

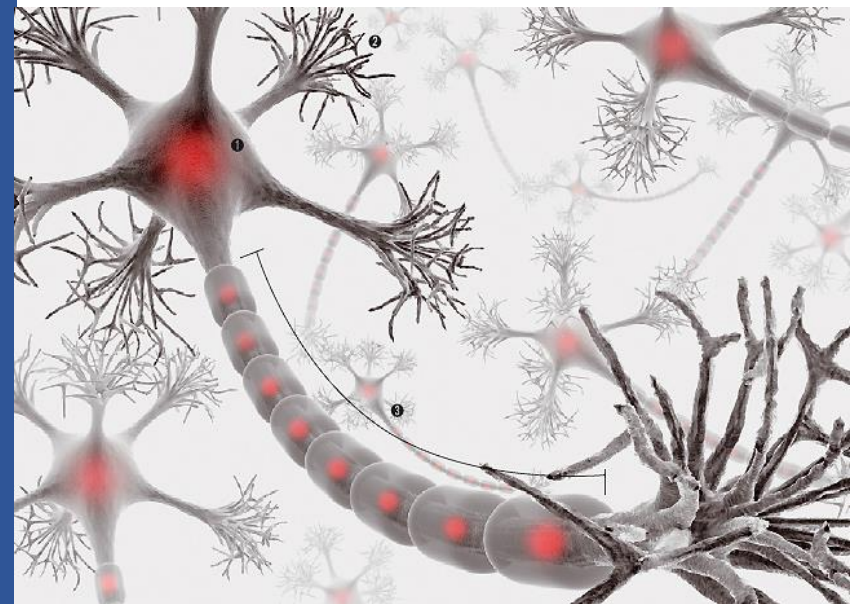
# 최적화 알고리즘

## 학습 목표

- 신경망을 학습시키기 위해 필요한 최적화 알고리즘을 이해한다.

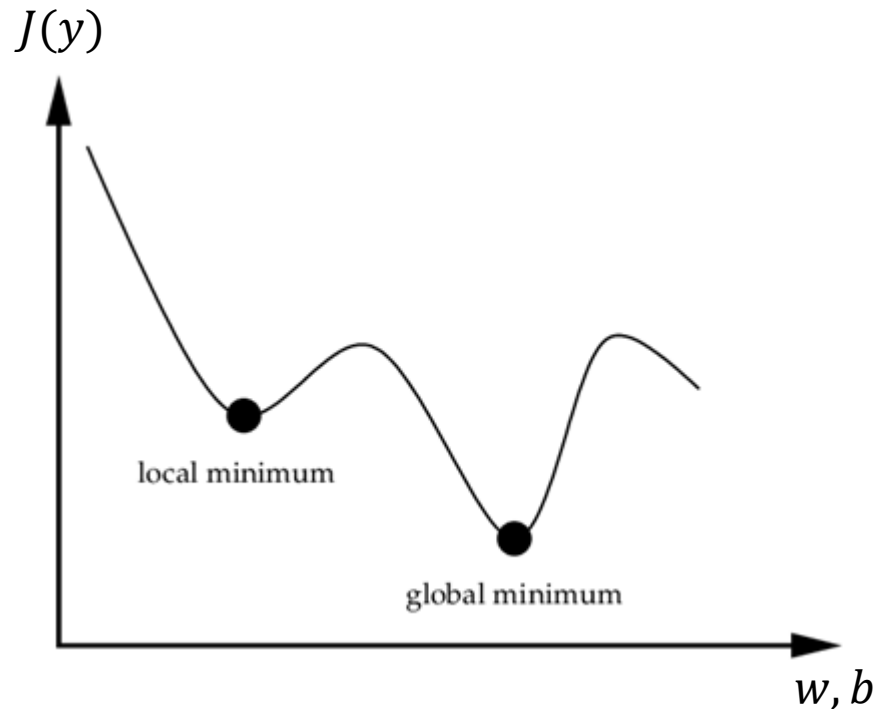
## 주요 내용

1. SGD의 문제점
2. SGD + Momentum
3. AdaGrad
4. RMSProp
5. Adam



# Loss Function을 최소화 하려면?

## Loss Function Minimization



## 최적화 알고리즘

### 1차 미분

- Gradient Descent
- Variants of Gradient Descent : SGD, Adagrad, Momentum, RMS prob, Adam

Deep Learning에서 주로 사용하는 방법

### 1.5차 미분

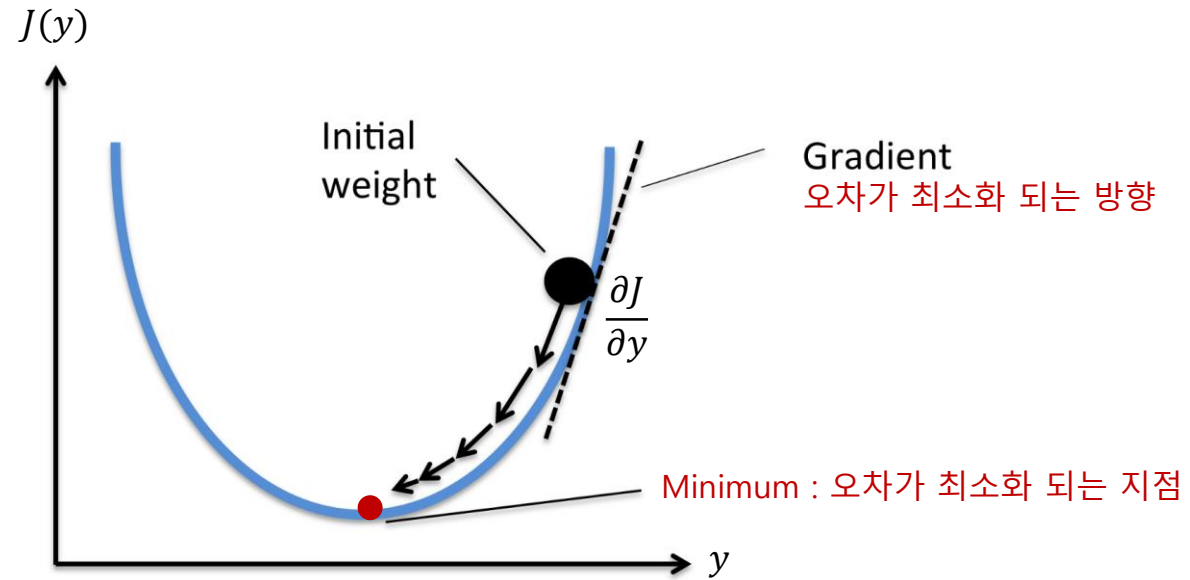
- Quasi-Newton Method
- Conjugate Gradient Descent
- Levenberg-Marquardt Method

### 2차 미분

- Newton Method
- Interior Point Method

# Gradient Descent

## Gradient Descent

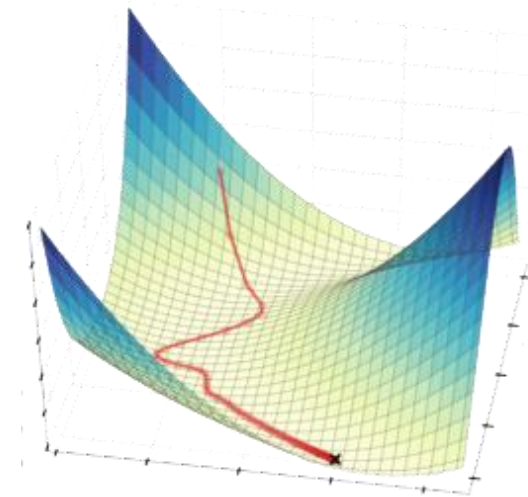


## Parameter Update

$$y^+ = y - \alpha \frac{\partial J}{\partial y}$$

Step Size  $\alpha$  Gradient  $\frac{\partial J}{\partial y}$

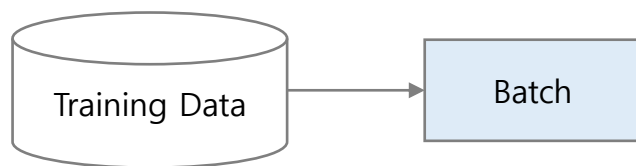
## 3D View



# 훈련 단위

## Batch

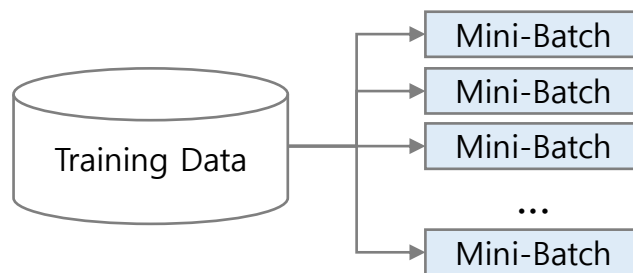
전체 훈련 데이터를 하나의 배치로  
만들어 훈련



훈련 집합이 너무 크면 불가능!

