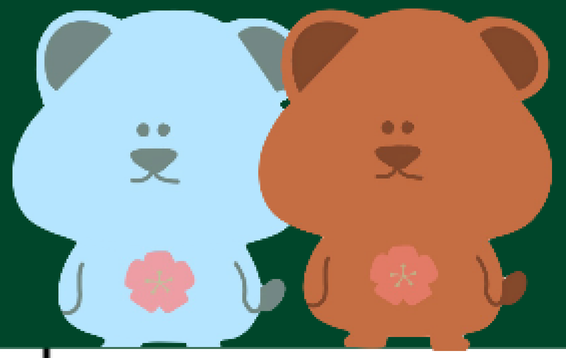


ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

장윤서

EWHA EURON

목차

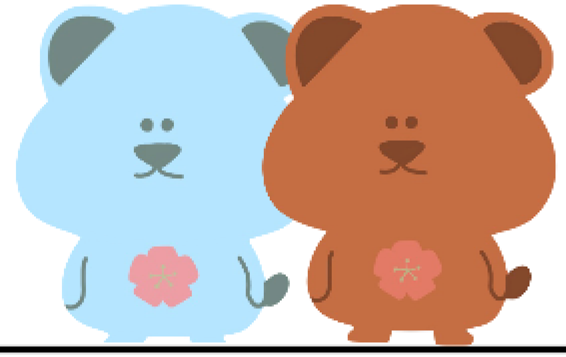
#01 논문 선정 동기

#02 Related Studies

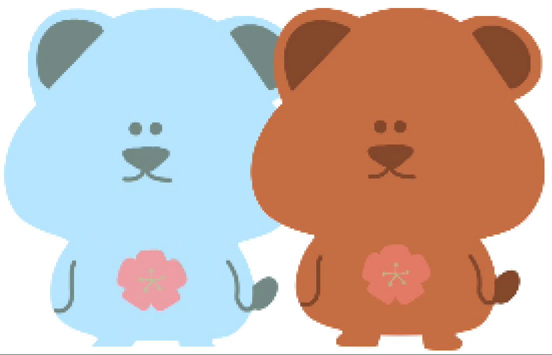
#03 Introduction

#04 Methods

#05 Results



논문 선정 동기



#01 논문을 고른 이유

1. 최적화 방법론의 전반적인 연구 흐름도 정리

2. Adam을 이해하고 사용하자

Related Studies

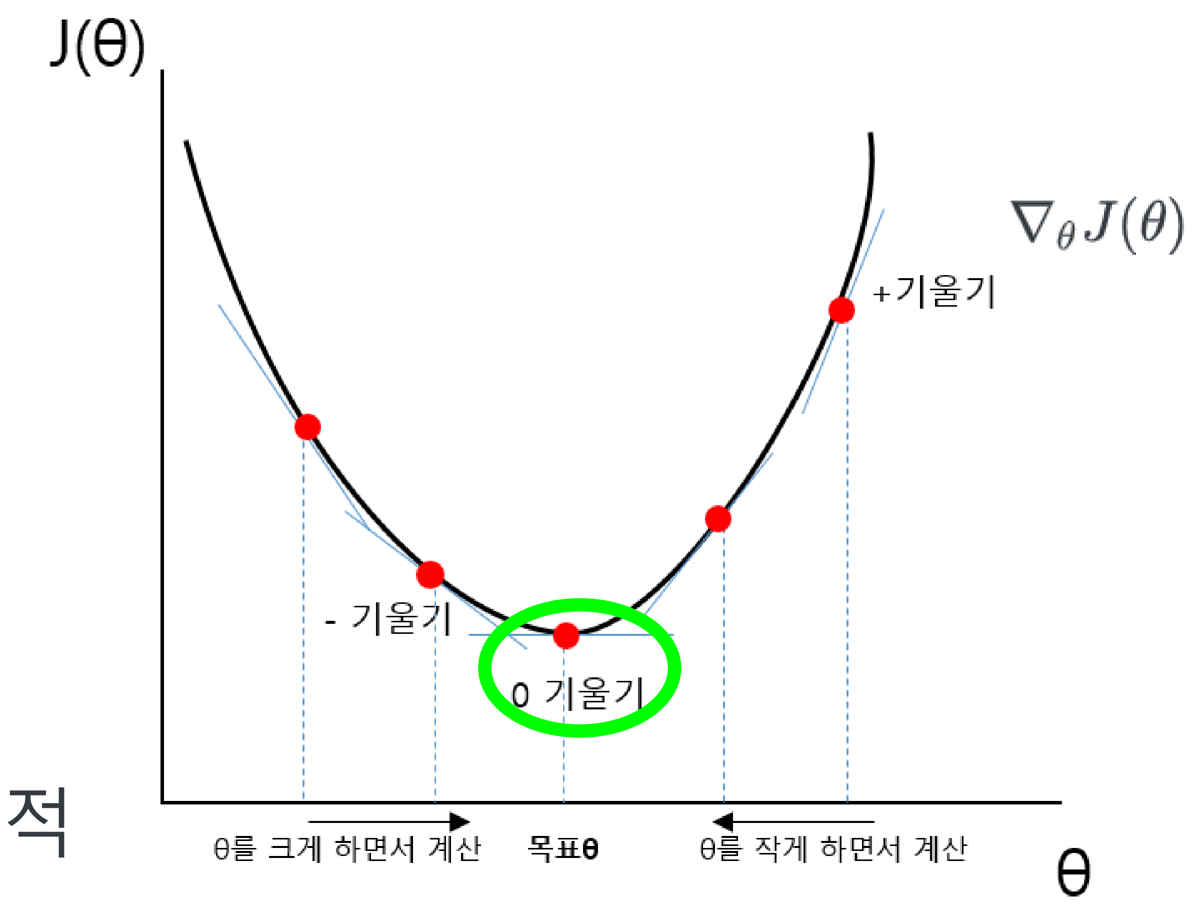
#01 Gradient-based optimization

최적화 (optimization)
Loss function J(θ)의 최솟값을 찾아가는 과정

Gradient based optimization

$\nabla_\theta J(\theta)$ Θ에 대해 손실함수 J(Θ)의 기울기

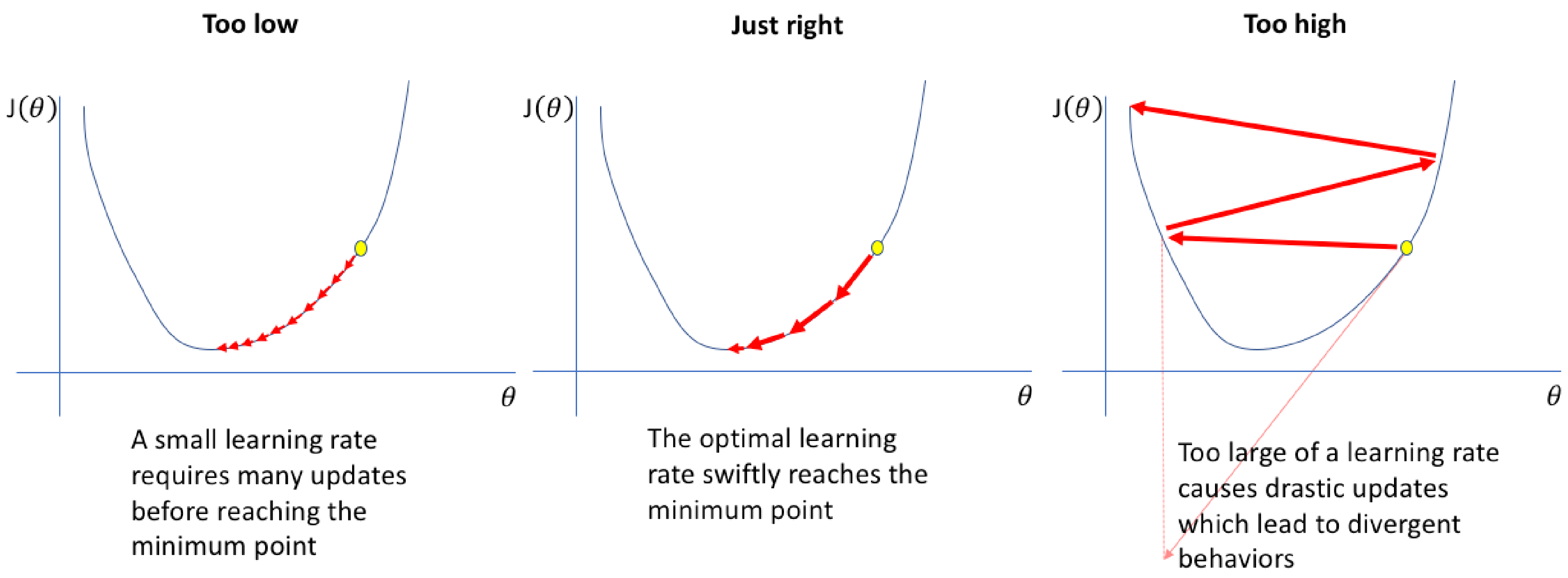
Θ 를 조금씩 변화시켜가면서 손실함수의 min값에 도달할 때까지 J(θ)의 기울기 $\nabla_\theta J(\theta)$ 를 계산함으로써 최소 loss 찾는 최적화 방법
Θ를 줄여나가는 경사하강법(Gradient Descent)이 대표적



