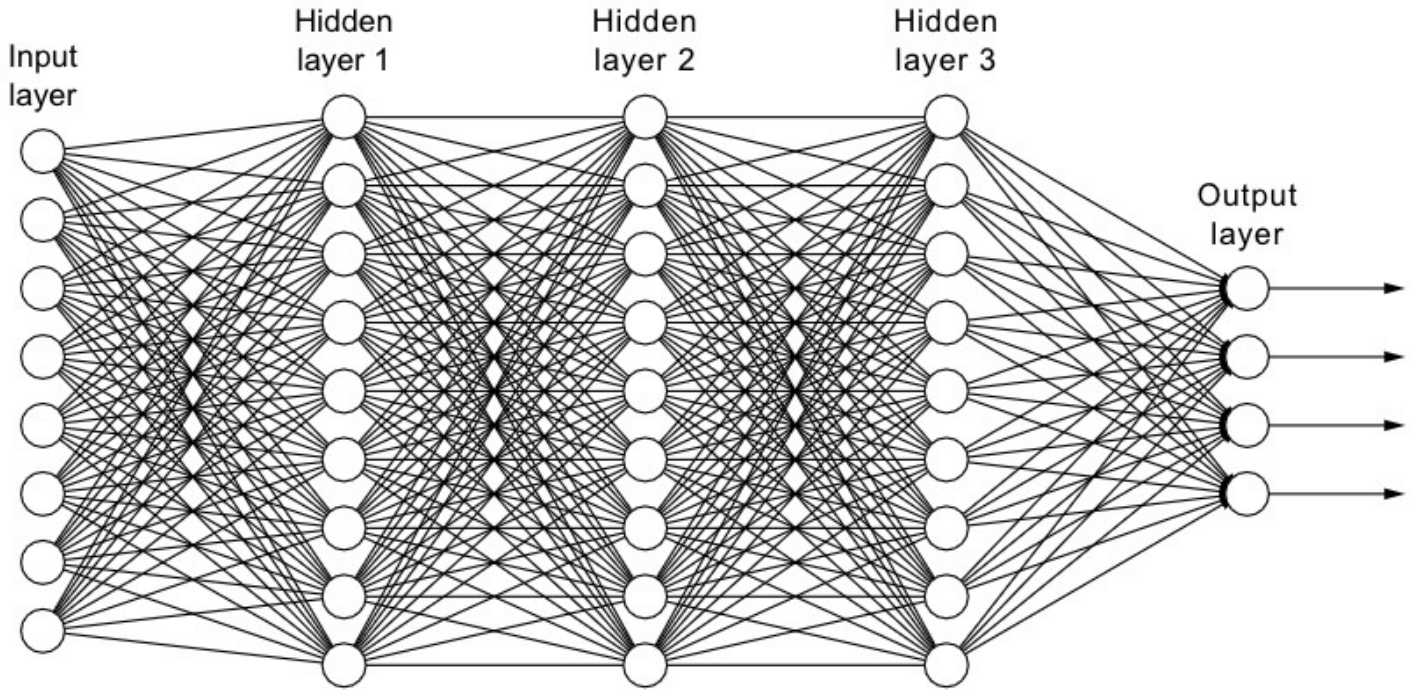
Gradient descent

Back propagation

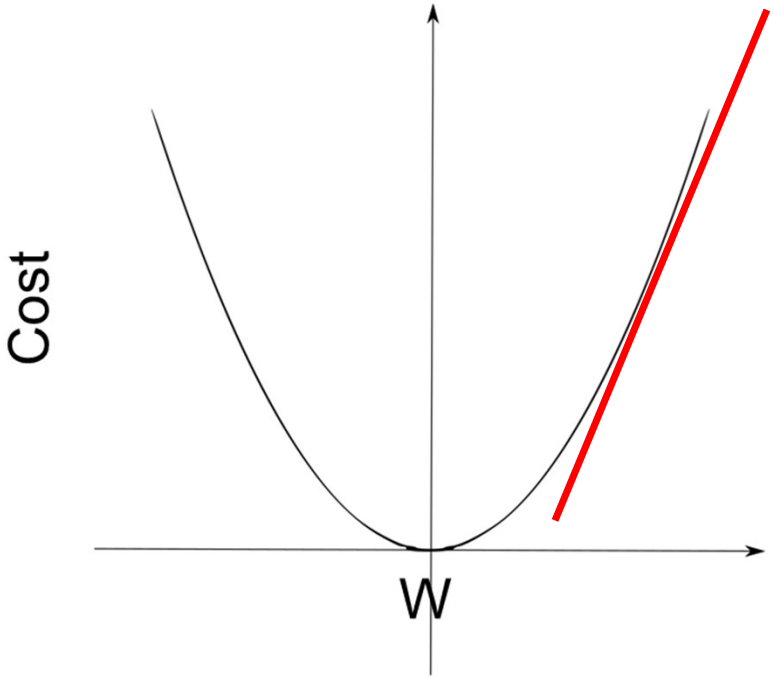


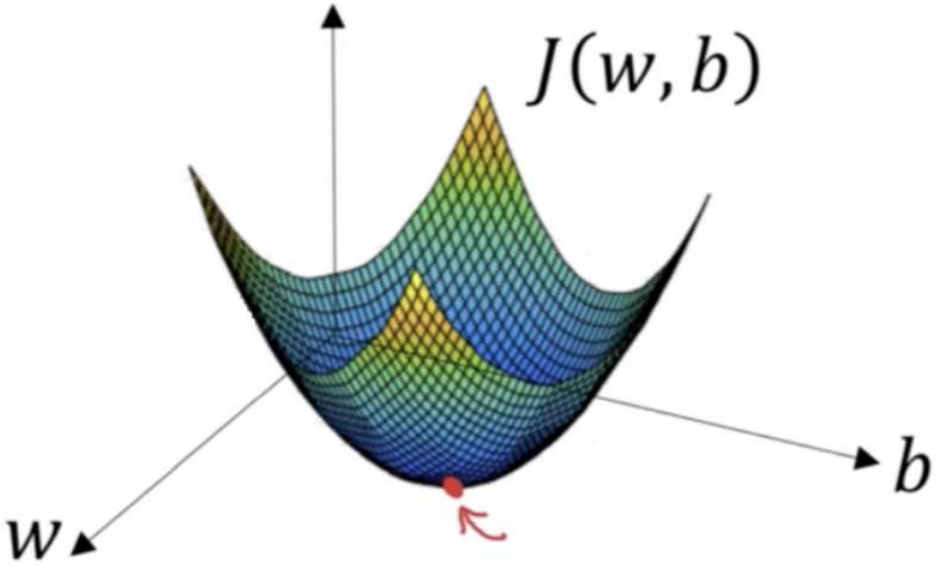
Each epoch: input → output

Loss measurement

Back propagation

Gradient descent





-Back propagation: learning → how? By gradient descent algorithm!
→ Let's find a minimum cost!

