

Optimization Algorithms

2022. 8

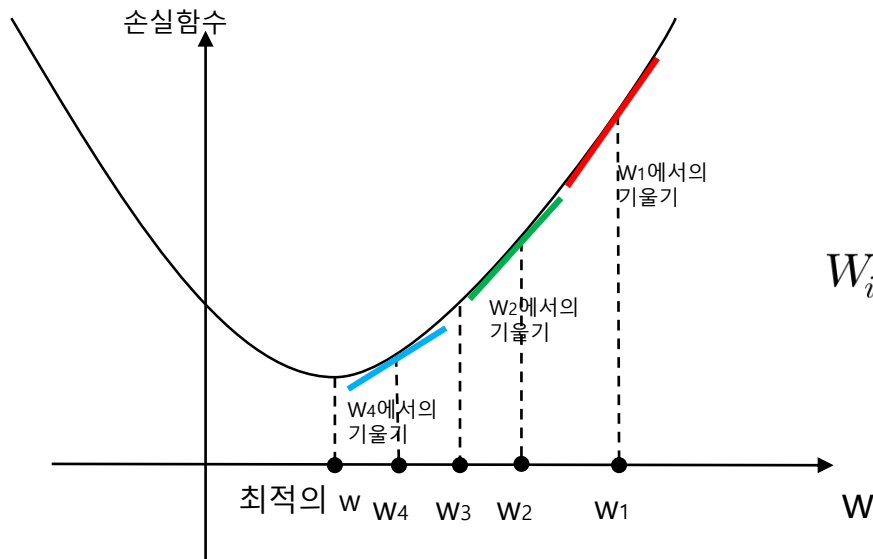
Yongjin Jeong, KwangWoon University

[참고] 본 자료에는 인터넷에서 다운받아 사용한 그림이나 수식들이 들어
있으니 다른 용도로 사용하거나 외부로 유출을 금해 주시기 바랍니다.

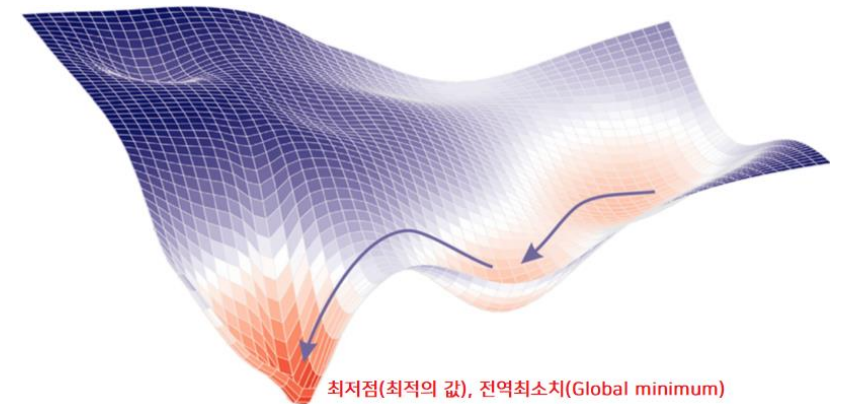
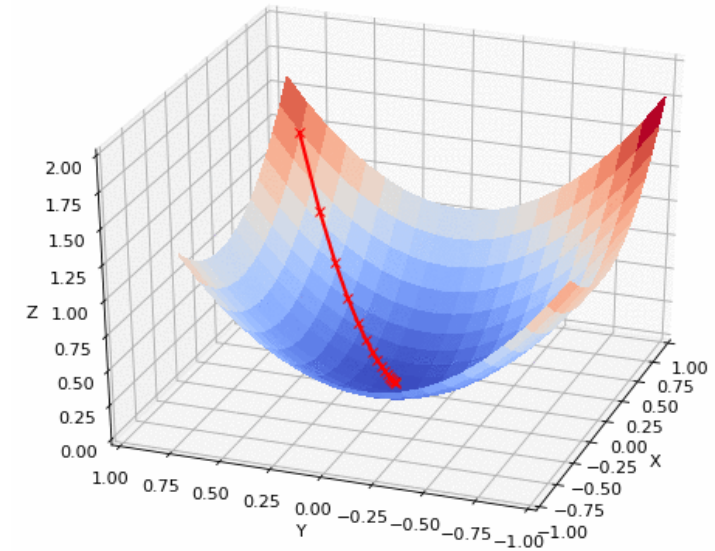
Optimization algorithms