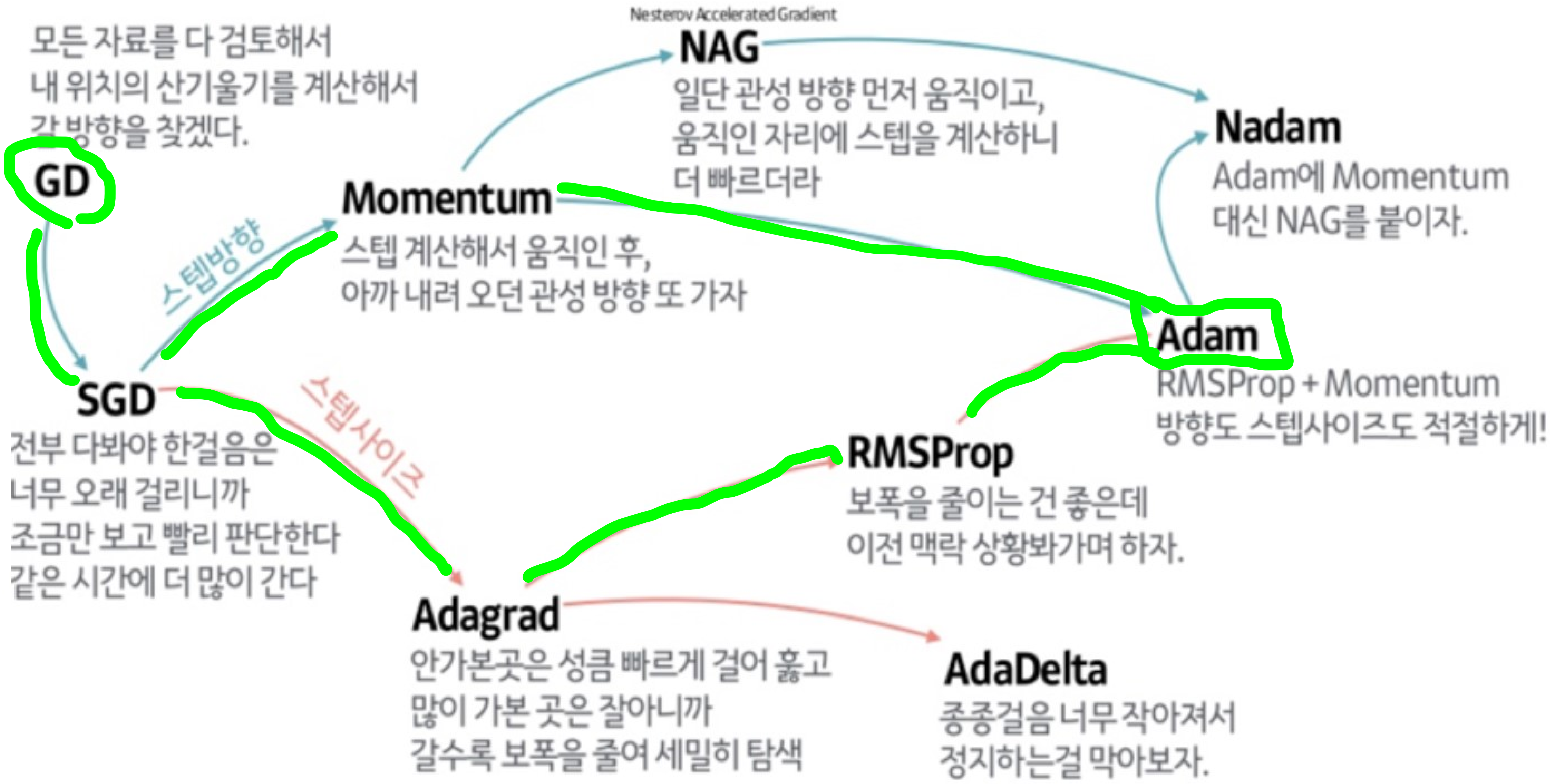
이때 Θ를 한번 갱신하는 각 단계를 Iteration이라고 한
한 번의 iteration에서 변화 식 :

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

η : 스텝의 크기, 학습 속도(Learning Rate)
Learning Rate가 클수록 Θ가 큰 차이로 갱신된다
Learning Rate가 너무 작으면 시간이 오래 걸림
Learning Rate가 너무 크면 min값을 지나칠 수 있음



#02 Gradient descent variants

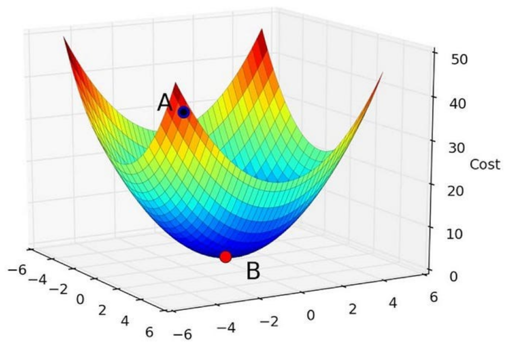


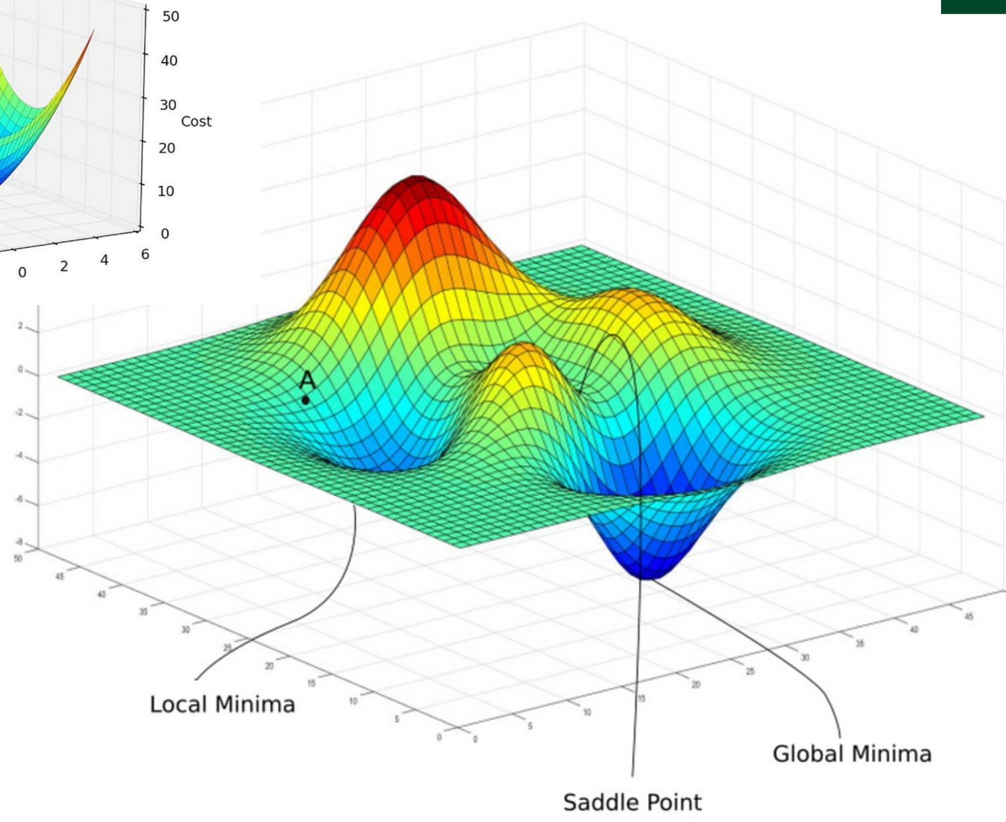
#03 Gradient descent variants

Batch gradient descent (Vanilla gradient descent)