Gradient descent

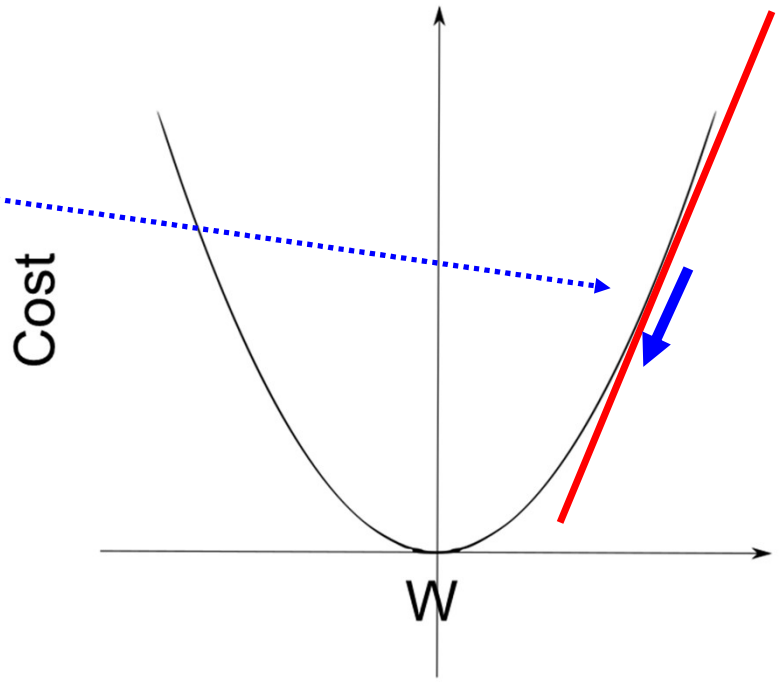
$$\theta = \theta - \eta \nabla J(\theta)$$

θ: *parameter set of the model*

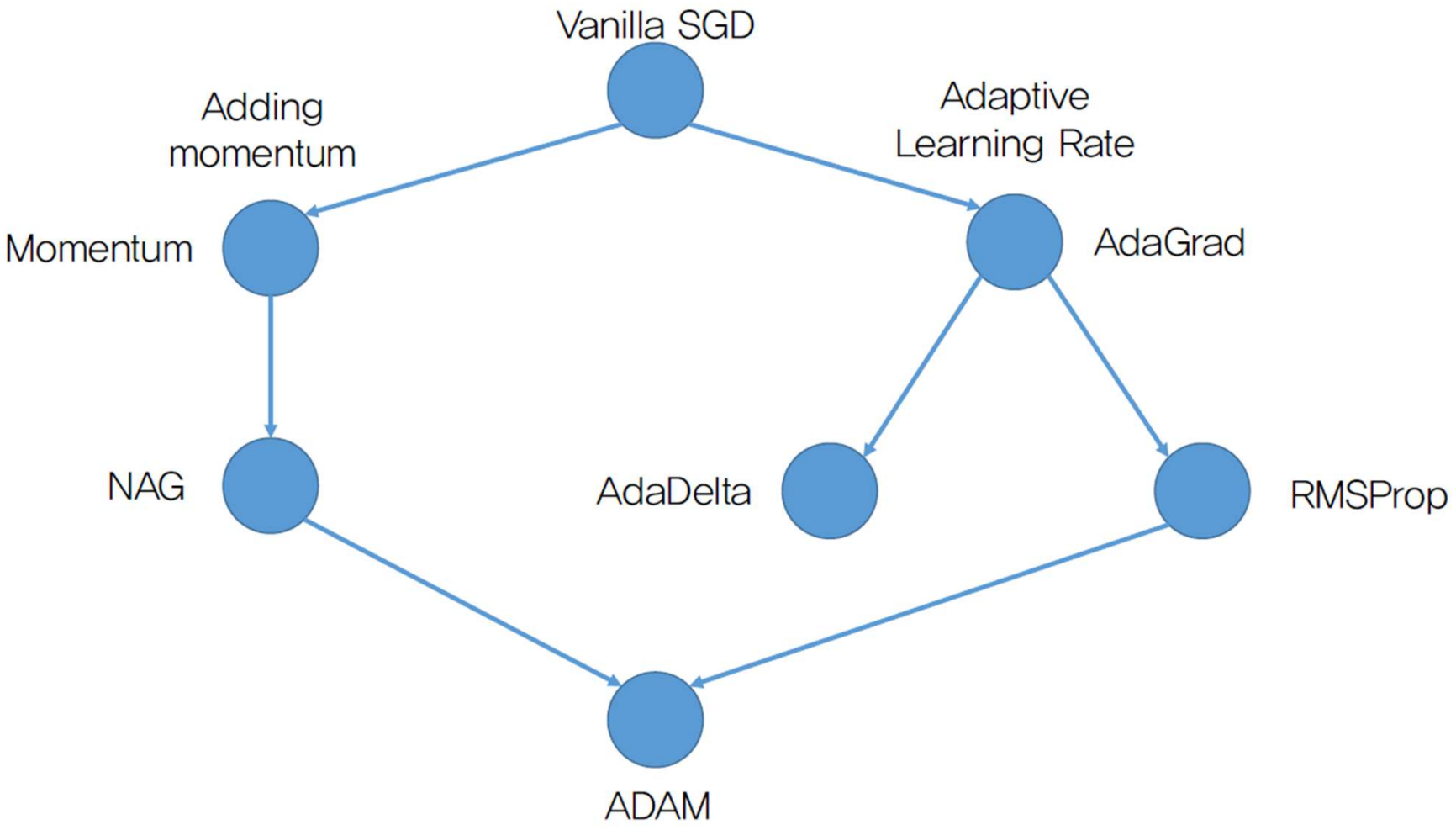
η: *learning rate*

$J(\theta)$: *Loss function*

∇: *gradient* $\rightarrow$ *derivative*



Gradient descent



Gradient descent

-Batch gradient descent
Calculate parameters of gradient for whole training dataset

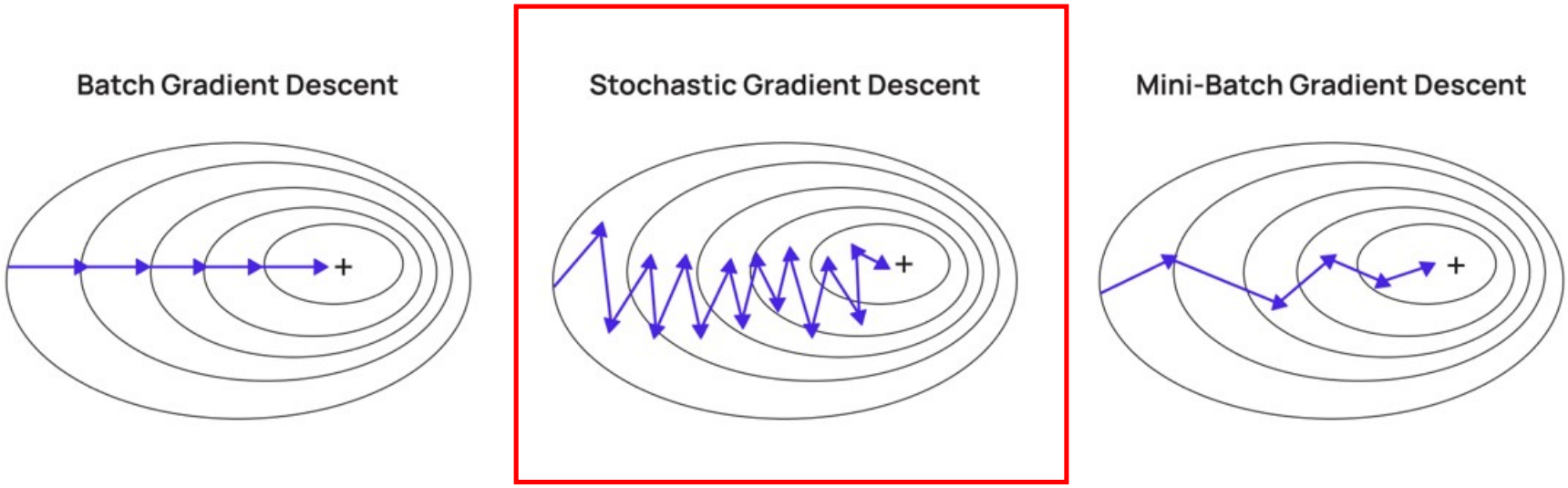
→ Nead a lot of memory
→ Calculation of gradient is too slow

Stochastic Gradient Descent (SGD)