- **Gradient Descent** (경사하강법)

- General optimization algorithm
- take repeated steps in the opposite direction of the **gradient** (or approximate **gradient**) of the function at the current point



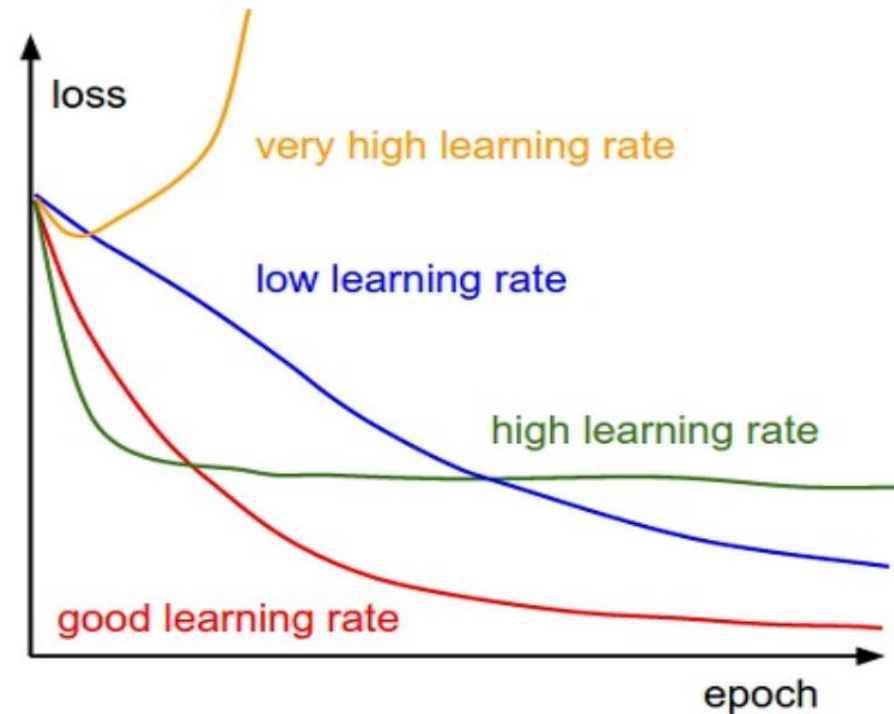
$$W_i = W_{i-1} - \eta \text{Grad}(i)$$



Optimization algorithms

- **Learning rate: η (eta)**

- low: takes time to converge, and may get stuck in an undesirable local minimum
- high: may jump over minima
- too high: may diverge
- Need adaptive adjustment



