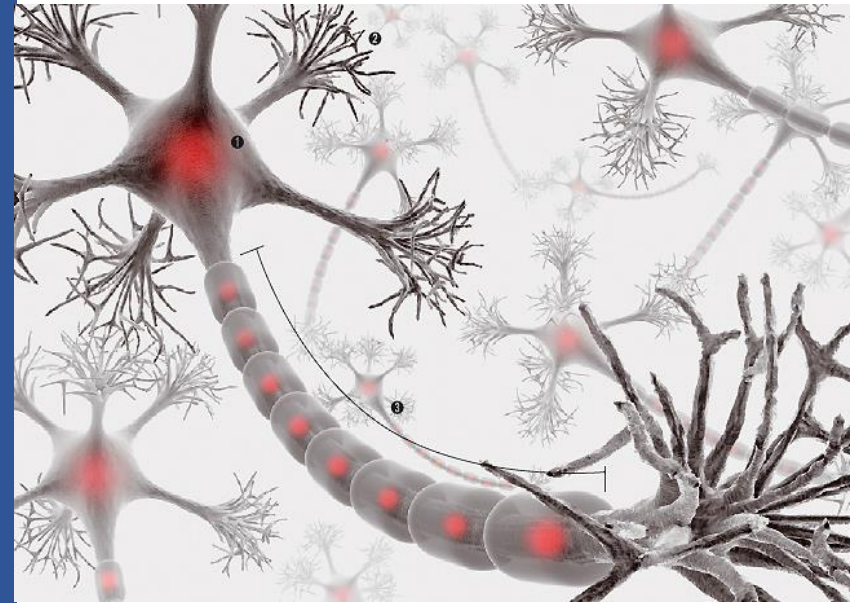


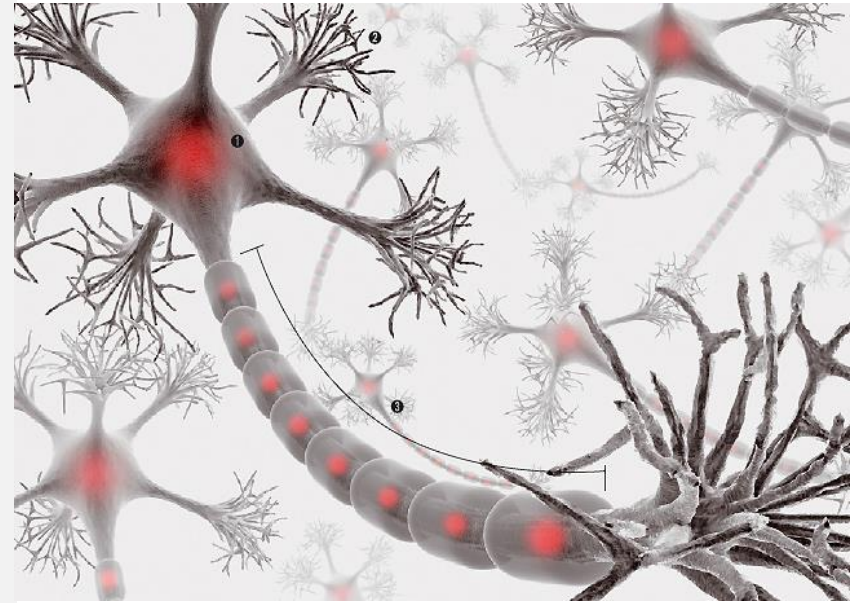
# [복습] 최적화, 초기화, 정규화

## 학습 목표

- Day2에 배웠던 내용들을 복습해본다.