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

한 Iteration에서 전체 학습 데이터셋(batch)에 대해 loss를 계산
➔ 많은 계산량, 너무 오래 걸림
➔ 그래프가 convex하지 않은 경우(=볼록그래프가 아닌 경우) local minima에 빠질 우려



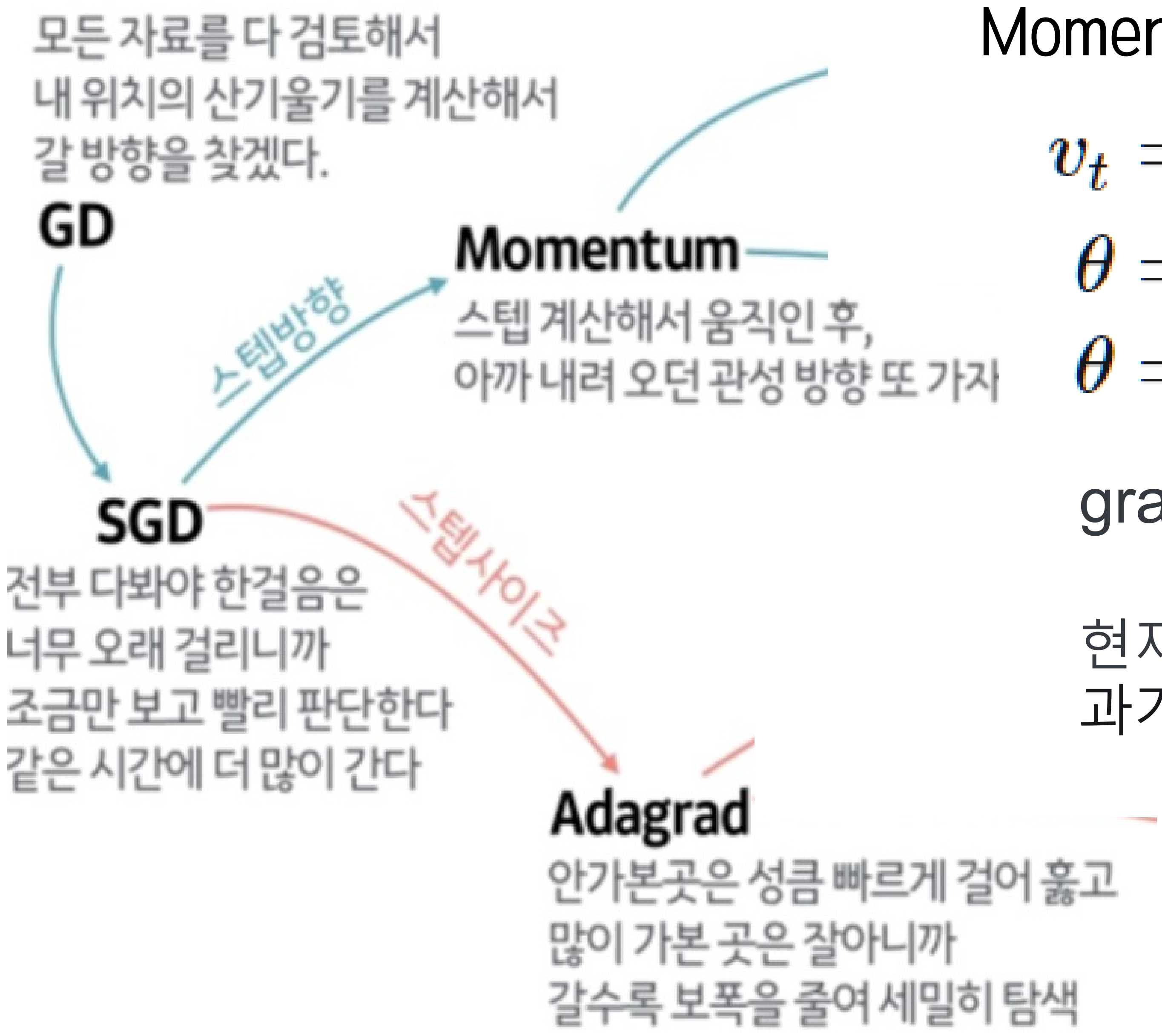


Stochastic Gradient Descent (SGD)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

전체 데이터(batch) 대신 일부 조그마한 데이터의 모음(mini-batch)에 대해서만 loss 계산
x(i), y(i)만을 loss 계산에 사용함
➔ Vanila보다 더 빠른 속도로 유사한 성능
➔ Local minia에 빠지지 않고 더 좋은 방향으로 수렴할 가능성이 있음

#04 SGD variants – Momentum



Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$
$$\theta = \theta - \left(\gamma v_{t-1} + \eta \nabla_\theta J(\theta)\right)$$

Vt : 각 time step에서의 이동 벡터
Γ : momentum을 얼마나 줄 것인지에 대한 변수

gradient descent를 통해 θ 를 갱신하는 과정에 관성을 주는 방법

현재 이동 방향과는 별개의 과거에 이동했던 방향을 기억해 θ 가 이동할 때
과거 이동 방향의 일정 정도를 추가적으로 이동시킴

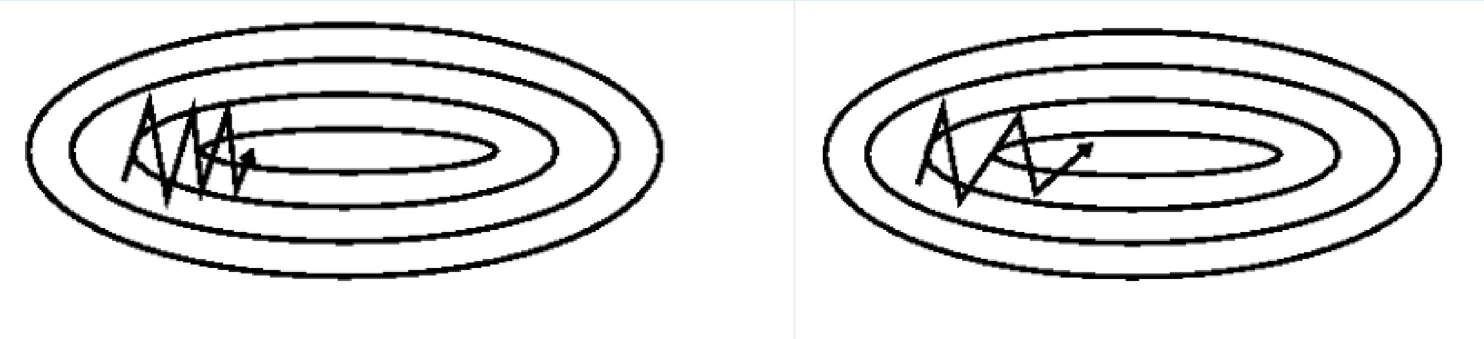
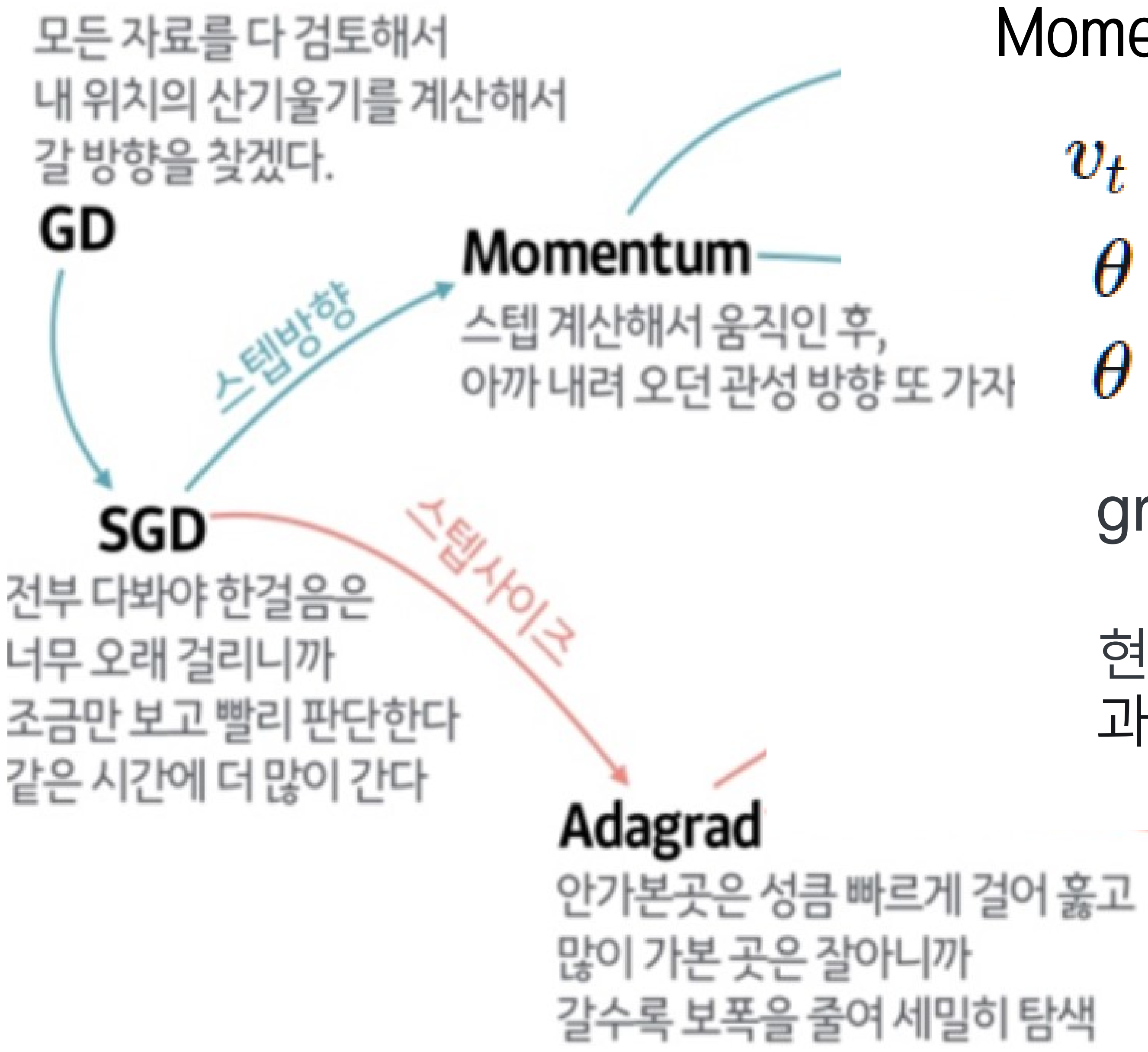


