최적화

서민재

목차

최적화란?

신경망 학습의 목적

손실 함수의 값을 가능한 한 낮추는 최적

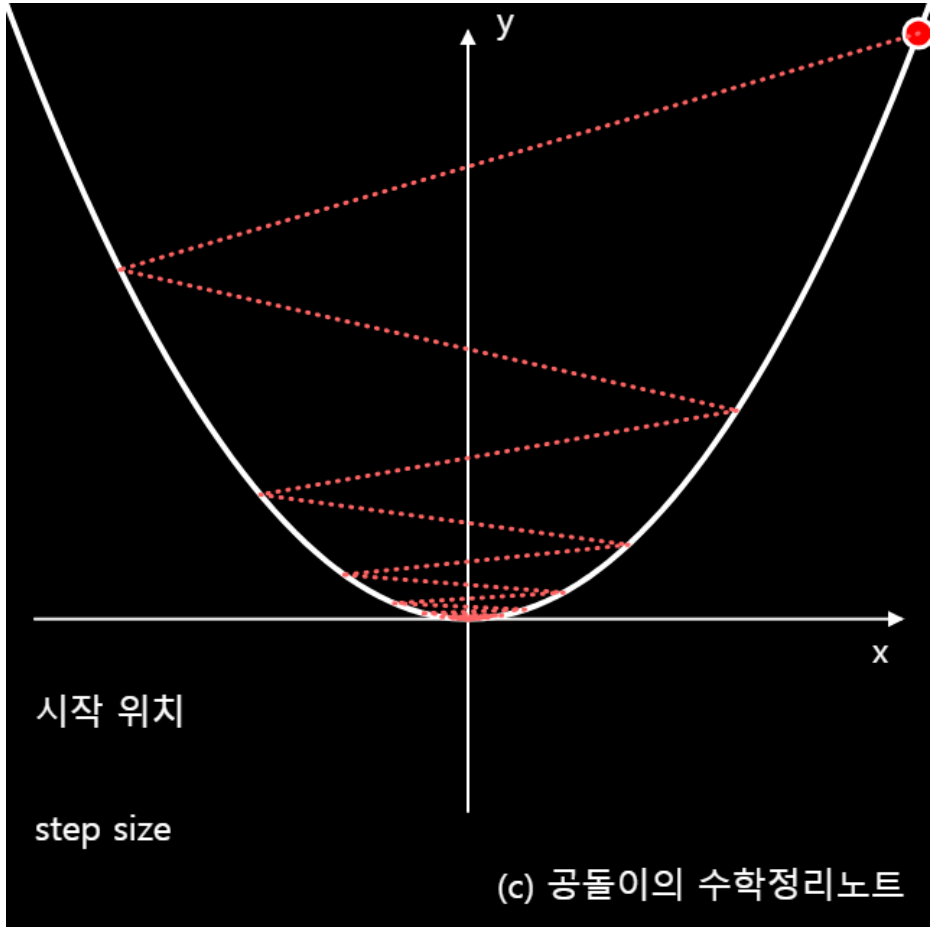
매개변수를 찾는 것

↓

해당 문제를 해결하는 것

= 최적화

경사하강법



함수 값이 낮아지는 방향으로 독립 변수 값을 변형시켜가면서 최종적으로는 최소 함수 값을 갖도록 하는 독립 변수 값을 찾는 방법

$$x_{i+1} = x_i - \alpha \frac{df}{dx}(x_i)$$

경사하강법

```
import tensorflow as tf

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x_data = [x_row[0] for x_row in data]
y_data = [y_row[0] for y_row in data]
```

```
a = tf.Variable(tf.random.uniform([1], 0, 10, dtype = tf.float64, seed = 0))
b = tf.Variable(tf.random.uniform([1], 0, 100, dtype = tf.float64, seed = 0))
```

```
y = a*x_data + b
```

```
rmse = tf.sqrt(tf.reduce_mean(tf.square(y - y_data)))
```

```
learning_rate = 0.1
```