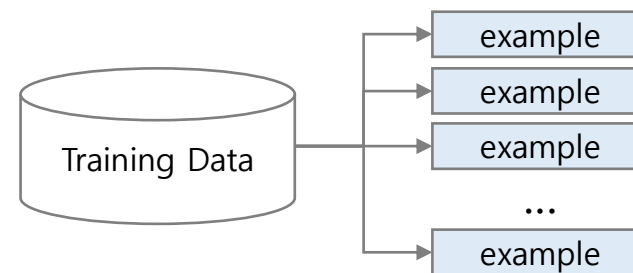
## Mini-Batch

n개 샘플을 묶은 미니배치 단위로 훈련



## Stochastic

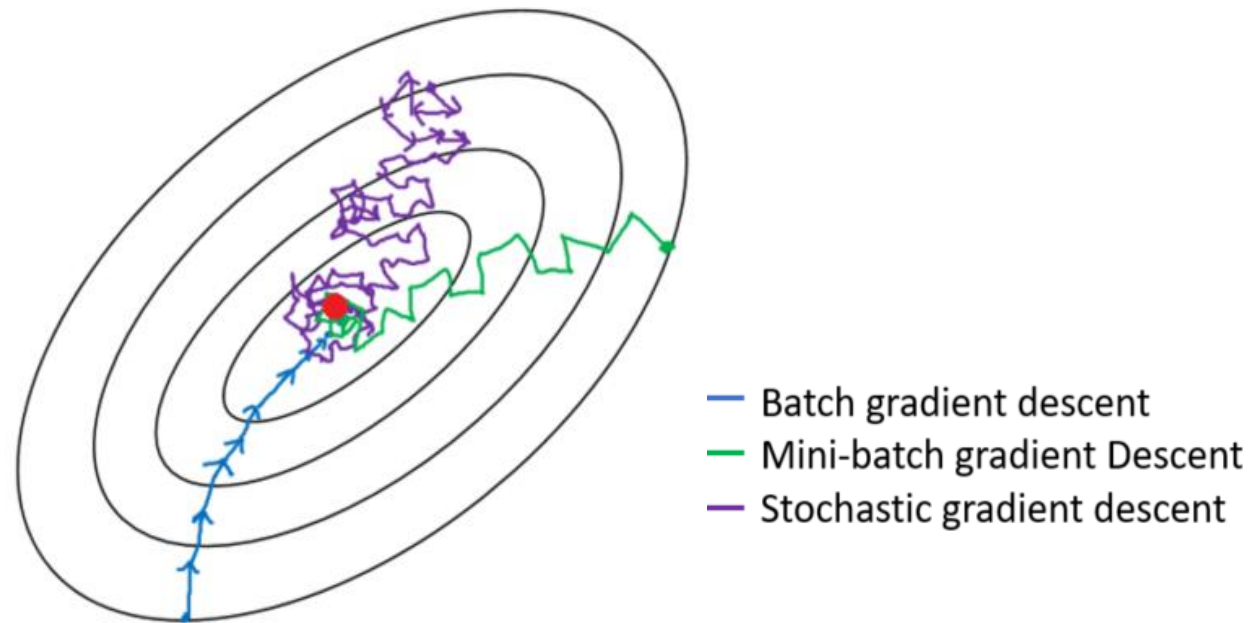
각 example 단위로 훈련



Too Noisy!

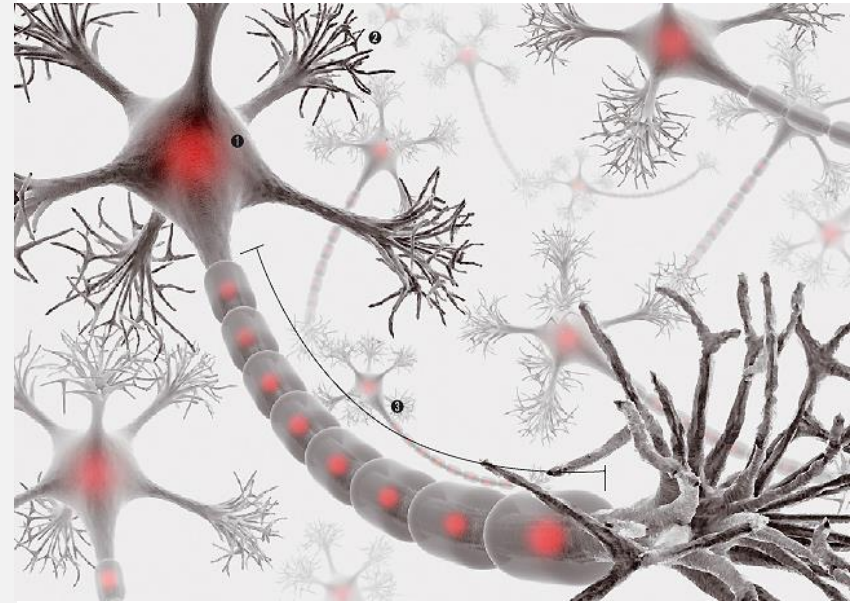
# 훈련 단위

## Gradient Descent Trajectory



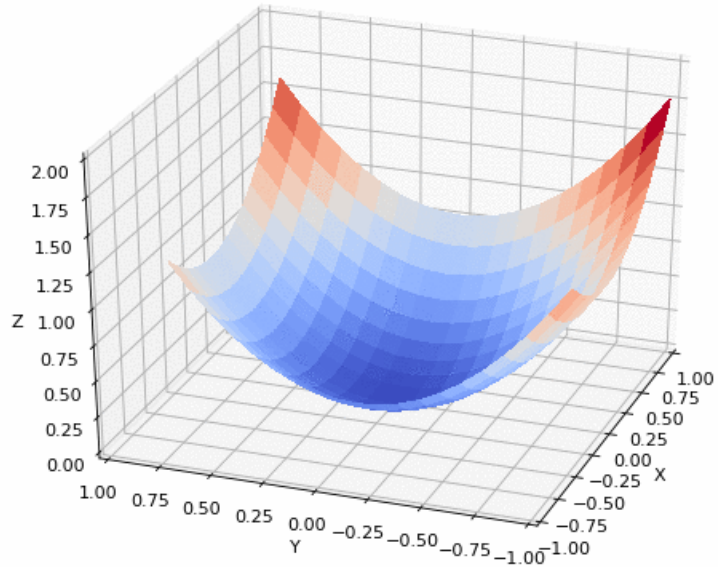
[그림] <https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>

# 1 SGD의 문제점

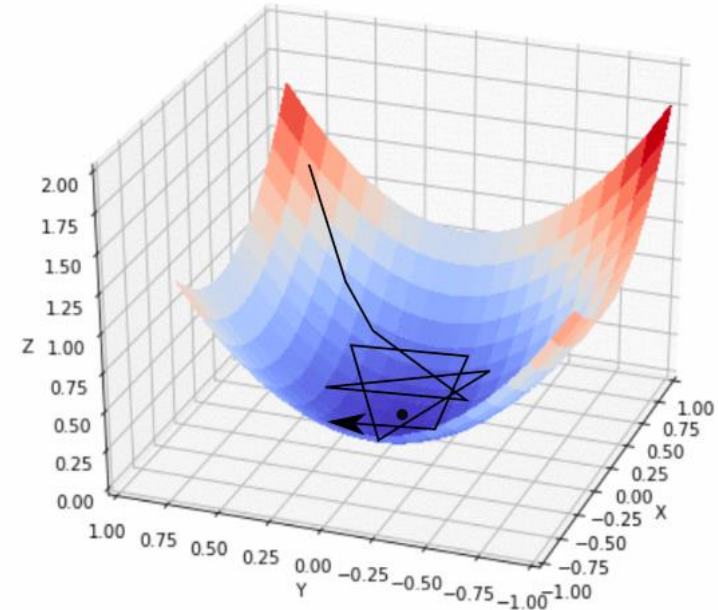


Issue 학습률이 자동으로 조정되지 않는다.

