

Section 8. 최적화 (Optimization)

목차

- 섹션 7. 활성 함수 (Activation Function)
- **섹션 8. 최적화 (Optimization)**
- 섹션 9. PyTorch로 만들어보는 Fully Connected NN
- 섹션 10. 정규화 (Regularization)
- 섹션 11. 학습 속도 스케줄러 (Learning Rate Scheduler)
- 섹션 12. 초기화 (Initialization)
- 섹션 13. 표준화 (Normalization)

Recap from previous chapters

Gradient Descent의 종류들 (Section 5.)

- **Full-batch** Gradient Descent:

$$\nabla_w L \approx \frac{1}{N} \sum_{i=1}^N \nabla_w L|_{\mathbf{x}_i}$$

- **Stochastic** Gradient Descent:

$$\nabla_w L \approx \nabla_w L|_{\mathbf{x}_i}$$

- **Mini-batch** Stochastic Gradient Descent ($B \ll N$):

$$\nabla_w L \approx \frac{1}{B} \sum_{i=1}^B \nabla_w L|_{\mathbf{x}_i}$$

What we will learn!

- “Full-batch” / “Stochastic” / “Mini-batch Stochastic” Gradient Descent 외에 다른 최적화 방법은 없을까?
- 학습이 더 안정적이고, 더 빠르게 학습이 수렴하게 해주는 방법이 없을까?

Optimization

Objective

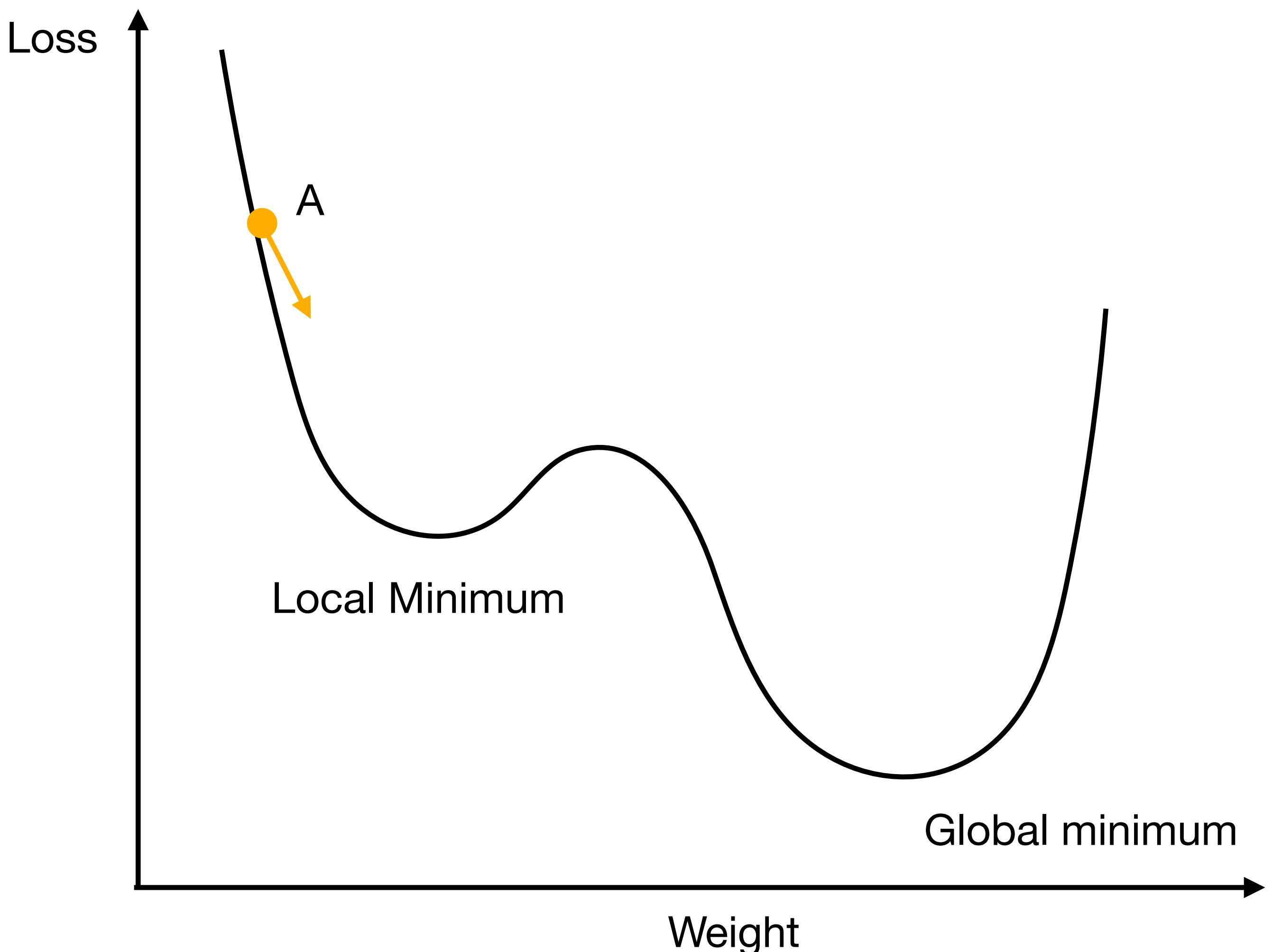
학습 목표

- Momentum (관성)의 개념 이해
- Acceleration (가속)의 개념 이해 (Nesterov's Accelerated Gradient)
- Adaptive Learning Rate 기반의 방법 이해
 - AdaGrad
 - AdaDelta
 - RMSProp
- Adaptive Moment Estimation (ADAM) 이해
- 각 방법의 장단점 이해

8-1. Momentum (관성)

Optimization Momentum

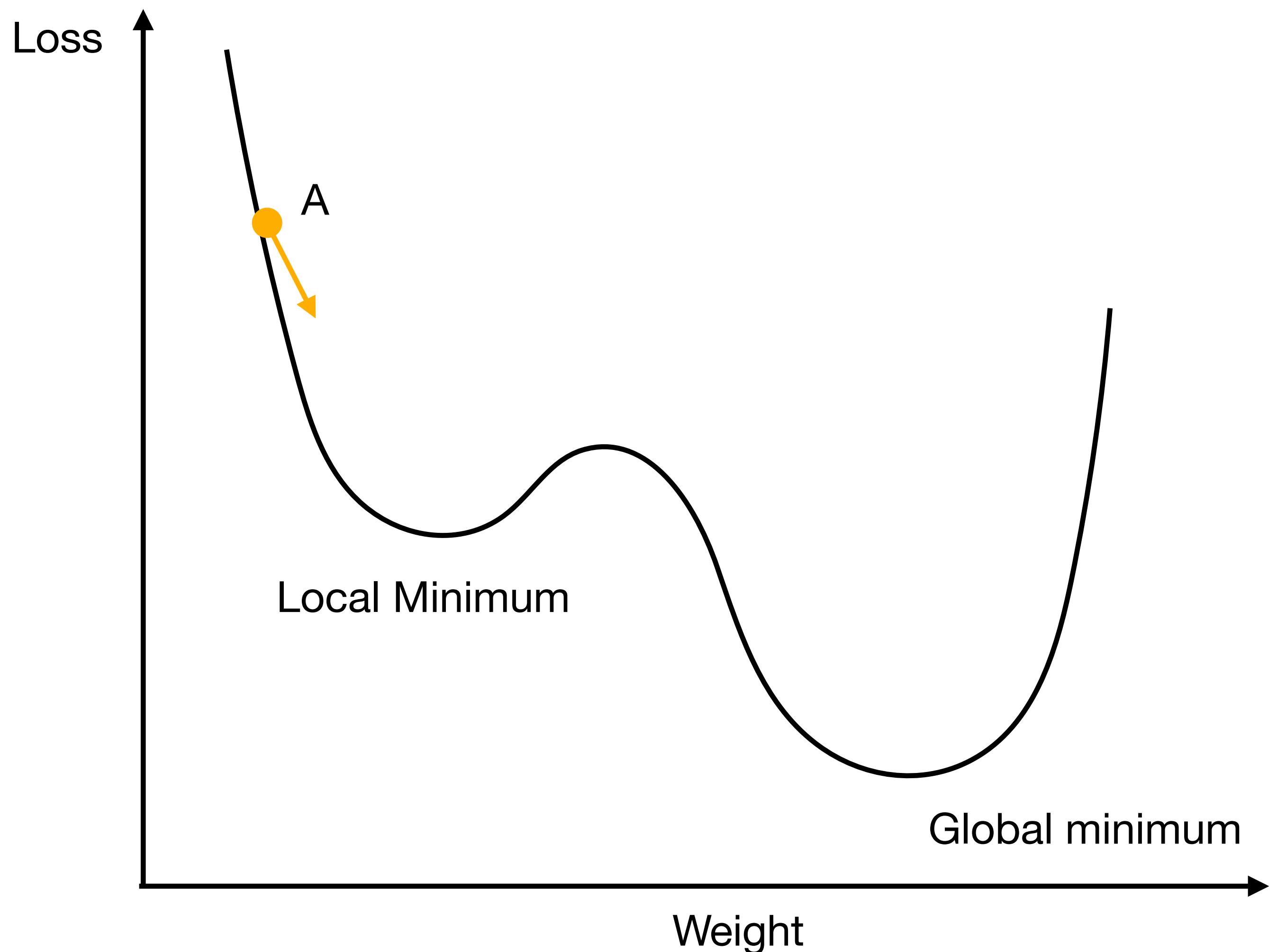
- A에서 initialize했을시, Gradient Descent은 Local minimum에 빠지게 됨.



Optimization

Momentum

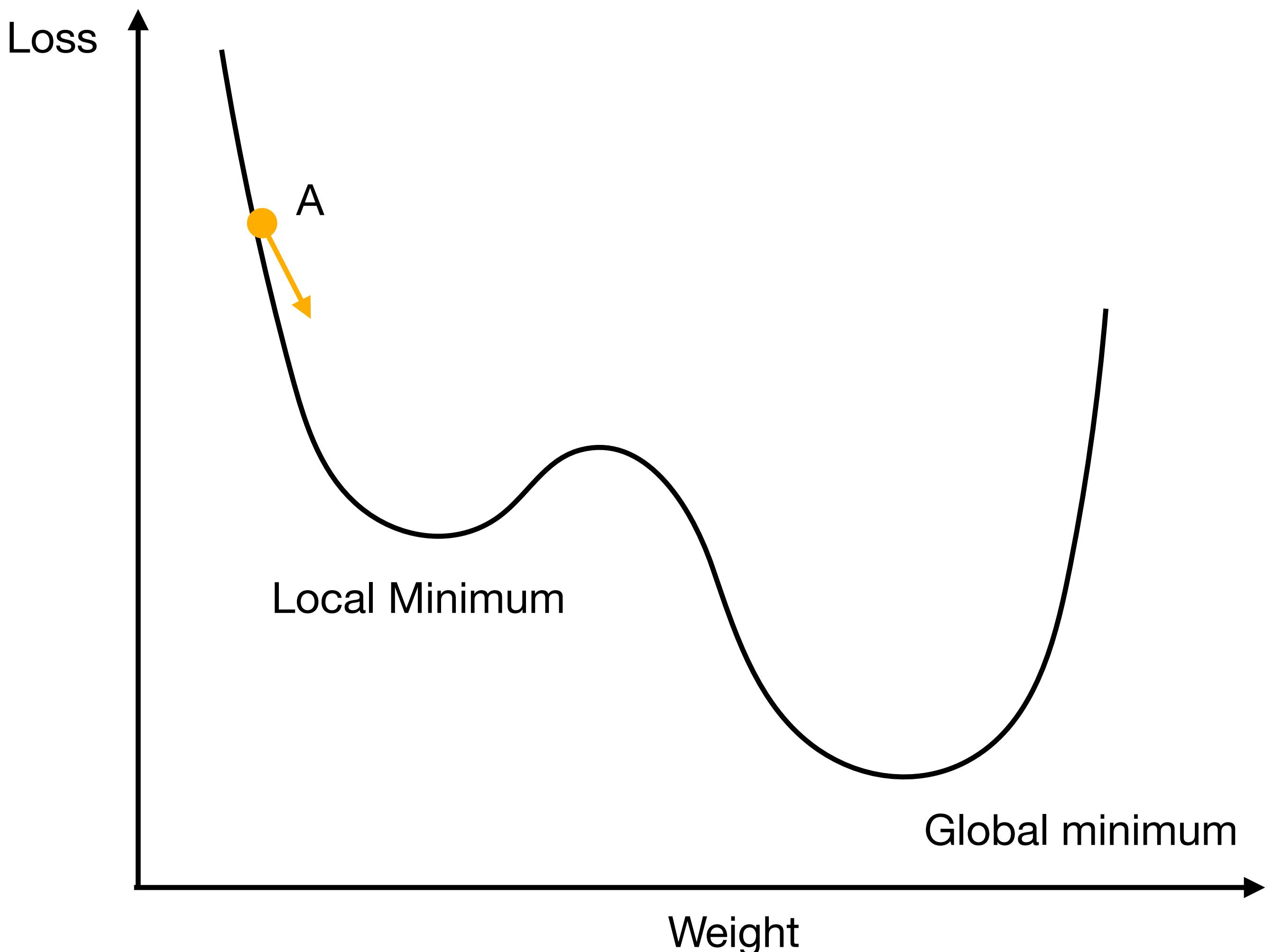
- A에서 initialize했을시, Gradient Descent은 Local minimum에 빠지게 됨.
- 어떻게 하면 Global Minimum에 도달할 수 있을까?



Optimization

Momentum

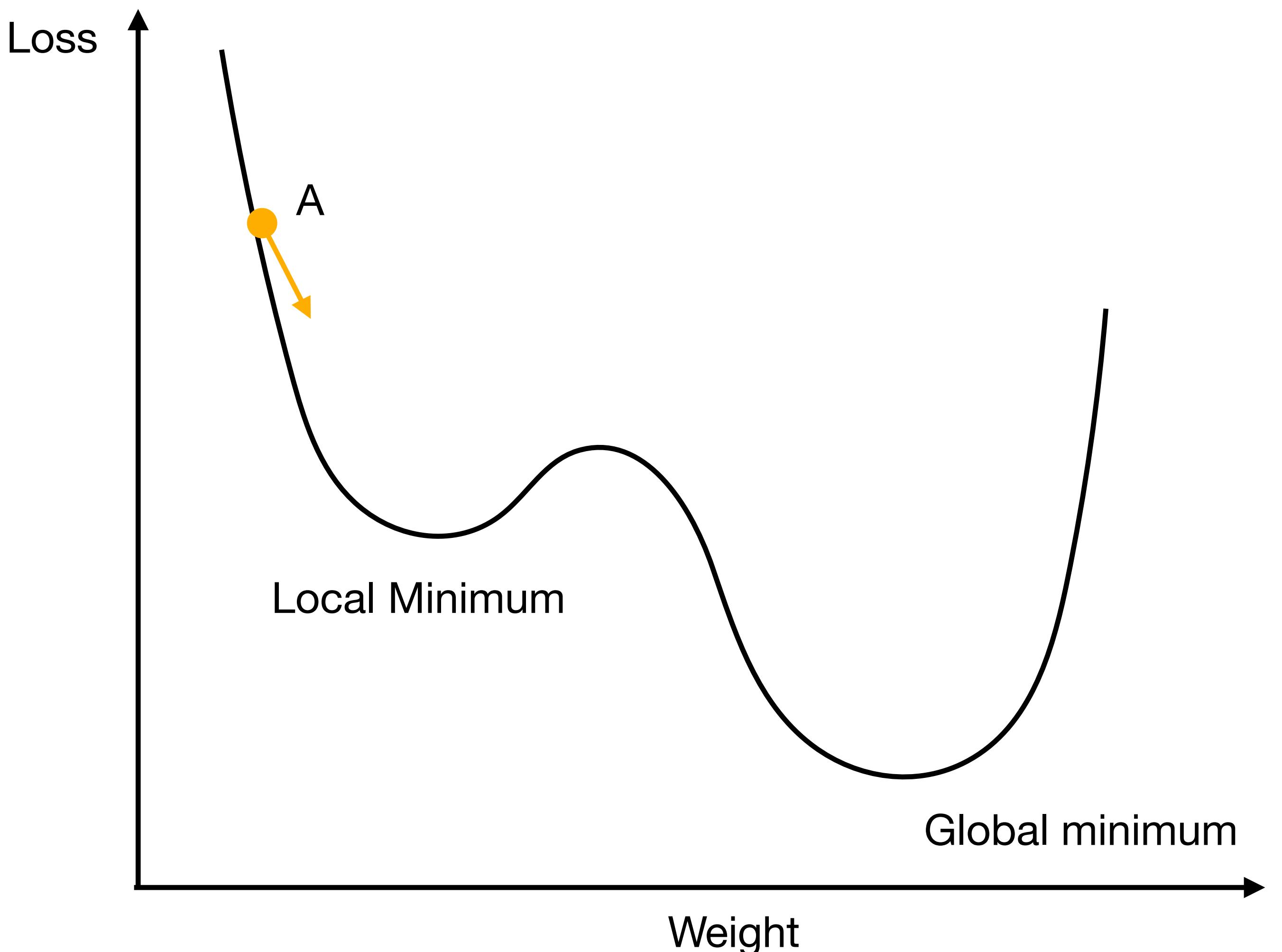
- A에서 initialize했을시, Gradient Descent은 Local minimum에 빠지게 됨.
- 어떻게 하면 Global Minimum에 도달할 수 있을까?
- **Momentum (관성)**을 활용!