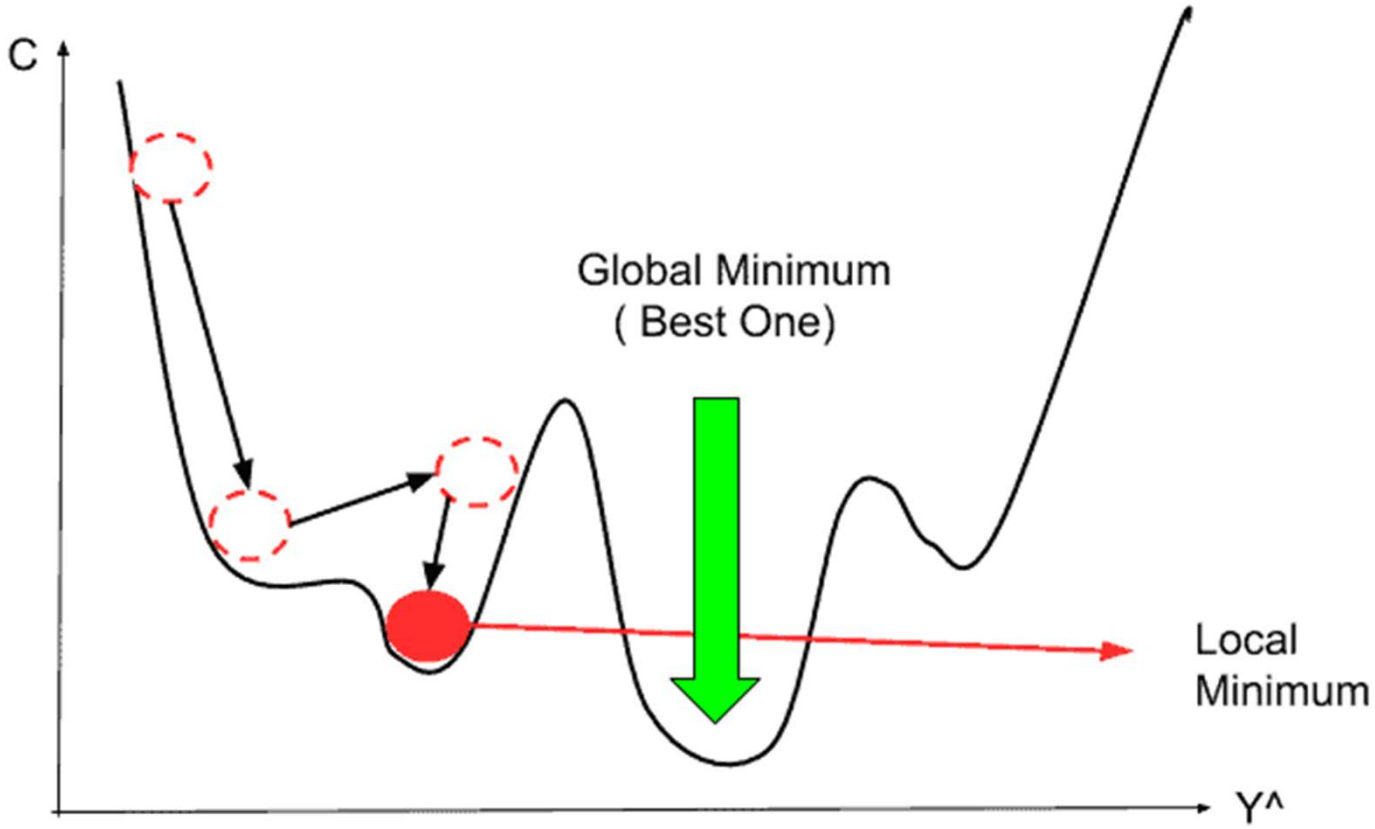
-Calculate gradient for **one sample** of whole training dataset,
rather than the whole training dataset (batch).

-Stochastic since the gradient is not deterministic, but stochastic depending
on the mini batch.

-Faster, while converging similar.
-prone to local minima



Local minima

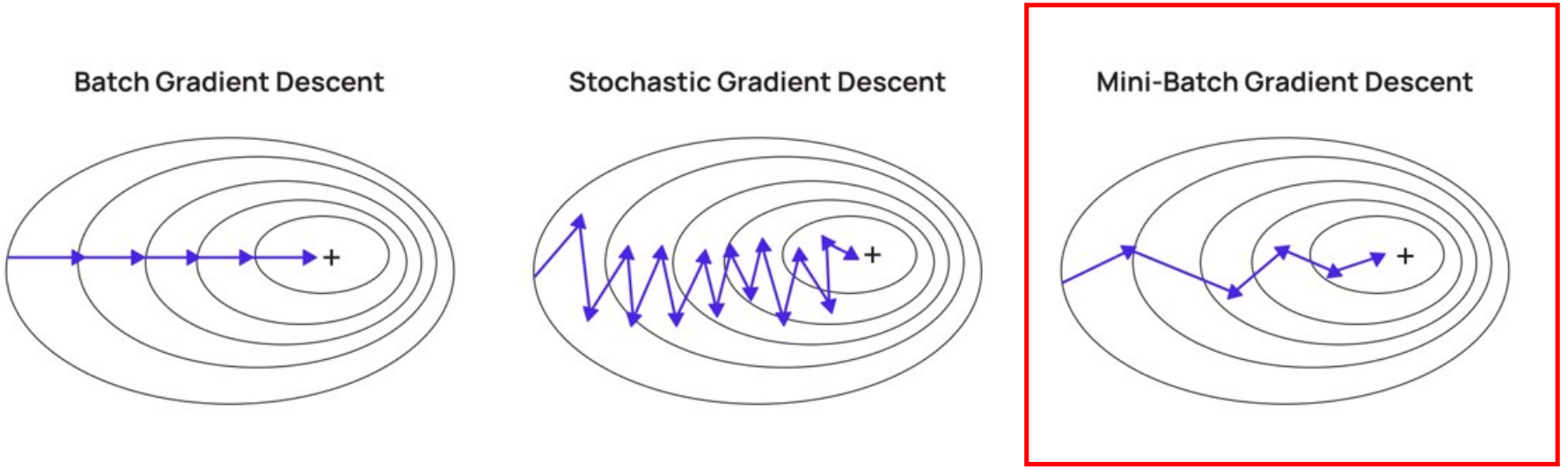


Mini-Batch Gradient Descent

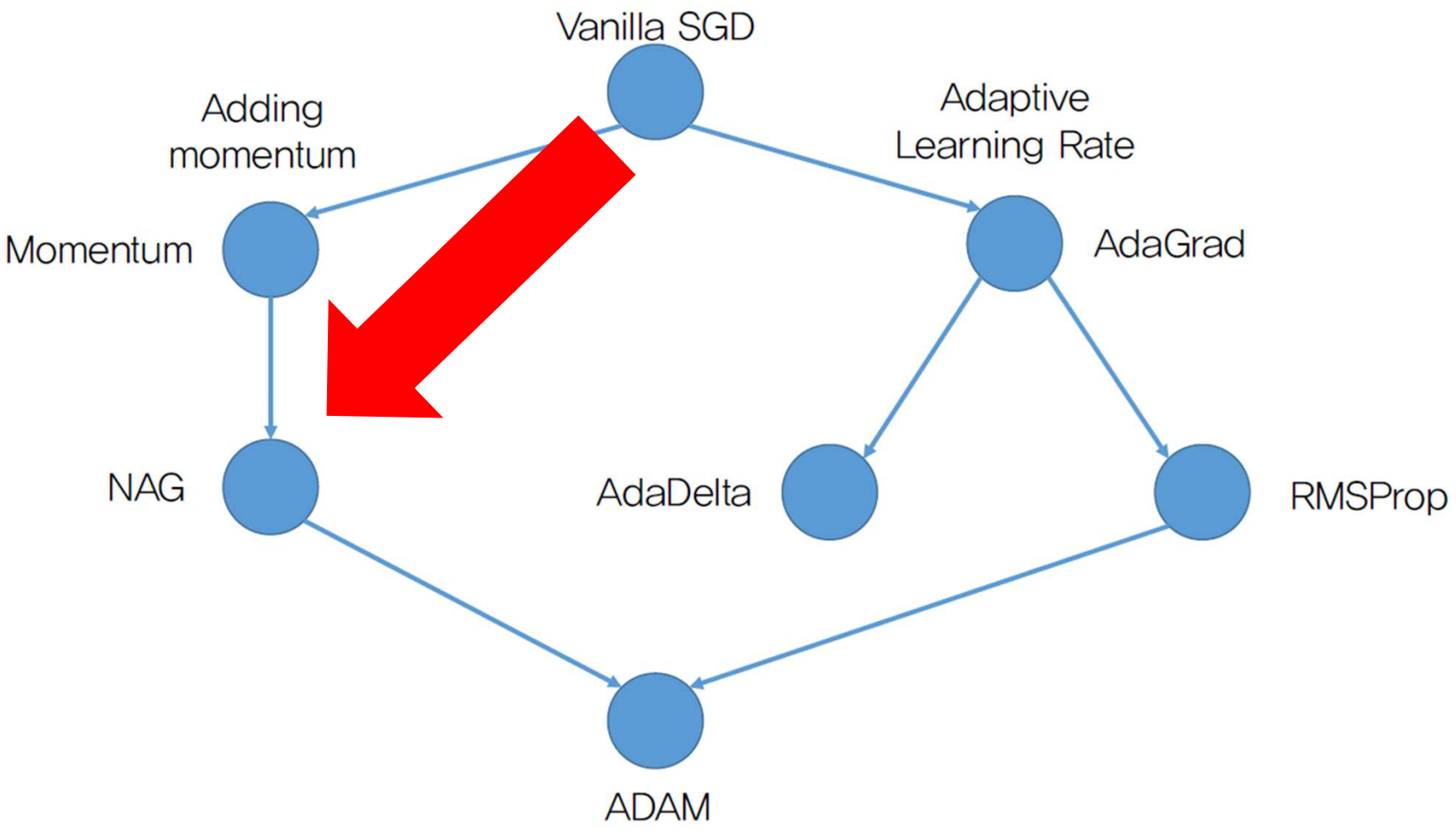
-Calculate gradient for **small chunk (mini-batch > 1)** of whole training dataset