Gradient Descent 수렴 경로



학습률이 큰 경우

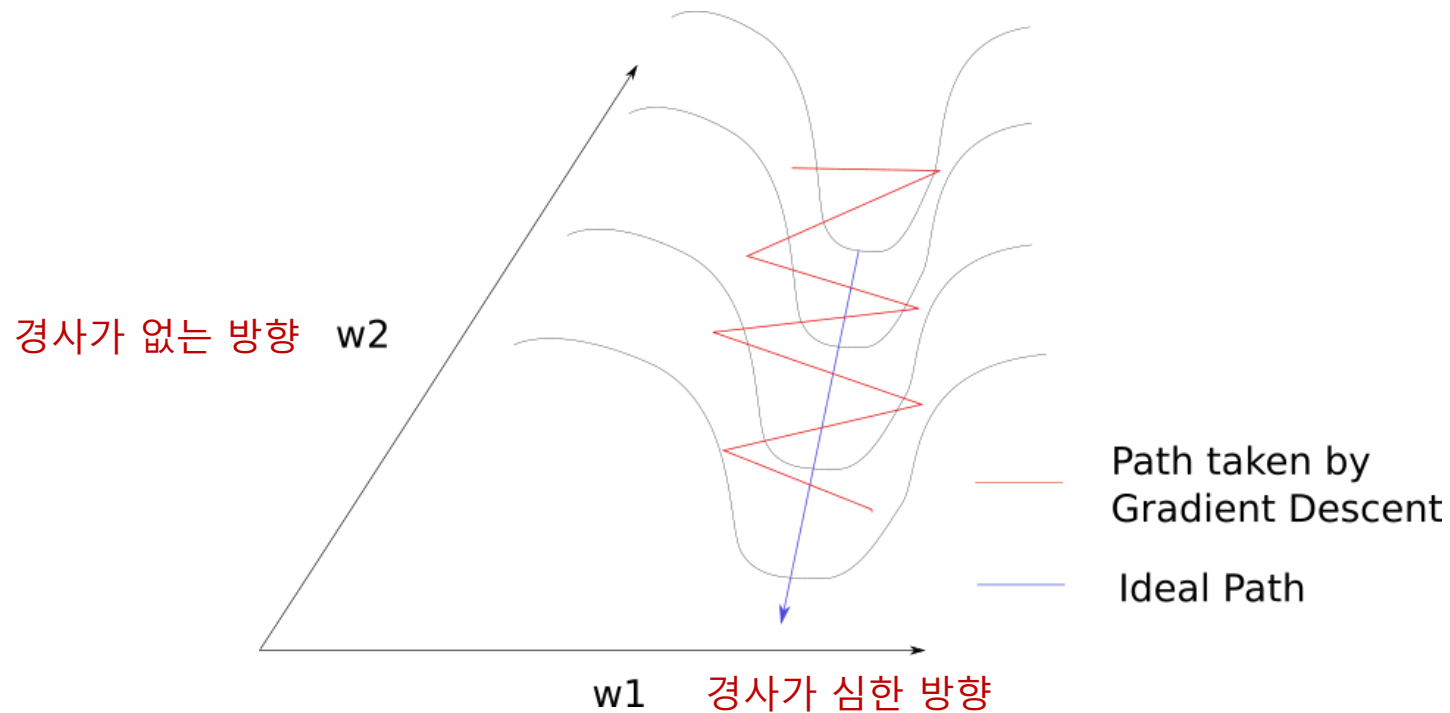


최소점 부근에서 수렴하지 못하고 진동

<https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/>

# Issue Ill-Conditioning 상태에서는 잘 수렴되지 않는다.

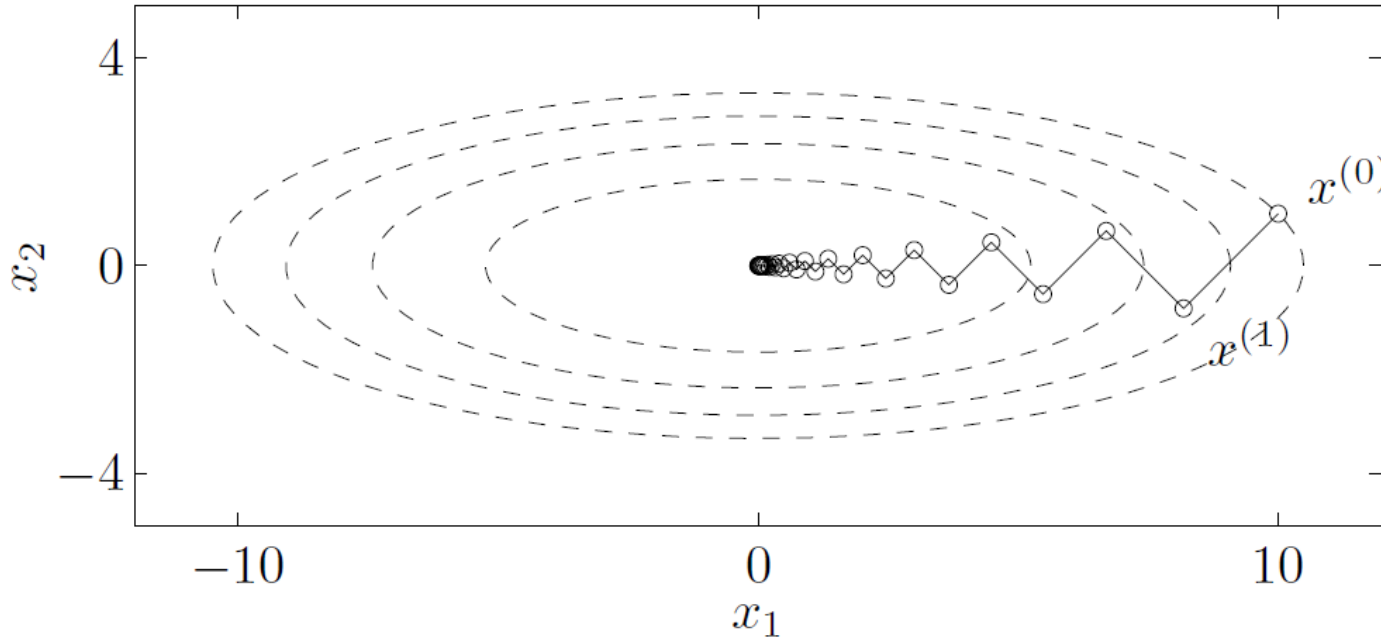
Loss 함수의 곡면이 좁은 계곡 모양인 경우?



진동하면서 학습이 중단



# Issue Ill-Conditioning 상태에서는 잘 수렴되지 않는다.



[그림] Convex Optimization, Stephen Boyd, Lieven Vandenbergh

$$f(x) = \frac{1}{2} (x_1^2 + 10x_2^2)$$

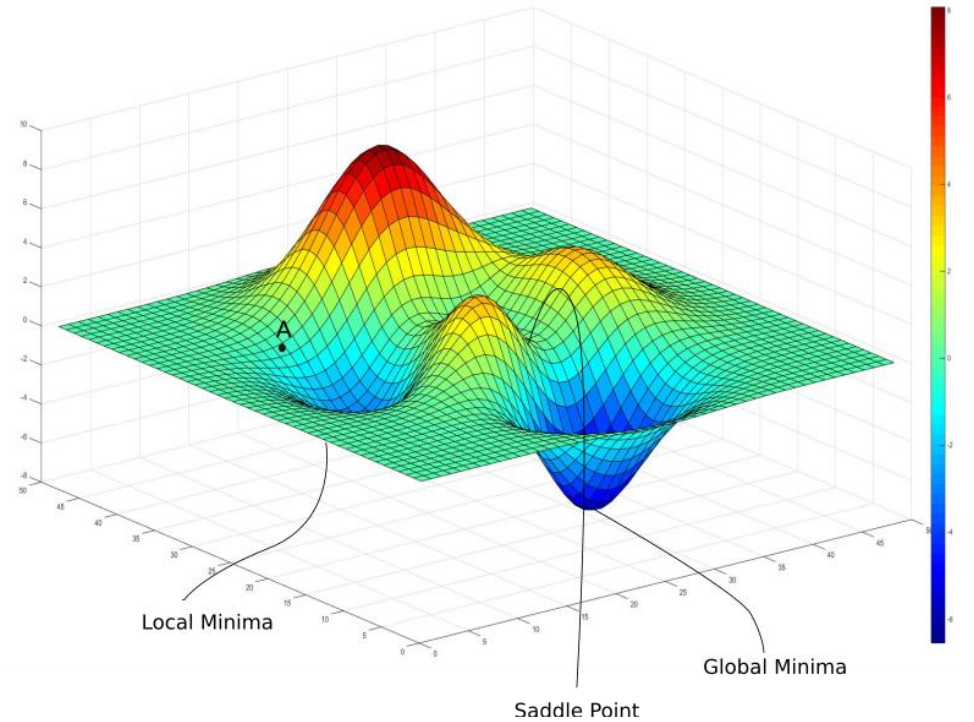
condition number : 10

## Condition number란?

- 타원에서의 장축과 단축의 비율
- Hessian 행렬에서 가장 큰 singular value와 가장 작은 singular value의 비율

# Issue 임계점에서 탈출하지 못한다.

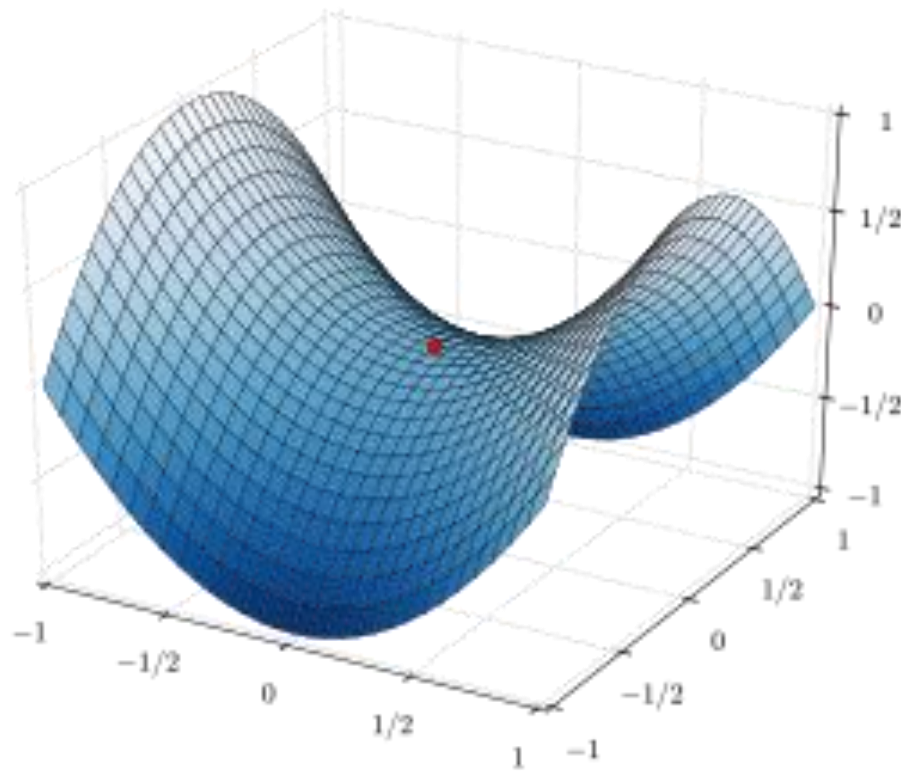
임계점 (Critical Point)에서 학습이 중단



<https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/>

Issue 임계점에서 탈출하지 못한다.

### Saddle Point

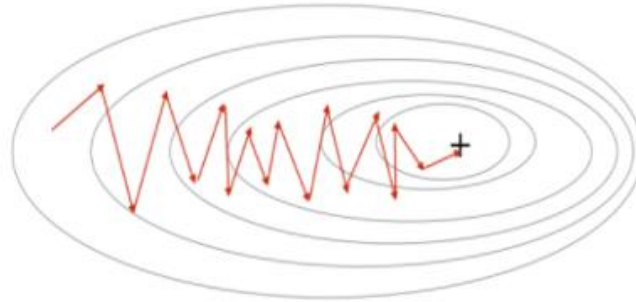


어떤 차원으로는 Maximum이고  
어떤 차원으로는 Minimum인 지점

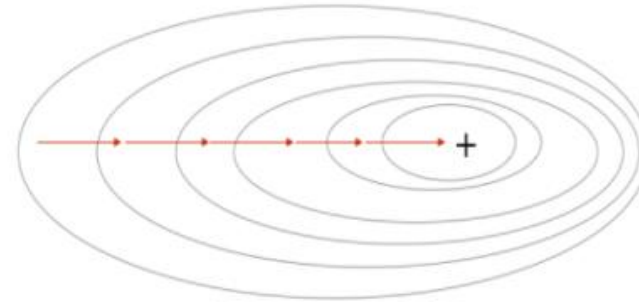
**차원이 높아질수록 Saddle Point가 많아진다!**

Issue 수렴 경로가 많이 왔다 갔다 한다.