Optimization

Momentum

- A에서 initialize했을시, Gradient Descent은 Local minimum에 빠지게 됨.
- 어떻게 하면 Global Minimum에 도달할 수 있을까?
- **Momentum (관성)을 활용!**
- 관성 = 원래 이동하던 방향으로 계속 이동하려는 경향성!



Optimization

Momentum

- **Momentum (관성)** = 원래 이동하던 방향으로 계속 이동하려는 경향성!
- i 번째 mini-batch에 대한 Loss Gradient:

$$g_i = \frac{\partial L(y_i, \hat{y}_i)}{\partial w}$$

Optimization

Momentum

- **Momentum (관성)** = 원래 이동하던 방향으로 계속 이동하려는 경향성!
- i 번째 mini-batch에 대한 Loss Gradient:

$$g_i = \frac{\partial L(y_i, \hat{y}_i)}{\partial w}$$

- Momentum을 사용할시 weight에 대한 update은 다음과 같다:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

Optimization

Momentum

- Momentum을 사용할시 weight에 대한 update은 다음과 같다:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

- λ = learning rate
- μ_i = Momentum parameter (0~1 값)
- \tilde{g}_i = Moving average (이동평균)

Optimization

Momentum

Copyright©2023. Acadential. All rights reserved.

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

$$\tilde{g}_{i+1} = \mu_i \cdot (\mu_{i-1} \tilde{g}_{i-1} + \lambda \cdot g_{i-1}) + \lambda \cdot g_i$$

$$\tilde{g}_{i+1} = \lambda \cdot g_i + \lambda \cdot \mu_i \cdot g_{i-1} + \mu_i \cdot \mu_{i-1} \cdot \tilde{g}_{i-1}$$

$$\tilde{g}_{i+1} = \lambda \cdot g_i + \lambda \cdot \mu_i \cdot g_{i-1} + \lambda \cdot \mu_i \cdot \mu_{i-1} \cdot g_{i-2} + \dots$$

Optimization

Momentum

- Momentum을 사용할시 weight에 대한 update은 다음과 같다:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

- λ = learning rate
- μ_i = Momentum parameter (0~1 값)
 - 해당 값이 작을수록 최근의 Loss Gradient g_i 를 \tilde{g}_{i+1} 에 더 많이 반영한다.
 - 해당 값이 클수록 이전 Loss Gradient을 더 많이 반영하고, 원래 이동하는 방향으로 계속 나아가려는 경향을 가지게 된다.

Optimization

Momentum의 효과

Copyright©2023. Acadential. All rights reserved.

학습의 수렴 속도를 더 높여줄 수 있다.

- 왜냐하면 (smooth한 loss surface을 가정했을시) minimum에 가까워지면서 Gradient의 크기가 줄어들어 GD의 학습 속도가 줄어든다.
- 관성은 \tilde{g}_{i+1} 가 이전의 Gradient들의 평균 크기를 반영하여 학습 속도가 줄어드는 것을 완화시켜준다.

Optimization

Momentum의 효과

Copyright©2023. Acadential. All rights reserved.

학습의 진동폭을 줄여준다.

- Learning Rate (학습률)이 너무 크면 GD은 oscillate (진동)할 수 있다.
- 관성은 \tilde{g}_{i+1} 가 이전의 Gradient들의 평균 방향을 반영하여 진동폭을 줄여줄 수 있다.

Optimization

Momentum의 효과

Copyright©2023. Acadential. All rights reserved.

관성은 Gradient가 noisy한 경우에 효과적이다.

- 관성은 이전의 Gradient들의 평균을 반영하여 noise가 평균화되어 noise들끼리 서로 상쇄되어 줄어드는 효과를 볼 수 있다.
- 그리고 Gradient의 평균 방향으로 이동할시 noise가 적을 수 있다.

Optimization

Momentum의 효과

관성은 saddle인 지점을 피하는데 효과적이다.

관성은 Local Minima를 빠져나오는데 도움을 줄 수 있다.

- 해당 지점에서의 Gradient은 0이지만 관성은 이전 Gradient을 반영하므로 해당 지점으로부터 벗어나는데 효과적이다.

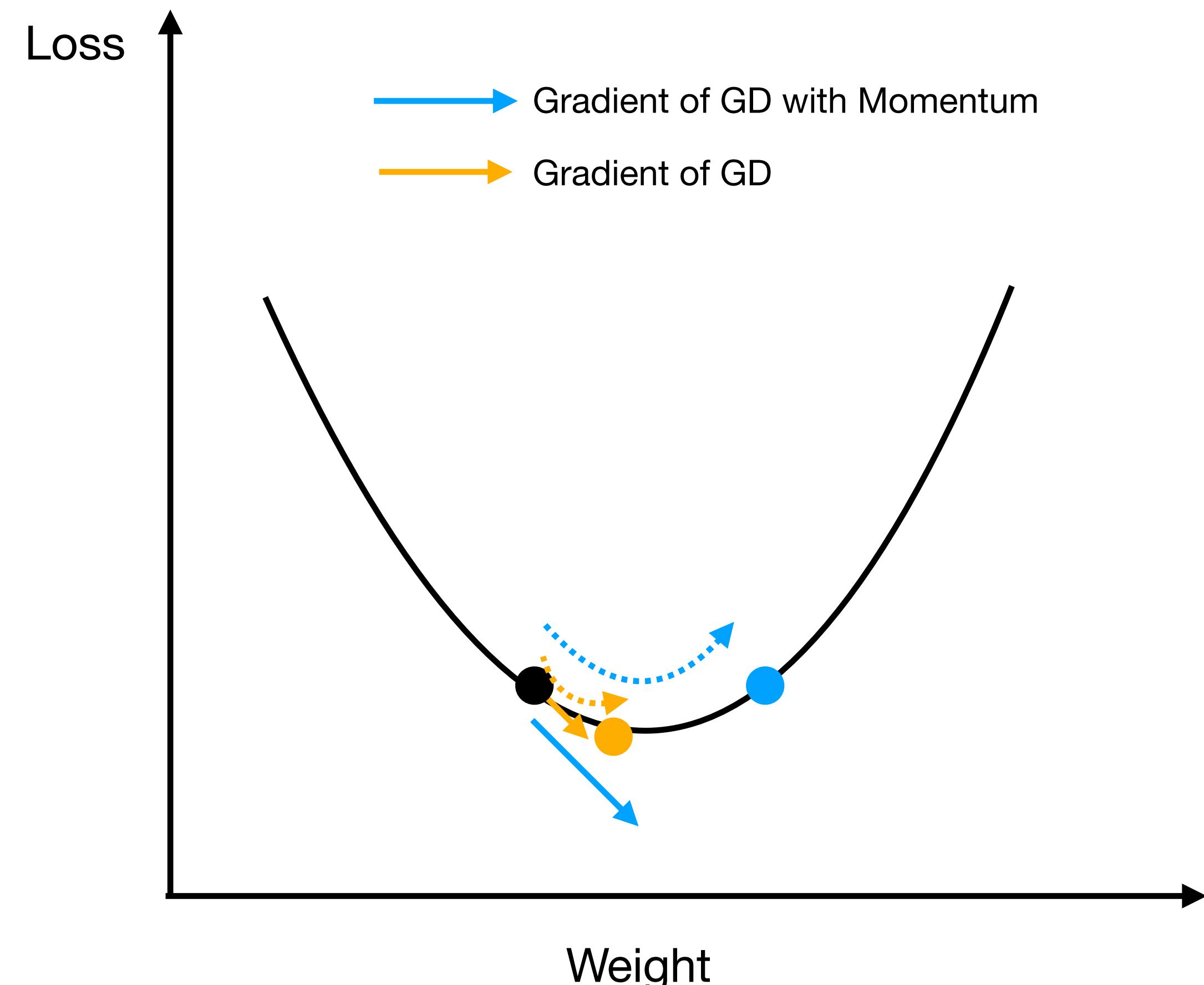
Optimization

Momentum 참고 사항

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

- 여기에서 μ_i 상수값은 **GD iteration**을 거듭할수록 줄여주는 것이 좋을 수 있다.
- 왜냐하면 minimum에 근처에 도달했을 시 수렴해야되는데 관성은 **minimum 주변으로 oscillate**할 수 있기 때문이다.



8-2. Nesterov Accelerated Gradient (NAG)

Optimization

Nesterov's Accelerated Gradient (NAG)

- Nesterov's Accelerated Gradient (NAG):

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g(w_i - \mu_i \tilde{g}_i)$$

- Recap on GD with Momentum:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

Optimization

Nesterov's Accelerated Gradient (NAG)

- Nesterov's Accelerated Gradient (NAG):

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g(w_i - \mu_i \tilde{g}_i)$$

- Recap on GD with Momentum:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

- NAG와 GD with Momentum은 서로 매우 유사하지만 로 표시된 부분이 서로 다르다!

Optimization

Nesterov's Accelerated Gradient (NAG)

- Nesterov's Accelerated Gradient (NAG):

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g(w_i - \mu_i \tilde{g}_i)$$

- Recap on GD with Momentum:

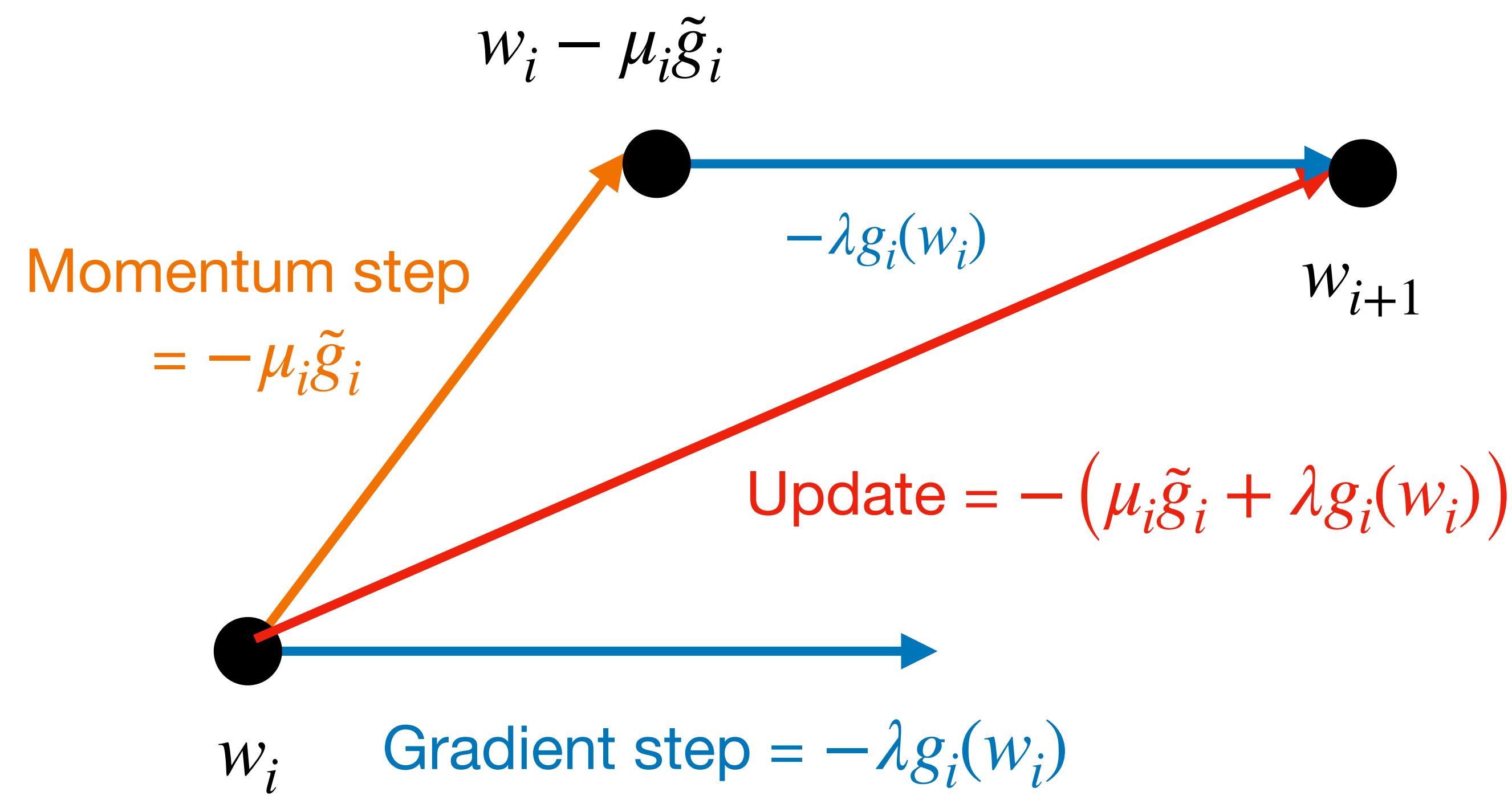
$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

- Momentum에서 $g_i(w_i)$ 의 의미 = “현재 지점 w_i 의 gradient $g_i(w_i)$ ”
- NAG에서 $g(w_i - \mu_i \tilde{g}_i)$ 의 의미는 = “weight parameter가 모멘텀에 의해 이동한 지점 $w_i - \mu_i \tilde{g}_i$ 에서의 gradient”

$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

모멘텀:

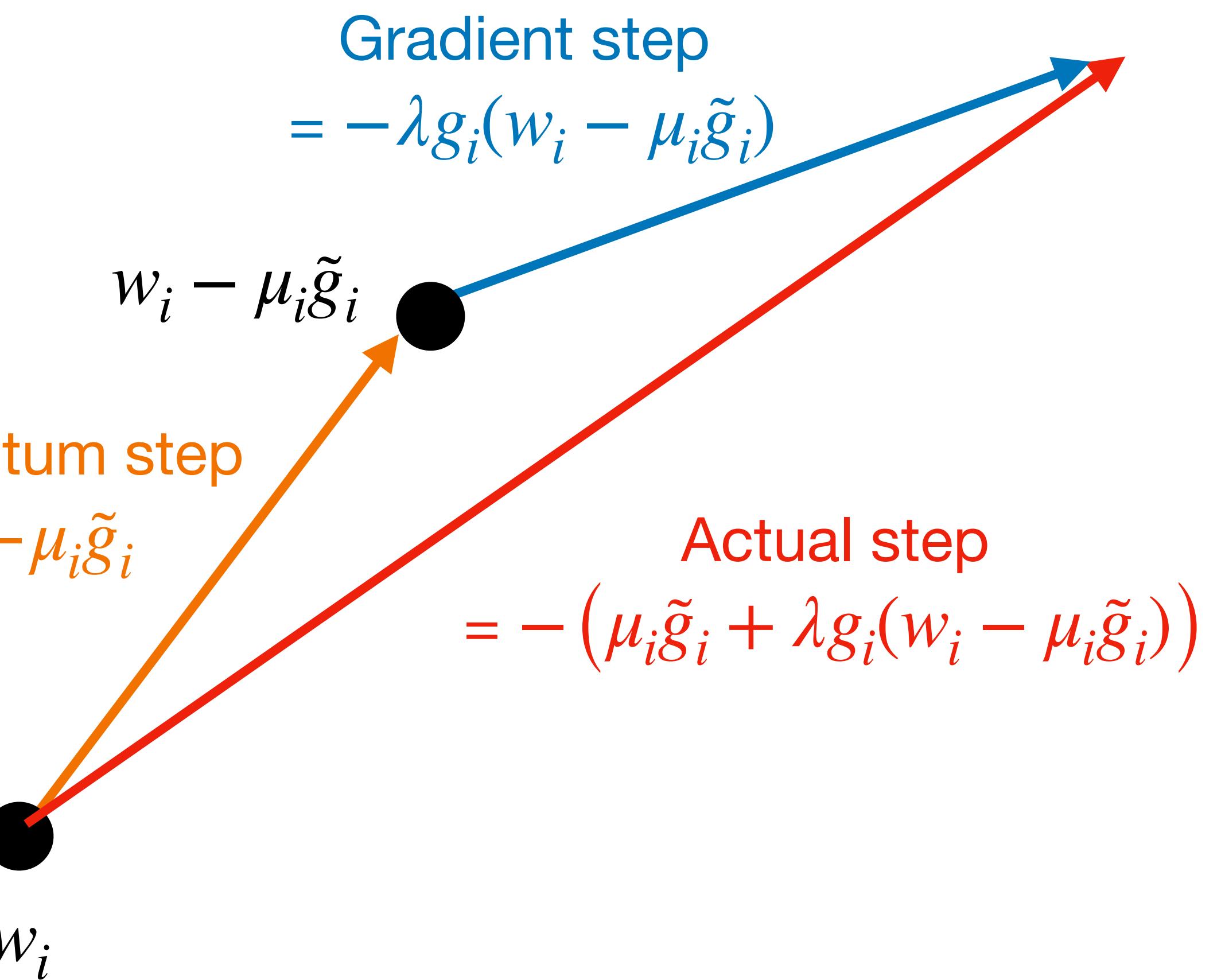
$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$



$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

NAG:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g(w_i - \mu_i \tilde{g}_i)$$



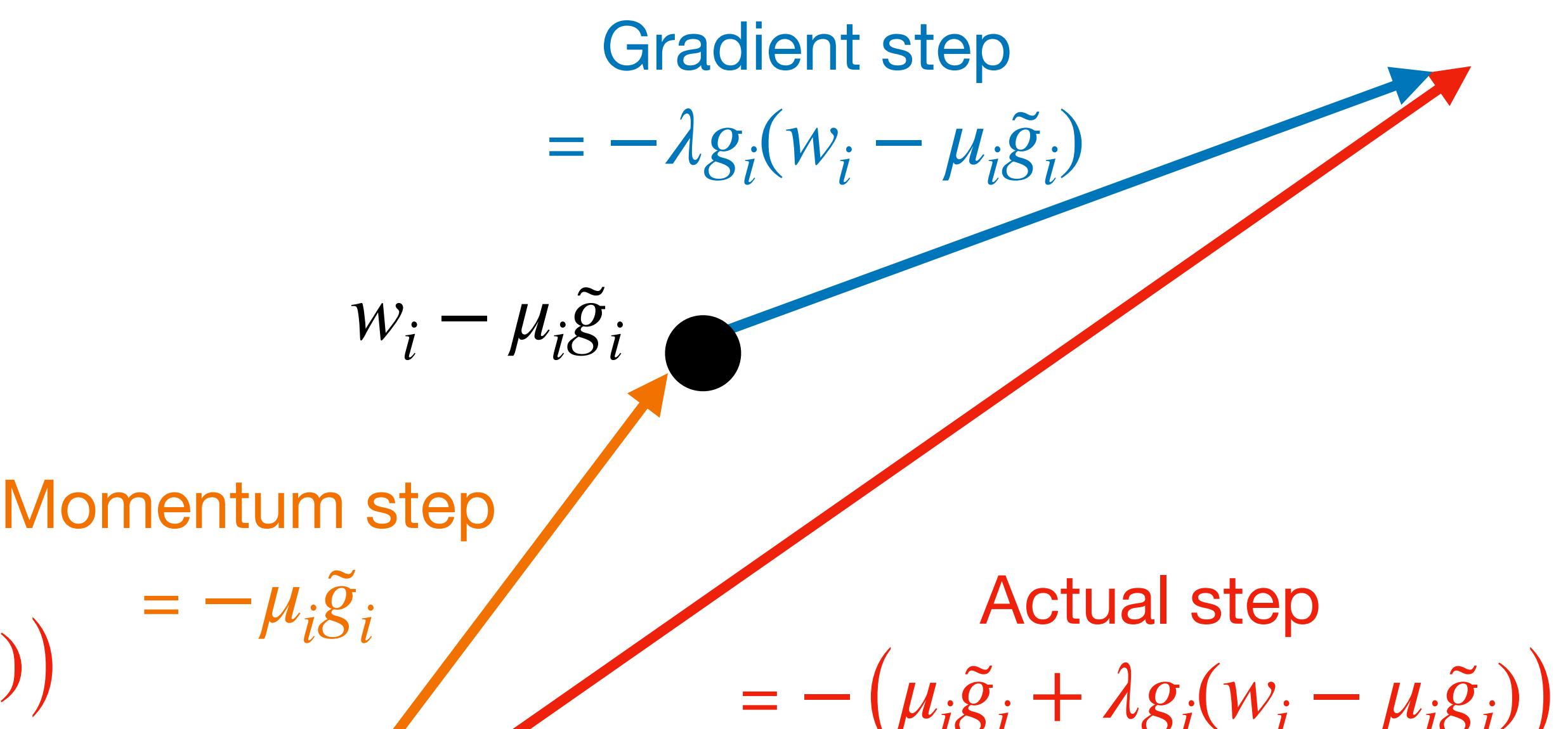
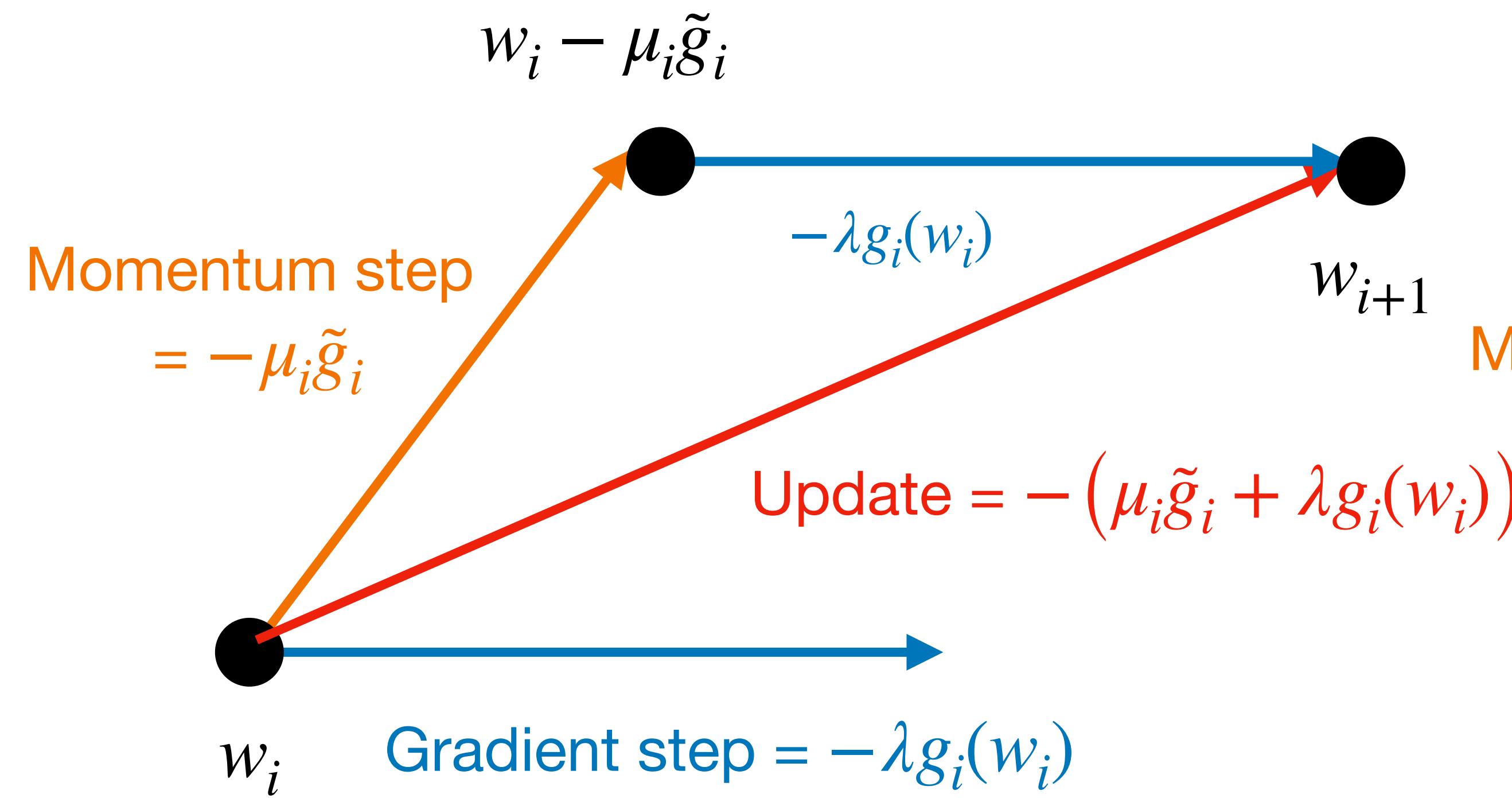
$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

모멘텀:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

NAG:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g(w_i - \mu_i \tilde{g}_i)$$



Optimization

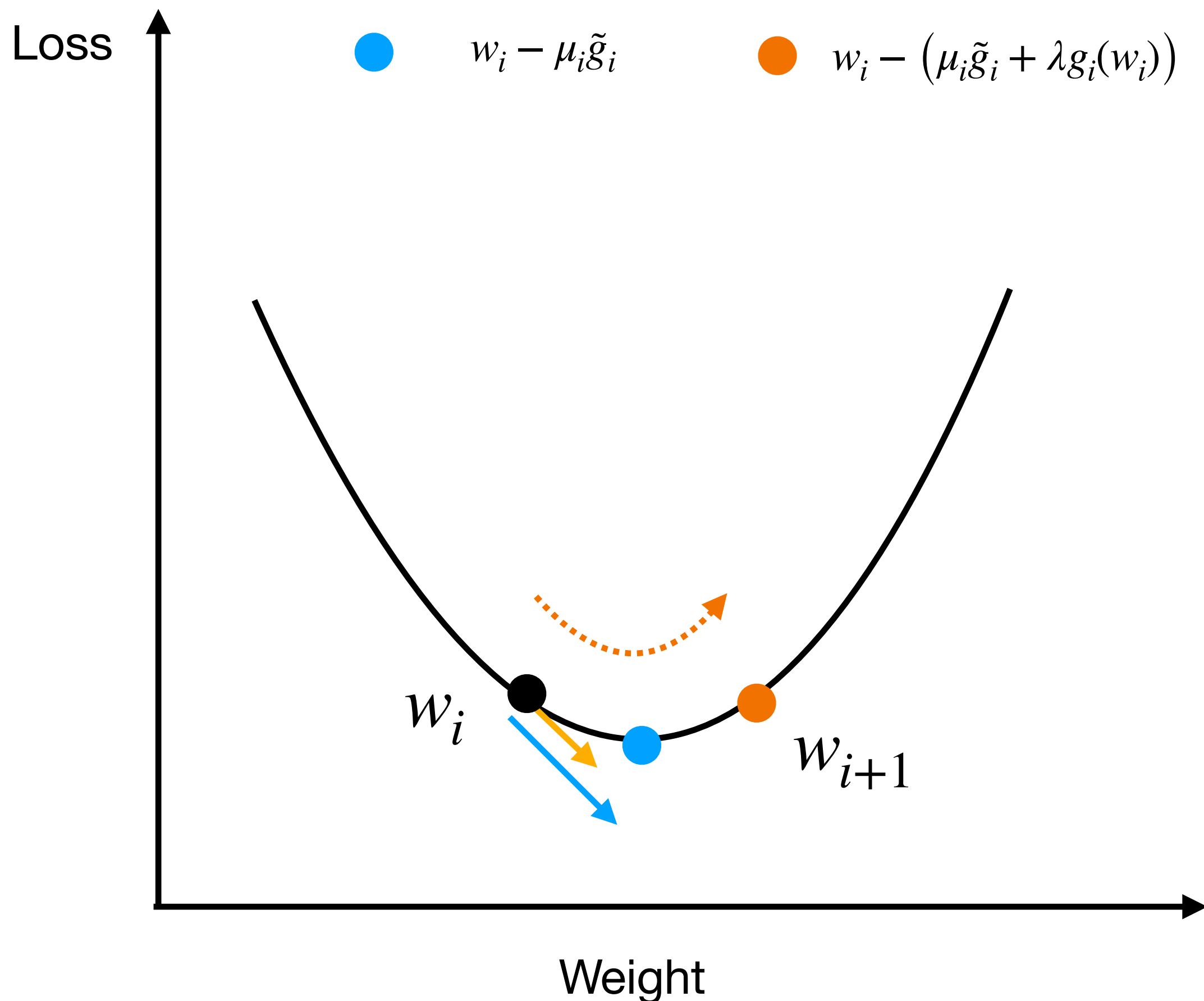
Nesterov's Accelerated Gradient (NAG)

Copyright©2023. Acadential. All rights reserved.

- Momentum Step = $-\mu_i \tilde{g}_i$
- Gradient at $w_i = -\lambda g_i$

Momentum

- Weight가 Minimum에 어느 정도 도달한 경우
- Momentum step이 충분히 작지않아 Overshoot해버릴 수 있다.
- Minimum 주변을 oscillate해버린다.

