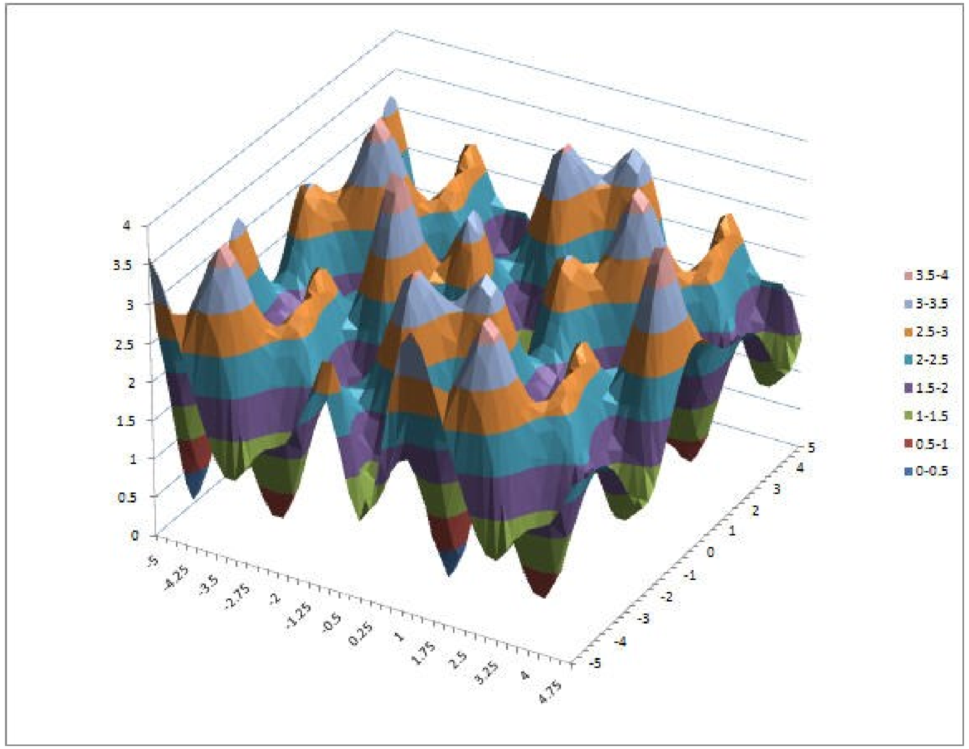
Optimization Methods

SGD Bells and Whistles



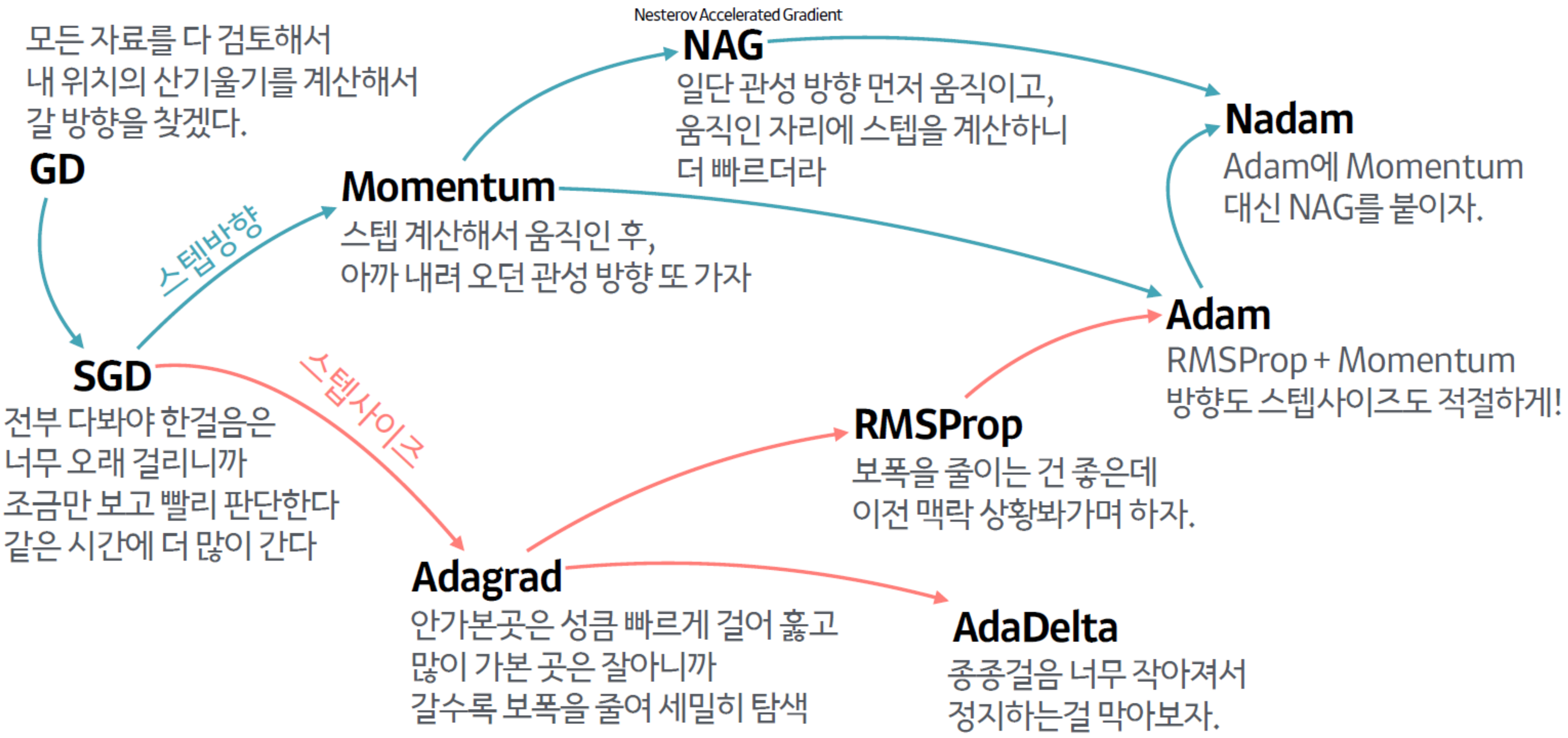
Fast Campus
Start Deep Learning with Tensorflow

3 Weeks Ago…

- We studied about
  - Multi-Layer Perceptron
  - Back Propagation(using chain rule)
  - Softmax function
  - Cross Entropy
  - Underfitting vs Overfitting
  - Regularization Methods
  - Data Augmentation
  - Weight Initialization