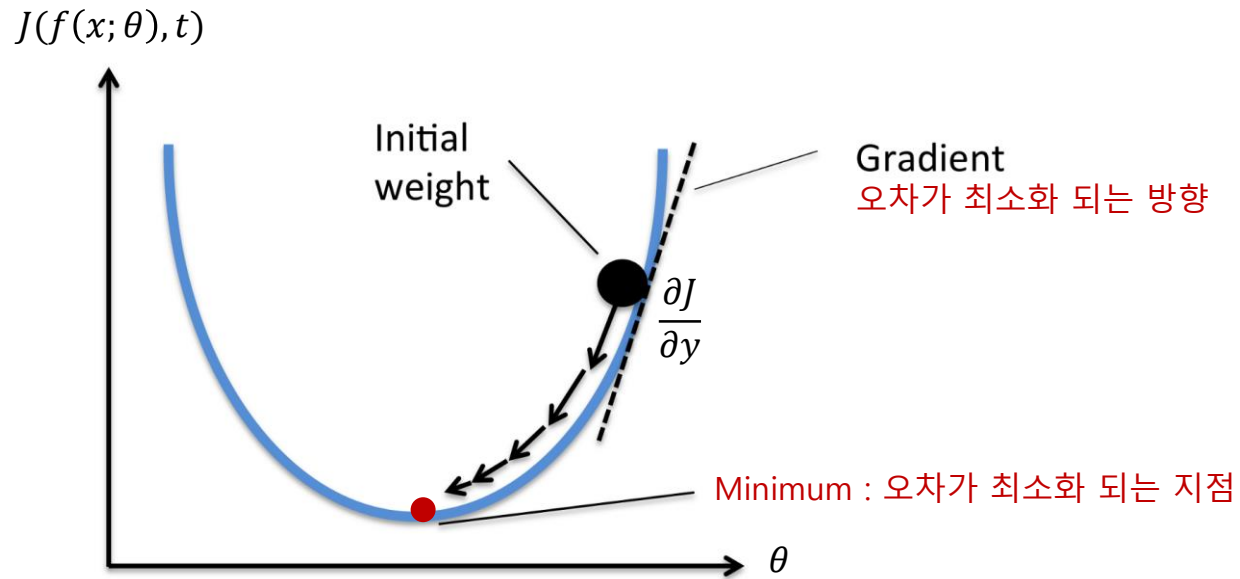


# 최적화

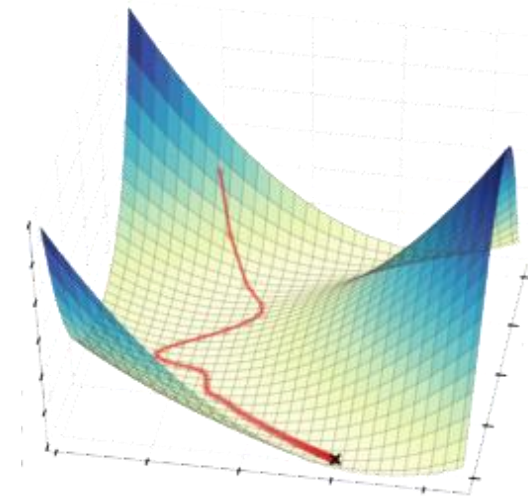


# Gradient Descent

## Gradient Descent



## 3D View



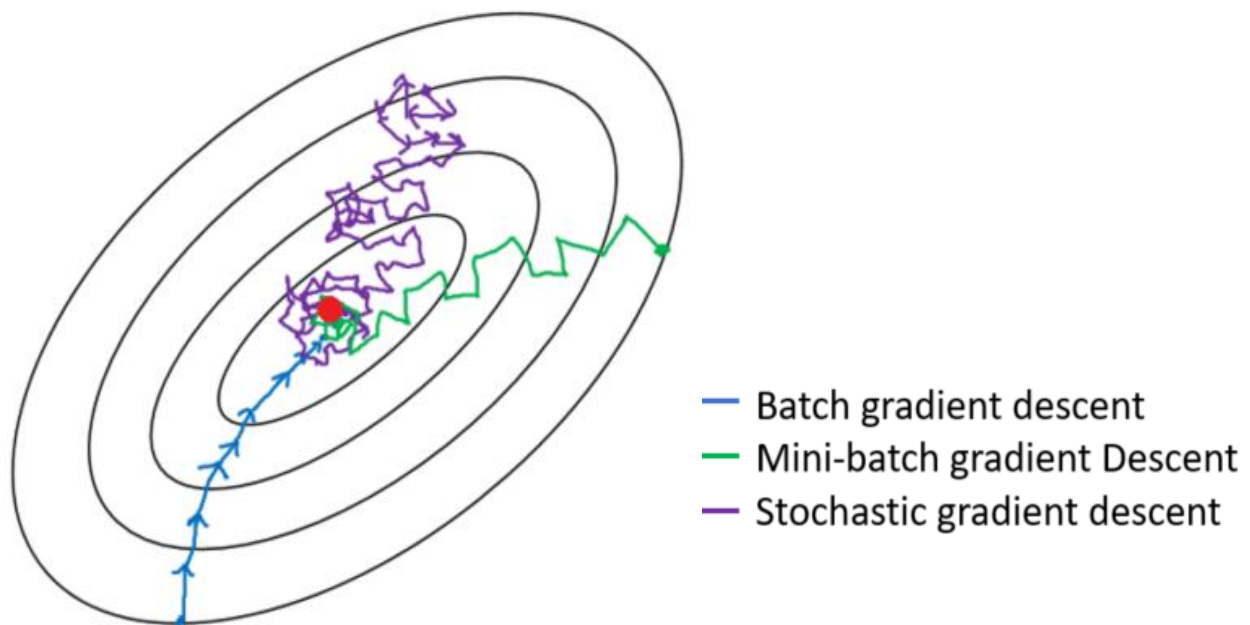
## Parameter Update

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