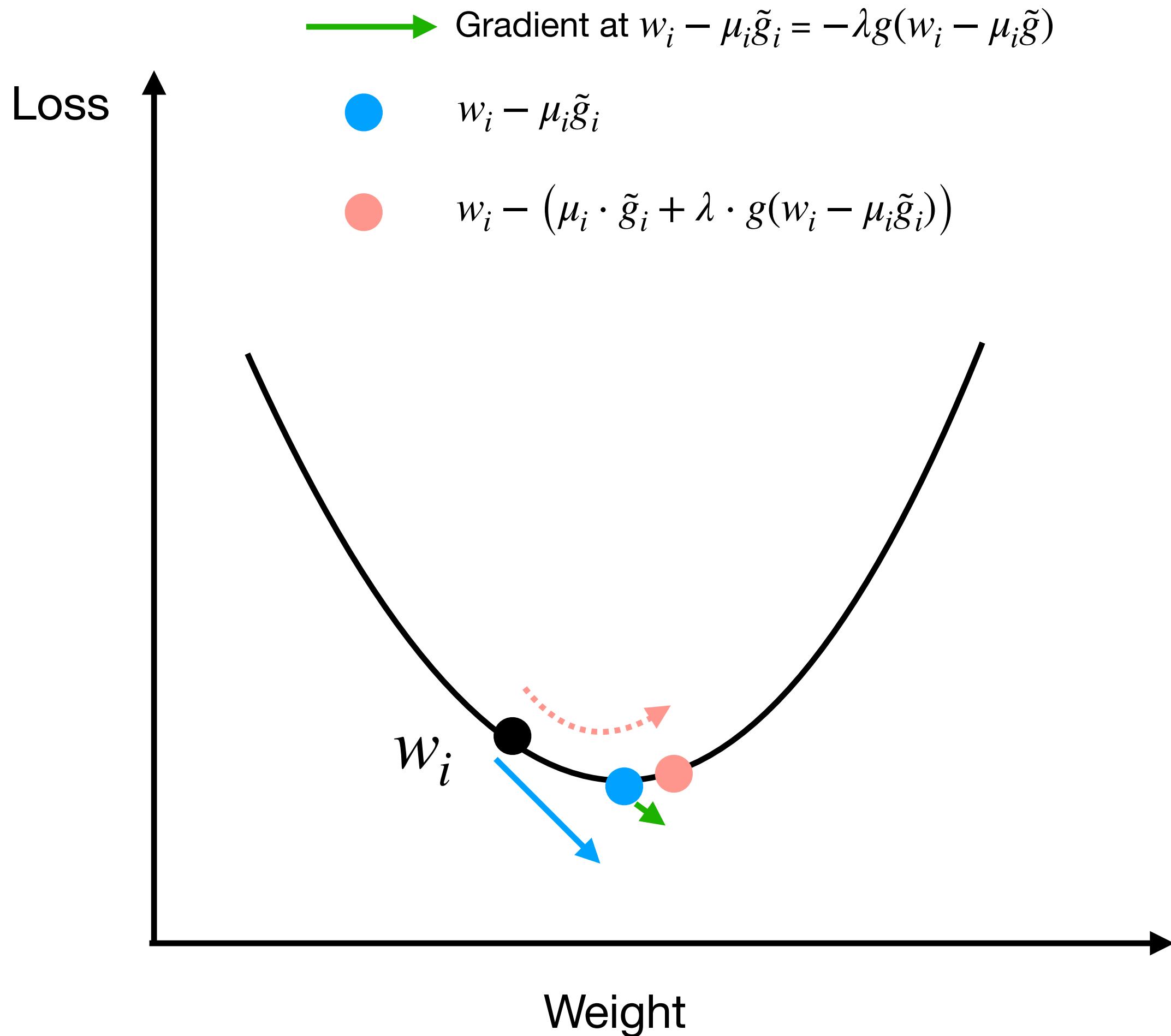


Optimization

Nesterov's Accelerated Gradient (NAG)

Nesterov's Accelerated Gradient (NAG)

- Momentum step으로 이동한 지점에서의 Gradient 사용.
- (Minimum 근처에서) NAG가 계산한 Gradient 은 더 작은 값을 가짐.
→ Momentum에서 발생하는 overshoot 문제를 완화할 수 있음.



Optimization

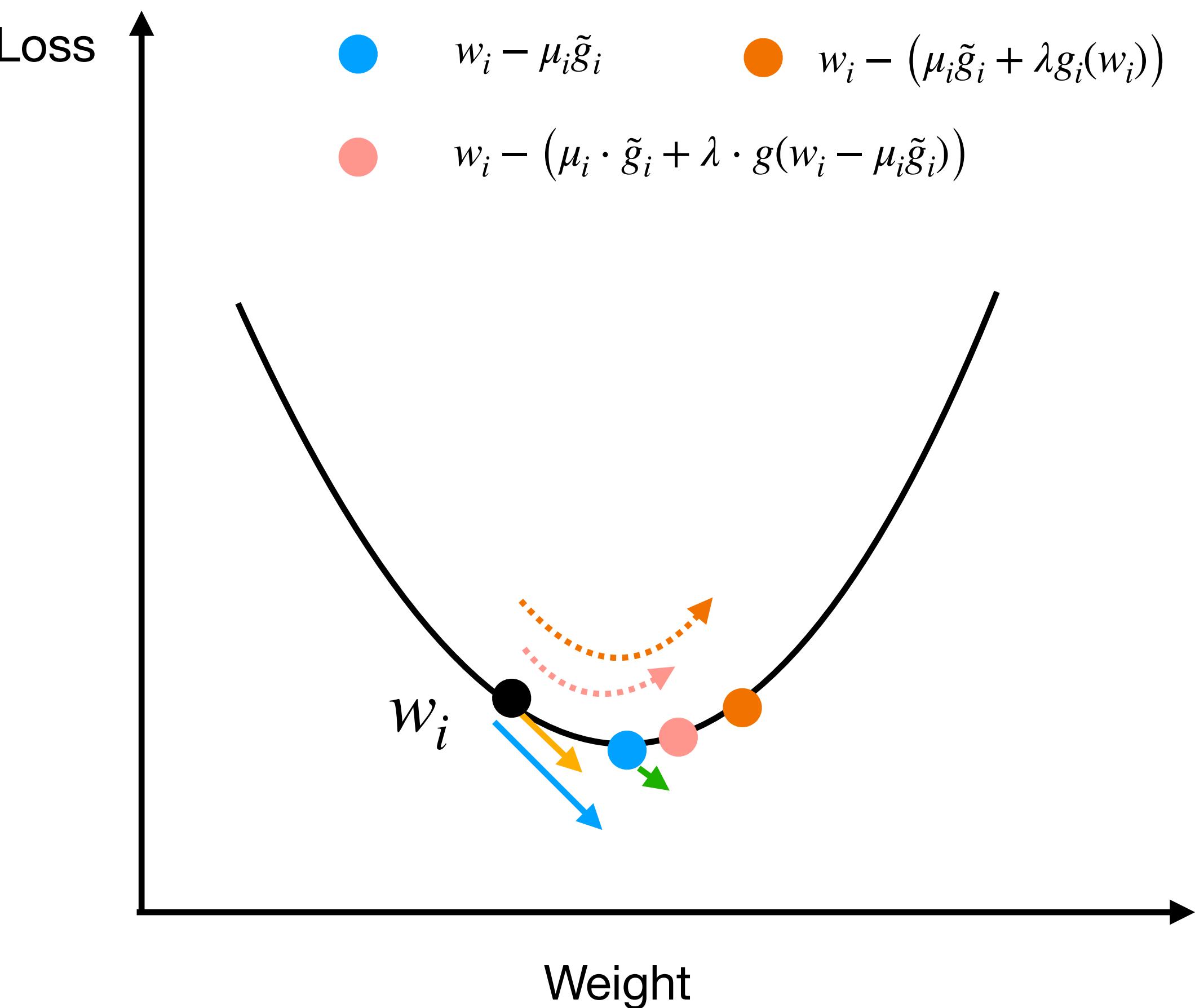
Nesterov's Accelerated Gradient (NAG)

Copyright©2023. Acadential. All rights reserved.

- Momentum Step = $-\mu_i \tilde{g}_i$
- Gradient at $w_i = -\lambda g_i$
- Gradient at $w_i - \mu_i \tilde{g}_i = -\lambda g(w_i - \mu_i \tilde{g})$

Momentum vs. NAG

- NAG은 Momentum의 Overshoot, oscillation 문제를 완화한다.



8-3. AdaGrad

Optimization

AdaGrad

- Weight parameter별로 gradient의 learning rate을 (Adaptive하게) 다르게 주는 방식.

Optimization

AdaGrad

Copyright©2023. Acadential. All rights reserved.

- Weight parameter별로 gradient의 learning rate을 (Adaptive하게) 다르게 주는 방식.
- Parameter들의 update가 균일하도록 learning rate을 다르게 배정.

Optimization

AdaGrad

Copyright©2023. Acadential. All rights reserved.

- Weight parameter별로 gradient의 learning rate을 (Adaptive하게) 다르게 주는 방식.
- Parameter들의 update가 균일하도록 learning rate을 다르게 배정.
- 많이 update되는 parameter일수록 learning rate을 작게 설정하고, 조금씩만 update되는 parameter일수록 learning rate을 크게 설정.

Optimization

AdaGrad

AdaGrad:

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Optimization

AdaGrad

AdaGrad:

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$w_{t,i}$: t번째 GD step 후 weight parameter인 w_t 의 i 번째 component.

Optimization

AdaGrad

AdaGrad:

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$w_{t,i}$: t번째 GD step 후 weight parameter인 w_t 의 i 번째 component.

$G_{t,ii} = \sum_{t'=0}^t g_{t',i}^2$: G_t 의 diagonal (row=i, column=i)에 해당되는 요소.

Optimization

AdaGrad

AdaGrad:

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

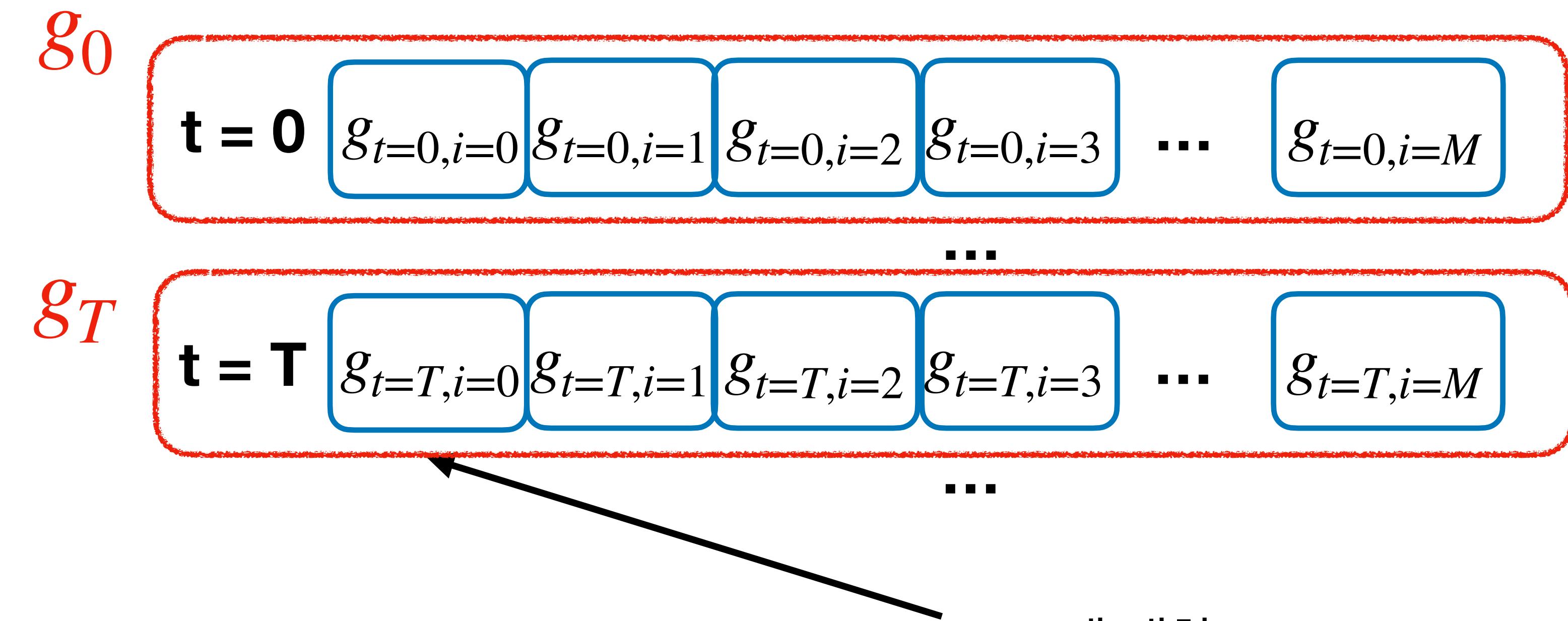
$w_{t,i}$: t번째 GD step 후 weight parameter인 w_t 의 i 번째 component.

$$G_{t,ii} = \sum_{t'=0}^t g_{t',i}^2 : G_t의 diagonal (row=i, column=i)에 해당되는 요소.$$

$$G_t = \sum_{t'=0}^t g_{t'} g_{t'}^T \text{ (diagonal 요소들은 i번째 gradient component의 제곱의 합이다)}$$

Optimization

AdaGrad



AdaGrad:

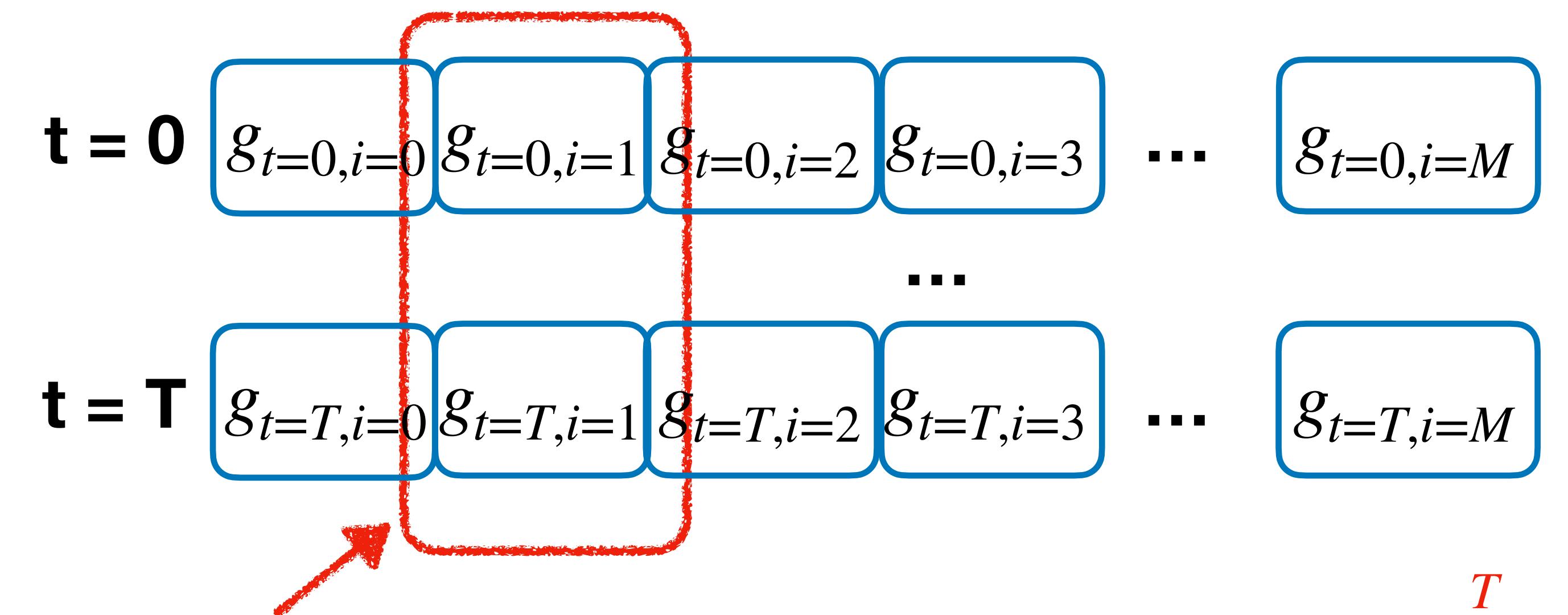
$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$G_{t,ii} = \sum_{t'=0}^t g_{t',i}^2$: G_t 의 diagonal (row=i, column=i)에 해당되는 요소.