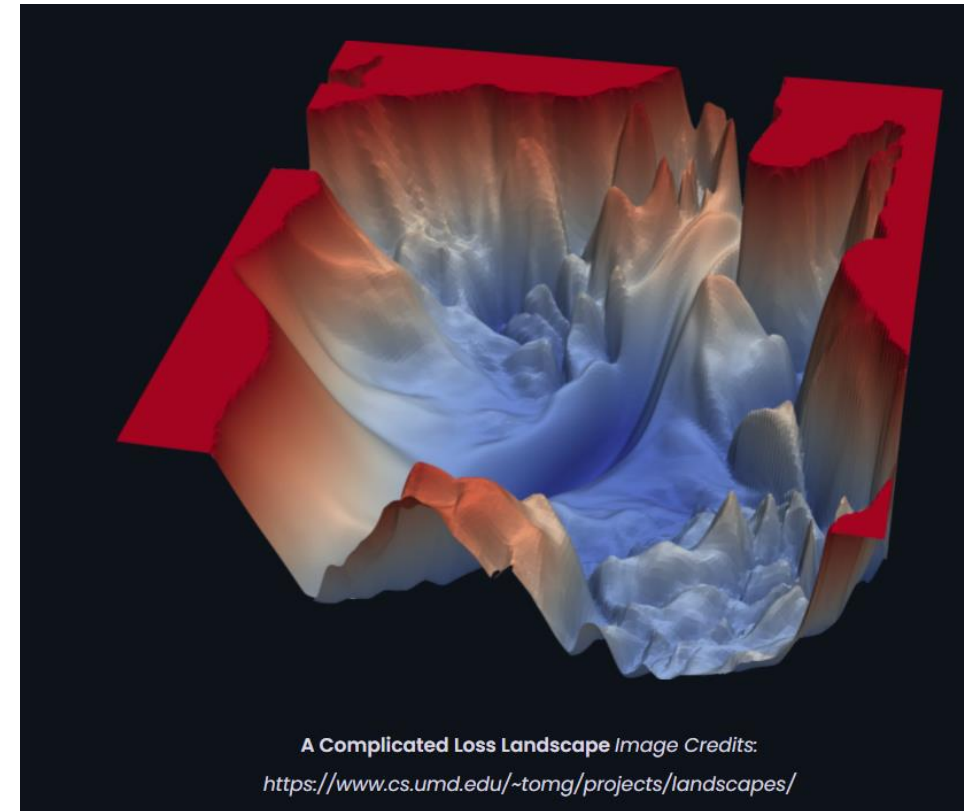
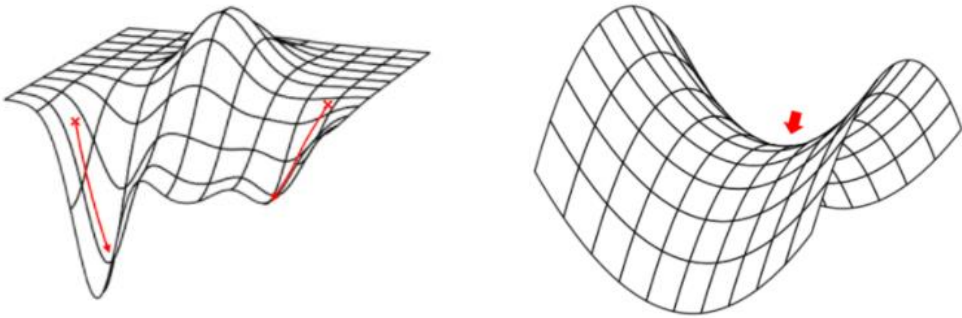
Optimization algorithms

- **Gradient descent**

- computes the gradient of the cost function w.r.t. to the parameters θ for the **entire** training dataset for each update (too much computation)

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$$

- Local minimum
- Saddle Point

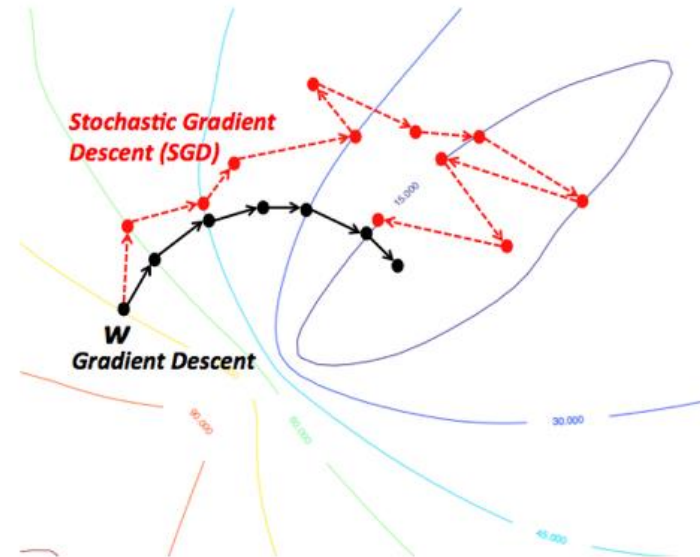


Optimization algorithms

- **Stochastic gradient descent (SGD)**

- performs a parameter update for **each** training example $x^{(i)}$ and label $y^{(i)}$
- usually much faster and can also be used to learn online
- It performs frequent updates with a high variance that cause the objective function to fluctuate heavily.
- higher probability not to fall in local minima

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



Optimization algorithms

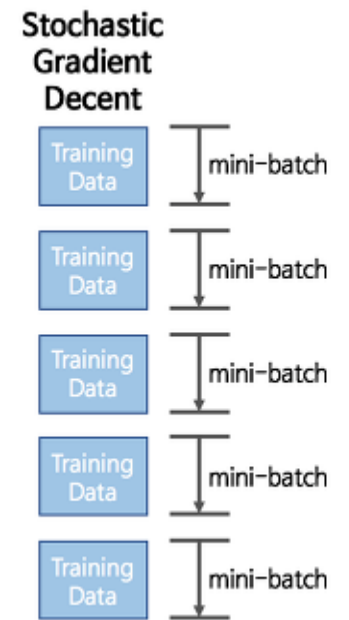
- **Mini-batch gradient descent**

- performs an update for every **mini-batch** of n training examples
- common mini-batch sizes range between 50 and 256, but can vary for different applications