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

# 훈련 단위

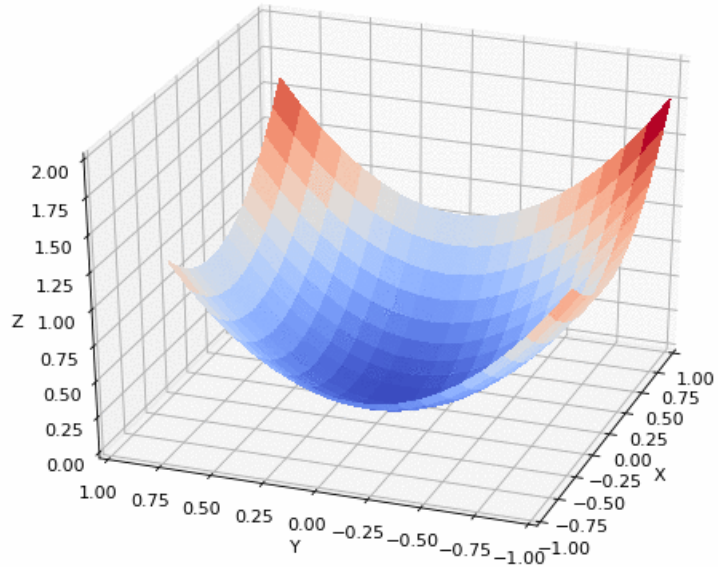
## Gradient Descent Trajectory



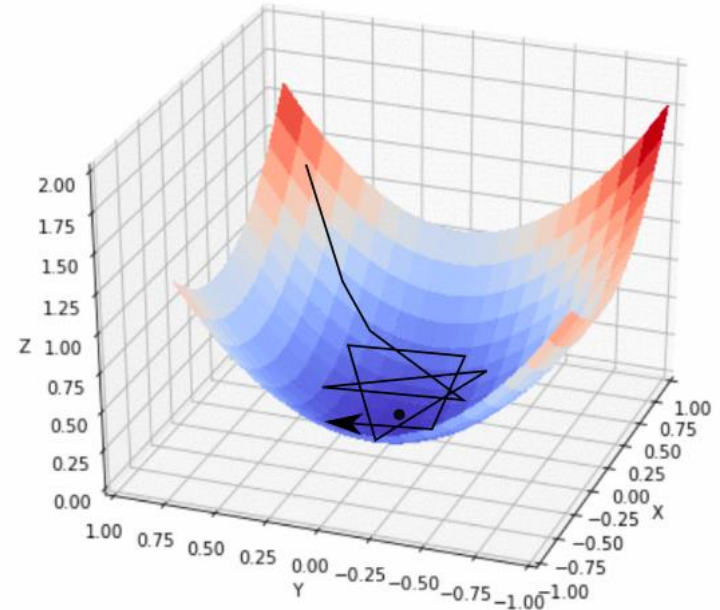
[그림] <https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>

Issue 학습률이 자동으로 조정되지 않는다.