Stochastic Gradient Descent



Gradient Descent



SGD vs. GD의 비교 동영상 : <https://youtu.be/6a5Nn49MsYY?t=38>

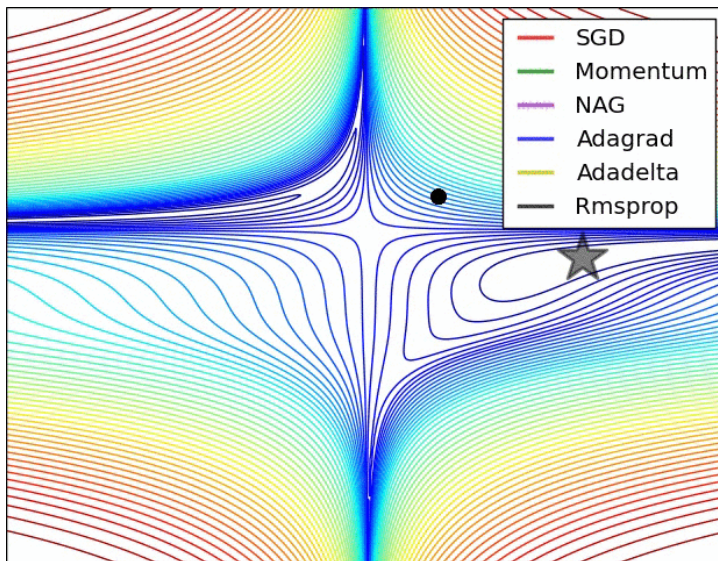
## Issue Summary

### Stochastic Gradient Descent는

- 경사에 따라 학습률이 자동으로 조정되지 않는다.
- Ill-Conditioning 상태에서는 잘 수렴되지 않는다.
- 임계점에서 탈출하지 못한다.
- 수렴 경로가 많이 왔다 갔다 한다.

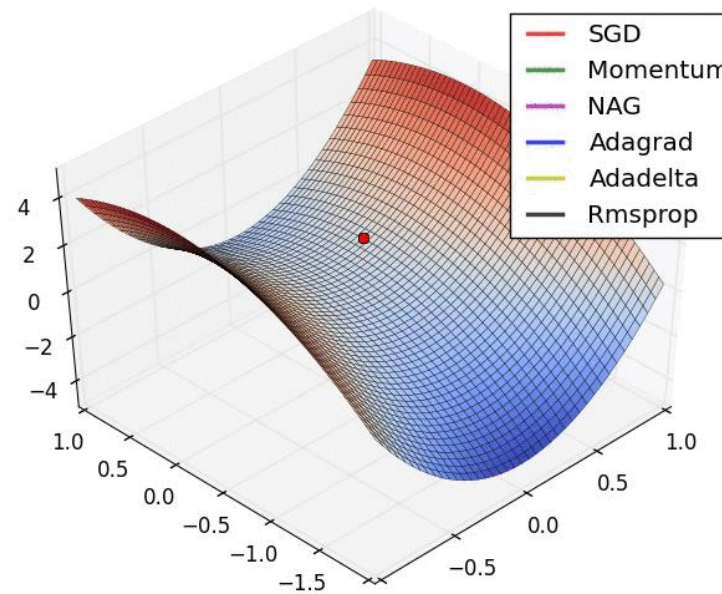
# Variants of Gradient Descent

Beale's Function

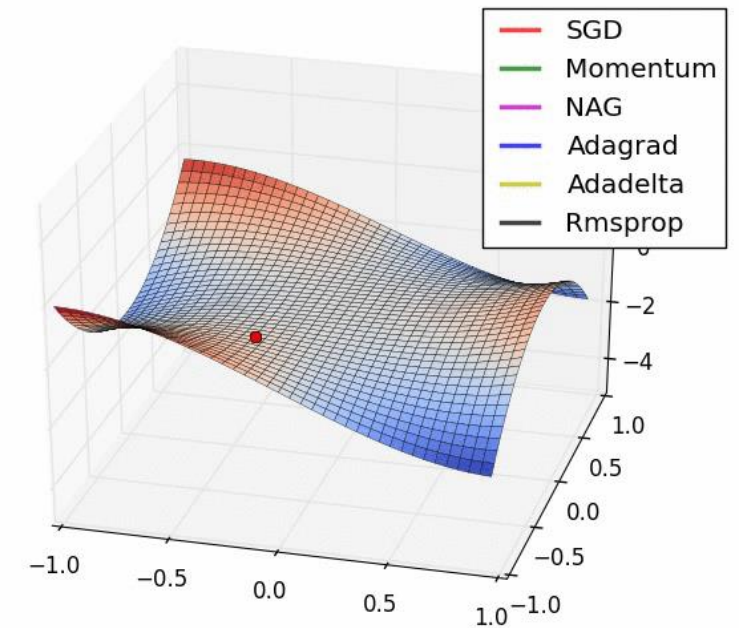


NAG : Nesterov accelerated gradient

Long Valley



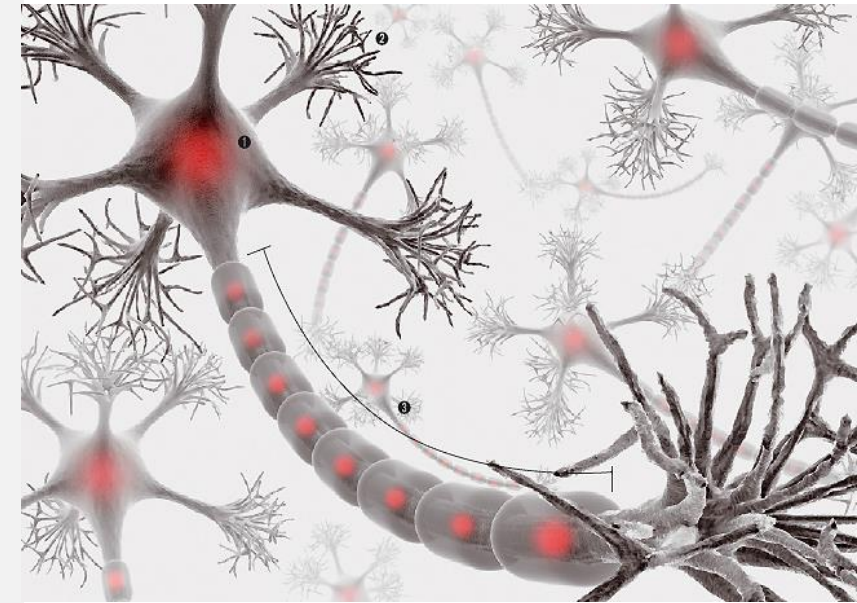
Saddle Point



An overview of gradient descent optimization algorithms, Sebastian Ruder



## 2 SGD + Momentum



# SGD + Momentum

## SGD + Momentum

이전 단계의  
Velocity

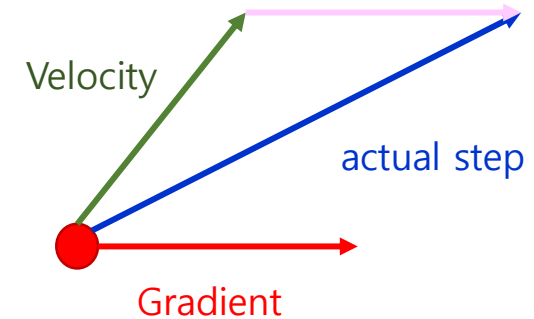
현재 Gradient

$$\mathbf{v}_{t+1} = \rho \mathbf{v}_t + \nabla f(x_t)$$

Velocity = 누적 Gradient

- $\rho$  : “friction” 마찰 계수로 0.9이나 0.99를 사용

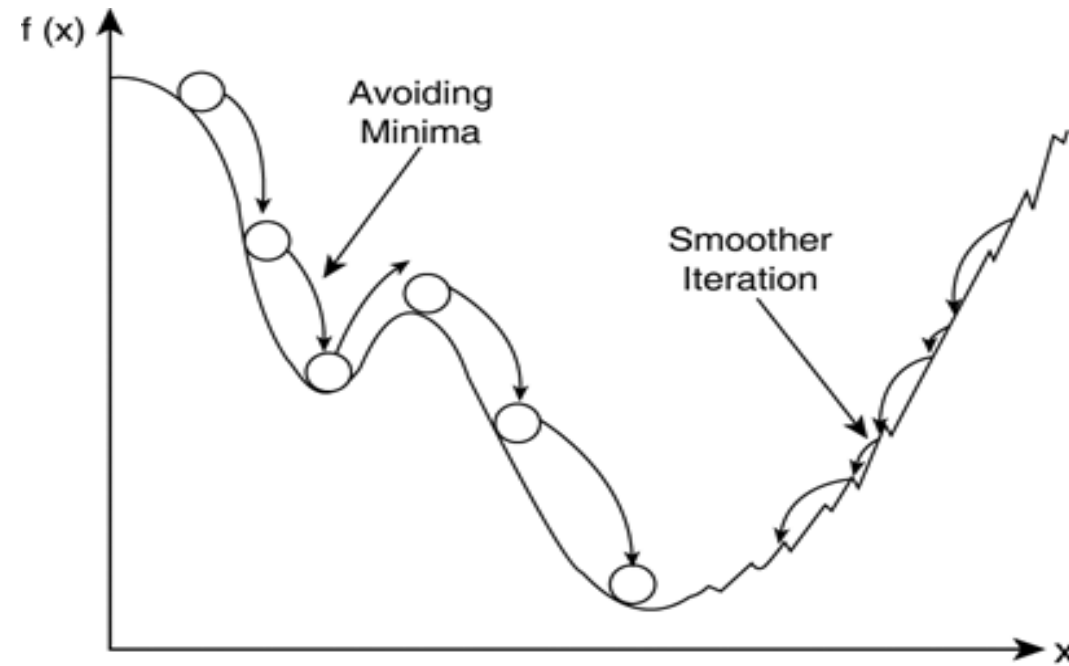
$$x_{t+1} = x_t - \alpha \mathbf{v}_{t+1}$$