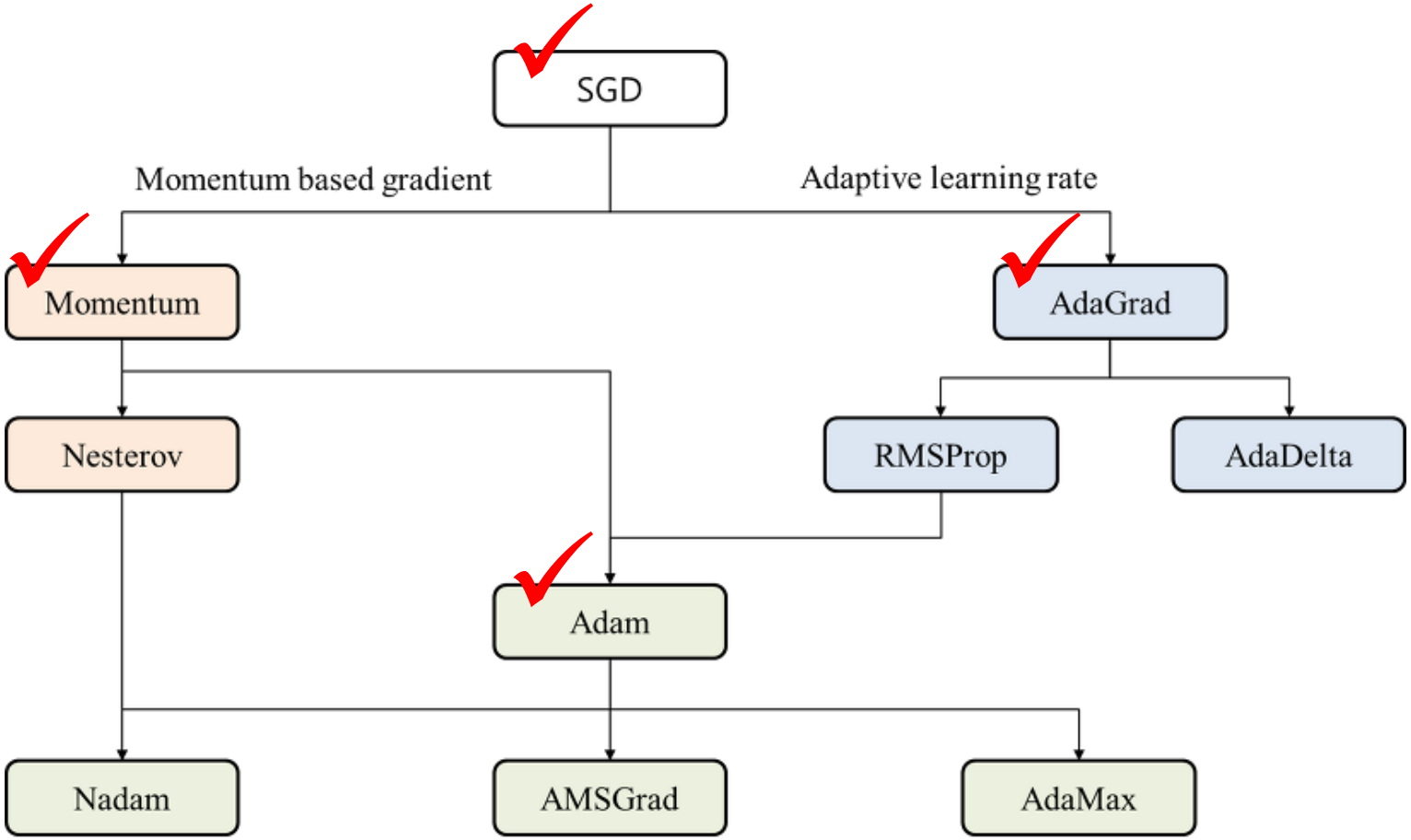
```
gradient_descent = tf.train.GradientDescentOptimizer(learning_rate).minimize(rmse)
```

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(5001):
        sess.run(gradient_descent)
        if step  % 100 == 0:
            print("Epoch: %.f, RMSE = %.04f, 기울기 a = %.4f, y 절편 b = %.4f" % (step, sess.run(rmse), sess.run(a), sess.run(b)))
```

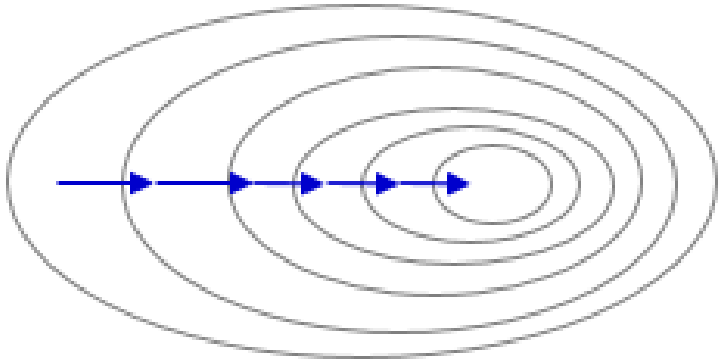
경사하강법