Gradient Descent 수렴 경로



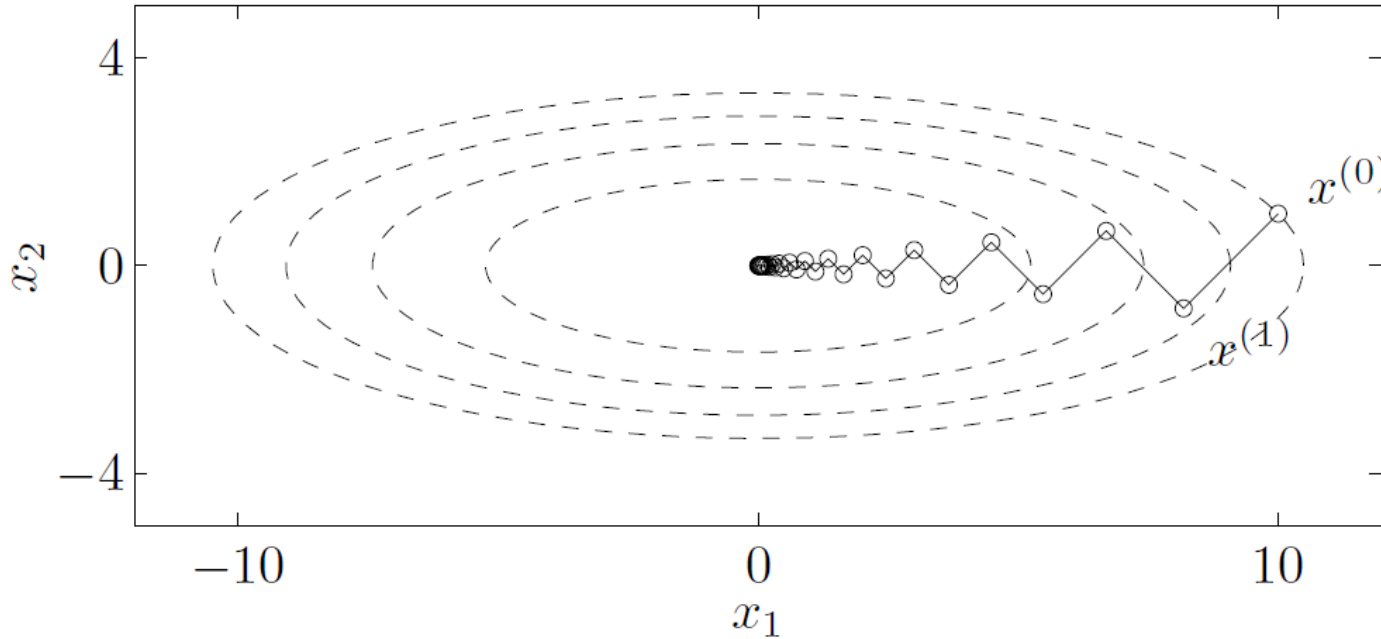
학습률이 큰 경우



최소점 부근에서 수렴하지 못하고 진동

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

## Issue Ill-Conditioning 상태에서는 잘 수렴되지 않는다.



[그림] Convex Optimization, Stephen Boyd, Lieven Vandenberghe

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

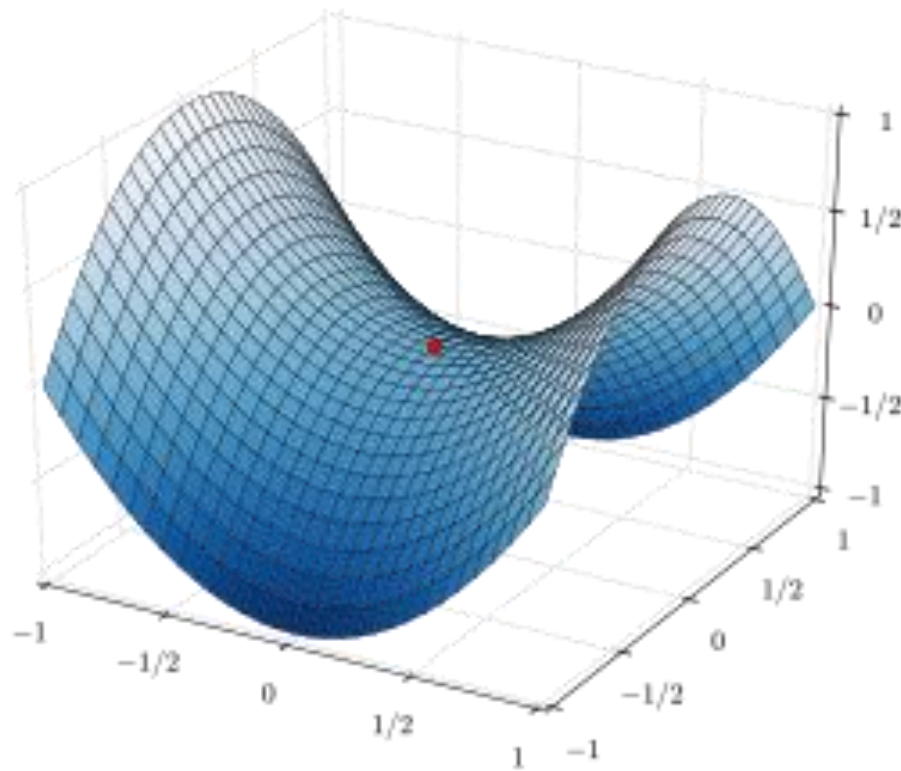
condition number : 10

### Condition number란?

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

Issue 임계점에서 탈출하지 못한다.