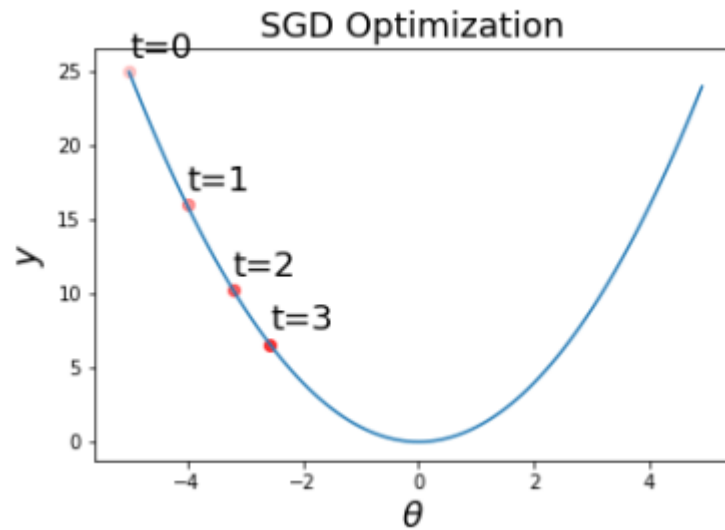
# 장점

## Local Minima & Saddle Point

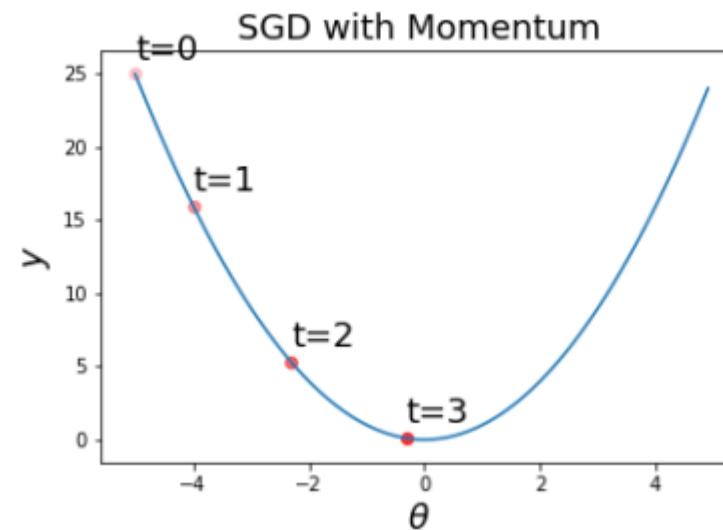


# 장점

## SGD



## SGD+Momentum

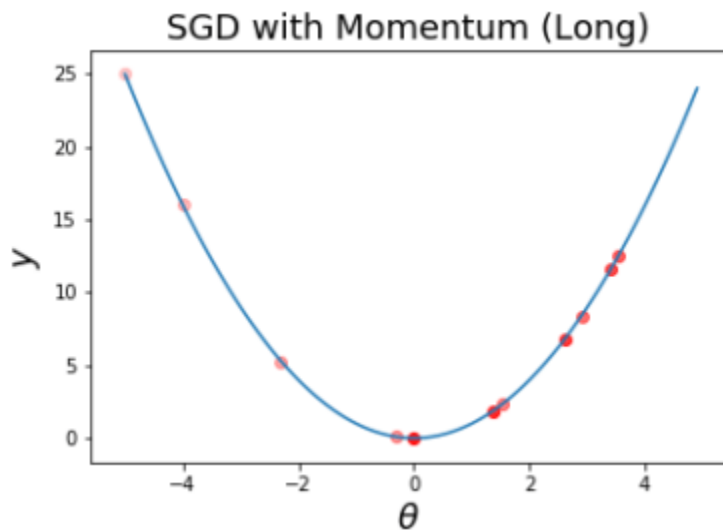


- + Gets to the optimal quicker

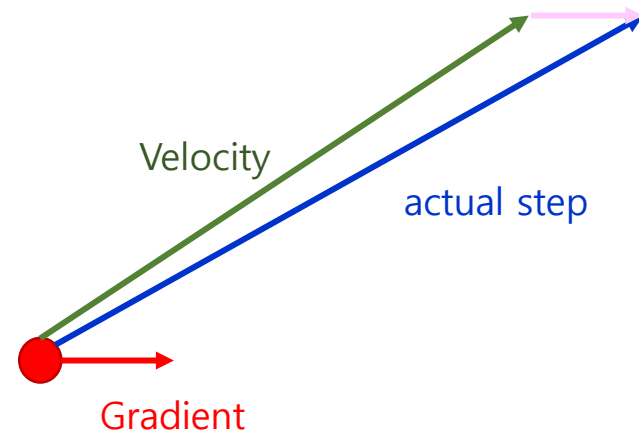
<http://www.thushv.com/deep-learning/a-practical-guide-to-understanding-stochastic-optimization-methods-workhorse-of-machine-learning/>

# 단점

## SGD+Momentum



- 최소점에 도달한 이후에도 Overshooting 될 수 있음
- 즉, 현재 Gradient가 작더라도 Velocity가 크면 최적화가 계속 진행됨



- Can overshoot if not careful with the learning rate

# 코드

## SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

## SGD+Momentum

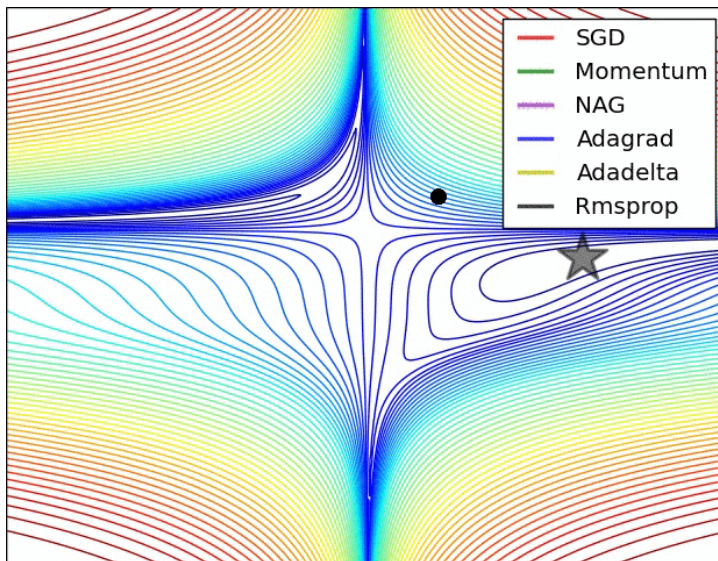
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