```
Epoch: 0, RMSE = 114.2489, 기울기 a = 7.5434, y 절편 b = 80.5910
Epoch: 100, RMSE = 30.2481, 기울기 a = -11.4110, y 절편 b = 74.0875
Epoch: 200, RMSE = 28.6267, 기울기 a = -10.7494, y 절편 b = 70.1156
Epoch: 300, RMSE = 27.0053, 기울기 a = -10.0840, y 절편 b = 66.1443
Epoch: 400, RMSE = 25.3839, 기울기 a = -9.4185, y 절편 b = 62.1731
Epoch: 500, RMSE = 23.7625, 기울기 a = -8.7530, y 절편 b = 58.2018
Epoch: 600, RMSE = 22.1412, 기울기 a = -8.0875, y 절편 b = 54.2305
Epoch: 700, RMSE = 20.5198, 기울기 a = -7.4221, y 절편 b = 50.2593
Epoch: 800, RMSE = 18.8984, 기울기 a = -6.7566, y 절편 b = 46.2880
Epoch: 900, RMSE = 17.2770, 기울기 a = -6.0911, y 절편 b = 42.3167
Epoch: 1000, RMSE = 15.6556, 기울기 a = -5.4256, y 절편 b = 38.3455
Epoch: 1100, RMSE = 14.0342, 기울기 a = -4.7602, y 절편 b = 34.3742
Epoch: 1200, RMSE = 12.4129, 기울기 a = -4.0947, y 절편 b = 30.4029
Epoch: 1300, RMSE = 10.7915, 기울기 a = -3.4292, y 절편 b = 26.4316
Epoch: 1400, RMSE = 9.1701, 기울기 a = -2.7637, y 절편 b = 22.4604
Epoch: 1500, RMSE = 7.5487, 기울기 a = -2.0983, y 절편 b = 18.4891
Epoch: 1600, RMSE = 5.9273, 기울기 a = -1.4328, y 절편 b = 14.5178
Epoch: 1700, RMSE = 4.3059, 기울기 a = -0.7673, y 절편 b = 10.5466
Epoch: 1800, RMSE = 2.6846, 기울기 a = -0.1018, y 절편 b = 6.5753
Epoch: 1900, RMSE = 1.0632, 기울기 a = 0.5636, y 절편 b = 2.6040
Epoch: 2000, RMSE = 1.5416, 기울기 a = 0.6112, y 절편 b = 0.6709
Epoch: 2100, RMSE = 1.5419, 기울기 a = 0.6851, y 절편 b = 0.2030
Epoch: 2200, RMSE = 1.5419, 기울기 a = 0.7117, y 절편 b = 0.0406
Epoch: 2300, RMSE = 1.5419, 기울기 a = 0.7211, y 절편 b = -0.0159
Epoch: 2400, RMSE = 1.5419, 기울기 a = 0.7244, y 절편 b = -0.0355
Epoch: 2500, RMSE = 1.5419, 기울기 a = 0.7256, y 절편 b = -0.0423
Epoch: 2600, RMSE = 1.5419, 기울기 a = 0.7259, y 절편 b = -0.0446
Epoch: 2700, RMSE = 1.5419, 기울기 a = 0.7261, y 절편 b = -0.0455
Epoch: 2800, RMSE = 1.5419, 기울기 a = 0.7261, y 절편 b = -0.0457
Epoch: 2900, RMSE = 1.5419, 기울기 a = 0.7262, y 절편 b = -0.0458
Epoch: 3000, RMSE = 1.5419, 기울기 a = 0.7262, y 절편 b = -0.0459
```

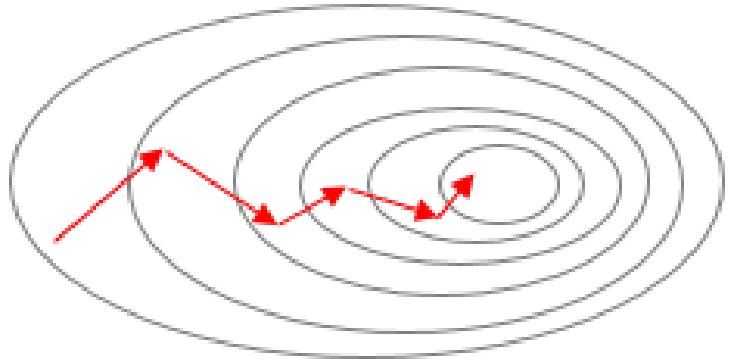
최적화 알고리즘


SGD
Momentum based gradient
Adaptive learning rate
Momentum
AdaGrad
Nesterov
RMSProp
AdaDelta
Adam
Nadam
AMSGrad
AdaMax

최적화 알고리즘_ SGD



경사 하강법



SGD

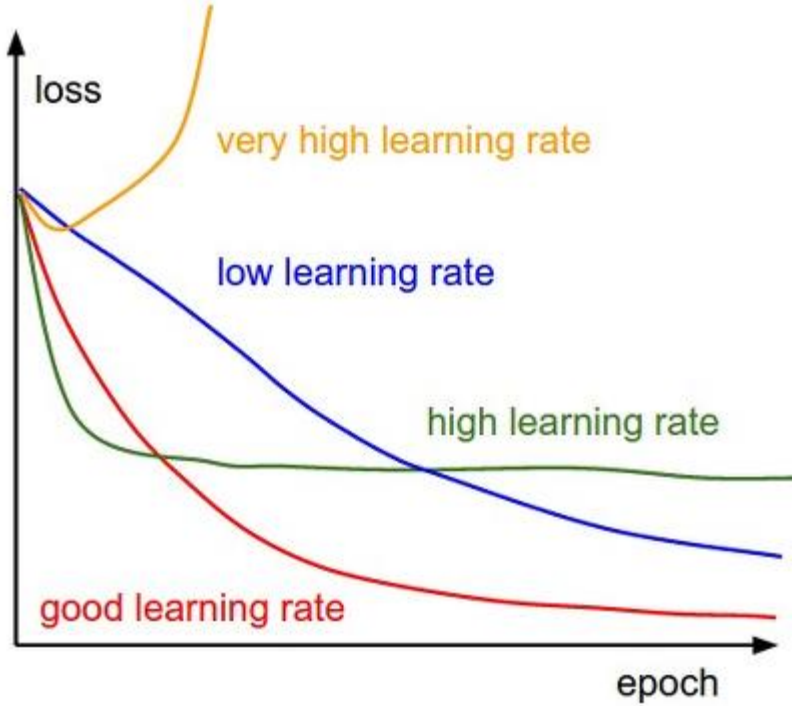
최적화 알고리즘 _ SGD

$$W(t+1) = W(t) - \alpha \frac{\partial L}{\partial W}$$