Image 2: SGD without momentum

Image 3: SGD with momentum

Momentum은 SGD가 진동 현상을 겪을 때 한번에 이동하는 step size를
늘리면서도 자주 이동하는 방향에 대한 관성이 생기게 되어 최적값으로 가는
방향에 힘을 얻게 됨 -> 더 빠른 최적화

#04 SGD variants – Momentum



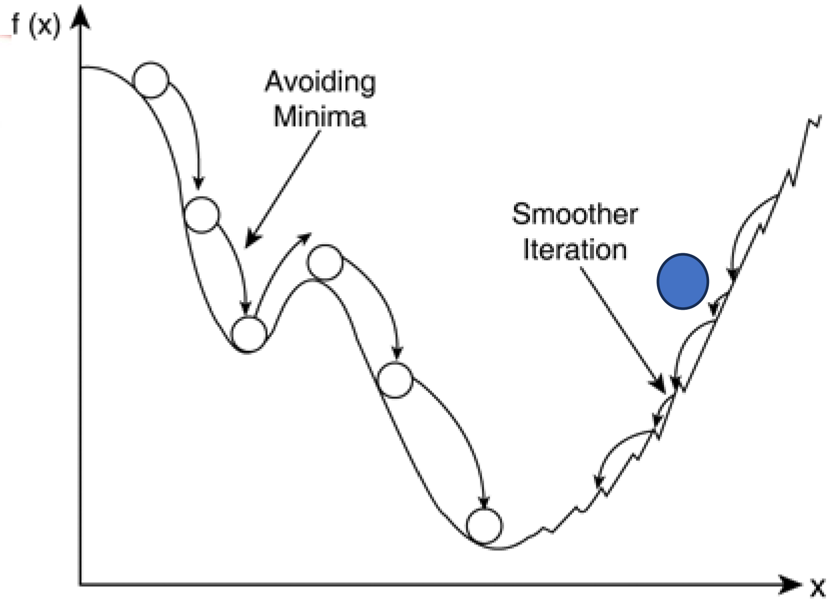
Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$
$$\theta = \theta - \left(\gamma v_{t-1} + \eta \nabla_\theta J(\theta)\right)$$

Vt : 각 time step에서의 이동 벡터
Γ : momentum을 얼마나 줄 것인지에 대한 변수

gradient descent를 통해 θ 를 갱신하는 과정에 관성을 주는 방법

현재 이동 방향과는 별개의 과거에 이동했던 방향을 기억해 θ 가 이동할 때
과거 이동 방향의 일정 정도를 추가적으로 이동시킴

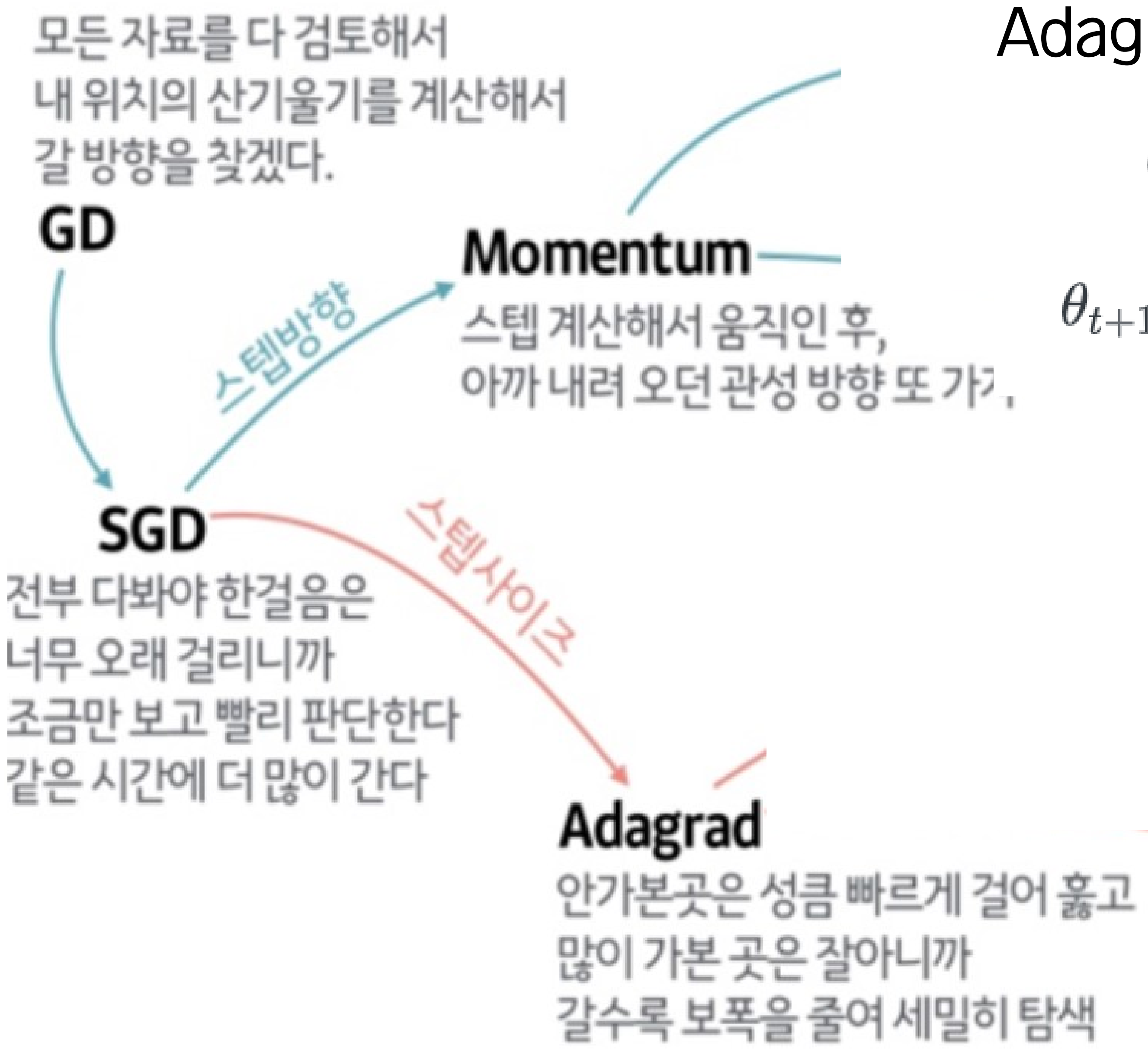


Momentum은 local minima를 빠져나올 수 있는
효과 또한 기대할 수 있음

기존에 이동했던 방향에 관성이 있어 local
minima를 빠져나올 수 있기 때문 (탱탱볼)