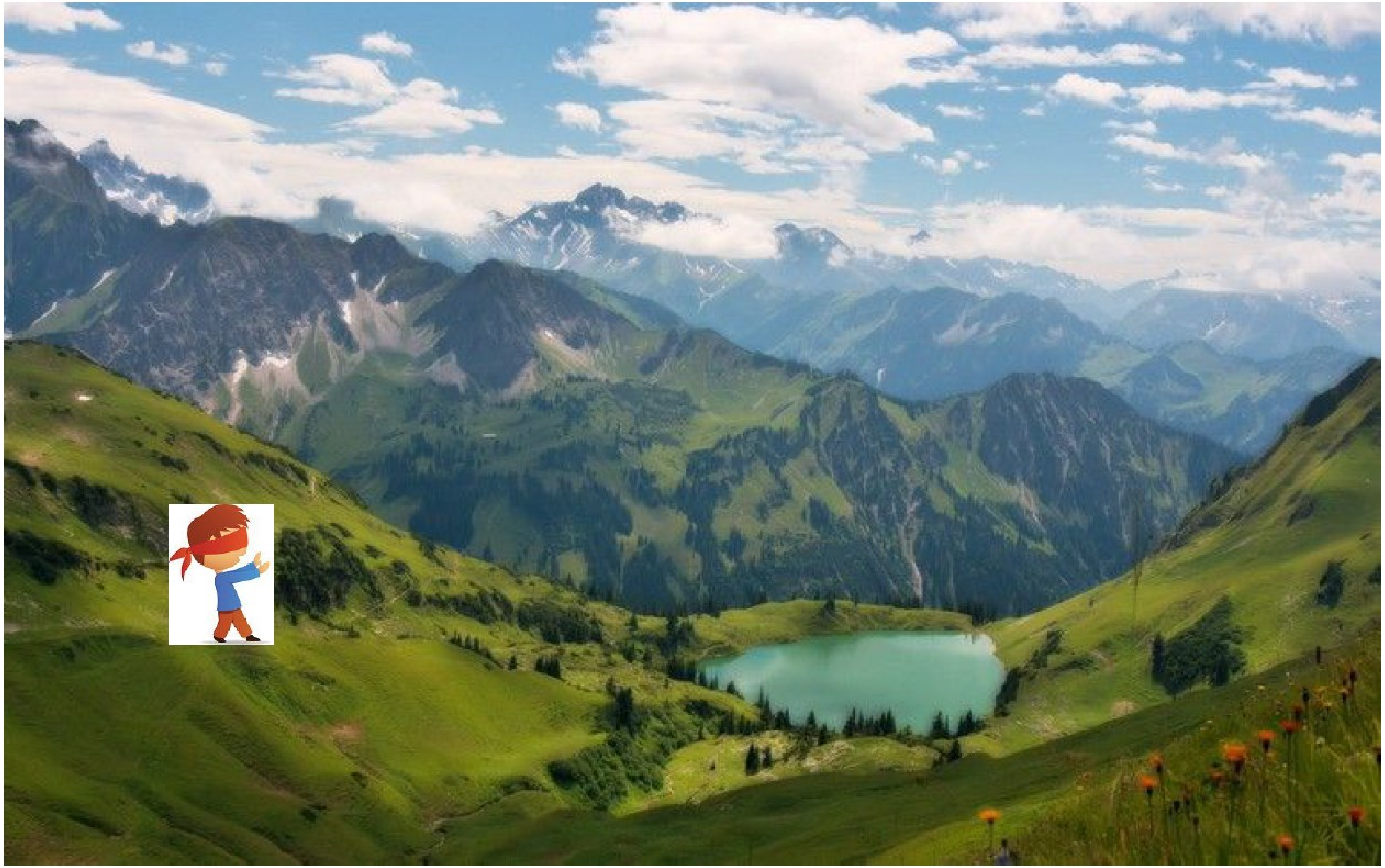
Now we will study

- Optimization Methods



Slide Credit : 하용호@Kakao

Recap : Gradient Descent

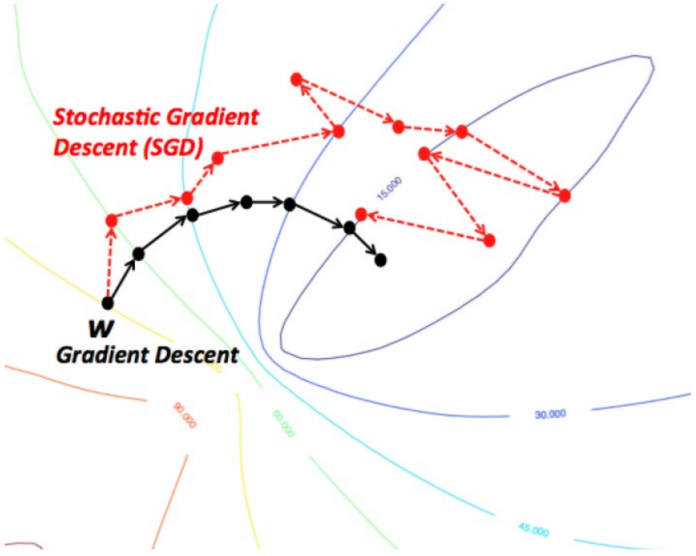


Slide Credit : Stanford CS231n

Recap : Gradient Descent

- Batch gradient descent

$$\theta_t = \overset{\text{Previous parameter}}{\boxed{\theta_{t-1}}} - \underset{\text{Learning Rate}}{\boxed{\eta}} \cdot \overset{\text{Gradient}}{\boxed{\nabla_\theta J(\theta)}}$$