```python
class SGD:
    def __init__(self, lr = 0.01):
        self.lr = lr

    def update(self, params, grads):
        for key in params.key():
            params[key] -= self.lr * grads[key]
```

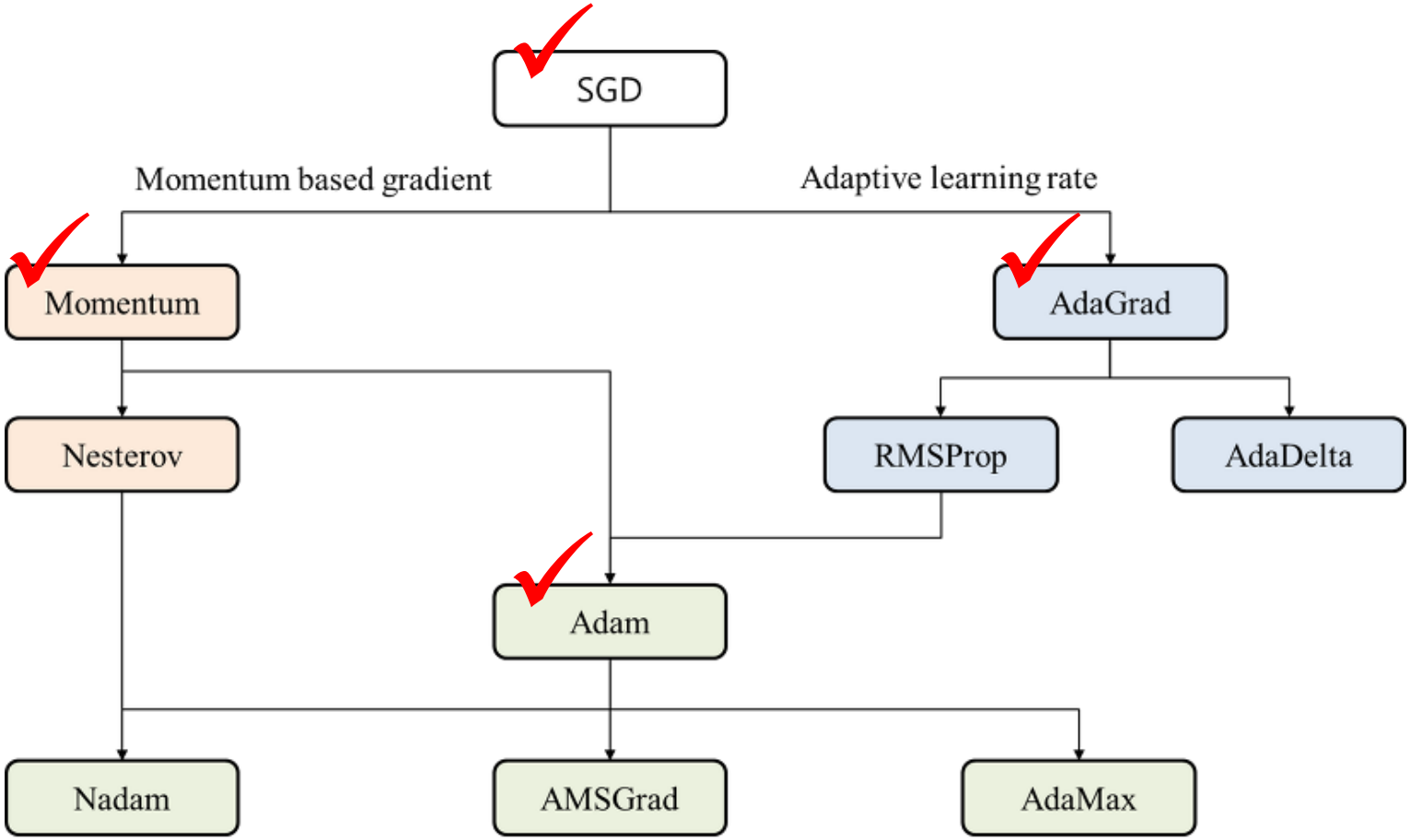
최적화 알고리즘 _ SGD

탐색 방향



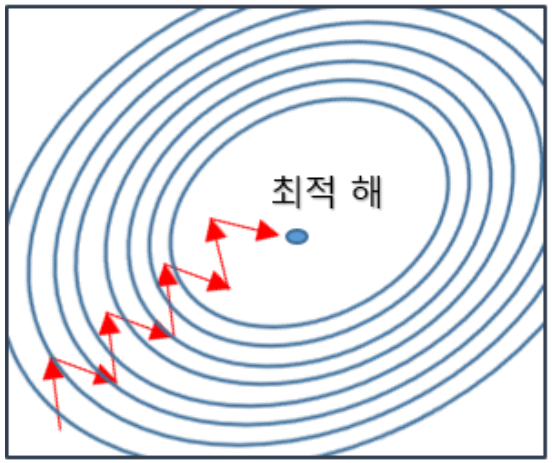
탐색 거리

최적화 알고리즘

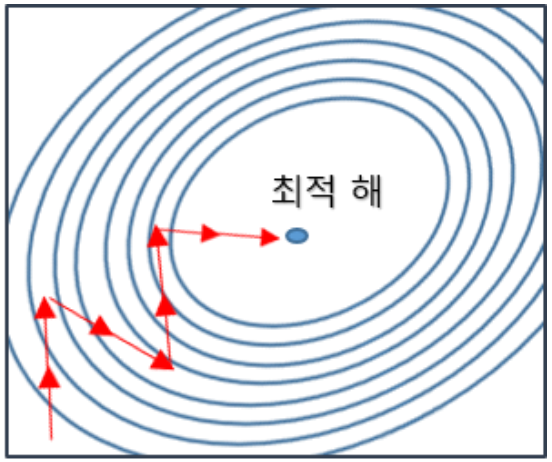