But 기존 방향을 저장해야 하기 때문에
메모리가 두배로 쓰인다는 단점 있음

#05 SGD variants - Adagrad



Adagrad(Adaptive Gradient)

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \boxed{\frac{\eta}{\sqrt{G_t + \epsilon}}} \cdot \nabla_\theta J(\theta_t)$$

Gt : time step까지 각 변수가 이동한 gradient의 sum of squares (기울기 제곱의 합)

learning rate η를 Gt의 제곱근으로 나눈 값 x J 기울기 만큼 θ를 갱신함

ε : 0으로 나눠지는 것을 방지하는 극소값

변수들을 update할 때 각각의 변수마다 step size를 다르게 설정해서 이동하는 방식

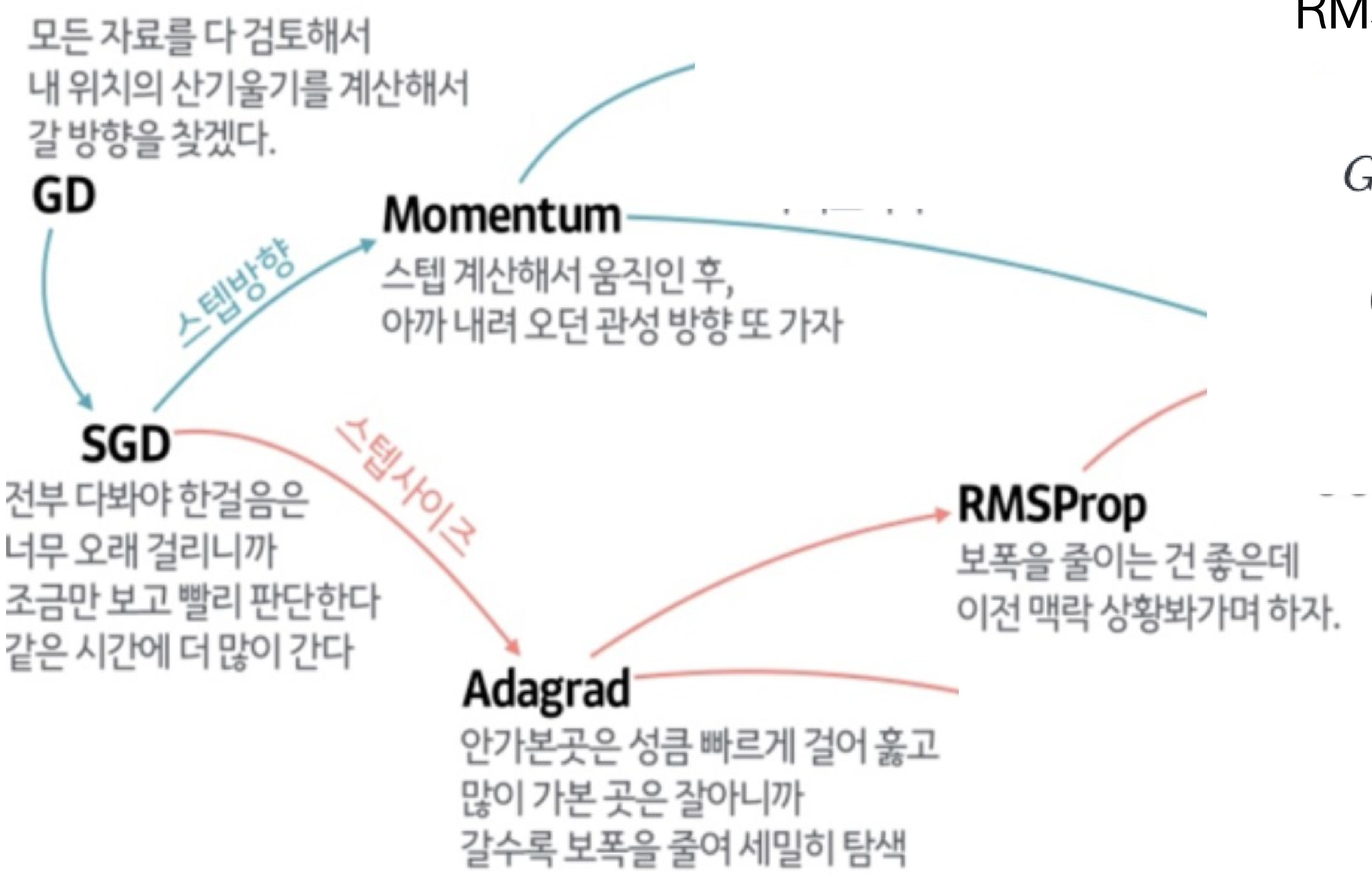
- 변화가 많았던 변수들 : optimum에 가까히 있을 확률이 높기에 작은 크기로 이동하면서 세밀하게 값을 조정
- 적게 변화한 변수들 : optimum값에 도달하기 위해 많이 이동해야할 확률이 높기에 먼저 빠르게 loss값을 줄이는 방향으로 탐색

word representation을 학습시킬 경우 단어의 등장 확률에 따라 variable의 사용 비율이 확연하게 차이나기 때문에 Adagrad가 효율적임

➔ Step size가 점차 줄어들기 때문에 따로 learning rate schedule (step size decay)를 지정해주지 않아도 된다는 장점

➔ G가 점점 커지기 때문에 시간이 지나면서 step size가 0에 가까워진다는 문제점

#06 RMSProp



RMSprop

$$G = \boxed{\gamma G + (1-\gamma)(\nabla_\theta J(\theta_t))^2}$$

$$\theta = \theta - \frac{\eta}{\sqrt{G+\epsilon}} \cdot \nabla_\theta J(\theta_t)$$