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

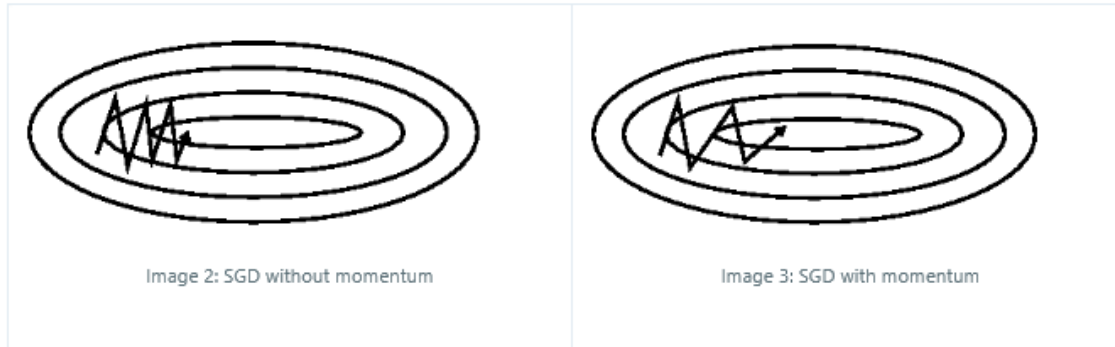
- **Challenges**

- GD does **not guarantee good convergence**, but offers few challenges.
- How to find **optimal learning rates**? -> decaying (step size)
- How often should the parameters be **updated**? -> batch size
- How to **escape from local minima**? -> momentum (step direction)



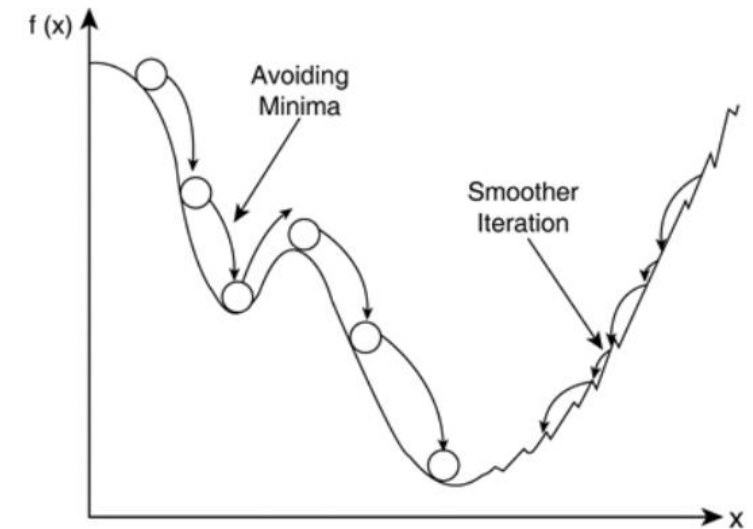
Optimization algorithms

- **Momentum (batch GD with Momentum)**
 - movement(update) vector (v) introduced
 - Also use the results from previous batch (momentum term m is close to 0.9)
 - Converge faster, reduced vibration, and helps to escape from local minimums.