SGD
Momentum based gradient
Adaptive learning rate
Momentum
AdaGrad
Nesterov
RMSProp
AdaDelta
Adam
Nadam
AMSGrad
AdaMax

최적화 알고리즘 _ 모멘텀

Momentum
= 관성



확률적 경사 하강법



모멘텀

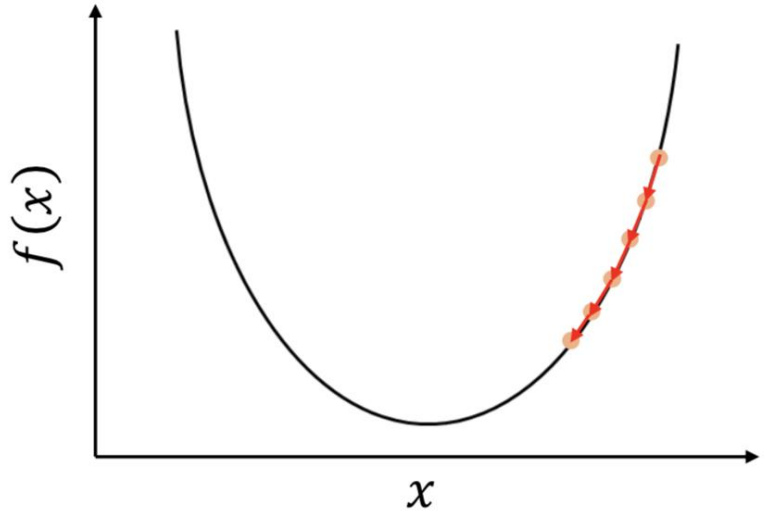
최적화 알고리즘 _ 모멘텀

$$V(t) = m*V(t-1) - \alpha \frac{\partial L}{\partial W}$$

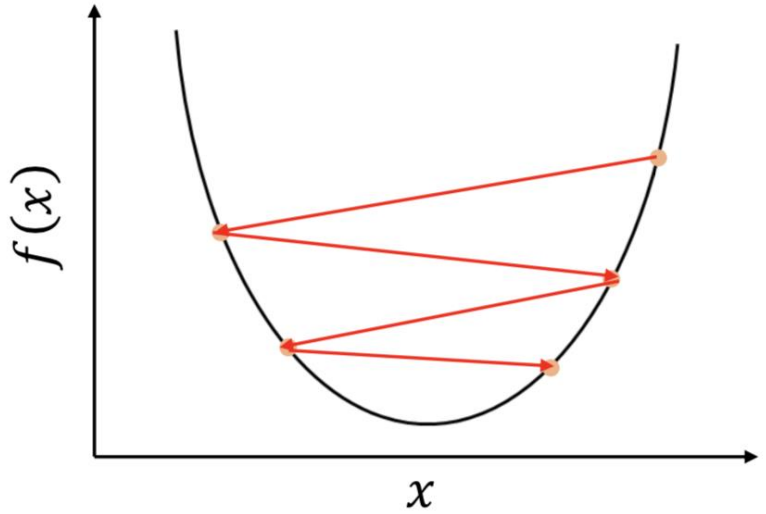
$$W(t+1) = W(t) + V(t)$$