- Stochastic gradient descent

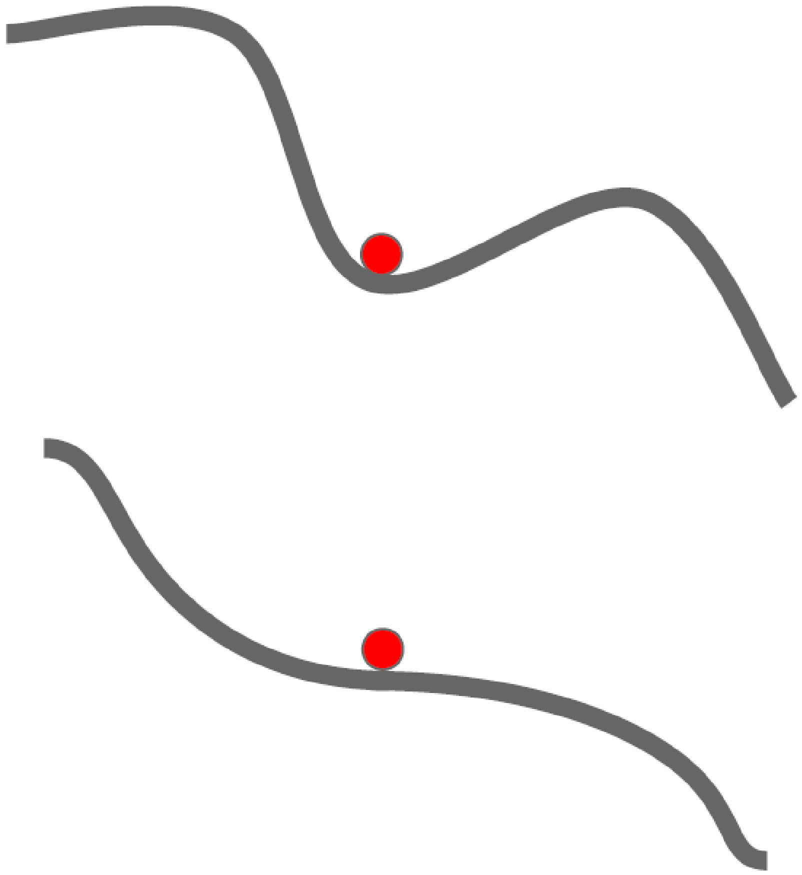
$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

- Mini-batch gradient descent

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Problems of Gradient Descent

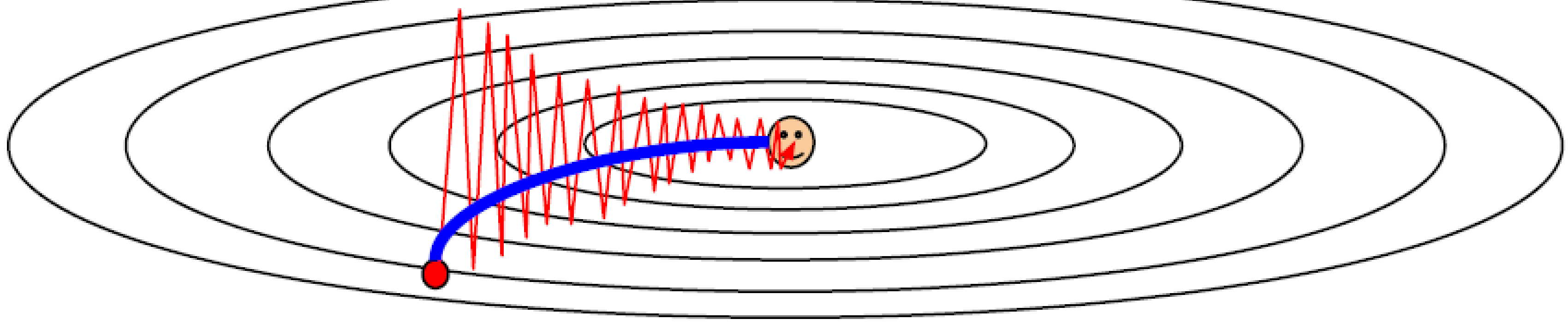
- (It can) stuck at local minima or saddle point(zero gradient)