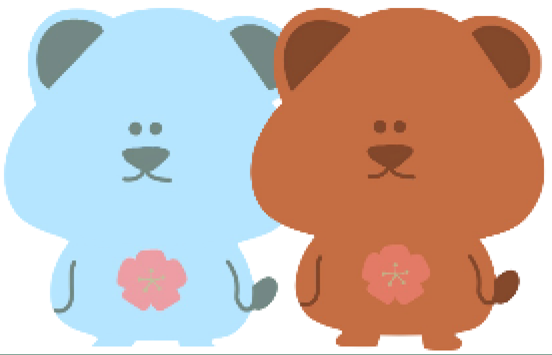
Gt의 기울기 제곱의 합 부분을 지수평균으로 대체함

예시 : $E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$

Adagrad의 단점을 보완한 방법

- 최근 변화량의 변수간 상대적인 크기 차이를 반영하면서 Adagrad처럼 Gt가 무한정 커져 step이 0으로 수렴하는 문제를 방지

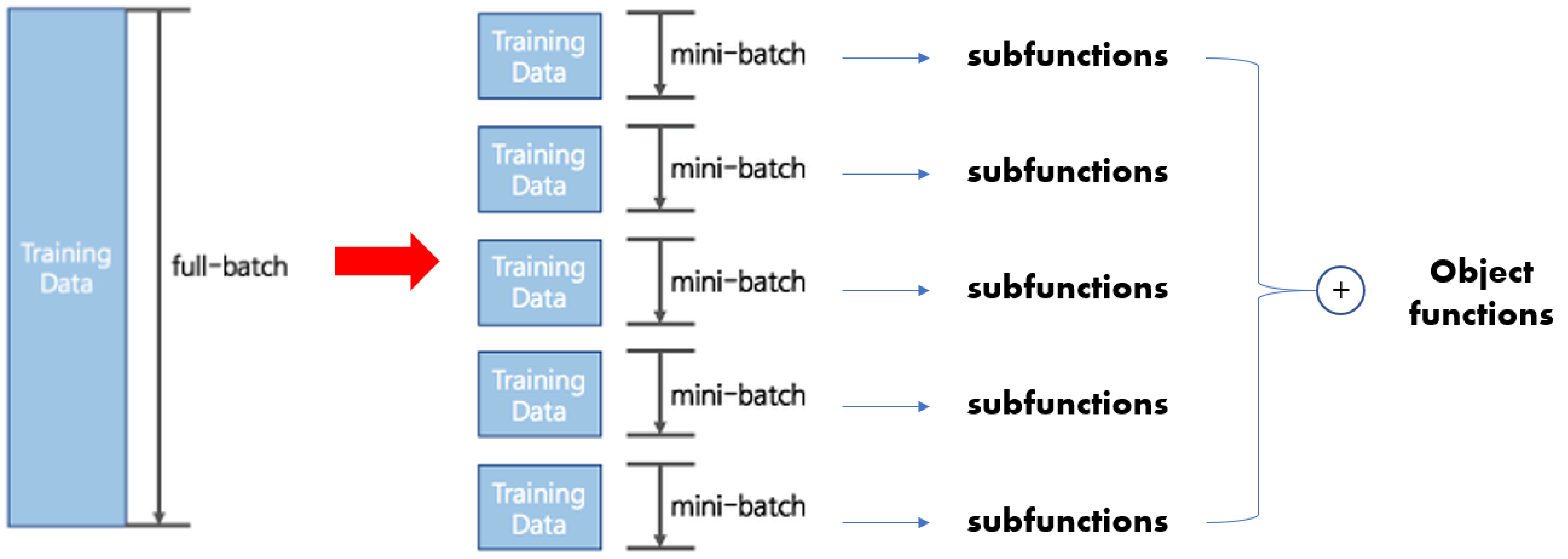
Introduction



EWHA EURON

#01 Objective Function

Objective Function is Stochastic



많은 목적함수들(object functions)은 다른 subsample 데이터(mini-batch로 분할된 학습 데이터)에서 평가된 subfunctions의 합으로 구성됨

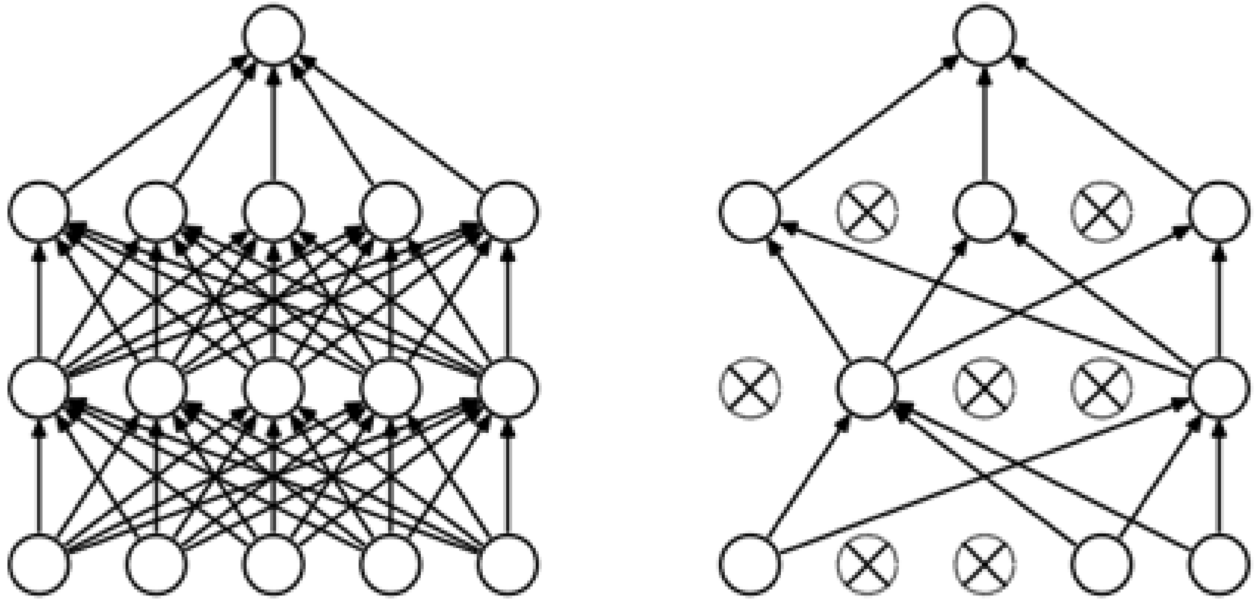
➔목적 함수는 확률적 성질을 가짐

이 경우 각 미니 배치마다 gradient step size을 다르게 설정할 수 있기 때문에 학습에 효율적임 (ex. SGD 방법)