Optimization

AdaGrad

AdaGrad:



$t=T$ 까지의 gradient들의 제곱의 총합이 바로 $G_{T,11} = \sum_{t=0}^T g_{t,1}^2$

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

- $G_{t,ii} = \sum_{t'=0}^t g_{t',i}^2 : G_t$ 의 diagonal (row=i, column=i)에 해당되는 요소.

Optimization

AdaGrad

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

- 즉 $G_{t,ii}$ 가 크다는 의미는 w_i 에 대한 gradient의 magnitude가 평균적으로 크다는 의미이고,
- 따라서 이것에 반비례하게 learning rate을 조정함.

Optimization

AdaGrad

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

- AdaGrad의 문제점 1:
 - $G_{t,ii}$ 은 t 가 커지면서 무한정으로 계속 커지고 $\frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}}$ 은 0에 수렴한다.
 - $\lim_{t \rightarrow \infty} \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} = 0$
 - 따라서 시간이 지나면서 Model이 더 이상 update되지 않는 상태에 빠지게 된다.

Optimization

AdaGrad

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

- AdaGrad의 문제점 2:

- $\frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}}$ 의 scale이 [constant]/[gradient]의 unit을 가진다.
- 원래 learning rate은 [constant]의 unit을 가져야한다.

8-4. AdaDelta & RMSProp

Optimization

AdaDelta

- **AdaDelta:** AdaGrad의 단점들
 - “AdaGrad의 adaptive한 learning rate가 0에 수렴되는 문제점”
 - “Learning rate의 unit이 [상수] / [gradient]인 문제점”
- 을 보완하기 위해서 제안된 방법.

Optimization

AdaDelta

Copyright©2023. Acadential. All rights reserved.

- **AdaDelta**: AdaGrad의 단점들 “AdaGrad의 adaptive한 learning rate가 0에 수렴되는 문제점”, “learning rate의 unit이 [상수] / [gradient]인 문제점”을 보완하기 위해서 제안된 방법.

AdaDelta에서는 gradient의 제곱의 **“moving average”**을 사용한다.

또한 learning rate의 unit을 [constant]로 맞춰준다.

Optimization

AdaDelta

- **AdaGrad:**

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

- **AdaDelta**

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

What is Moving Average?

- **Moving average (aka running average)**

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

어떤 일련의 값 $(\theta_0, \theta_1, \theta_2, \dots, \theta_N)$ 들이 있을때, 해당 값들의 이동 평균.

What is Moving Average?

- **Moving average (aka running average)**

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

어떤 일련의 값 $(\theta_0, \theta_1, \theta_2, \dots, \theta_N)$ 들이 있을때, 해당 값들의 이동 평균.

위 식을 일련의 값들로 풀어서 표현하면:

$$\text{MA}[\theta]_t = (1 - \gamma)(\theta_t + \gamma\theta_{t-1} + \dots + \gamma^{t-t'}\theta_{t'} + \dots + \gamma^t\theta_0)$$

γ 은 0~1사이의 값

What is Moving Average?

Copyright©2023. Acadential. All rights reserved.

- **Moving average (aka running average)**

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

어떤 일련의 값 $(\theta_0, \theta_1, \theta_2, \dots, \theta_N)$ 들이 있을때, 해당 값들의 이동 평균.

위 식을 일련의 값들로 풀어서 표현하면:

$$\text{MA}[\theta]_t = (1 - \gamma)(\theta_t + \gamma\theta_{t-1} + \dots + \gamma^{t-t'}\theta_{t'} + \dots + \gamma^t\theta_0)$$

γ 은 0~1사이의 값

t' 가 과거의 값일수록, $\gamma^{t-t'}$ 은 더욱 작으며, $\theta_{t'}$ 이 $\text{MA}[\theta]_t$ 에 기여하는 바는 더 작아진다.

Optimization

AdaDelta

- AdaDelta

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

γ = moving average constant (클수록 이전 t의 값을 더 많이 반영함)

참고로 $0 < \gamma < 1$. 따라서 $\boxed{\quad}$ 이 무한히 커지지 않는다. 즉, “effective learning rate”가 0이 되지 않아 AdaGrad의 첫번째 문제점 해결!

Optimization

AdaDelta

- AdaDelta

“effective learning rate”

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

참고로 “effective learning rate”이란 g_t 앞에 붙는 scalar value로 “실질적인” learning rate의 의미를 가진다.

Optimization

AdaDelta

- AdaDelta

$=\Delta w_t$

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

Optimization

AdaDelta

즉, Momentum처럼 과거의 gradient 크기가 클수록 현재의 gradient step도 크게 만든다. (나중에 확인하겠지만 AdaGrad, RMSProp보다 더 빠르다)

Δw_{t-1}^2 에 대한 Moving Average이다

- AdaDelta

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t = \Delta w_t$$

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

■ 값은 이전 GD step의 업데이트 값이다! 즉 이전 step의 ■ 2 의 Moving Average이다.

Optimization

AdaDelta

- AdaDelta

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

unit = $[w]$

unit = $[wt^{-1}]$

Optimization

AdaDelta

- AdaDelta

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

unit = $[w] / [wt^{-1}] = [t]$

unit = $[wt^{-1}]$

즉, unit이 weight의 unit과 동일하다!!!

(AdaGrad의 문제점 2 해결)

그리고 매뉴얼하게 직접 learning rate을 지정

하지 않아도 된다!

unit = $[t] \cdot [wt^{-1}] = [w]$

RMSProp

Optimization

RMSProp

Copyright©2023. Acadential. All rights reserved.

- Root Mean Squared Propagation (RMSProp)
- 학습이 너무 오래 진행되면 step size가 너무 작아지는 AdaGrad을 보완하기 위해서 제안된 또 다른 방법.
- RMSProp:

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

AdaDelta와 유사한데 AdaDelta와 달리 분자가 λ 인 점에서 차이가 있다. (즉, AdaGrad의 두번째 문제점 해소 X)

8-5. ADAM

Optimization

ADAM

- **Adaptive Moment Estimation (ADAM):**
 - $E[g^2]$ (2nd moment)와 $E[g]$ (first moment)을 Moving Average로 estimate하여 이를 adaptive한 learning rate 계산에 사용함!

Optimization

ADAM

- **Adaptive Moment Estimation (ADAM):**

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Optimization

ADAM

- **Adaptive Moment Estimation (ADAM):**

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

Estimation of 1st moment of gradient i.e. $E[g]$

\hat{v}_t

λ

ϵ

\hat{m}_t

Optimization

ADAM

- **Adaptive Moment Estimation (ADAM):**

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

β_1 은 상수 (default로 0.9)

즉, \hat{m}_t 은 t가 늘어날수록 m_t 에 수렴함

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

β_2 은 상수 (default로 0.999)

즉, \hat{v}_t 은 t가 늘어날수록 v_t 에 수렴함

Optimization

ADAM

- **Adaptive Moment Estimation (ADAM):**

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

즉, m_t 은 g_t 에 대한 Moving Average이다!

Optimization

ADAM

- **Adaptive Moment Estimation (ADAM):**

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

즉, v_t 은 g_t^2 에 대한 Moving Average이다!

Optimization

ADAM

- **Adaptive Moment Estimation (ADAM):**

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Momentum에서 차용됨!

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

RMSProp, AdaDelta
에서 차용됨!

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

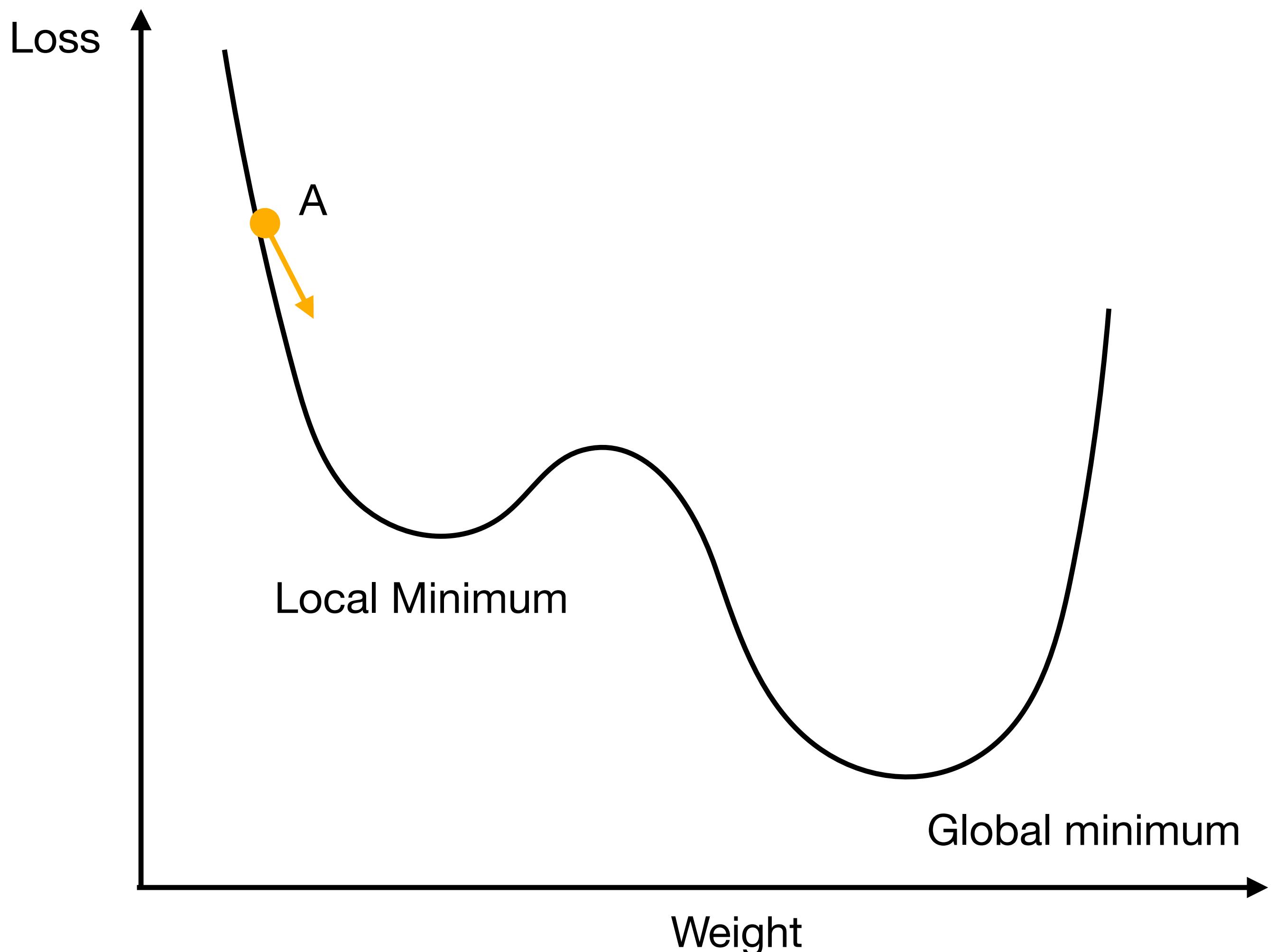
8-6. PyTorch로 구현해보는 Optimization

8-7. Section 8 요약

Section Summary

Momentum

- A에서 initialize했을시, Gradient Descent은 Local minimum에 빠지게 됨.
- 어떻게 하면 Global Minimum에 도달할 수 있을까?
- **Momentum (관성)을 활용!**
- 관성 = 원래 이동하던 방향으로 계속 이동하려는 경향성!



Section Summary

Momentum

- Momentum을 사용할시 weight에 대한 update은 다음과 같다:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

$$w_{i+1} = w_i - \tilde{g}_{i+1}$$

- λ = learning rate
- μ_i = Momentum parameter (0~1 값)
- \tilde{g}_i = Moving average (이동평균)

Section Summary

Momentum

효과:

- 학습의 수렴 속도를 더 높여줄 수 있다.
- 학습의 진동폭을 줄여준다.
- Gradient가 Noisy한 경우 안정적 학습에 기여한다.
- Saddle Point를 피하는데 효과적이다.

Section Summary

Nesterov's Accelerated Gradient (NAG)

- Nesterov's Accelerated Gradient (NAG):

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g(w_i - \mu_i \tilde{g}_i)$$

- Recap on GD with Momentum:

$$\tilde{g}_{i+1} = \mu_i \cdot \tilde{g}_i + \lambda \cdot g_i$$

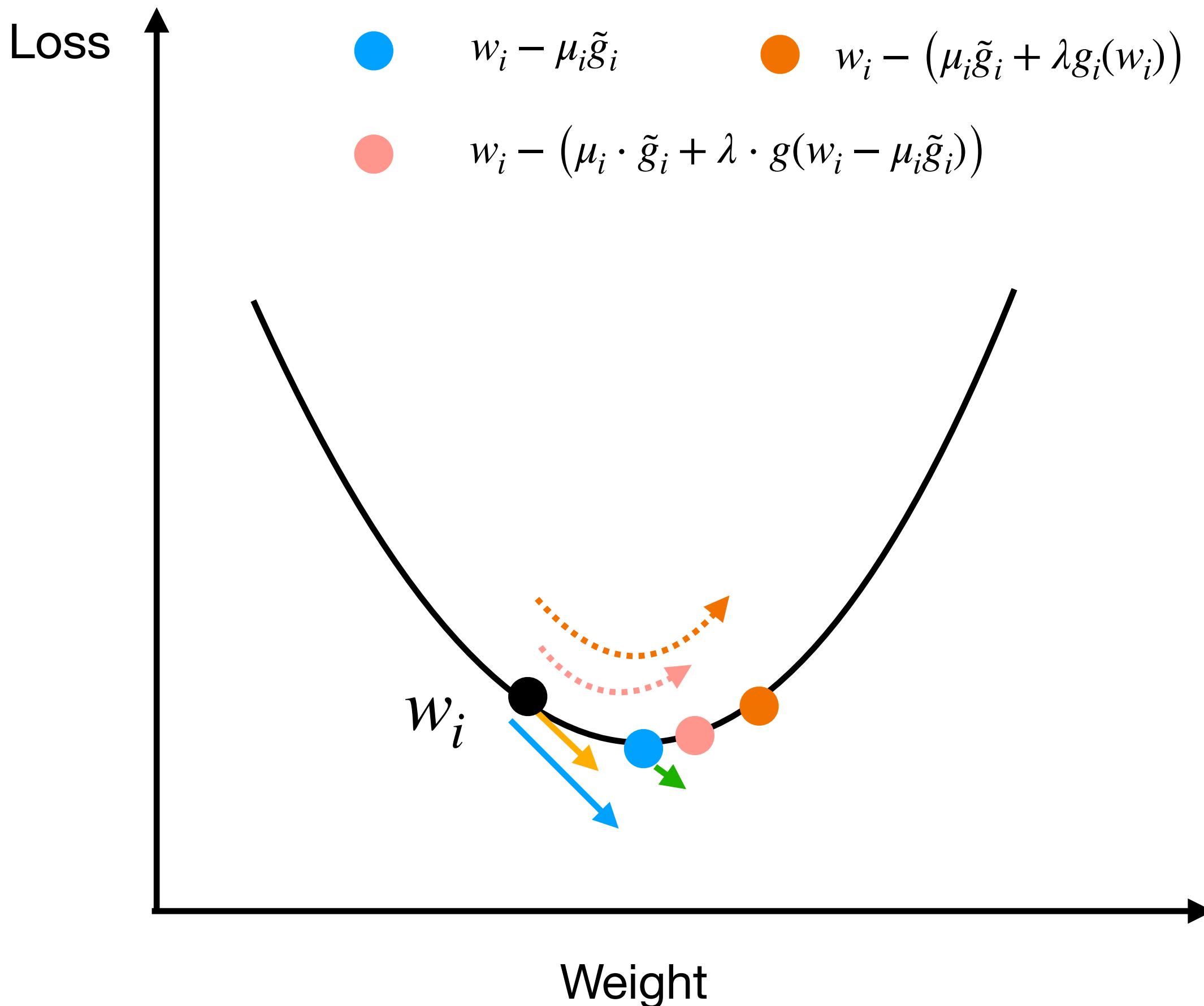
- Momentum에서 $g_i(w_i)$ 의 의미 = “현재 지점 w_i 의 gradient $g_i(w_i)$ ”
- NAG에서 $g(w_i - \mu_i \tilde{g}_i)$ 의 의미는 = “weight parameter가 모멘텀에 의해 이동한 지점 $w_i - \mu_i \tilde{g}_i$ 에서의 gradient”

Section Summary

Nesterov's Accelerated Gradient (NAG)

Copyright©2023. Acadential. All rights reserved.