-Stochastic since the gradient is not deterministic, but stochastic depending on the mini batch.

-Faster, while converging similar.
-avoid local minima



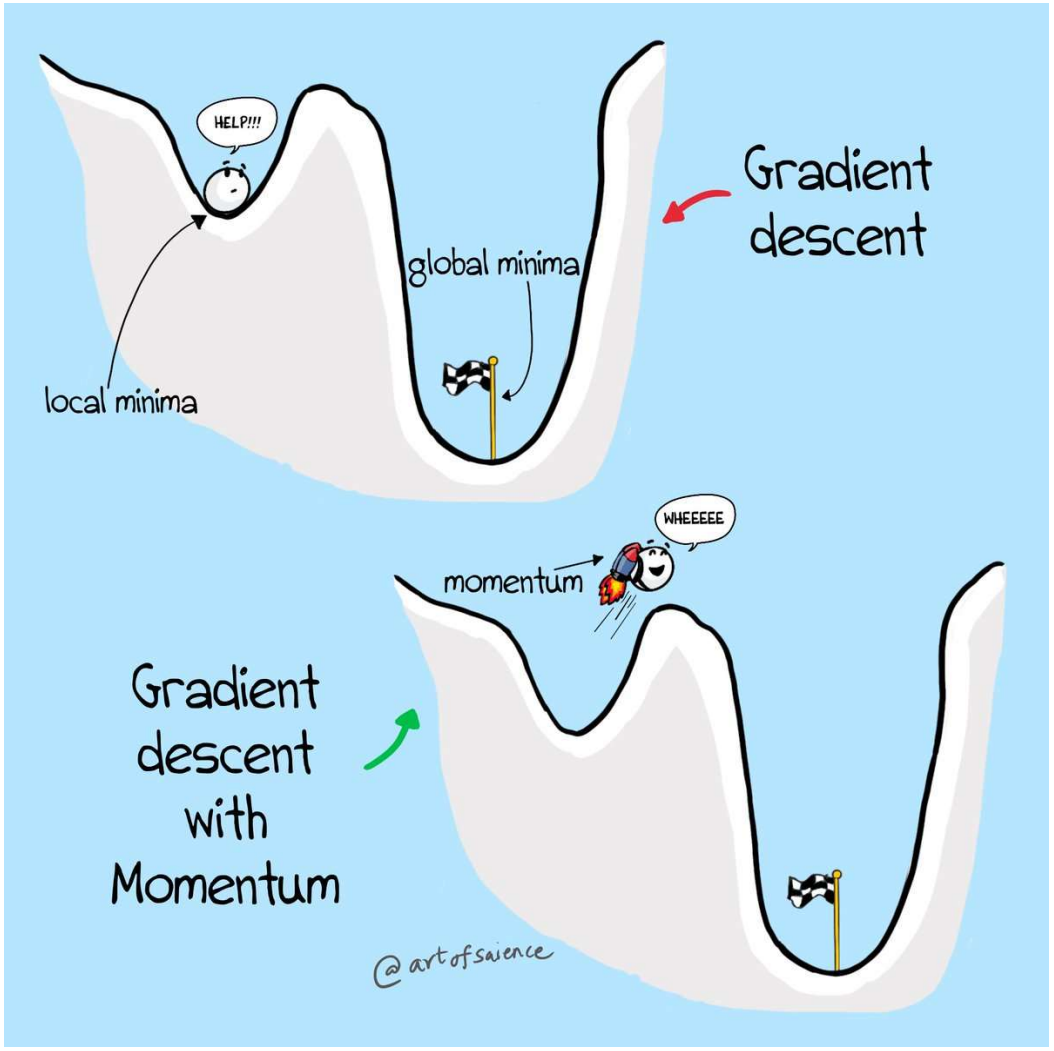
Gradient descent



Inertia



Momentum


HELP!!!
local minima
global minima
Gradient descent
WHEEEEE
momentum
Gradient descent with Momentum
@artofsaience

Momentum

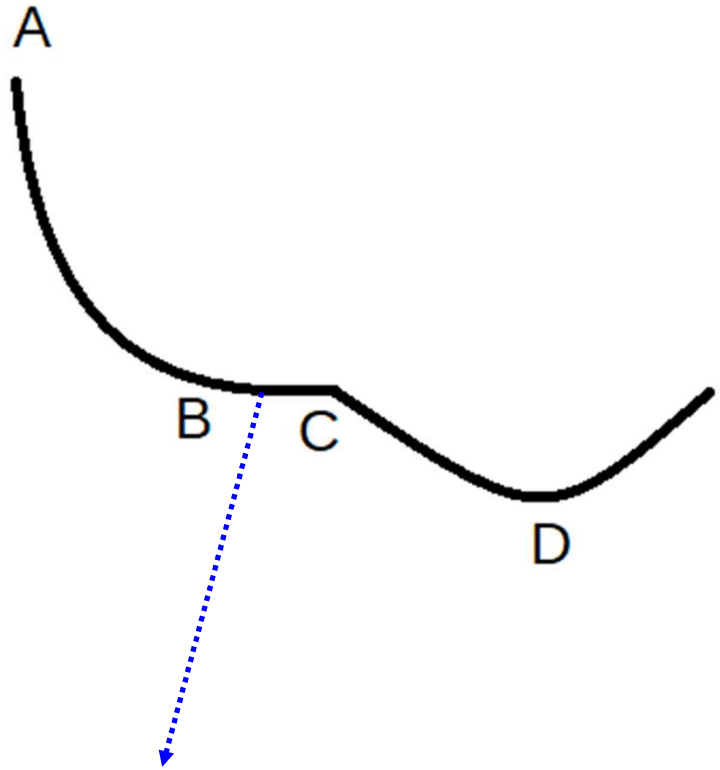
$$\theta = \theta - v_t$$

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

θ: *parameter set of the model*
η: *learning rate*
$J(\theta)$: *Loss function*
∇: *gradient* $\rightarrow$ *derivative*