$$V_t = m \times V_{t-1} - \eta \nabla_{\omega} J(\omega_t)$$

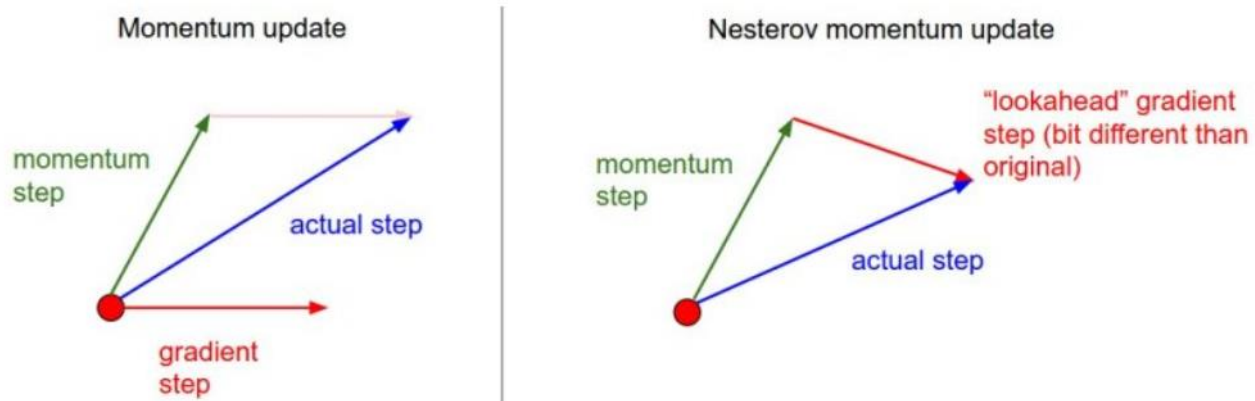
$$\omega_{t+1} = \omega_t + V_t$$



Avoiding Local Minima. Picture from <http://www.yalDEX.com>.

Optimization algorithms

- Improved Momentum (**Nesterov accelerated gradient (NAG)**)
 - Momentum 방법과 동일하지만 Gradient 계산이 약간 다름
 - Momentum 에서는 현재 위치의 기울기와 모멘텀을 따로 계산하고 나중에 더하지만, 여기서는 먼저 모멘텀을 계산 후 기울기 를 계산



Difference between Momentum and NAG. Picture from CS231.

$$V_t = m \times V_{t-1} - \eta \nabla_{\omega} J(\omega_t - m \times V_{t-1})$$

$$\omega_{t+1} = \omega_t + V_t$$

Optimization algorithms

- Separate adaptive learning rates: 학습률을 weight 에 따라 다르게 함.
- **Adagrad** (Adaptive gradient)
 - 파라미터 별 update (different learning rates for ω_i 's)
 - 과거에 많이 변경되지 않은 매개 변수에 더 큰 learning rate 적용 -> step-size 감소
 - 아래 식에서 squaring 과 dot(.) 연산은 element-wise 연산

$$G_t = G_{t-1} + (\nabla_{\omega} J(\omega_t))^2 = \sum_{i=1}^k (\nabla_{\omega_i} J(\omega_i))^2$$

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{G_t} + \epsilon} \cdot \nabla_{\omega} J(\omega_t)$$

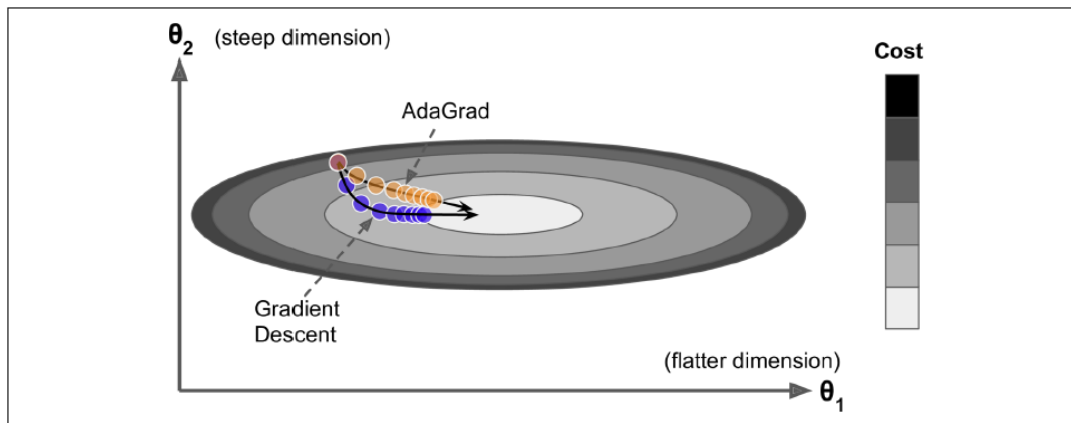


Figure 11-7. AdaGrad versus Gradient Descent

파이썬 소스 코드

```
1 | g += gradient**2
2 | weight[i] += - learning_rate ( gradient / (np.sqrt(g) + e)
```