- Momentum Step = $-\mu_i \tilde{g}_i$
- Gradient at $w_i = -\lambda g_i$
- Gradient at $w_i - \mu_i \tilde{g}_i = -\lambda g(w_i - \mu_i \tilde{g})$



Momentum vs. NAG

- NAG은 Momentum의 Overshoot, oscillation 문제를 완화한다.

Section Summary

AdaGrad

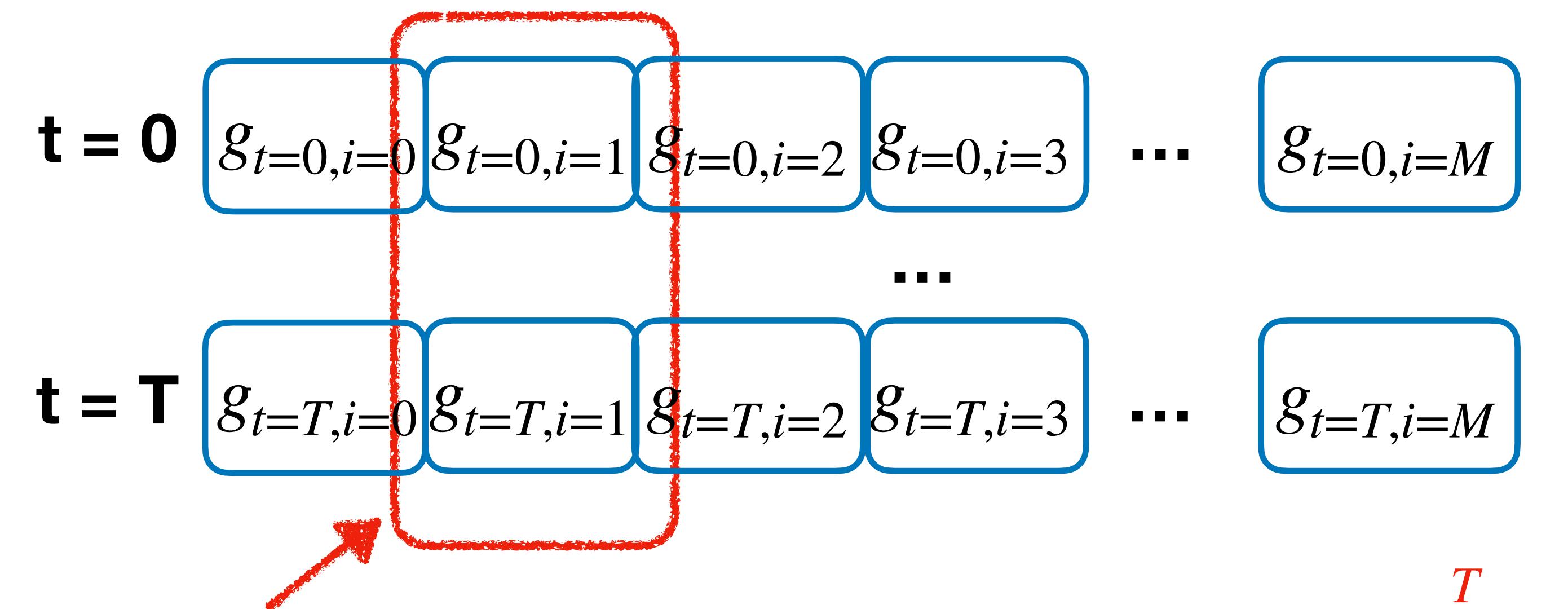
Copyright©2023. Acadential. All rights reserved.

- Weight parameter별로 gradient의 learning rate을 (Adaptive하게) 다르게 주는 방식.
- Parameter들의 update가 균일하도록 learning rate을 다르게 배정.
- 많이 update되는 parameter일수록 learning rate을 작게 설정하고, 조금씩만 update되는 parameter일수록 learning rate을 크게 설정.

Section Summary

AdaGrad

AdaGrad:



t=T까지의 gradient들의 제곱의 총합이 바로 $G_{T,11} = \sum_{t=0}^T g_{t,1}^2$

$$w_{t+1,i} = w_{t,i} - \frac{\lambda}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

- $G_{t,ii} = \sum_{t'=0}^t g_{t',i}^2$: G_t 의 diagonal (row=i, column=i)에 해당되는 요소.

Section Summary

AdaDelta

Copyright©2023. Acadential. All rights reserved.

- **AdaDelta:** AdaGrad의 단점들 “AdaGrad의 adaptive한 learning rate가 0에 수렴되는 문제점”, “learning rate의 unit이 [상수] / [gradient]인 문제점” 을 보완하기 위해서 제안된 방법.

AdaDelta에서는 gradient의 제곱의 **“moving average”**을 사용한다.

또한 learning rate의 unit을 [constant]로 맞춰준다.

Section Summary

AdaDelta

- AdaDelta

$$w_{t+1} = w_t - \frac{\sqrt{\text{MA}[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

$= \Delta w_t$

$$\text{MA}[\theta]_t = \gamma \cdot \text{MA}[\theta]_{t-1} + (1 - \gamma) \cdot \theta_t$$

Section Summary

RMSProp

Copyright©2023. Acadential. All rights reserved.

- Root Mean Squared Propagation (RMSProp)
- 학습이 너무 오래 진행되면 step size가 너무 작아지는 AdaGrad을 보완하기 위해서 제안된 또 다른 방법.
- RMSProp:

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\text{MA}[g^2]_t + \epsilon}} \cdot g_t$$

AdaDelta와 유사한데 AdaDelta와 달리 분자가 λ 인 점에서 차이가 있다. (즉, AdaGrad의 두번째 문제점 해소 X)

Section Summary

ADAM

- **Adaptive Moment Estimation (ADAM):**

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

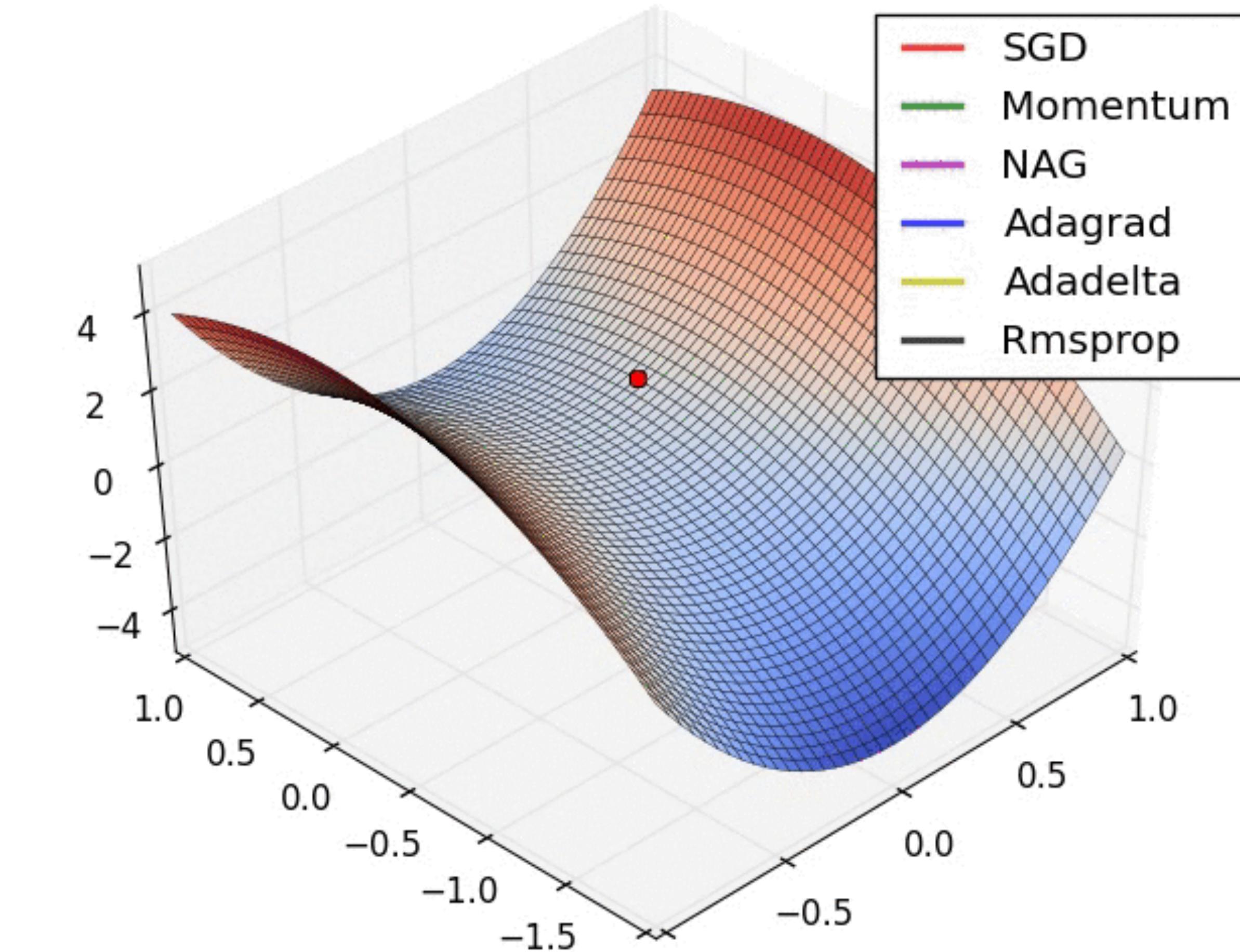
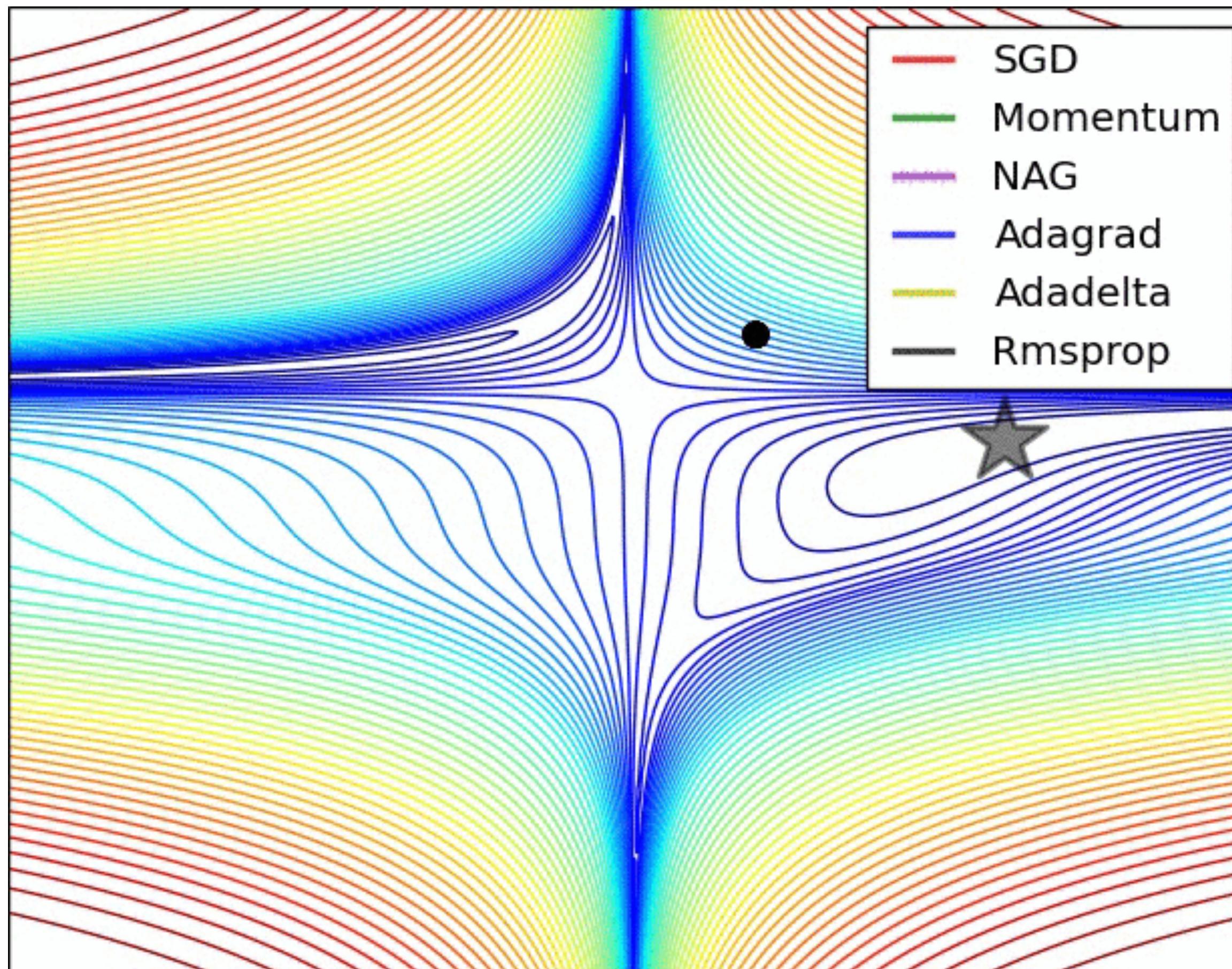
Momentum에서 차용됨!

AdaDelta, RMSProp에서
차용됨!

Section Summary

Training dynamics of different optimizers

Copyright©2023. Acadential. All rights reserved.