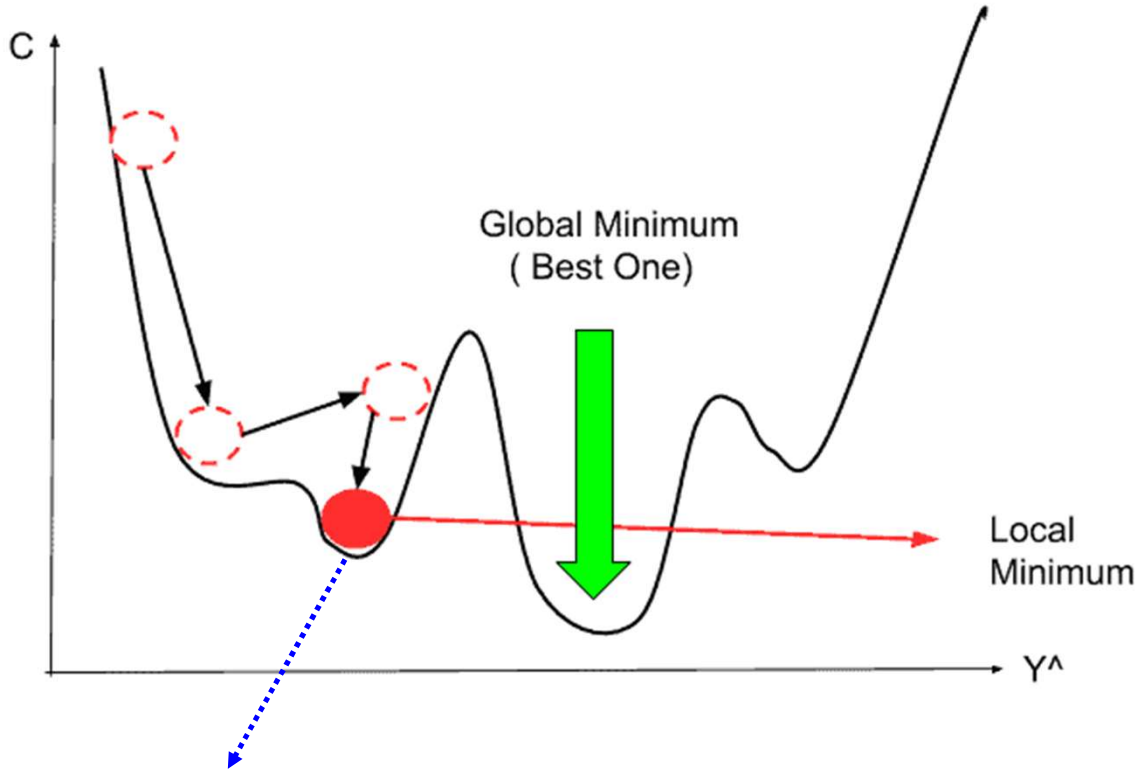
v_t: *current velocity* (*gradient descent*)
γv_{t-1}: *previous velocity* $*$ *weight*

Momentum



-Limitation: cannot proceed at the saddle point (very low momentum ~ 0)

Momentum

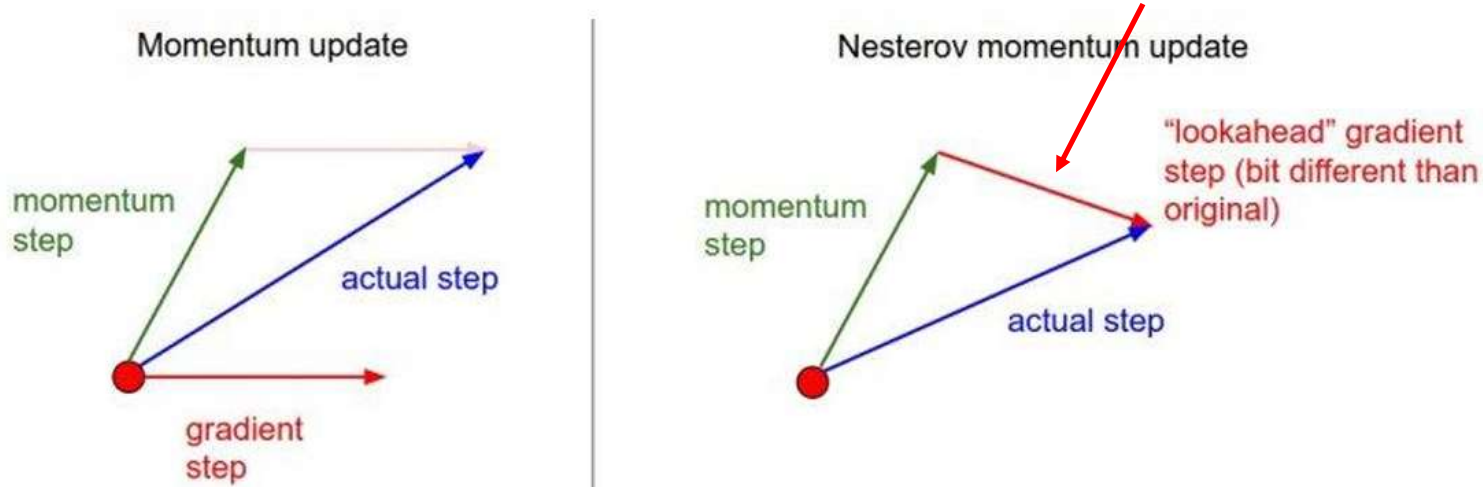


-Limitation: stuck at the local minimum

Nesterov Accelerated Gradient (NAG)

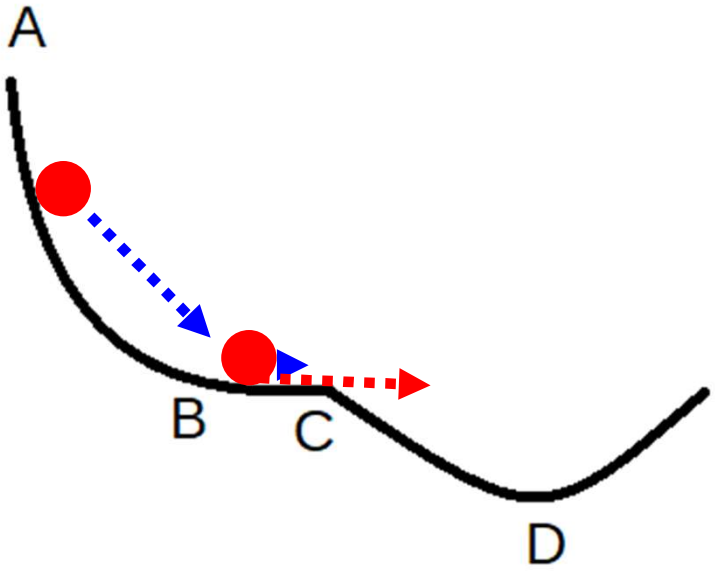
$$\theta = \theta - v_t$$

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

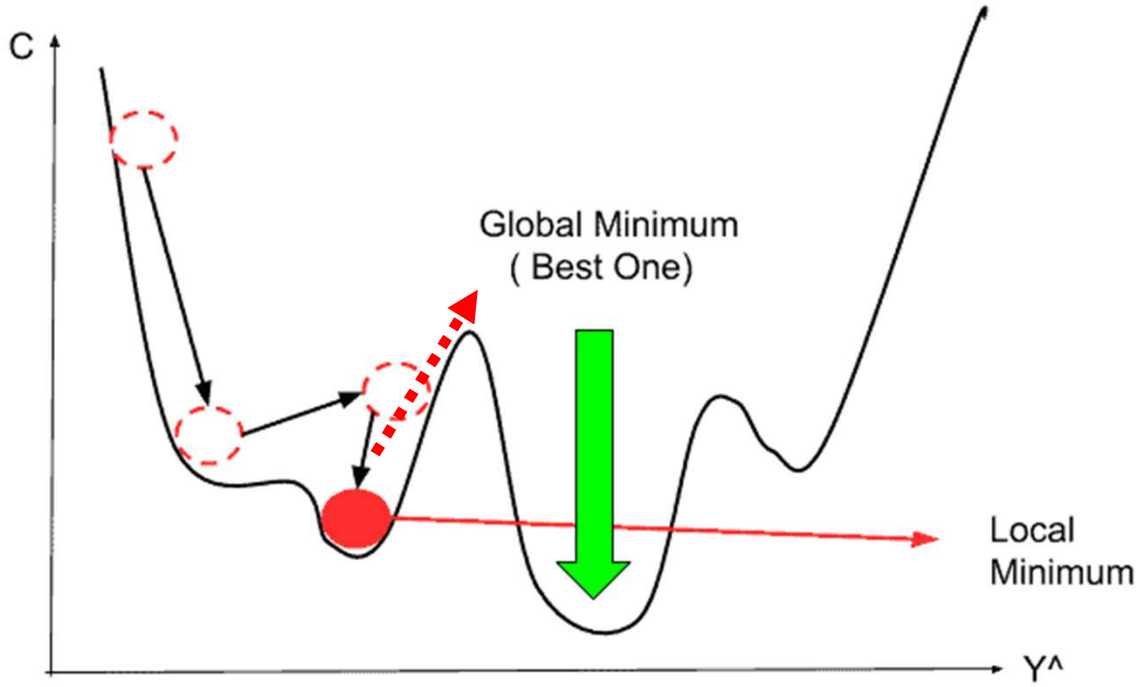


-Another step after “momentum” step → seek more from the saddle point

Momentum



Momentum



Nesterov Accelerated Gradient (NAG)