Tensorflow 소스 코드

```
1 | optimizer = tf.train.AdagradOptimizer(learning_rate=0.01).minimize(loss)
```

Keras 소스 코드

```
1 | keras.optimizers.Adagrad(lr=0.01, epsilon=1e-6)
```

Ref: Hands-on-Machine Learning (2nd)

Optimization algorithms


- **RMSProp** (root mean square propagation)
 - Adagrad 알고리즘은 너무 급격히 감소하여 global optimum 에 도달하지 못하는 경우 발생
 - 처음부터 모든 gradient G_t 를 합산하는 대신 **지수 평균** (exponential moving average) 사용하여 최근 것 사용 (more weights on the recent gradient): typical decay rate $\gamma = 0.9 \sim 0.999$
 - **Always performs much better than Adagrad, and most preferred until Adam came around.**

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

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_{\omega} J(\omega_t)$$

`optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9)`

gamma



Optimization algorithms

- **Adam** (Adaptive Moment Estimation)

- RMSProp + Momentum
- Momentum 과 유사하게 기울기의 지수 평균 반영 (과거 기울기 (past gradient) 의 지수적으로 감소하는 평균을 유지)
- RMSProp 과 유사하게 기울기 제곱 값의 지수 평균 반영 (각 매개 변수에 대한 적응형 학습률 (adaptive learning rate))
- Recommended values: $\epsilon=10^{-8}$, $\beta_1=0.9$, $\beta_2=0.999$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\omega} J(\omega_t) \quad \text{first moment}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\omega} J(\omega_t))^2 \quad \text{second moment}$$

$$\omega_{t+1} = \omega_t - m_t \frac{\eta}{\sqrt{v_t + \epsilon}}$$

$$\begin{aligned} M(t) &= \beta_1 M(t-1) + (1 - \beta_1) \frac{\partial}{\partial w(t)} \text{Cost}(w(t)) \\ V(t) &= \beta_2 V(t-1) + (1 - \beta_2) \left(\frac{\partial}{\partial w(i)} \text{Cost}(w(i)) \right)^2 \\ \hat{M}(t) &= \frac{M(t)}{1 - \beta_1^t} \quad \hat{V}(t) = \frac{V(t)}{1 - \beta_2^t} \\ W(t+1) &= W(t) - \alpha * \frac{\hat{M}(t)}{\sqrt{\hat{V}(t) + \epsilon}} \end{aligned}$$

- bias correction terms (exponential decay)
- 일반적으로 m, v 가 0 으로 초기화되어 있기 때문에 훈련 초기에 크게 bias 되는데 이를 교정하기 위해 bias correction term 이 필요함.
- 훈련 iteration 이 진행 되면서 bias correction term 은 1 로 접근하게 됨.

Optimization algorithms

- **Adam** (Adaptive Moment Estimation)

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

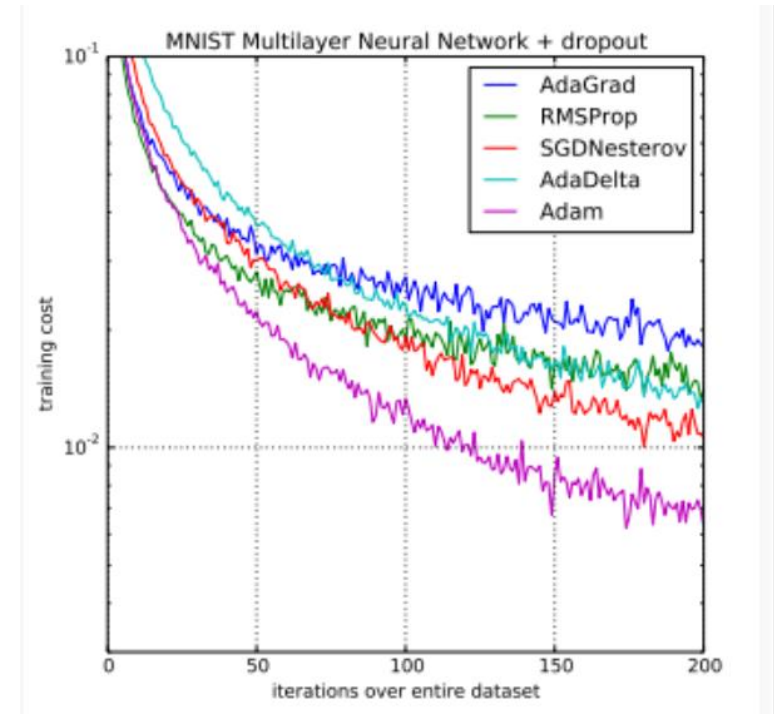
$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