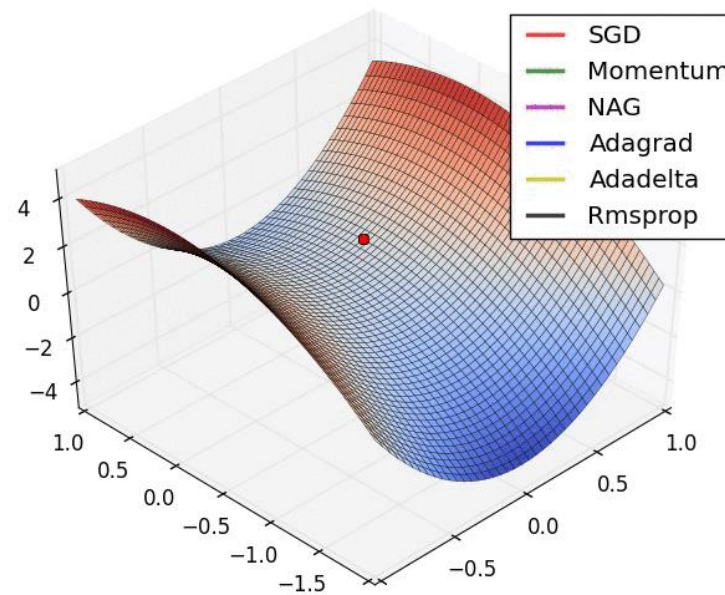
# Variants of Gradient Descent

Beale's Function

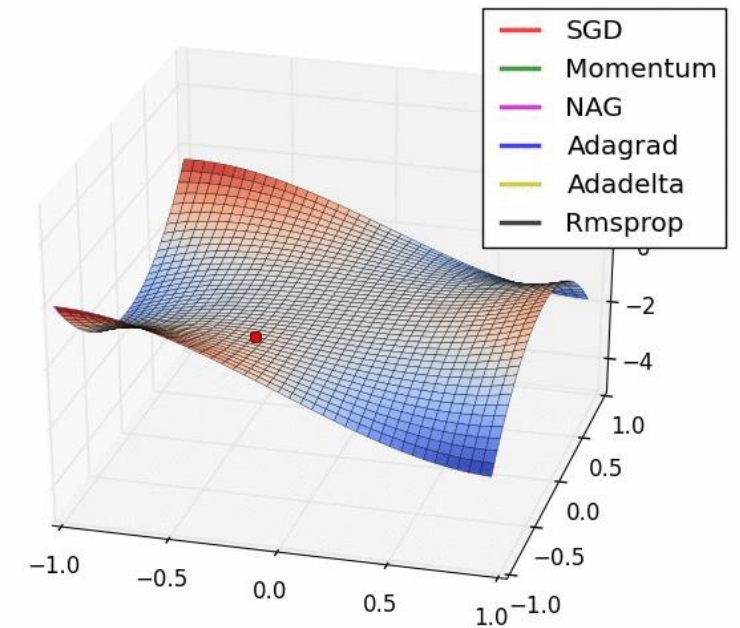


NAG : Nesterov accelerated gradient

Long Valley

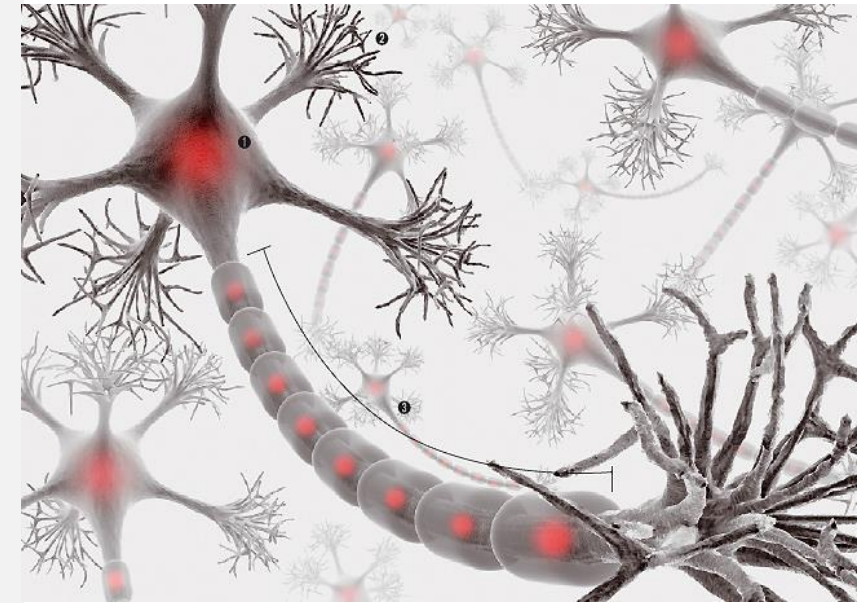


Saddle Point



An overview of gradient descent optimization algorithms, Sebastian Ruder

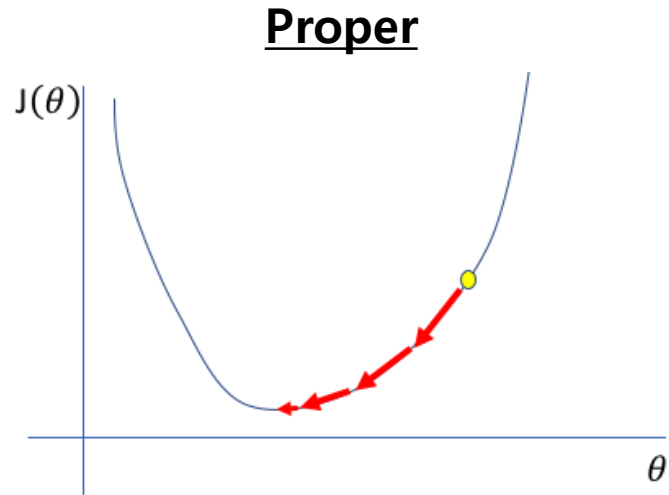
# 3 AdaGrad



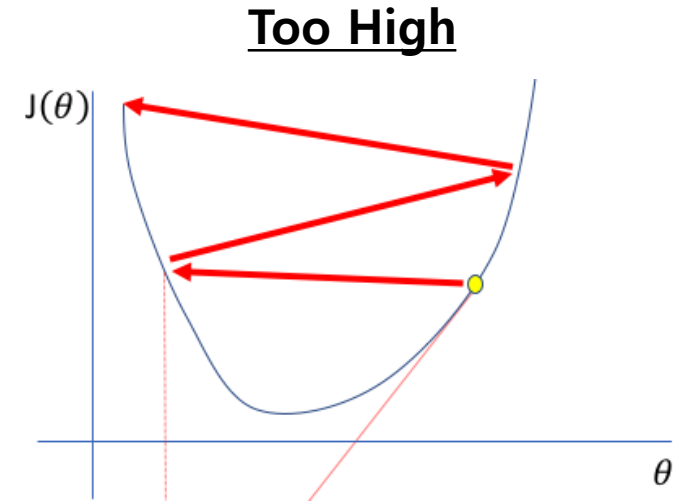
# Adaptive Learning Rate



작은 폭으로 이동하기 때문에 최  
솟점에 도달하기 위해 많은 step  
이 필요



초기에는 큰 폭으로 이동하다가  
최솟점에 가까이 갈수록 폭을 줄  
여서 이동



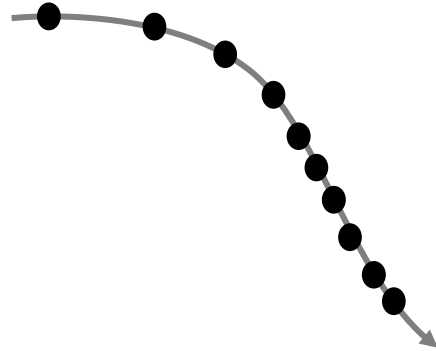
큰 폭으로 이동하기 때문에 최솟  
점을 지나쳐서 반대편으로 진동하  
거나 발산

곡면의 변화량에 따라 학습률을 조절해야 최소점에 빠르게 수렴할 수 있다!

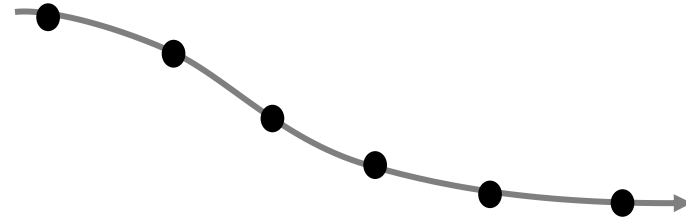
<https://www.jeremyjordan.me/nn-learning-rate/>

# AdaGrad

경로의 변화가 크면 적은 폭으로 이동하고 변화가 없으면 큰 폭으로 이동하자!



변화가 크면 적은 폭으로 이동



변화가 없으면 큰 폭으로 이동

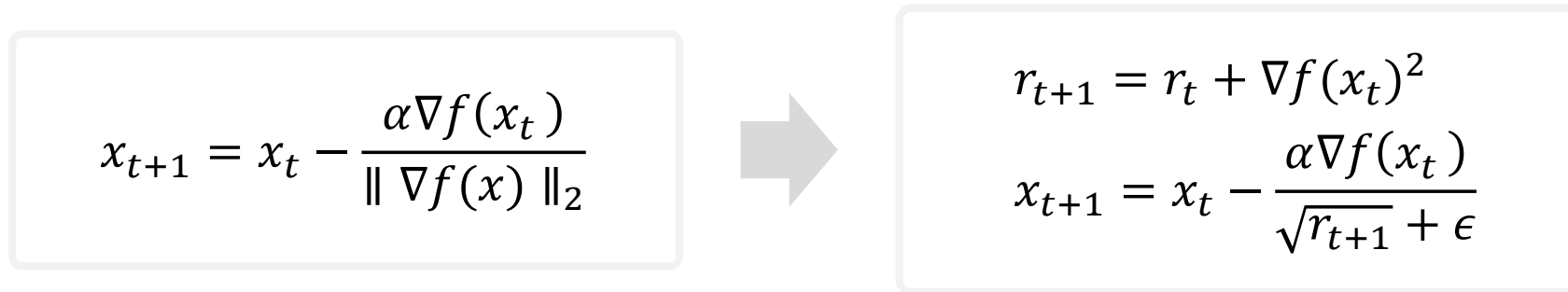
변화량에 따라 자동으로 조절되게 할 수는 없을까?



# AdaGrad

총 Gradient 크기는 전체 변화량이므로 이것으로 학습률을 정해보자!

$$\text{총 Gradient의 크기} = \|\nabla f(x)\|_2 = \sqrt{\nabla f(x_1)^2 + \nabla f(x_2)^2 + \dots + \nabla f(x_n)^2}$$


$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\|\nabla f(x)\|_2}$$
$$r_{t+1} = r_t + \nabla f(x_t)^2$$
$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\sqrt{r_{t+1}} + \epsilon}$$

$\epsilon$  : 분모가 0이 되기 않게 더해주는 상수

AdaGrad : adaptive gradient algorithm

# 장점

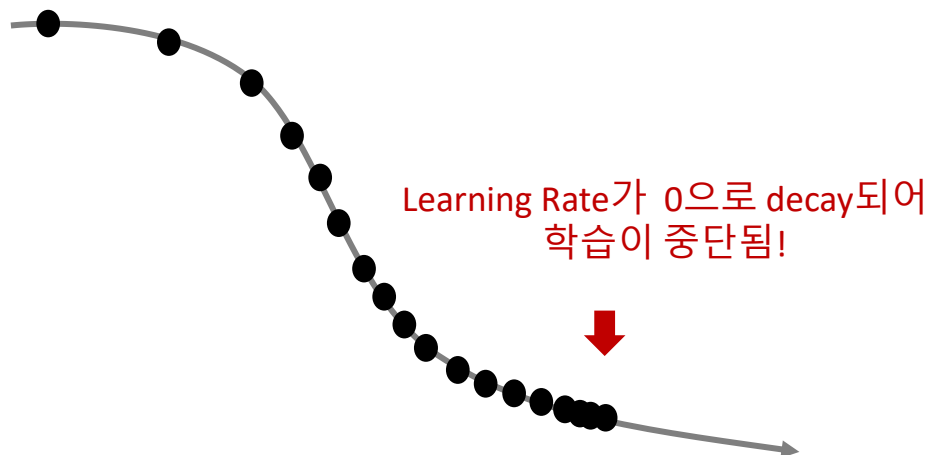
$$r_{t+1} = r_t + \nabla f(x_t)^2$$
$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\sqrt{r_{t+1}} + \epsilon}$$

$\epsilon$  : 분모가 0이 되기 않게 더해주는 상수

- 모델 파라미터 별로 개별적인 Learning Rate를 갖게 되는 효과
- “Per-parameter learning rates” or “adaptive learning rates”