### Saddle Point



어떤 차원으로는 Maximum이고  
어떤 차원으로는 Minimum인 지점

**차원이 높아질수록 Saddle Point가 많아진다!**

## 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.



# SGD + Momentum

## SGD + Momentum

이전 단계의  
Velocity

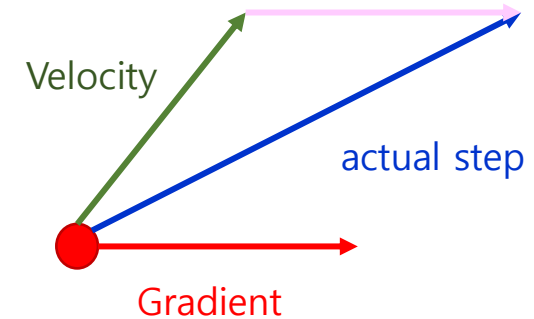
현재 Gradient

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

Velocity = 누적 Gradient

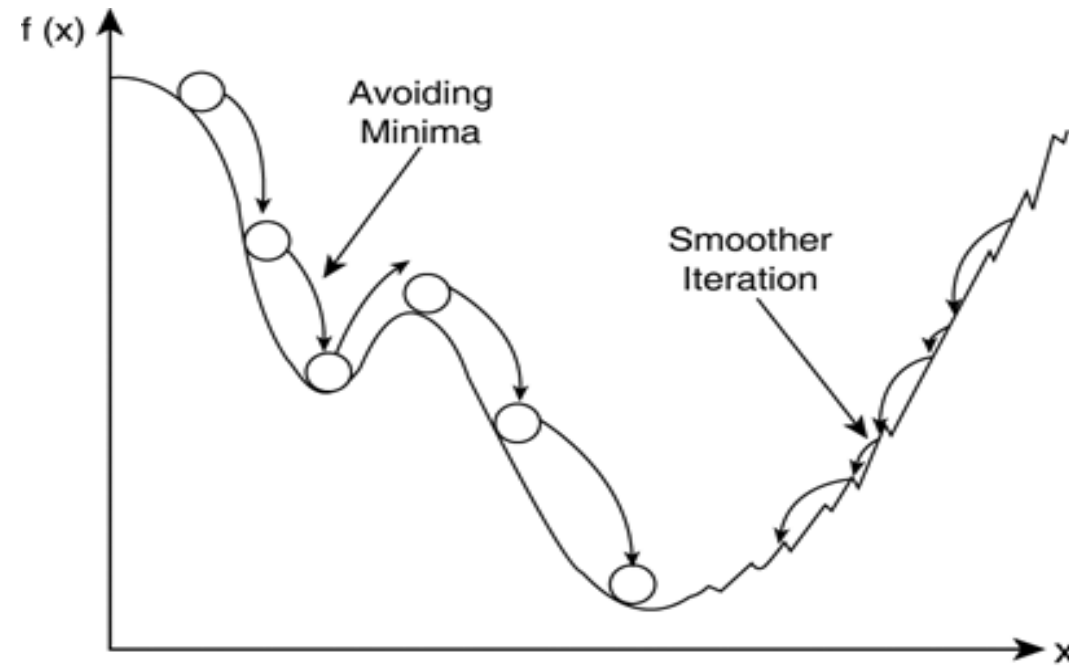
- $\rho$  : “friction” 마찰 계수로 0.9이나 0.99를 사용