```
class Momentum:
    def __init__(self, lr=0.01, momentum = 0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum*slef.v[key] - self.lr*grads[key]
            params[key] += self.v[key]
```

최적화 알고리즘 _ 아다그라드



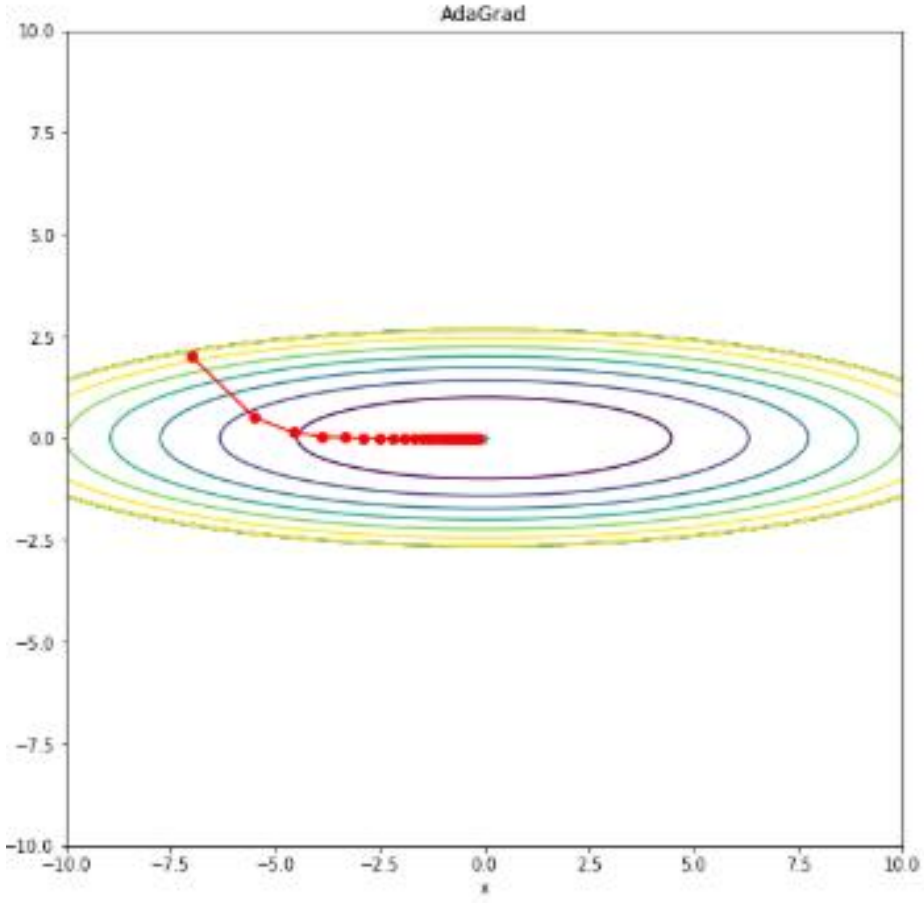
α가 너무 작은 경우



α가 너무 큰 경우