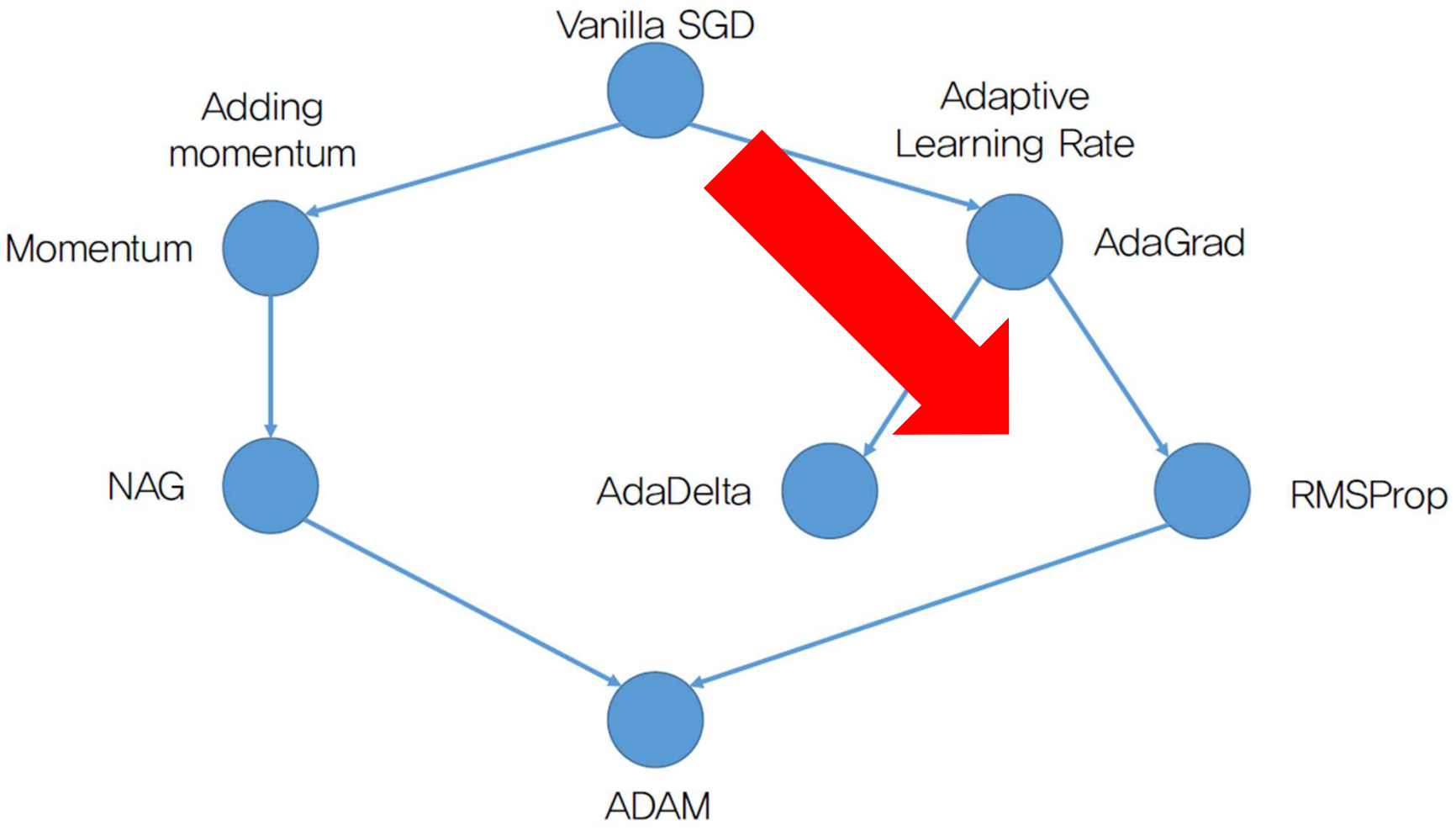
$$\theta = \theta - v_t$$

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



-Many weights for each parameter → same step (gradient)

Gradient descent



Adaptive Gradient (Adagrad)

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

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

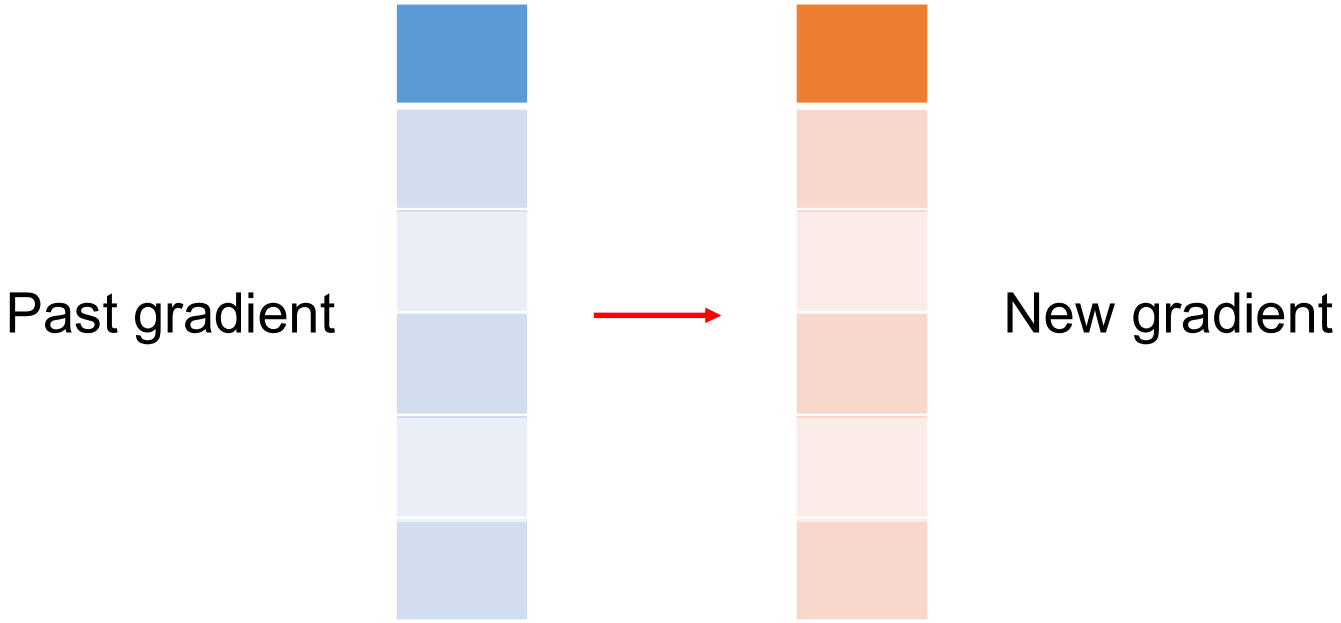
-Bigger step for small gradient in the past
-Smaller step for bigger gradient in the past
-Limitation: G will keep increasing → divergent (infinite) → gradient → 0