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



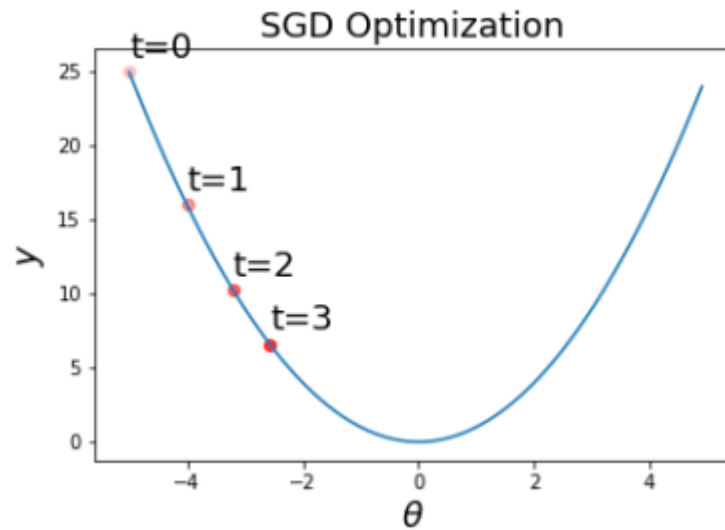
# 장점

## Local Minima & Saddle Point

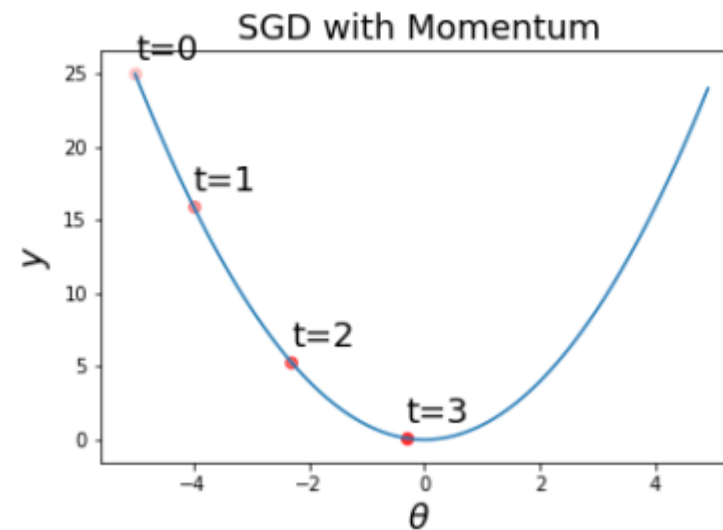


# 장점

## SGD



## SGD+Momentum

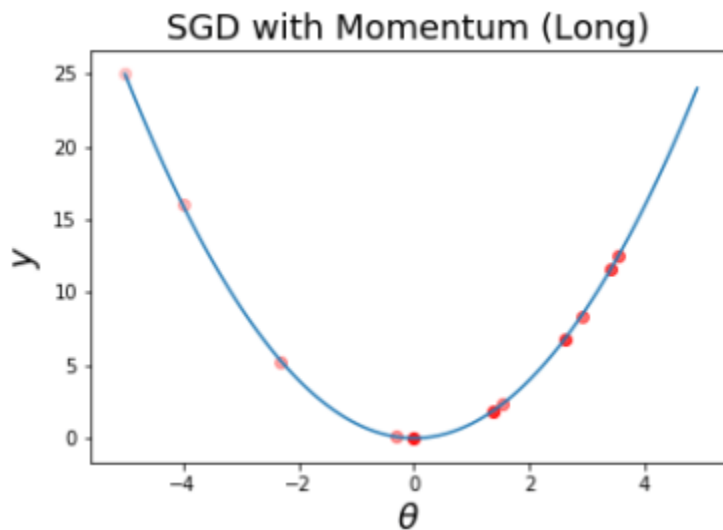


- + Gets to the optimal quicker

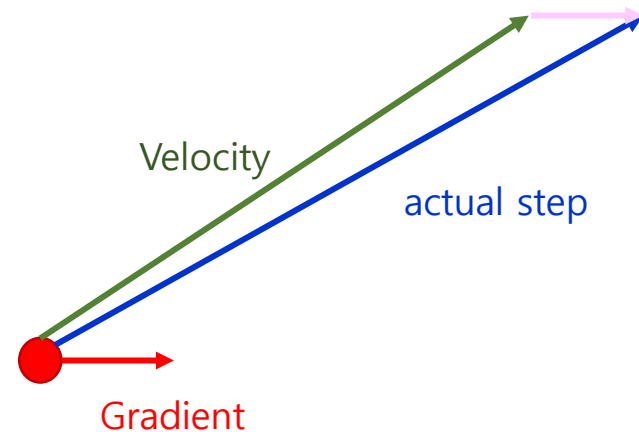
<http://www.thushv.com/deep-learning/a-practical-guide-to-understanding-stochastic-optimization-methods-workhorse-of-machine-learning/>