EWHA EURON

#02 Stochastic gradient-based optimization

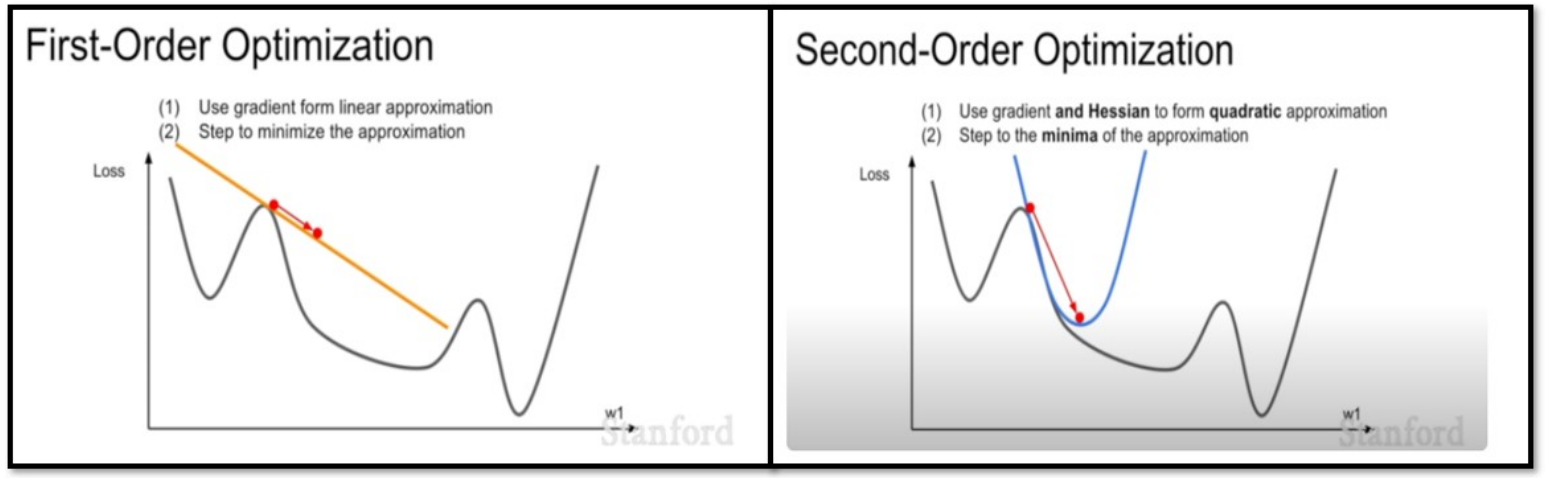
기존 확률적 그래디언트 기반 최적화 방법의 문제점



일반적인 신경망 구조　　Drop-out이 적용된 신경망

dropout regularization등 목적함수에 노이즈가 발생하면 목적함수의 최적화 성능이 저하됨, 따라서 더욱 효율적인 목적 함수가 필요

First-Order Optimization vs higher-order optimization

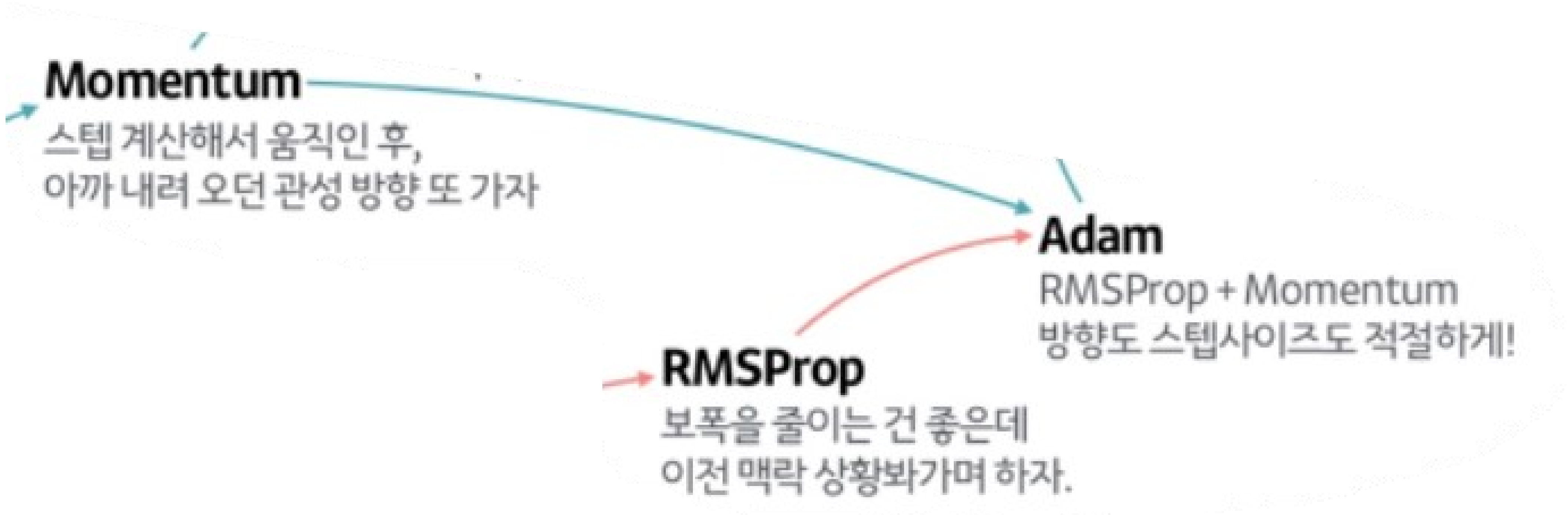


First-Order Optimization : 1차 미분한 가중치를 최적화에 반영하는 방식
1차 함수 방향아로만 최적화를 진행하기 때문에 탐색이 제한적

하지만 고차 함수 최적화의 경우 시간 복잡도가 크게 증가해 아직까지는 First-Order Optimization 만을 사용중

#03 Adam (Adaptive Moment Estimation)

RMSProp과 momentum 방식을 합친 것과 같은 알고리즘 방식



Momentum의 관성 + RMSProp의 각 변수마다 스텝 사이즈를 다르게 설정하는 방식을 동시 적용

Adam의 장점
- 간단한 구현과 효율적인 연산
- 데이터 및 모델 파라미터가 많이 필요한 경우에 적합함
- noisy gradient에 적합함
- 기존 최적화 방법 (SGD, Adagrad, RMSProp)에 비해 우수한 성능

Methods

#01 Algorithm

초기에 필요한 4가지 파라미터

1. Stepsize α (Learing Rate) 학습 속도
2. Decay Rates β1, β2 (0~1) : 그래디언트의 decay rate를 조정하는 역할을 하는 변수, Adam의 유일 하이퍼 파라미터로 최신값의 비중을 조정한다 (exponential decay)
3. 확률적 목적 함수 f(θ)
4. 초기 파라미터값 θ0

알고리즘

1) $m_0 \leftarrow 0$ (Initialize 1st moment vector) 화
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)

Mt : 모멘텀 역할 벡터 (방향 벡터)
Vt : RMSProp역할 벡터
업데이트가 많은 변수일 수록 높은 값이 배정됨

2) θt가 더 이상 수렴하지 않을 때까지 반복
   1) t=t+1
   2) t-1의 목적함수 그래디언트 계산 $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
   3) mt, vt 값 계산

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$$
$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$$

#01 Algorithm

4) bias-correction 적용

초기의 모멘텀 값이 0으로 초기화되는 경우를 방지

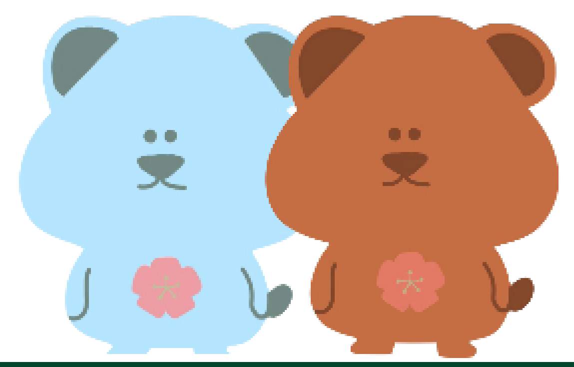
$$\widehat{m}_t \leftarrow m_t/(1-\beta_1^t)$$
$$\widehat{v}_t \leftarrow v_t/(1-\beta_2^t)$$

5) 최종 가중치 업데이트

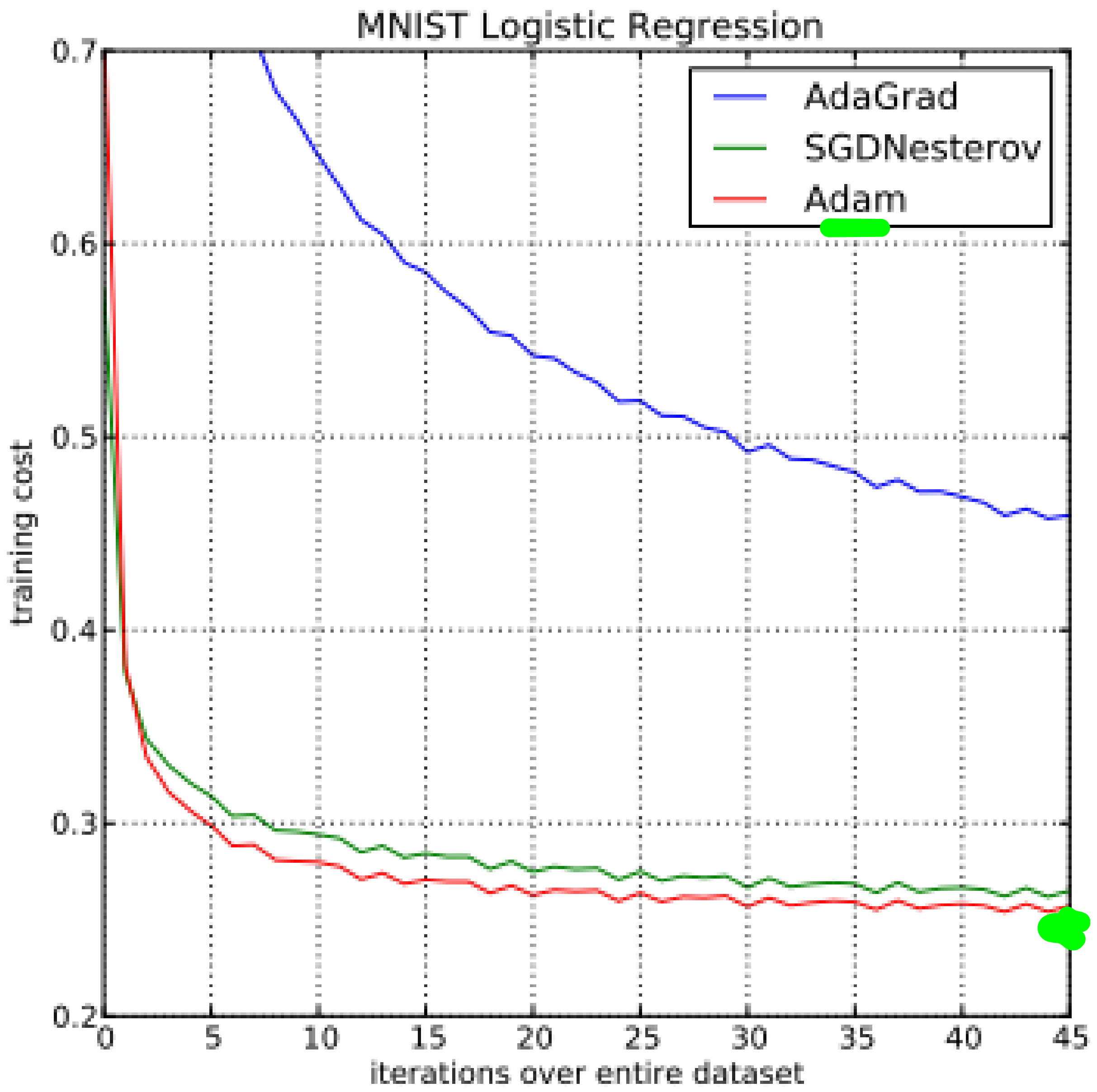
$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$$