Picture Credit : Stanford CS231n

Problems of Gradient Descent

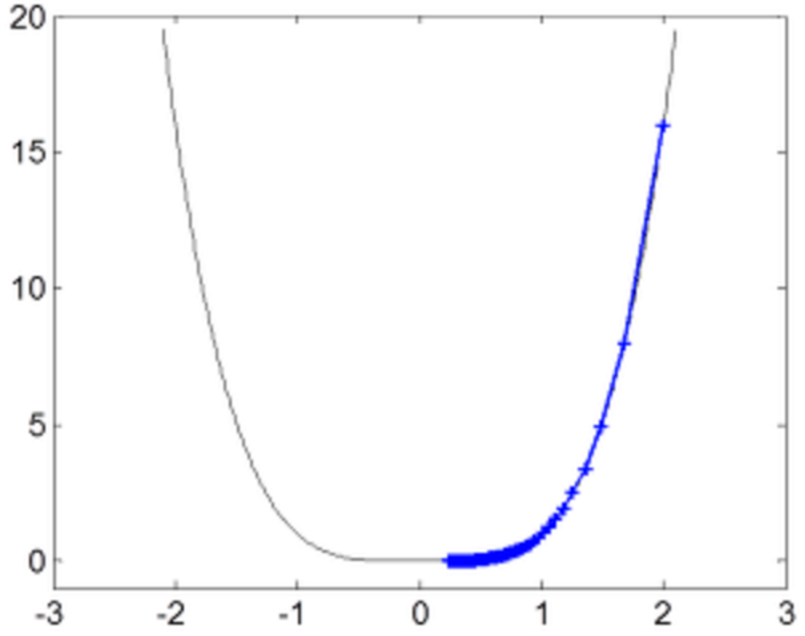
- Poor Conditioning



Picture Credit : Stanford CS231n

Problems of Gradient Descent

- Slow....
  - The closer to the optimal point, the smaller the gradient becomes.



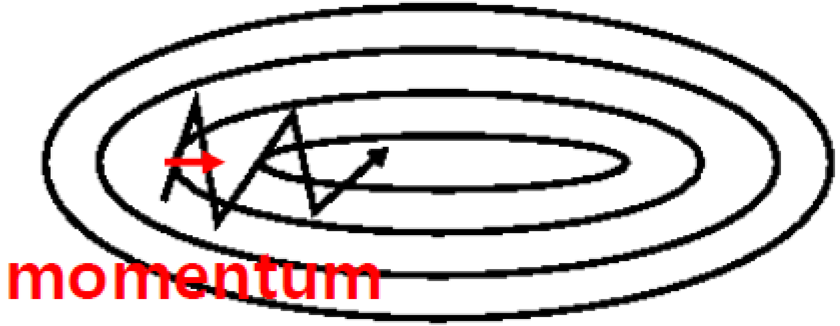
Momentum

- Let's move with inertia in the direction that we moved earlier.