# 단점

## SGD+Momentum



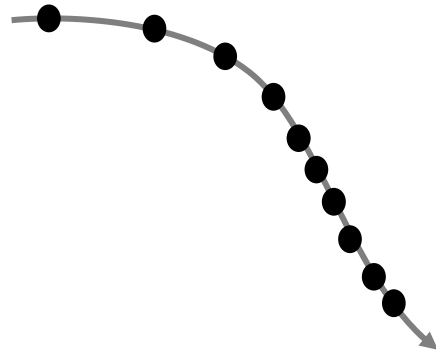
- 최소점에 도달한 이후에도 Overshooting 될 수 있음
- 즉, 현재 Gradient가 작더라도 Velocity가 크면 최적화가 계속 진행됨



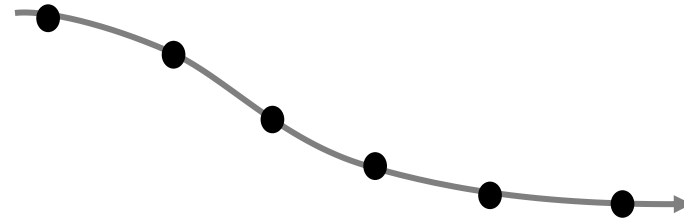
- Can overshoot if not careful with the learning rate

# AdaGrad

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



변화가 크면 적은 폭으로 이동



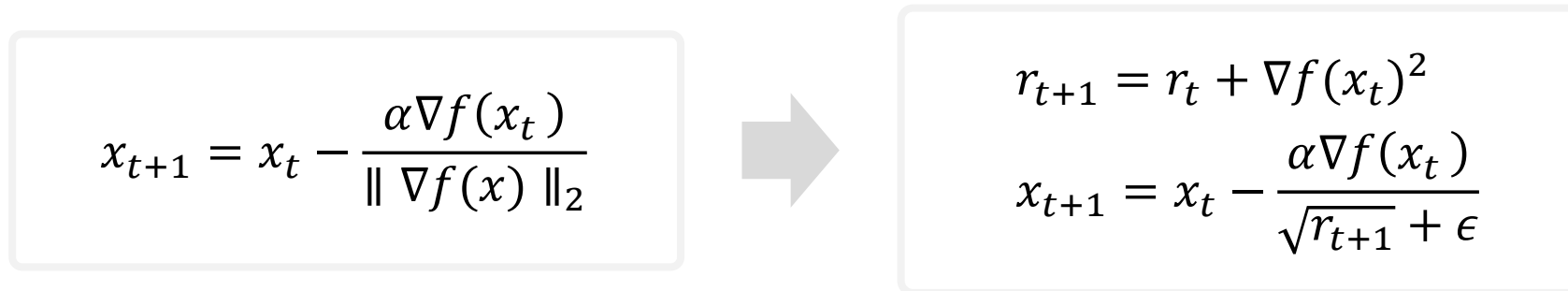
변화가 없으면 큰 폭으로 이동

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

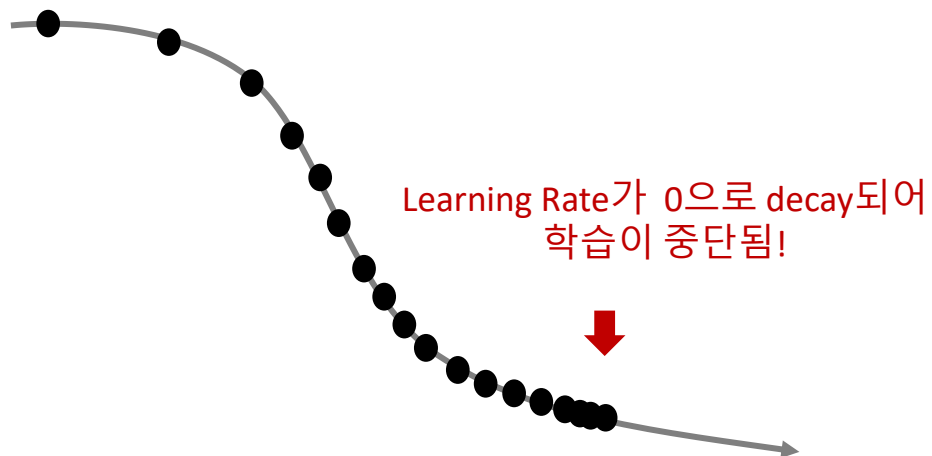
$$r_{t+1} = r_t + \nabla f(x_t)^2$$
$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\sqrt{r_{t+1}} + \epsilon}$$