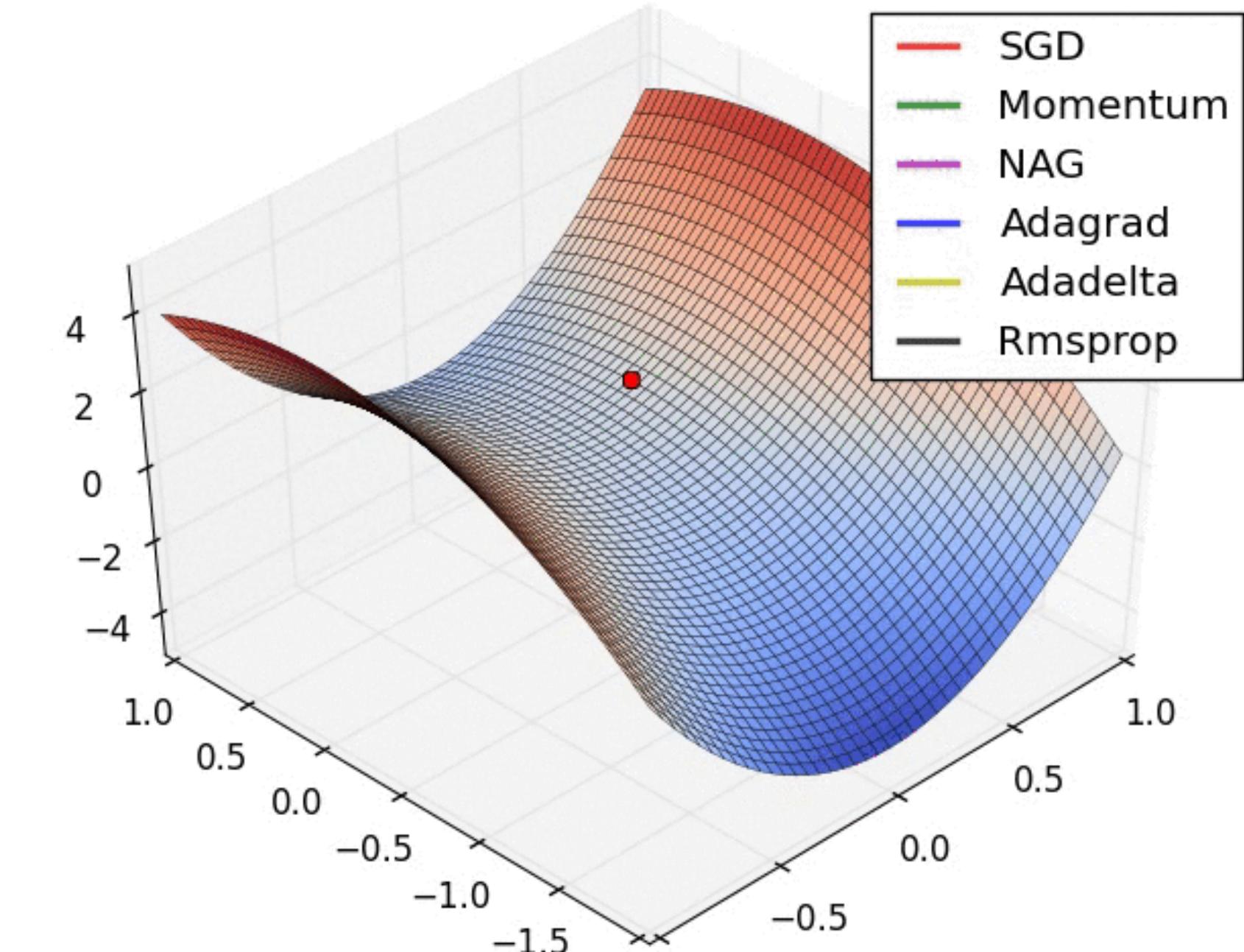
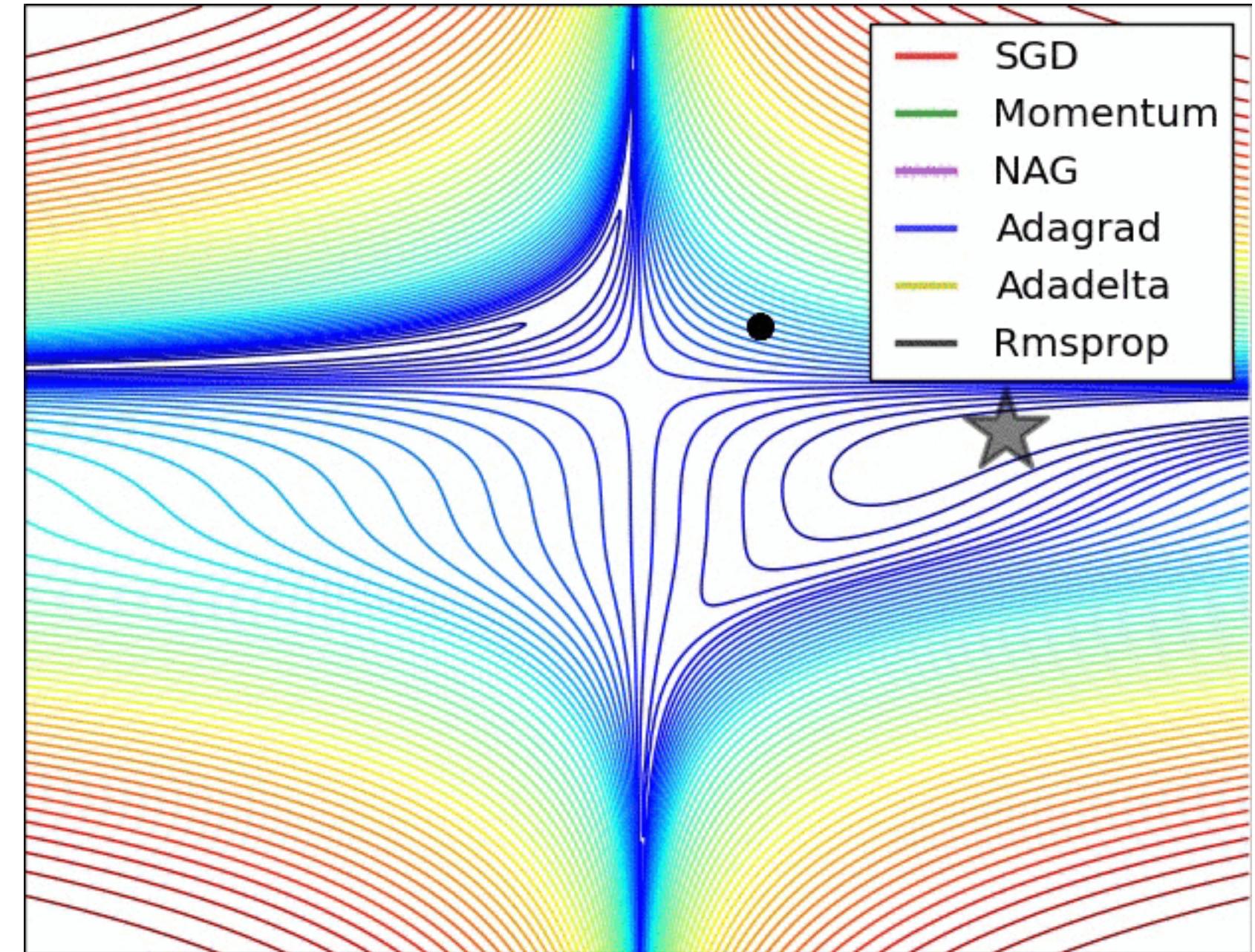


출처: <http://louistiao.me/notes/visualizing-and-animating-optimization-algorithms-with-matplotlib/>

Section Summary

Comments on optimizers

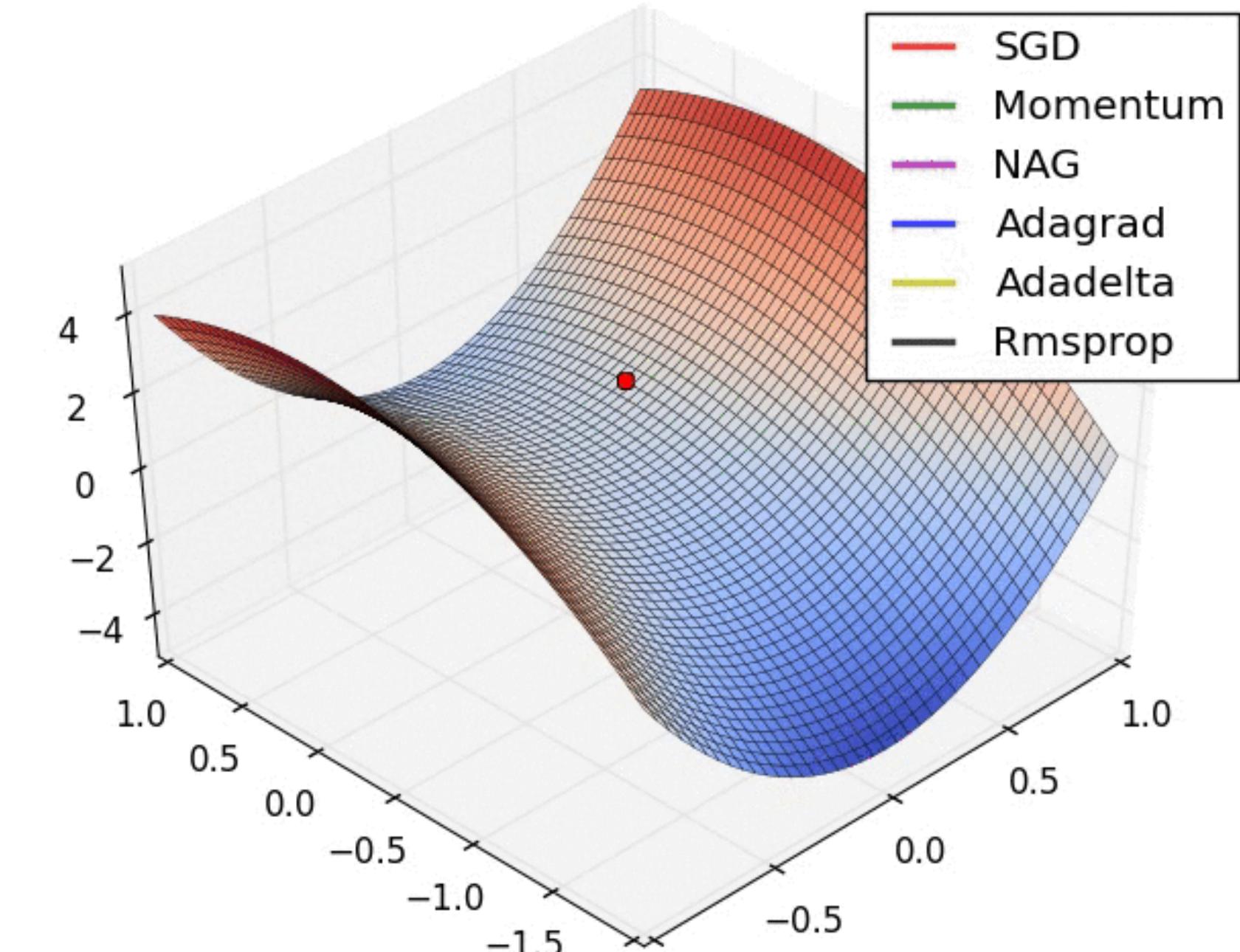
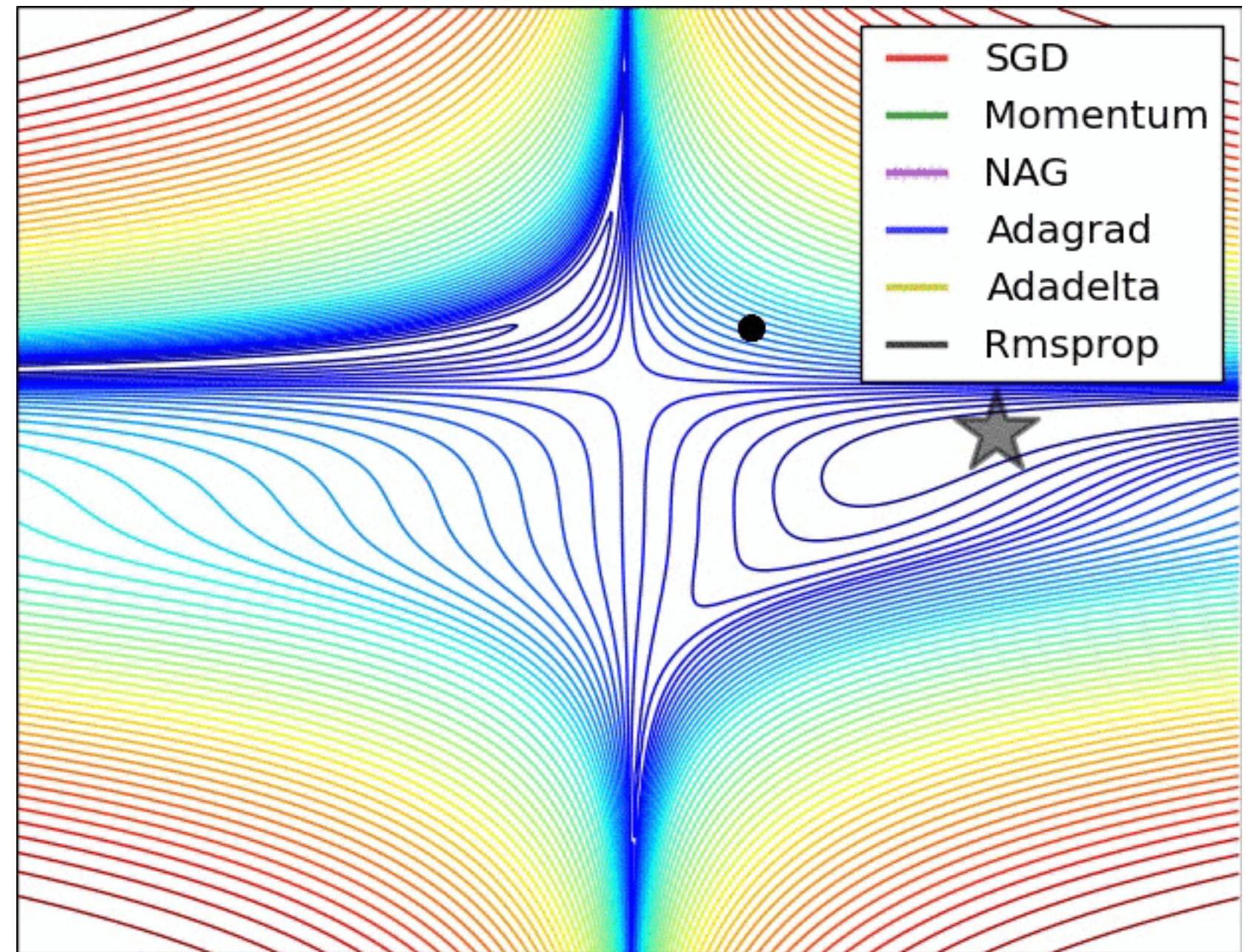
- SGD, Momentum, NAG의 경우 saddle point에서 쉽게 빠져나오지 못한다.
- 반면에 Adaptive한 LR 방법들인 AdaGrad, RMSProp, AdaDelta의 경우 쉽게 saddle point에서 빠져나온다.



Section Summary

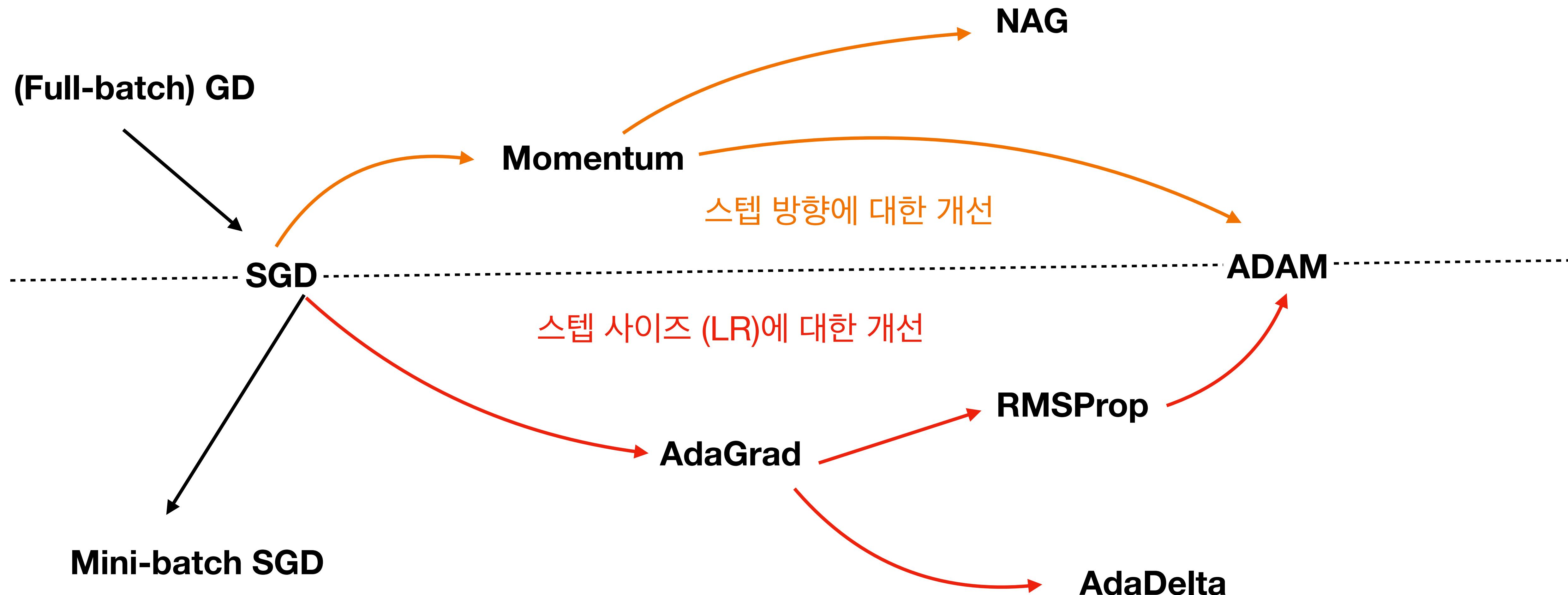
Comments on optimizers

- Saddle point을 벗어난 후에는 Momentum과 NAG가 빠르게 Minima을 향해 이동한다.
- AdaGrad, RMSProp의 경우 Momentum, NAG, AdaDelta에 비해서 느리다.