# 단점

경로가 길어질수록 총 Gradient 크기가 점점 커지는 문제 발생



- Convex 문제에 적합
- 신경망에서는 훈련 초반부터 학습률이 급격히 감소하는 문제가 있음

# 코드

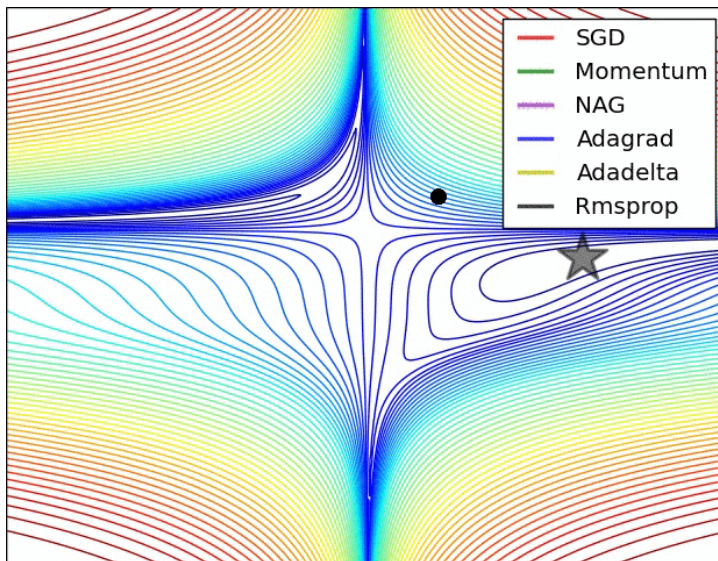
$$r_{t+1} = r_t + \nabla f(x_t)^2$$
$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\sqrt{r_{t+1}} + \epsilon}$$

$\epsilon$  : 분모가 0이 되기 않게 더해주는 상수

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

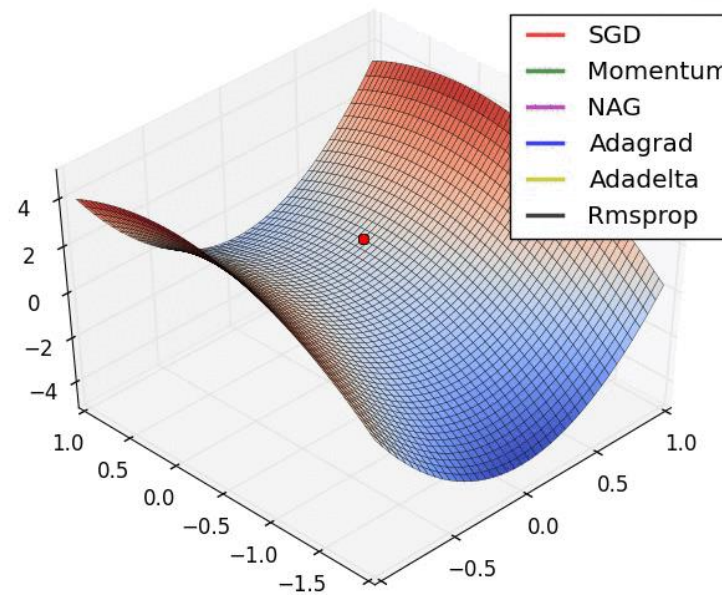
# Variants of Gradient Descent

Beale's Function

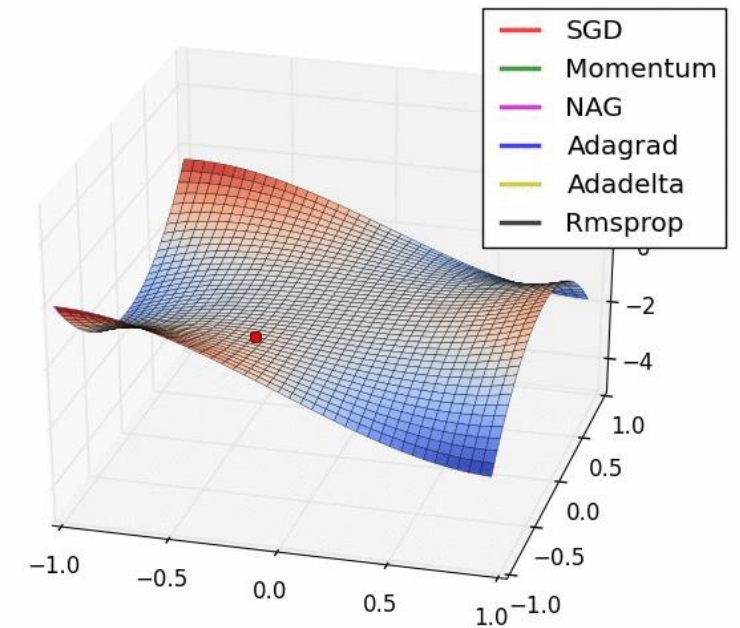


NAG : Nesterov accelerated gradient

Long Valley

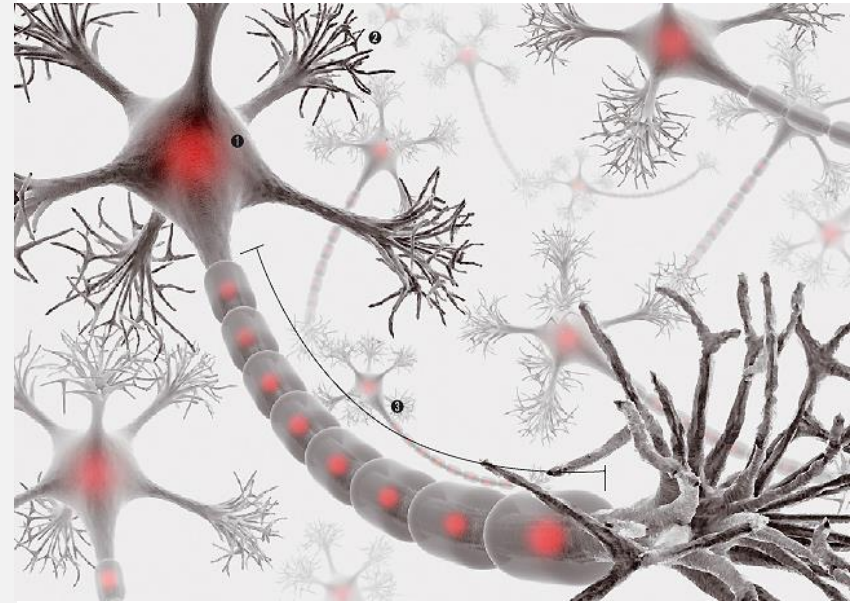


Saddle Point



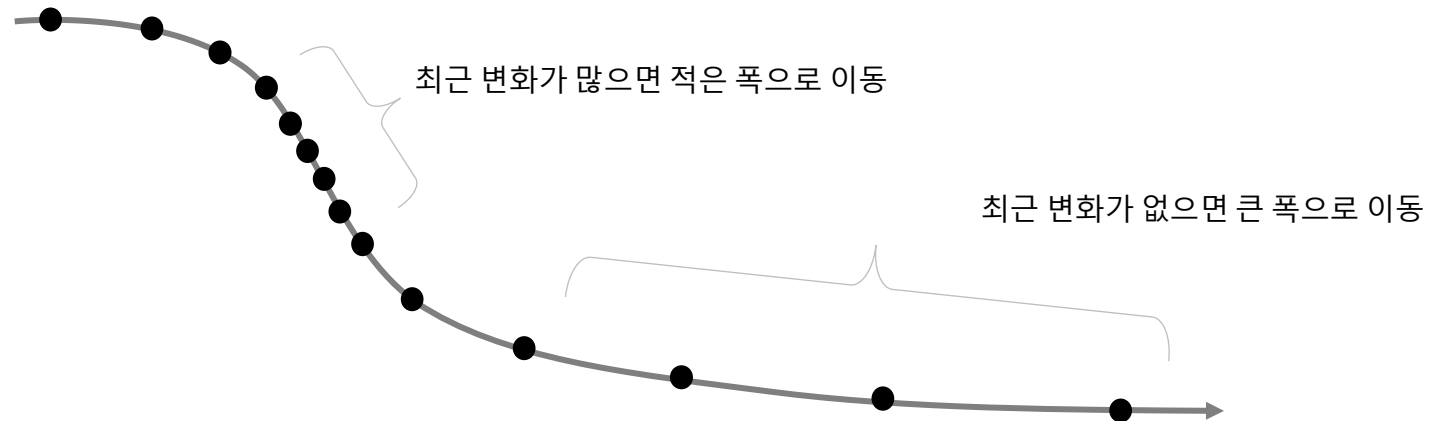
An overview of gradient descent optimization algorithms, Sebastian Ruder

# 4 RMSProp



# RMSProp

최근 변화량을 중심으로 이동 폭을 정해보자



# RMSProp

지수 가중 이동 평균 (Exponentially Weighted Moving Average)

$$r_{t+1} = \alpha r_t + (1 - \alpha) \nabla f(x_t)^2$$
$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\sqrt{r_{t+1}} + \epsilon}$$

$\alpha$  : 가중치 0.9 사용

$\epsilon$  : 분모가 0이 되기 않게 더해주는 상수 (1e-7 or 1e-8 사용)

RMSPop : Root Mean Square Propagation



# RMSProp

재귀식을 풀어보면 오래전 변화량은 적게 반영되고 최근 변화량은 많이 반영되는 것을 확인할 수 있다.

