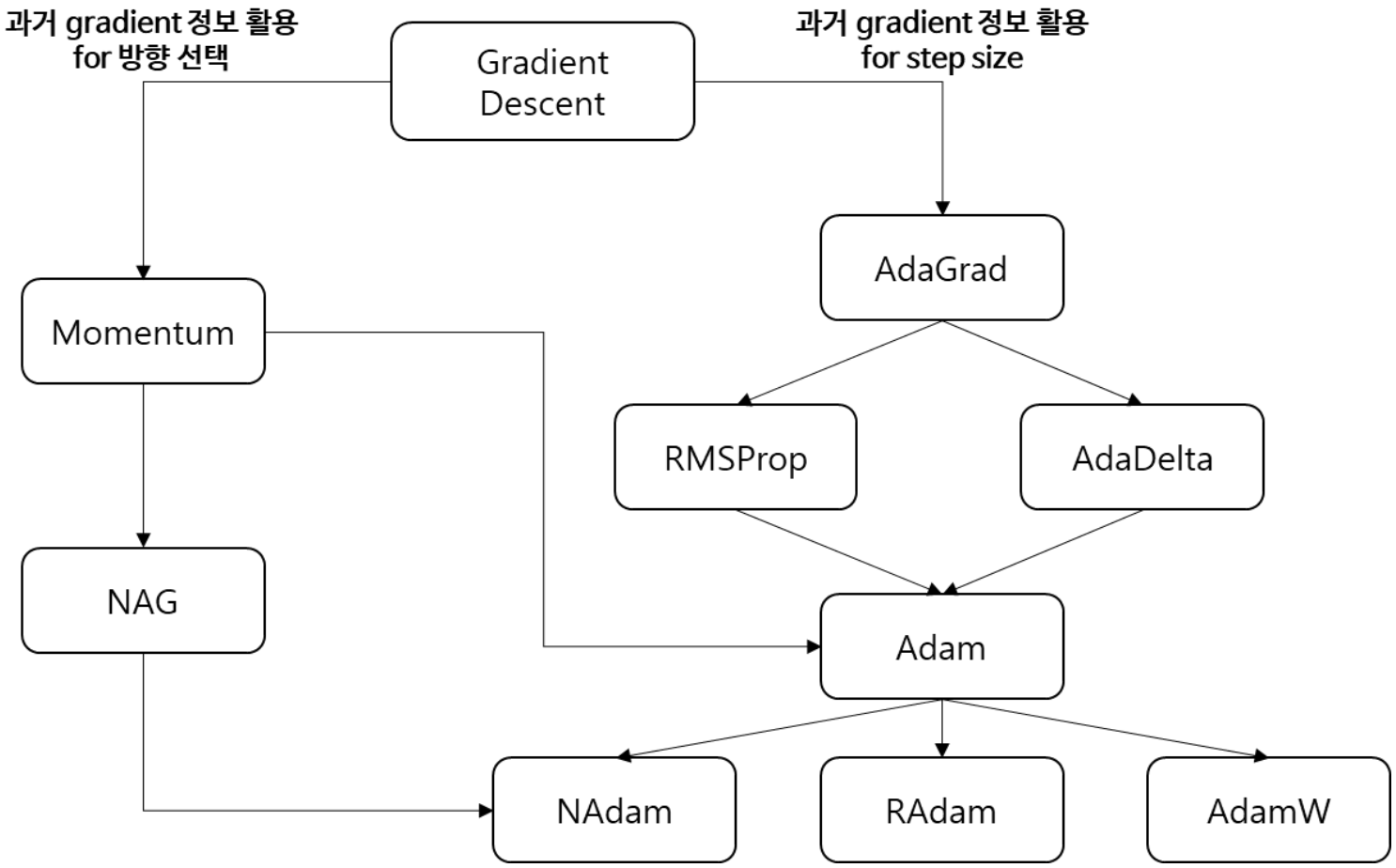
- 목적함수의 미분이 불가능한 상황에서 사용하는 최적화 알고리즘

- 1차 미분 정보인 gradient 활용
- 가장 일반적인 형태

- 고차원 미분 정보(Hessian 등) 활용
- 목적함수가 highly non-linear and ill-conditioned 일 때 사용
- 계산 비용이 큼

Conclusions

❖ Toward Optimal Optimizer



Reference

❖ Paper

- Sun, Shiliang, et al. "A survey of optimization methods from a machine learning perspective." IEEE transactions on cybernetics 50.8 (2019): 3668-3681.
- Kastrati, Muhamet, and Marenglen Biba. "A State-of-the-art Survey of Advanced Optimization Methods in Machine Learning." RTA-CSIT. 2021.
- Liu, Liyuan, et al. "On the variance of the adaptive learning rate and beyond." arXiv preprint arXiv:1908.03265 (2019).
- Loshchilov, Ilya, and Frank Hutter. "Decoupled weight decay regularization." arXiv preprint arXiv:1711.05101 (2017).

❖ Blogs

- https://dbstndi6316.tistory.com/297
- https://www.upgrad.com/blog/types-of-optimizers-in-deep-learning/
- https://zzaebok.github.io/deep_learning/RAdam/
- https://hiddenbeginner.github.io/deeplearning/paperreview/2019/12/29/paper_review_AdamW.html
- https://www.youtube.com/watch?v=_F5_hgX_lSE

감사합니다