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

end while

return θ_t (Resulting parameters)

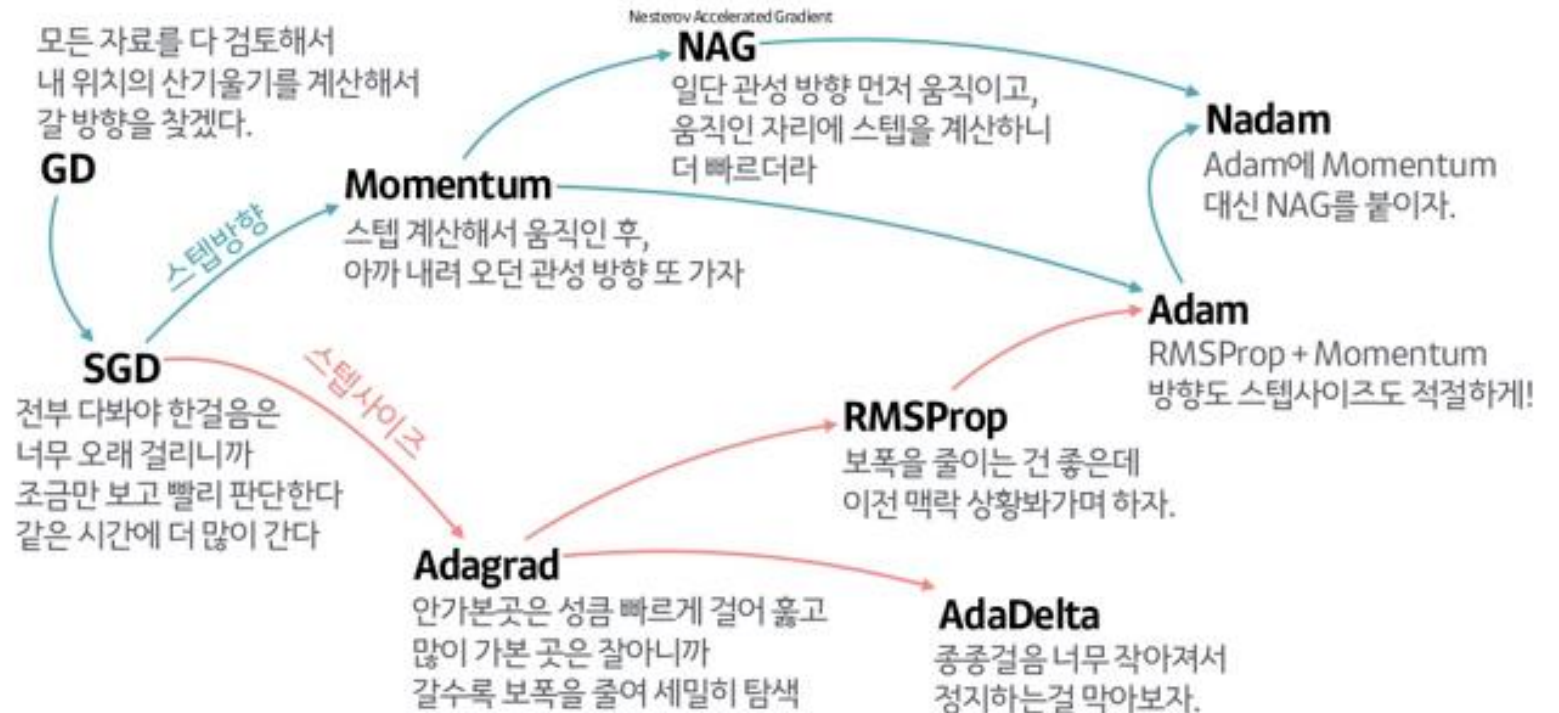


Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron

Taken from Adam: A Method for Stochastic Optimization, 2015.

Optimization algorithms

- Summary



- More:

- Adamax
- Nadam
- AMSGrad
- more ...

- Good animation

- <https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d>