EWHA EURON

Result



#01 Experiments



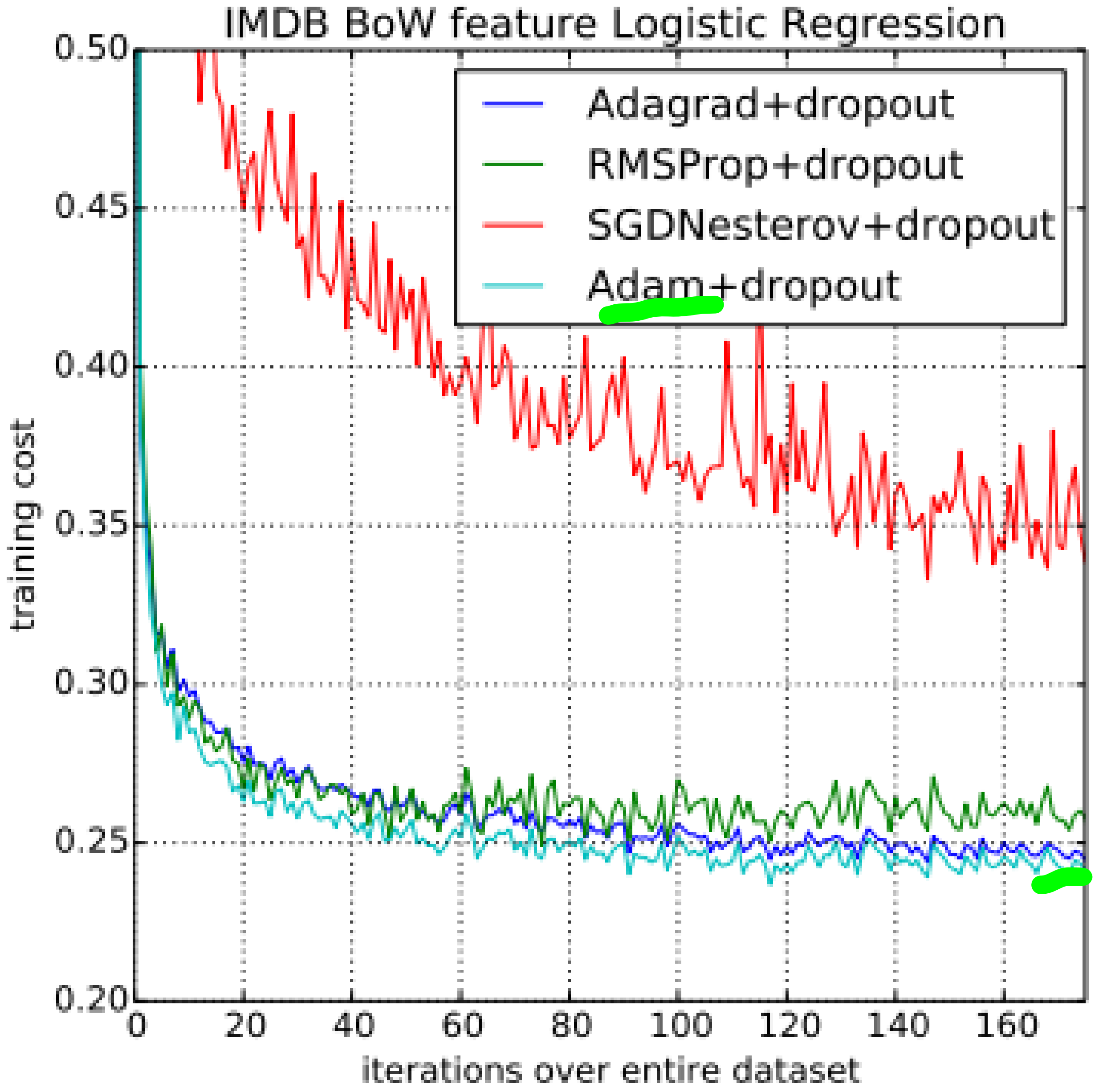
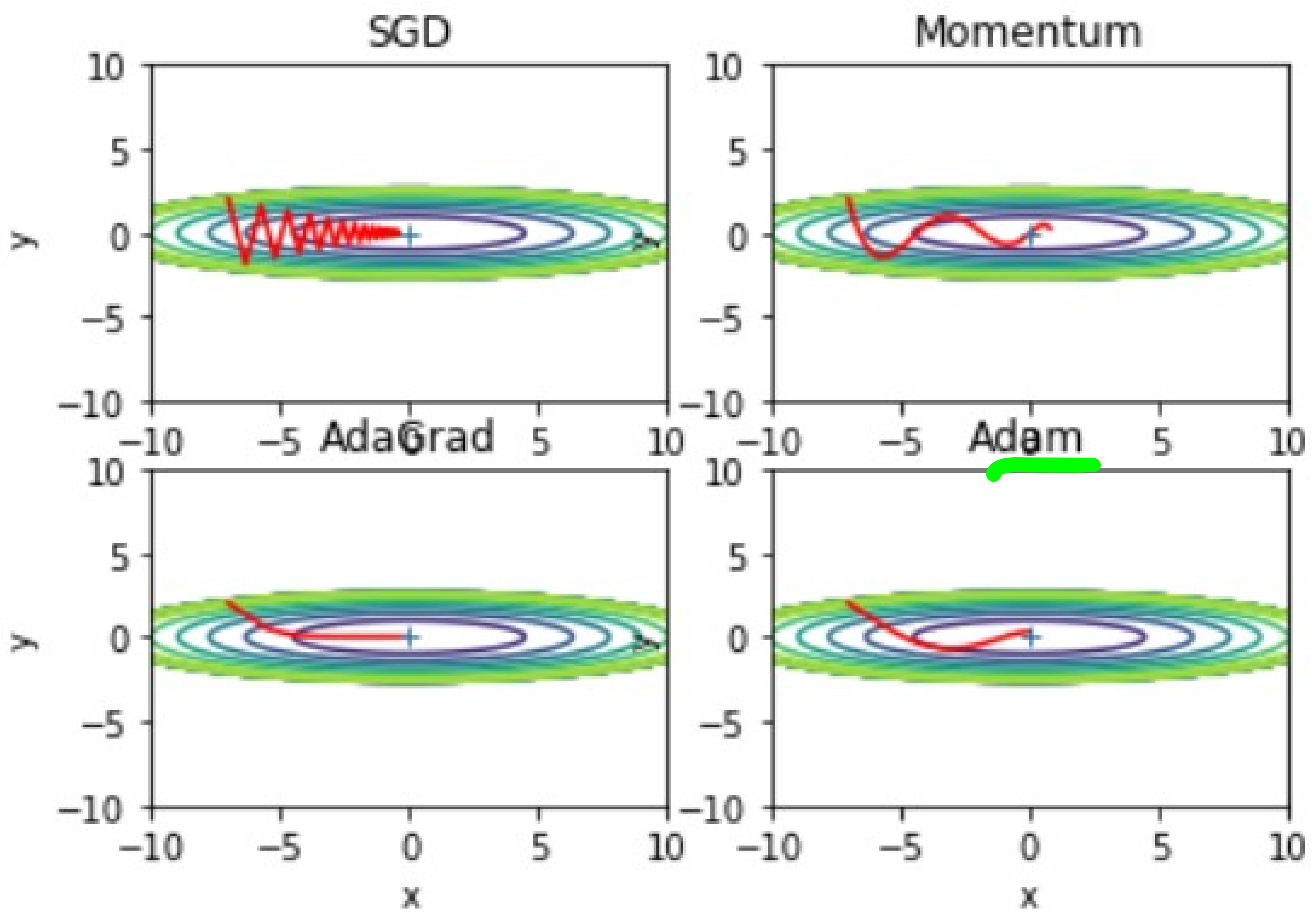


표 1. Logistic Regression에서 최적화 알고리즘의 비교 (좌 : MINIST, 우 : IMDB)

-> 기존 최적화 방법들보다 Adam의 성능이 뛰어남

#02 각 optimazation 비교



최적화 기법 비교: SGD, 모멘텀, AdaGrad, Adam

각 최적화 방법의 비교 이미지 (참고)

Discussion