Adaptive Gradient (Adagrad)

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

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

→ Bigger gradient:
Root (1/square)
next time



RMSProp

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

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

-γ: 0 ~ 1 (weight for past G vs current factor

-G(t-1): summation until now * [lower than 1] + current factor * [lower than 1]

→ Never divergent into infinite

RMSProp

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

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

u u u^{-1}

-Mathematically awkward → unit does not match

AdaDelta

$$\theta_{t+1} = \theta_t - \Delta_\theta$$

RMSProp

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

$$s_{t+1} = \gamma s_t + (1-\gamma)\Delta_\theta$$

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

-Match the unit

AdaDelta

$$\theta_{t+1} = \theta_t - \Delta_\theta$$

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

u

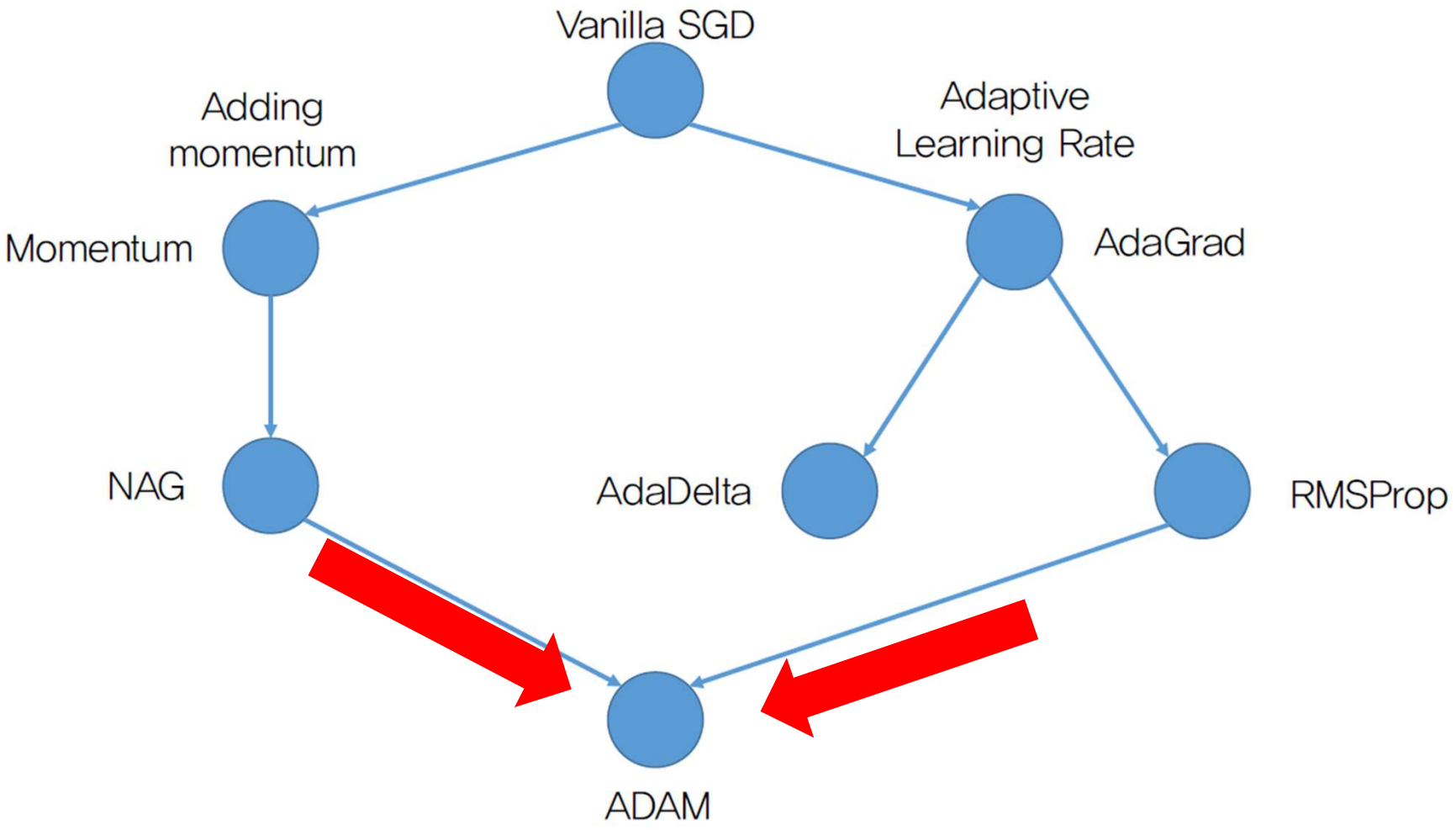
u

$$s_{t+1} = \gamma s_t + (1-\gamma)\Delta_\theta$$

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

-Match the unit

Gradient descent



Adaptive Moment Estimation (ADAM)

$$\nabla_\theta J(\theta_t)$$

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

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

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Proper initialization
(of m and v)

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

Nesterov ADAM (Nadam)

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

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

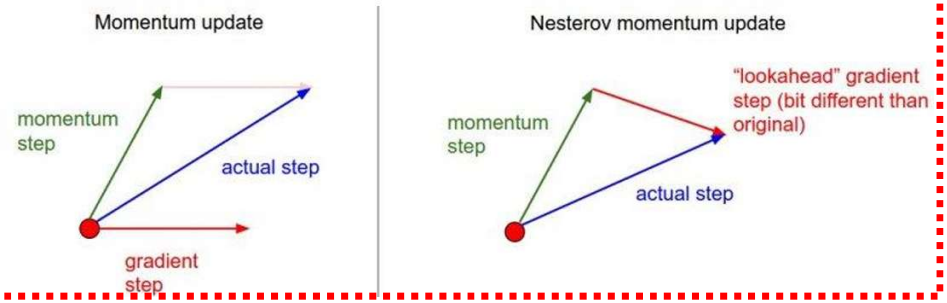
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

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

Nesterov Accelerated Gradient (NAG)

$$\theta = \theta - v_t$$

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



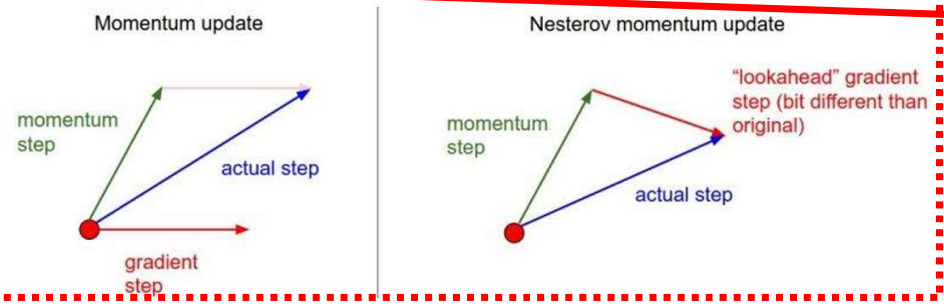
-ADAM momentum → NAG

Nesterov ADAM (Nadam)