학습률(Learning Rate) α에 비례하여 이동한다. 적절한 학습률을 선택하는 것은 매우 중요하다.

최적화 알고리즘 _ 아다그라드



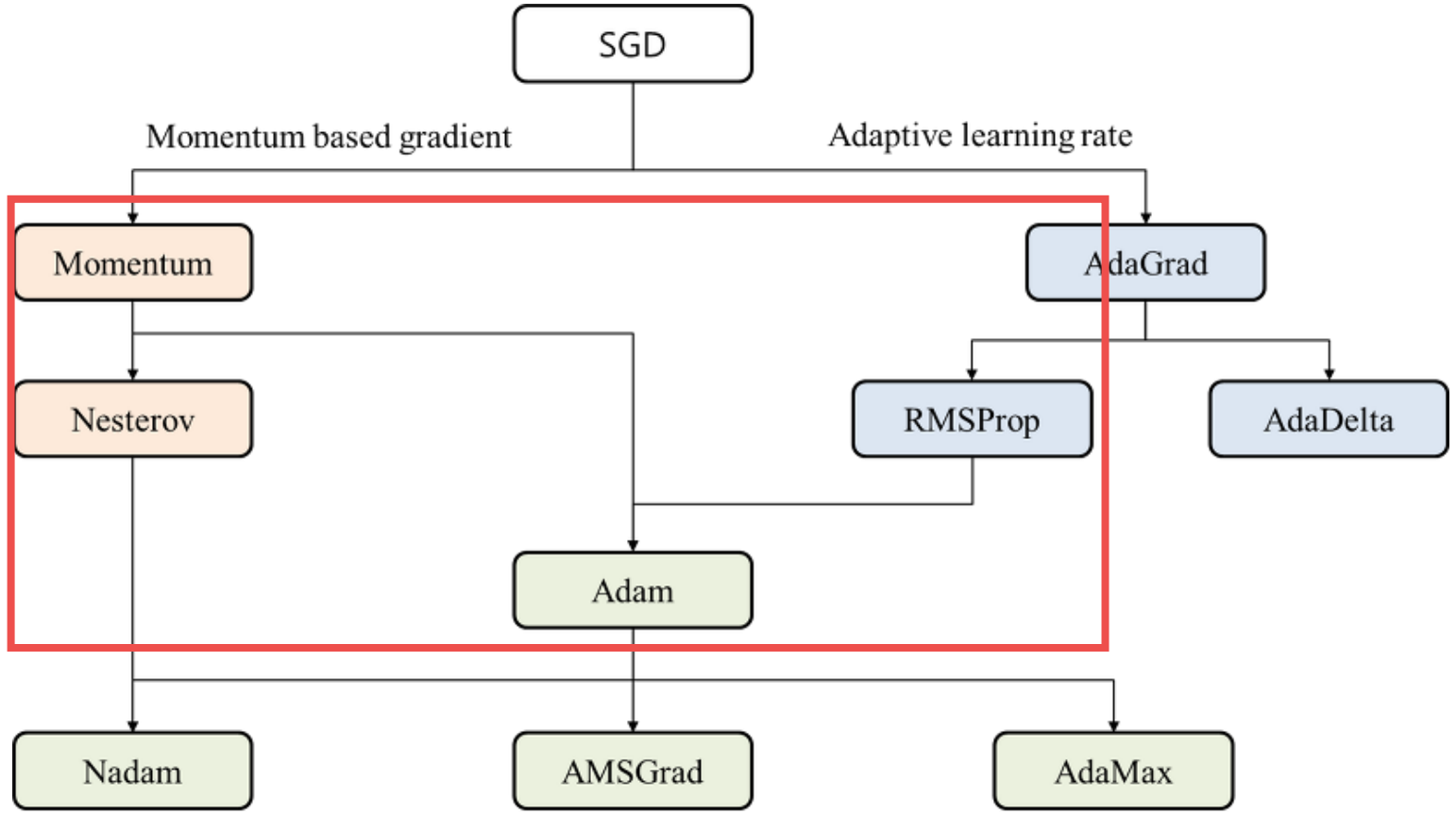
최적화 알고리즘 _ 아다그라드

$$h(t) = h(t-1) + \left(\frac{\partial L}{\partial W}\right)^2$$

$$W(t+1) = W(t) + \alpha \frac{1}{\sqrt{h(t)}} \frac{\partial L}{\partial W}$$