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

Gradient Descent

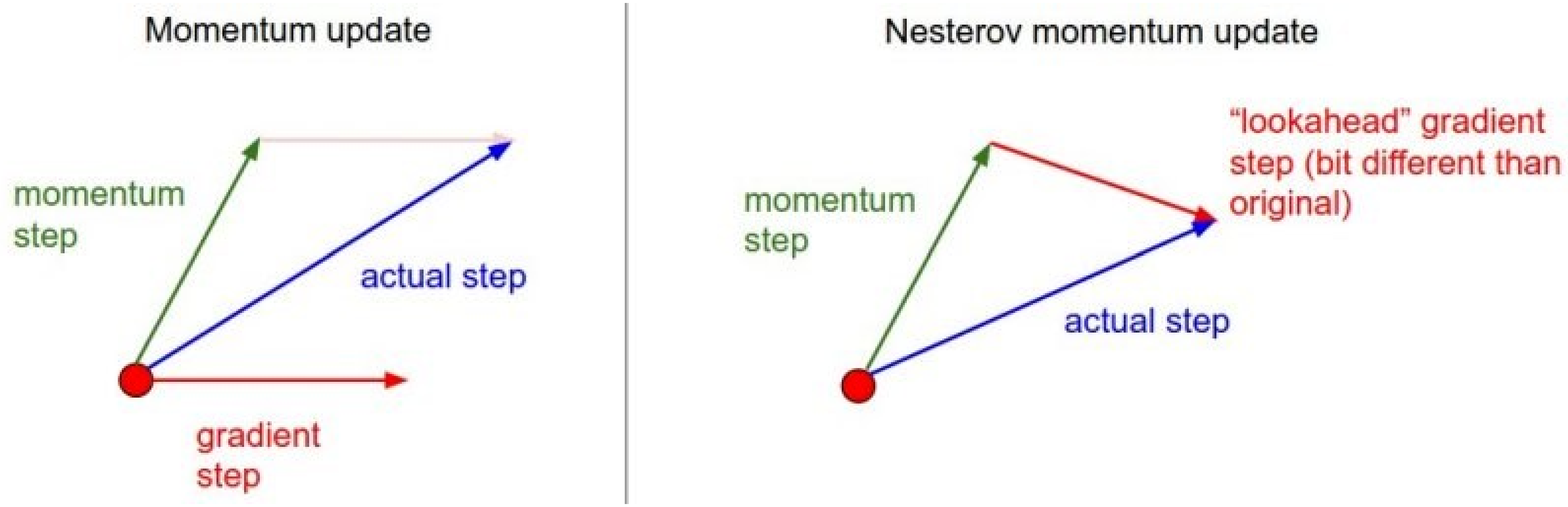
Momentum



NAG(Nesterov Accelerated Gradient)

- Move in the direction we were previously moving, and then calculate the gradient from where we moved.

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



Adagrad(Adaptive Gradient)

- It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters.

Element-wise Product

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

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

- As the learning continues, the G value continues to increase, so the step size becomes too small to learn

RMSprop

- Use exponentially decaying average of squared gradients

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

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

AdaDelta(Adaptive Delta)

- Similar to RMSprop
- Also use exponentially decaying average of step size

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

$$s_t = \gamma s_{t-1} + (1-\gamma)\Delta_\theta^2$$

$$\Delta_\theta = \boxed{\frac{\sqrt{s_t+\epsilon}}{\sqrt{G_t+\epsilon}}} \cdot \nabla_\theta J(\theta_t)$$

Learning rate
→ insensitive to hyper-parameters

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

Adam(Adaptive Moment Estimation)

- Adaptive Moment Estimation (Adam) stores both exponentially decaying average of past gradients and squared gradients
- Combination of Momentum and RMSprop
- Compensate the initial momentum biased towards zero

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

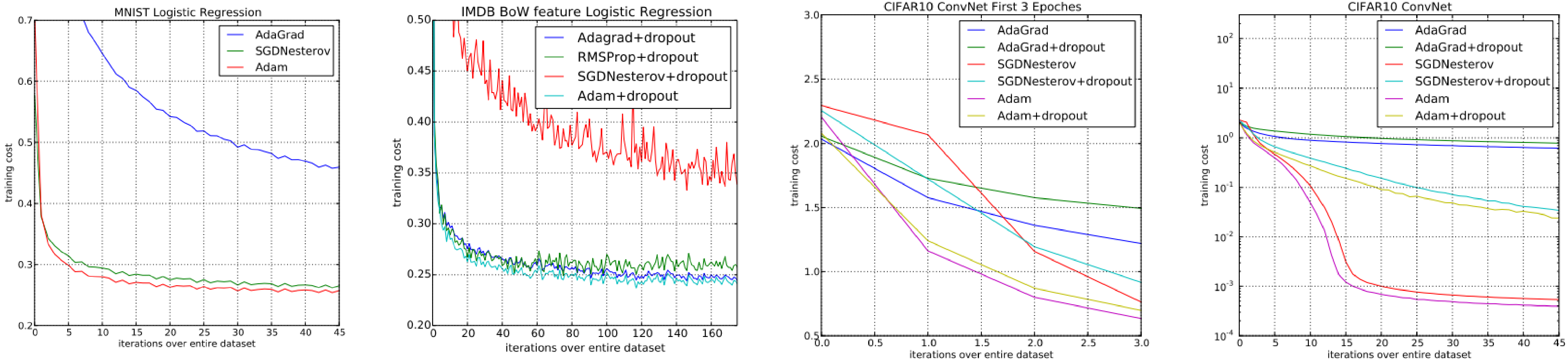
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\widehat{m_t} = {}^{m_t}/_{1-\beta_1^t} \quad \widehat{v_t} = {}^{v_t}/_{1-\beta_2^t}$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon} \widehat{m_t}$$