Nesterov Accelerated Gradient (NAG)

$$\theta = \theta - v_t$$

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



$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

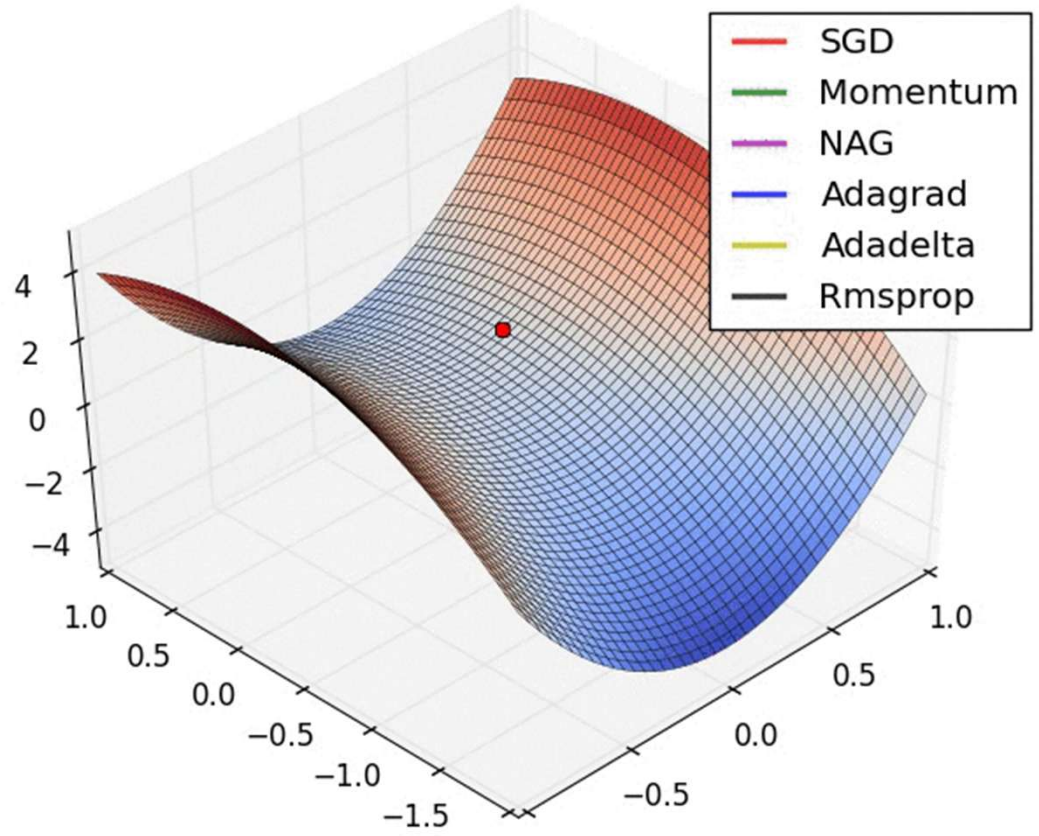
$$\theta_{t+1} = \theta_t - (\gamma m_t + \eta g_t)$$

-To avoid duplicate momentum
→ Current momentum

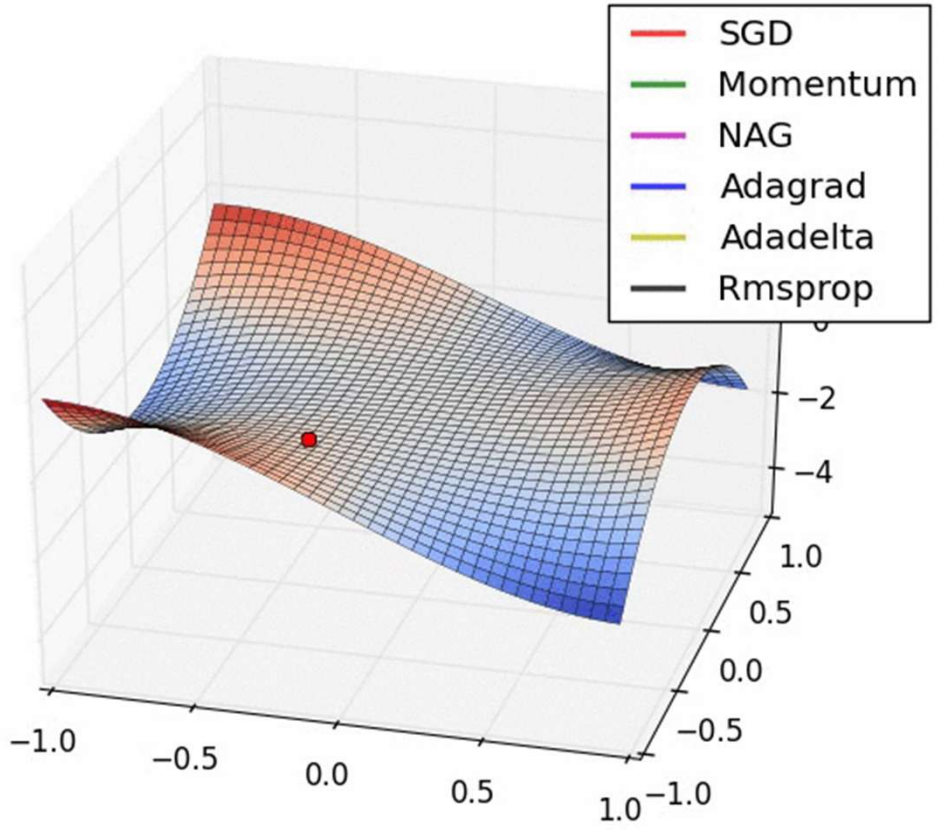
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} (\beta_1 \hat{m}_t + \frac{(1-\beta_1) g_t}{1-\beta_1^t})$$

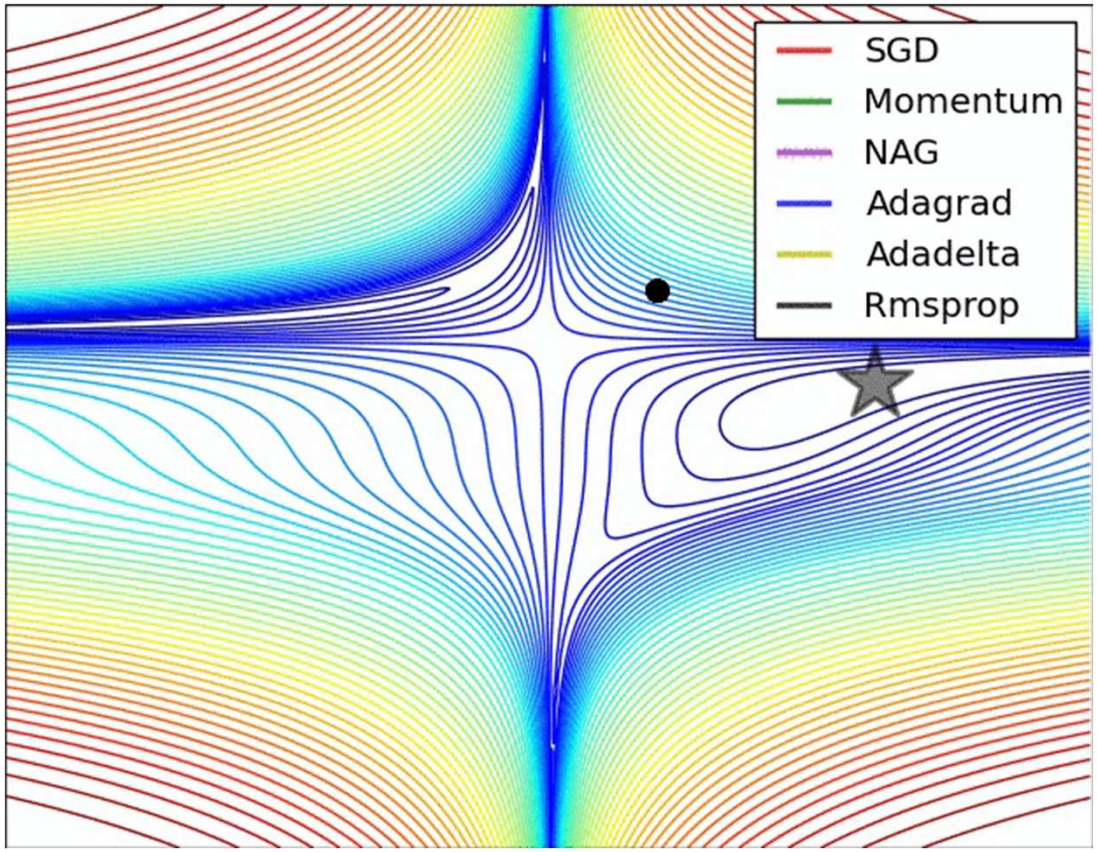
Gradient Descent simulation



Gradient Descent simulation



Gradient Descent simulation



Gradient Descent usage

```
optimizer = optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)
```

torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False, *, foreach=None, maximize=False, capturable=False, differentiable=False, fused=None, decoupled_weight_decay=False)