```
class AdaGrad:
    def __init__(self, lr = 0.01):
        self.lr = lr
        self.h = None

    def update(self, params, grads):
        if self.h is None:
            self.h = {}
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

            for key in params.keys():
                self.h[key] += grads[key] * grads[key]
                params[key] -= slef.lr * grads[key] / np.sqrt(self.h[key] + 1e-7)
```

최적화 알고리즘 _ 아담


SGD
Momentum based gradient
Adaptive learning rate
Momentum
AdaGrad
Nesterov
RMSProp
AdaDelta
Adam
Nadam
AMSGrad
AdaMax

최적화 알고리즘 _ 알엠에스프롭

이동 지수의 평균

$$x_k = \alpha p_k + (1-\alpha)x_{k-1} \qquad where \qquad \alpha = \frac{2}{N+1}$$

$$\begin{aligned}
x_k &= \alpha p_k + (1-\alpha)x_{k-1} \\
&= \alpha p_k + (1-\alpha)\{\alpha p_{k-1} + (1-\alpha)x_{k-2}\} \\
&= \alpha p_k + \alpha(1-\alpha)p_{k-1} + (1-\alpha)^2 x_{k-2} \\
&= \alpha p_k + \alpha(1-\alpha)p_{k-1} + (1-\alpha)^2\{\alpha p_{k-2} + (1-\alpha)x_{k-3}\} \\
&= \alpha p_k + \alpha(1-\alpha)p_{k-1} + \alpha(1-\alpha)^2 p_{k-2} + (1-\alpha)^3 x_{k-3} \\
&= \dots \\
&= \alpha\left(p_k + (1-\alpha)p_{k-1} + (1-\alpha)^2 p_{k-2} + \dots + (1-\alpha)^{N-1} p_{k-N+1}\right) + (1-\alpha)^N x_{k-N} \\
&= \alpha\sum_{i=0}^{N-1}(1-\alpha)^i p_{k-i} + (1-\alpha)^N x_{k-N}
\end{aligned}$$

Rmsprop

$$h(t) = \gamma h(t-1) + (1-\gamma)\left(\frac{\partial L}{\partial W}\right)^2$$

최적화 알고리즘 _ 아담

$$V(t) = \gamma_1 V(t-1) + (1-\gamma_1)\alpha \frac{\partial L}{\partial W}$$

$$h(t) = \gamma_2 h(t-1) + (1-\gamma_2)\left(\frac{\partial L}{\partial W}\right)^2$$

$$\hat{V}(t) = \frac{V(t)}{1-\gamma_1^t}, \quad \hat{h}(t) = \frac{h(t)}{1-\gamma_2^t}$$

$$W(t+1) = W(t) - \alpha \frac{\hat{h}(t)}{\sqrt{\hat{v}(t)}}$$

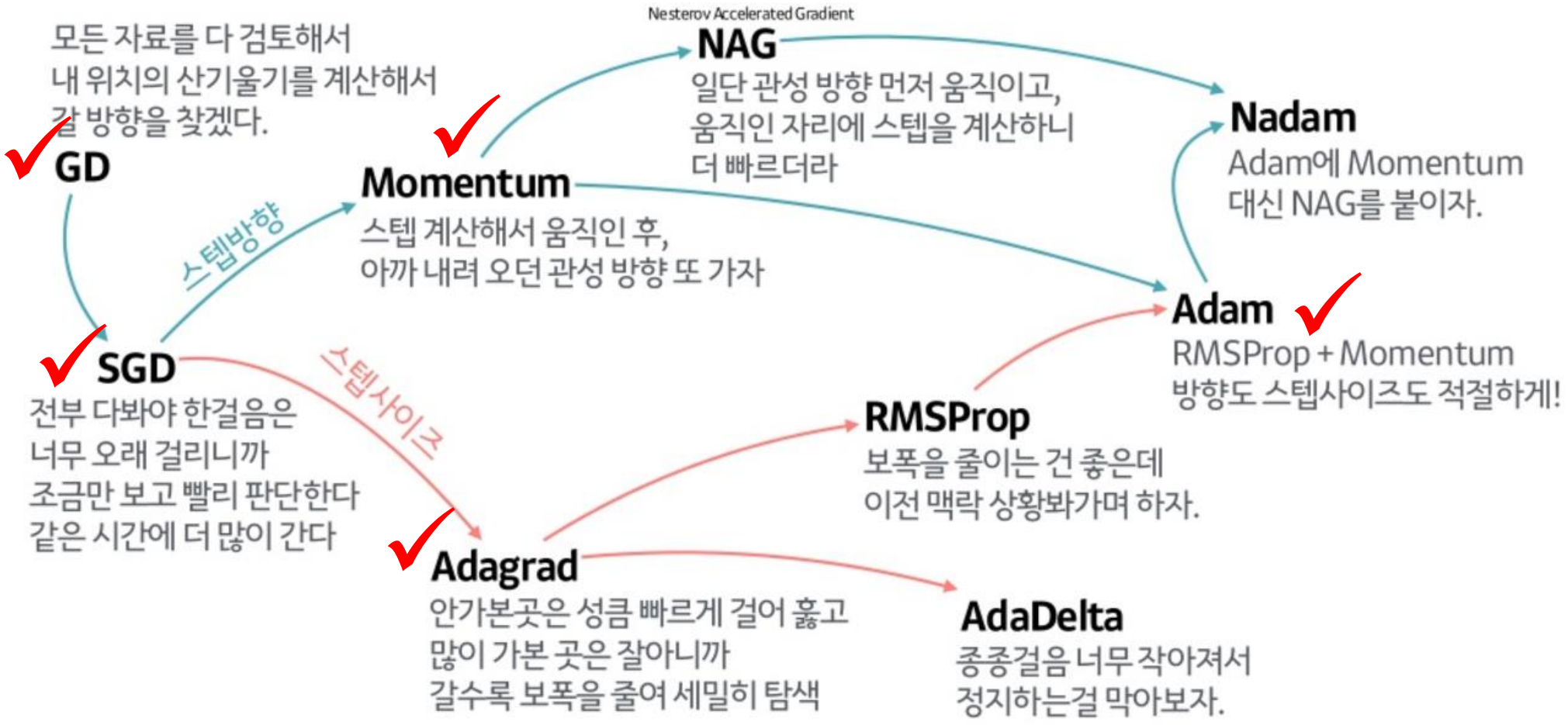
최적화 알고리즘 _ 아담

```python
class Adam:

    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)

        for key in params.keys():
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])

            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

$$\hat{V}(t) = \frac{V(t)}{1-\gamma_1^t}, \quad \hat{h}(t) = \frac{h(t)}{1-\gamma_2^t}$$

$$W(t+1) = W(t) - \alpha \frac{\hat{h}(t)}{\sqrt{\hat{v}(t)}}$$

정리


모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
스텝사이즈
Adagrad
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색
RMSProp
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
AdaDelta
종종걸음 너무 작아져서
정지하는걸 막아보자.

Q&A

Thank You