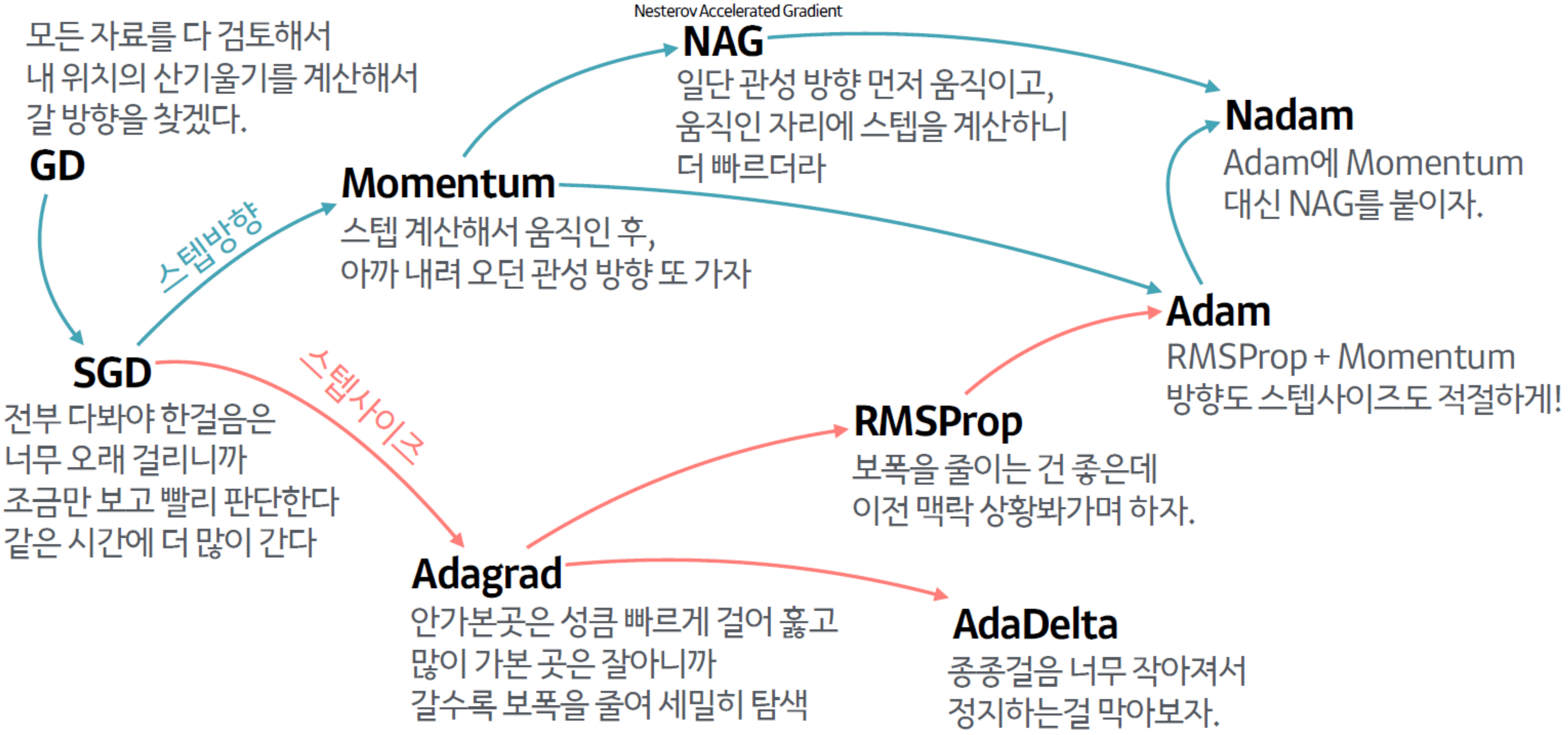
Adam

- Adam is a very popular method



We Studied

- Optimization Methods



Slide Credit : 하용호@Kakao

Optimizers in Tensorflow

- Just use these APIs!
  - `tf.train.Optimizer`
  - `tf.train.GradientDescentOptimizer`
  - `tf.train.AdadeltaOptimizer`
  - `tf.train.AdagradOptimizer`
  - `tf.train.AdagradDAOptimizer`
  - `tf.train.MomentumOptimizer`
  - `tf.train.AdamOptimizer`
  - `tf.train.FtrlOptimizer`
  - `tf.train.ProximalGradientDescentOptimizer`
  - `tf.train.ProximalAdagradOptimizer`
  - `tf.train.RMSPropOptimizer`