먼저,  $r_t = \beta r_{t-1} + (1 - \beta) \nabla f(x_{t-1})^2$ 를 첫번째 식에 대입해보자.

$$\begin{aligned} r_{t+1} &= \alpha r_t + (1 - \alpha) \nabla f(x_t)^2 && r_t = \beta r_{t-1} + (1 - \beta) \nabla f(x_{t-1})^2 \text{를 대입} \\ &= \beta (\beta r_{t-1} + (1 - \beta) \nabla f(x_{t-1})^2) + (1 - \beta) \nabla f(x_t)^2 \\ &= \beta^2 r_{t-1} + \beta (1 - \beta) \nabla f(x_{t-1})^2 + (1 - \beta) \nabla f(x_t)^2 \\ &= \beta^2 r_{t-1} + (1 - \beta) (\nabla f(x_t)^2 + \beta \nabla f(x_{t-1})^2) \end{aligned}$$

같은 방식으로  $r_{t-1}$ 부터  $r_1$ 까지 순서대로 대입하면 다음과 같이 식이 정리된다.

$$r_{t+1} = \beta^t r_1 + (1 - \beta) (\nabla f(x_t)^2 + \beta \nabla f(x_{t-1})^2 + \cdots + \beta^{t-1} \nabla f(x_1)^2)$$

최근 변화(Gradient)는 많이 반영됨

오래전 변화(Gradient)는 적게 반영됨

# 코드

## AdaGrad

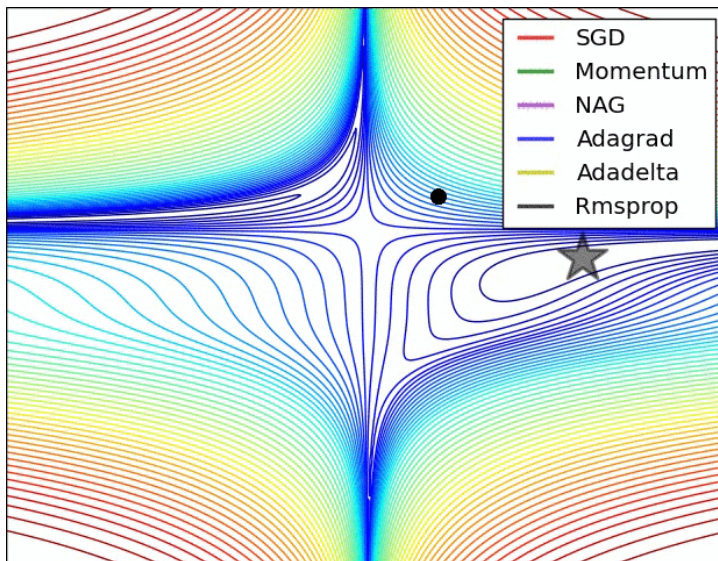
```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

## RMSProp

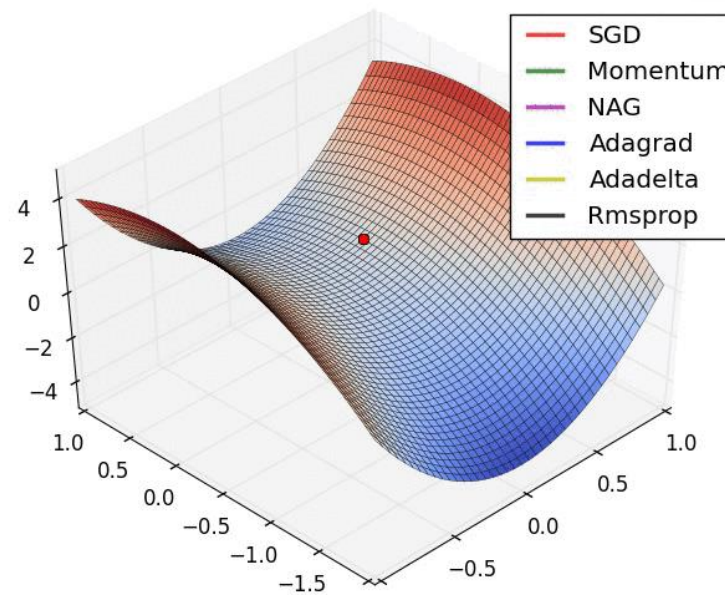
```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

# Variants of Gradient Descent

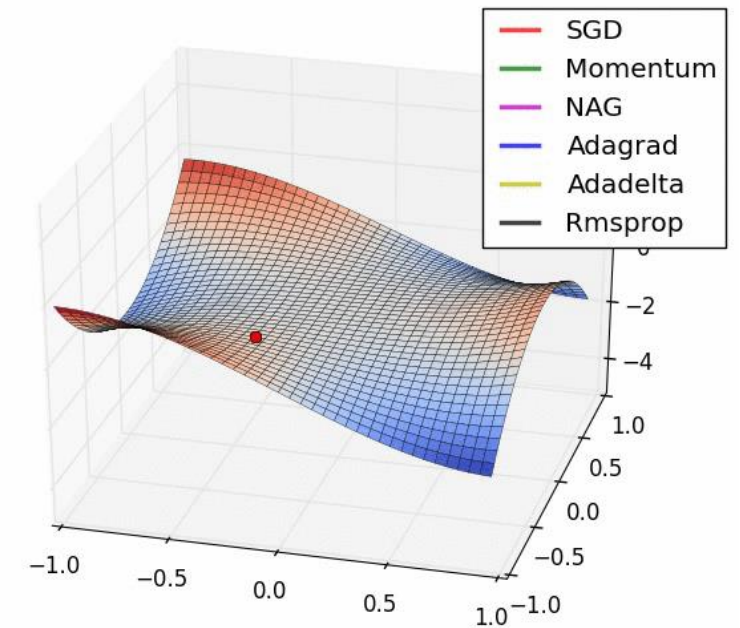
Beale's Function



Long Valley



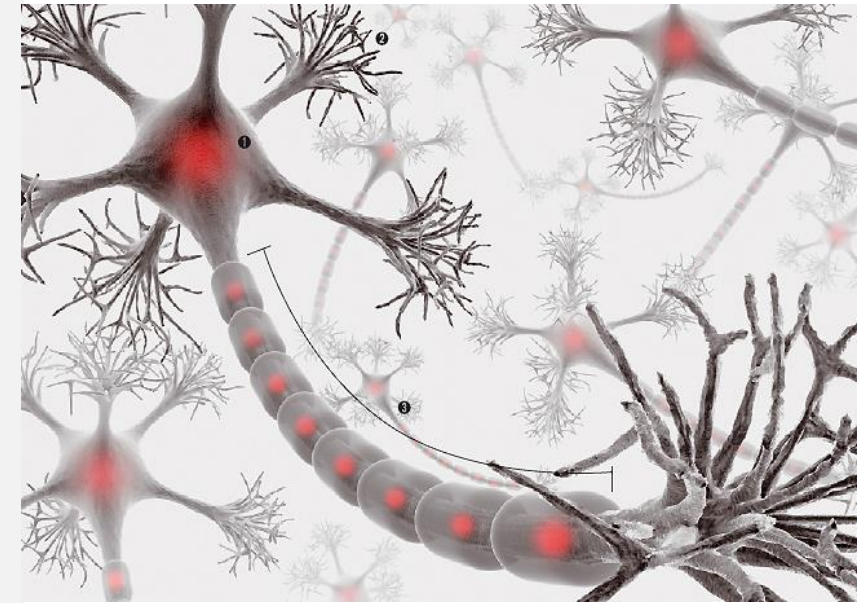
Saddle Point



NAG : Nesterov accelerated gradient

An overview of gradient descent optimization algorithms, Sebastian Ruder

# 5 Adam



# Adam

## Adaptive Moments 전략



$$v_{t+1} = \beta_1 v_t + (1 - \beta_1) \nabla f(x_t) \quad \text{first momentum (velocity)}$$

$$r_{t+1} = \beta_2 r_t + (1 - \beta_2) \nabla f(x_t)^2 \quad \text{second momentum (sum of squared gradient)}$$

$$x_{t+1} = x_t - \frac{\alpha v_{t+1}}{\sqrt{r_{t+1}} + \epsilon}$$

$\epsilon$  : 분모가 0이 되기 않게 더해주는 상수

# Adam (almost)

- $\text{beta1} = 0.9, \text{beta2} = 0.999$
- $\text{learning\_rate} = 1\text{e-}3 \text{ or } 5\text{e-}4$

$$v_0 = 0, r_0 = 0$$

for  $t$  in range(1, num\_iterations) :

$$v_{t+1} = \beta_1 v_t + (1 - \beta_1) \nabla f(x_t)$$

$$r_{t+1} = \beta_2 r_t + (1 - \beta_2) \nabla f(x_t)^2$$

$$x_{t+1} = x_t - \frac{\alpha v_{t+1}}{\sqrt{r_{t+1}} + \epsilon}$$

첫번째 단계에서 어떤 일이 벌어질까요?

$$v_1 = 0.1 * \nabla f(x_0)$$

$$r_1 = 0.001 * \nabla f(x_0)^2$$

- $r_0 = 0$ 으로 시작하므로  $r_1$ 이 매우 작은 숫자가 됨
- 따라서, step size가 매우 커져서 최적화에 좋지 않은 지점으로 이동할 수 있음

RMSProp 역시 훈련 초반에 크게 편향되는 문제가 있음

# Algorithm

$$v_0 = 0, r_0 = 0$$

for t in range(1, num\_iterations) :

$$v_{t+1} = \beta_1 v_t + (1 - \beta_1) \nabla f(x_t)$$

$$r_{t+1} = \beta_2 r_t + (1 - \beta_2) \nabla f(x_t)^2$$

$$v_{t+1} = \frac{v_{t+1}}{(1 - \beta_1^t)}$$

$$r_{t+1} = \frac{r_{t+1}}{(1 - \beta_2^t)}$$

$$x_{t+1} = x_t - \frac{\alpha v_{t+1}}{\sqrt{r_{t+1}} + \epsilon}$$



**훈련 초반에 발생하는 편향을 제거**

$$v_1 = \nabla f(x_0)$$

$$r_1 = \nabla f(x_0)^2$$

- t가 커질수록 분모항이 1로 수렴하므로 원래의 식으로 복원됨



# 코드

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

- $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$
  - $\text{learning\_rate} = 1\text{e-}3$  or  $5\text{e-}4$
- is a great starting point for many models!

Adam은 하이퍼파라미터에 대해 가장 Robust한 방법



**Thank you!**