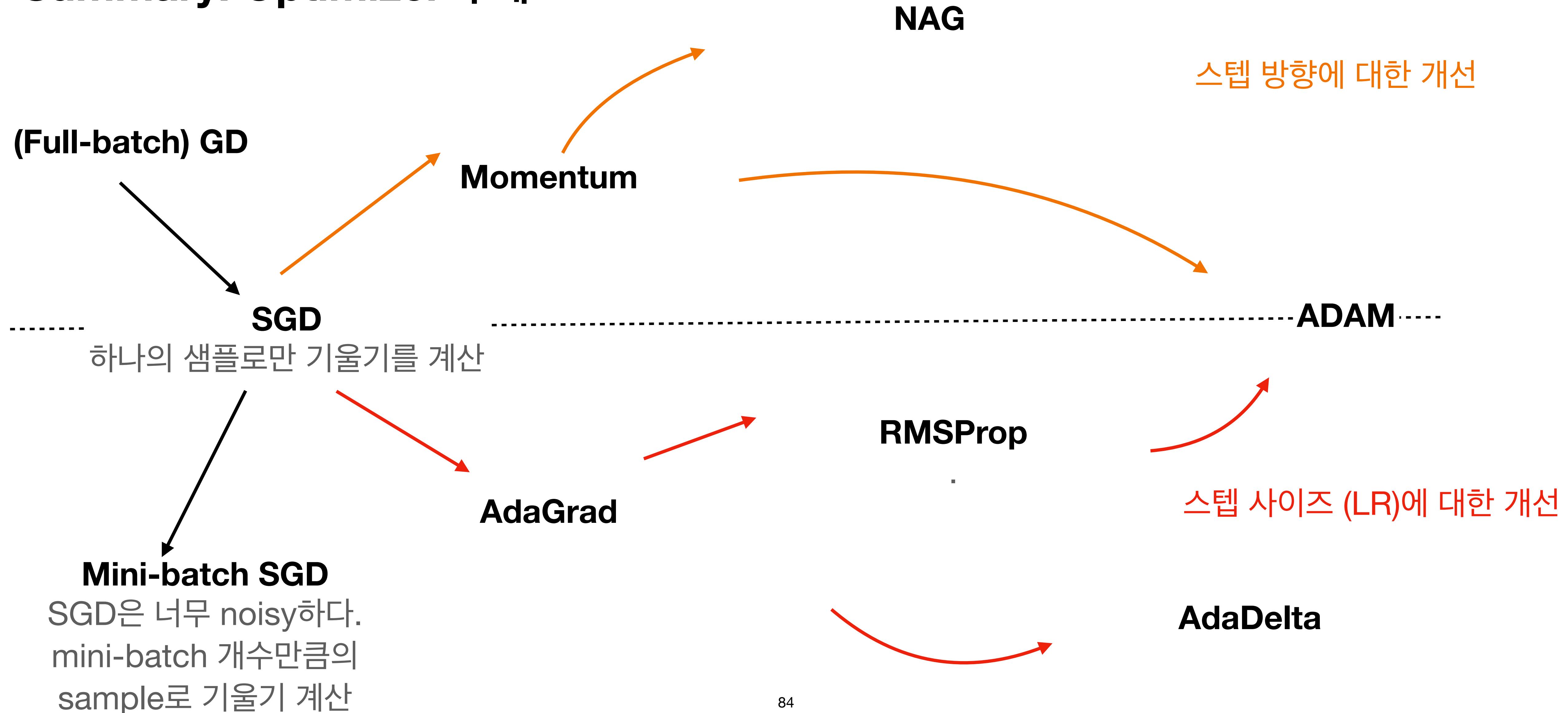


Section Summary

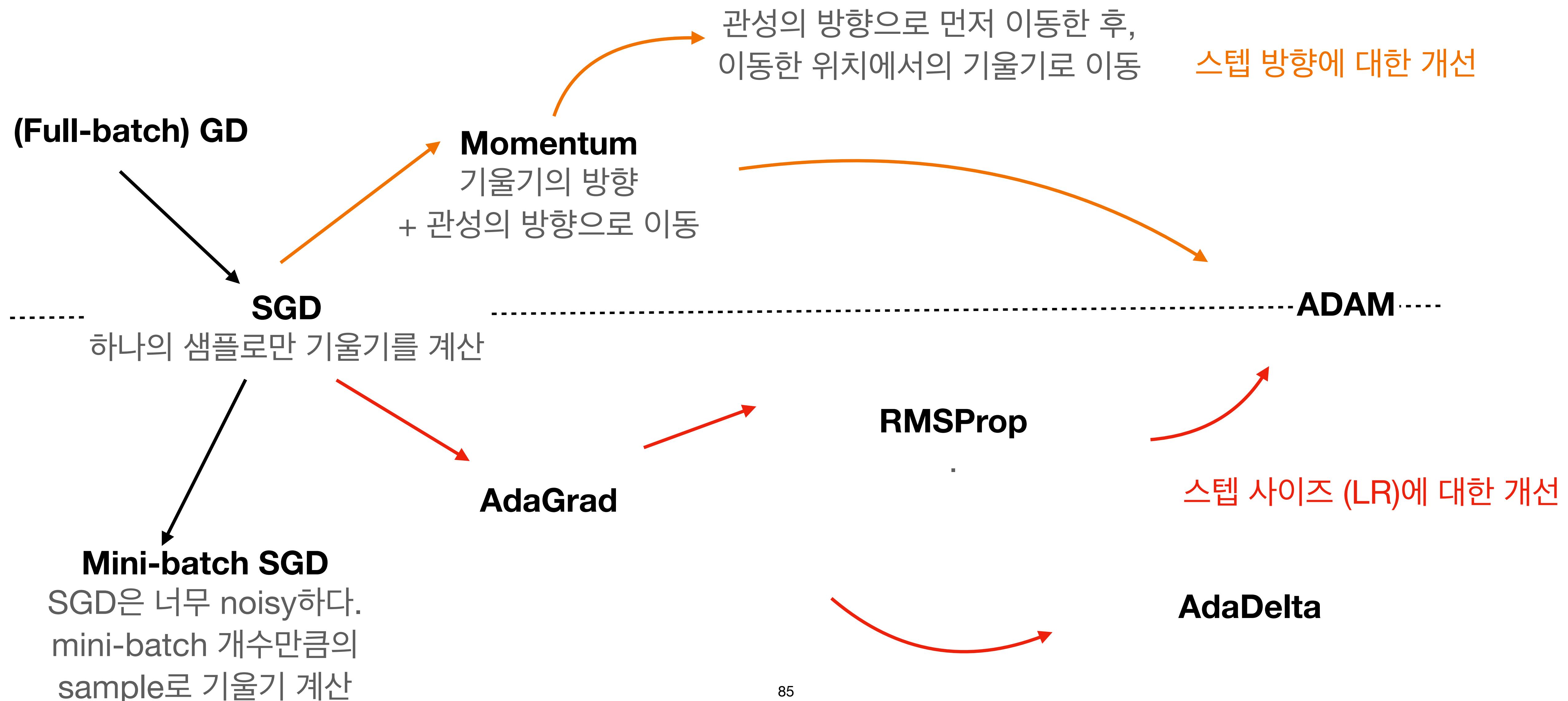
Summary: Optimizer의 계보



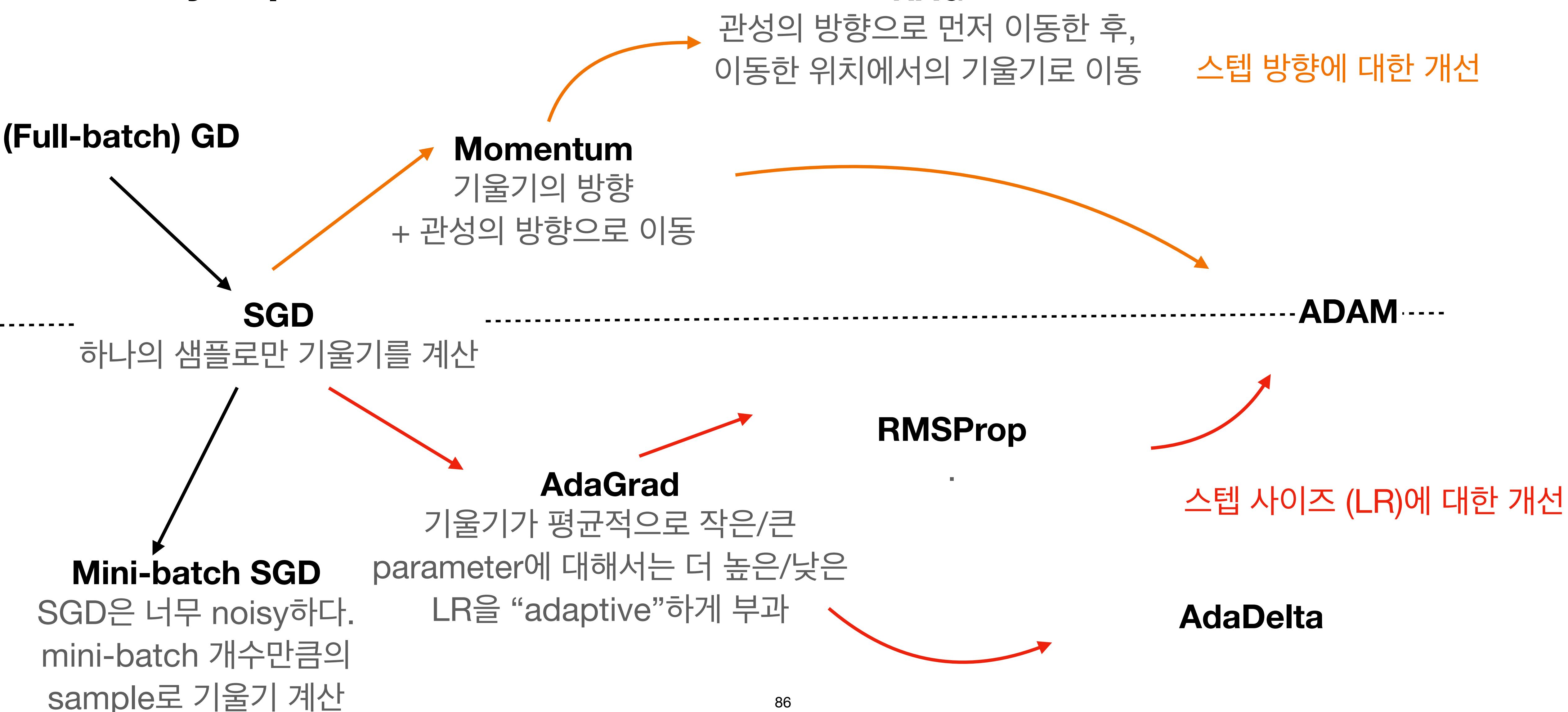
Summary: Optimizer의 계보



Summary: Optimizer의 계보

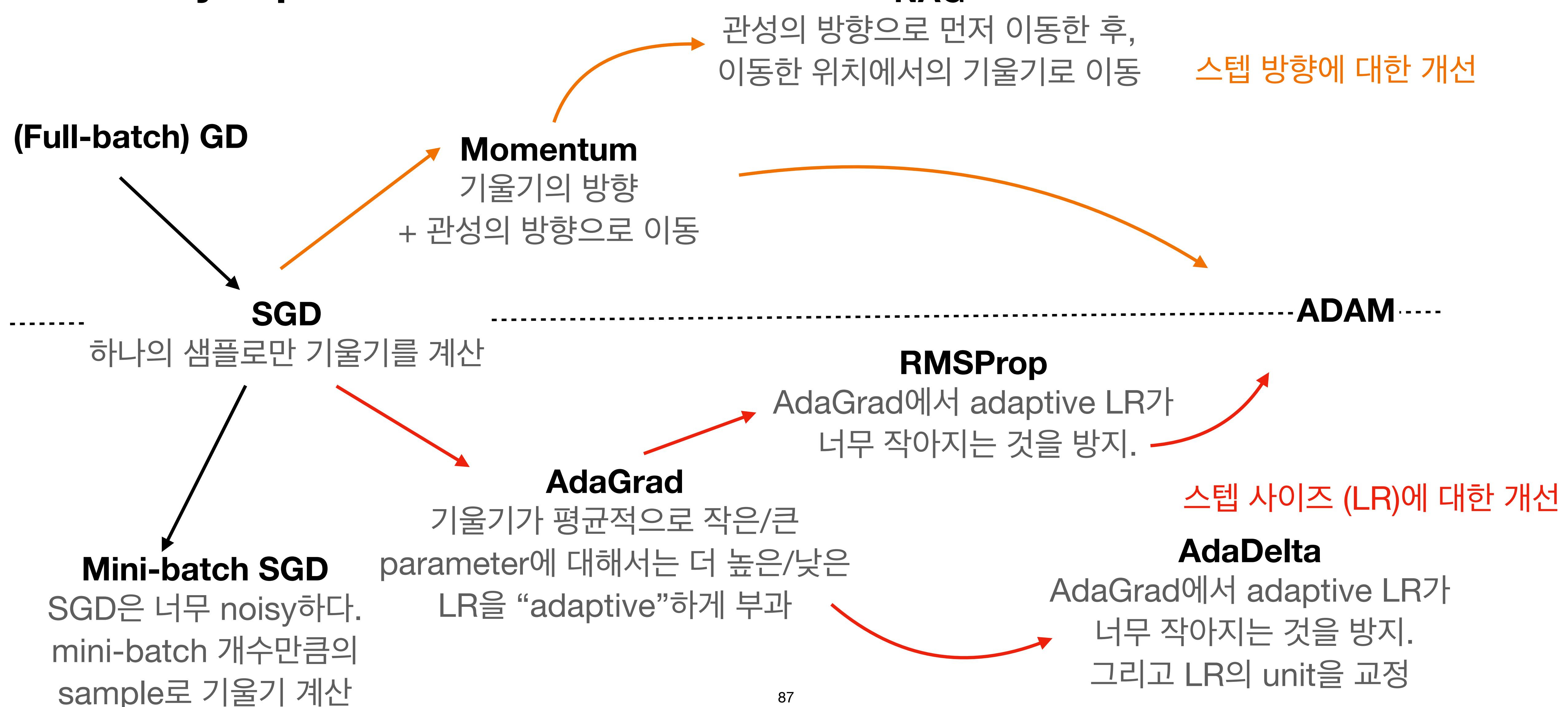


Summary: Optimizer의 계보



SGD은 너무 noisy하다.
mini-batch 개수만큼의
sample로 기울기 계산

Summary: Optimizer의 계보



Mini-batch SGD

SGD은 너무 noisy하다.
mini-batch 개수만큼의
sample로 기울기 계산

Summary: Optimizer의 계보