$\epsilon$  : 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

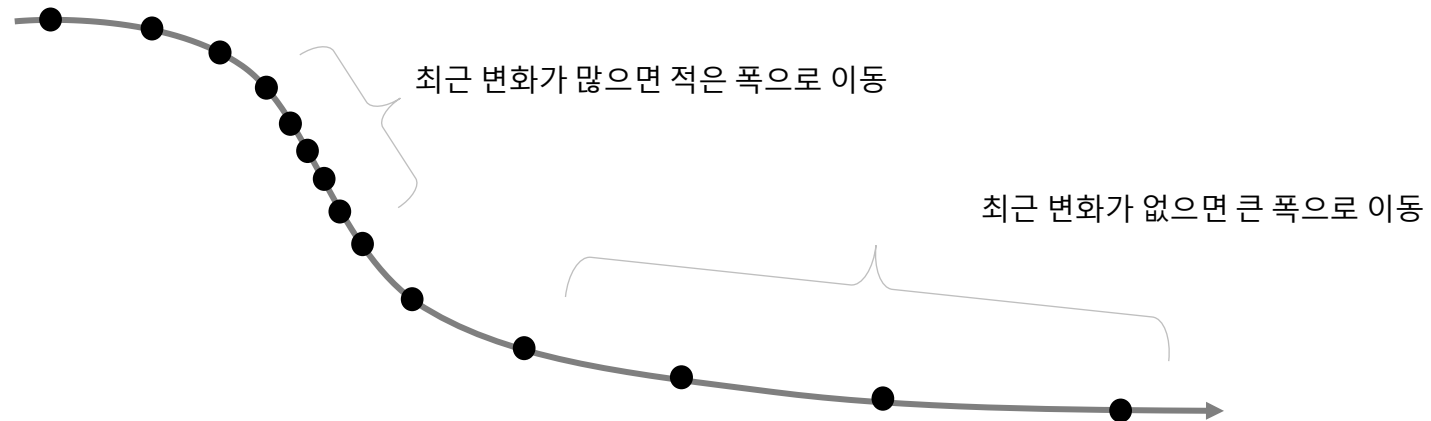


- Convex 문제에 적합
- 신경망에서는 훈련 초반부터 학습률이 급격히 감소하는 문제가 있음



# 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

# 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)

- $\beta_1 = 0.9, \beta_2 = 0.999$
- learning\_rate  $\alpha = 1\text{e-}3$  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) :

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

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

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

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

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



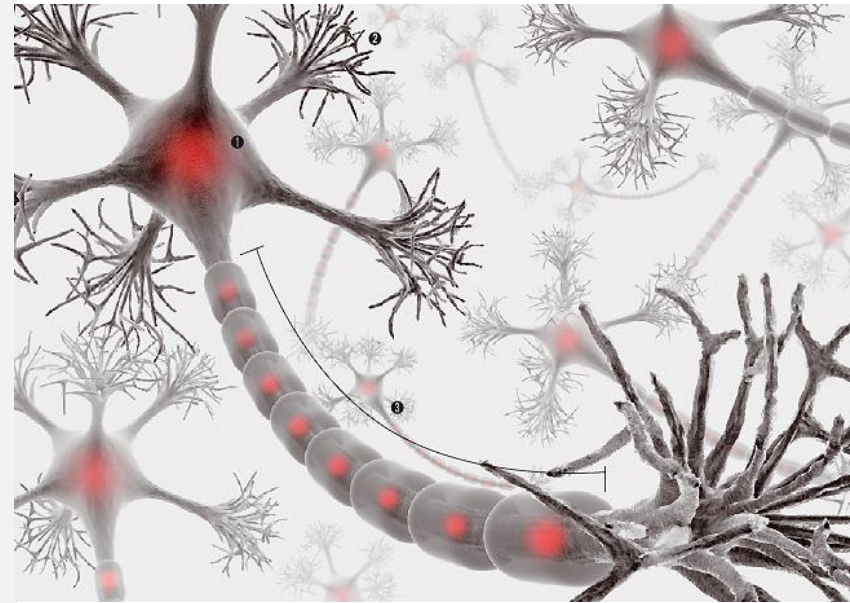
**훈련 초반에 발생하는 편향을 제거**

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

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

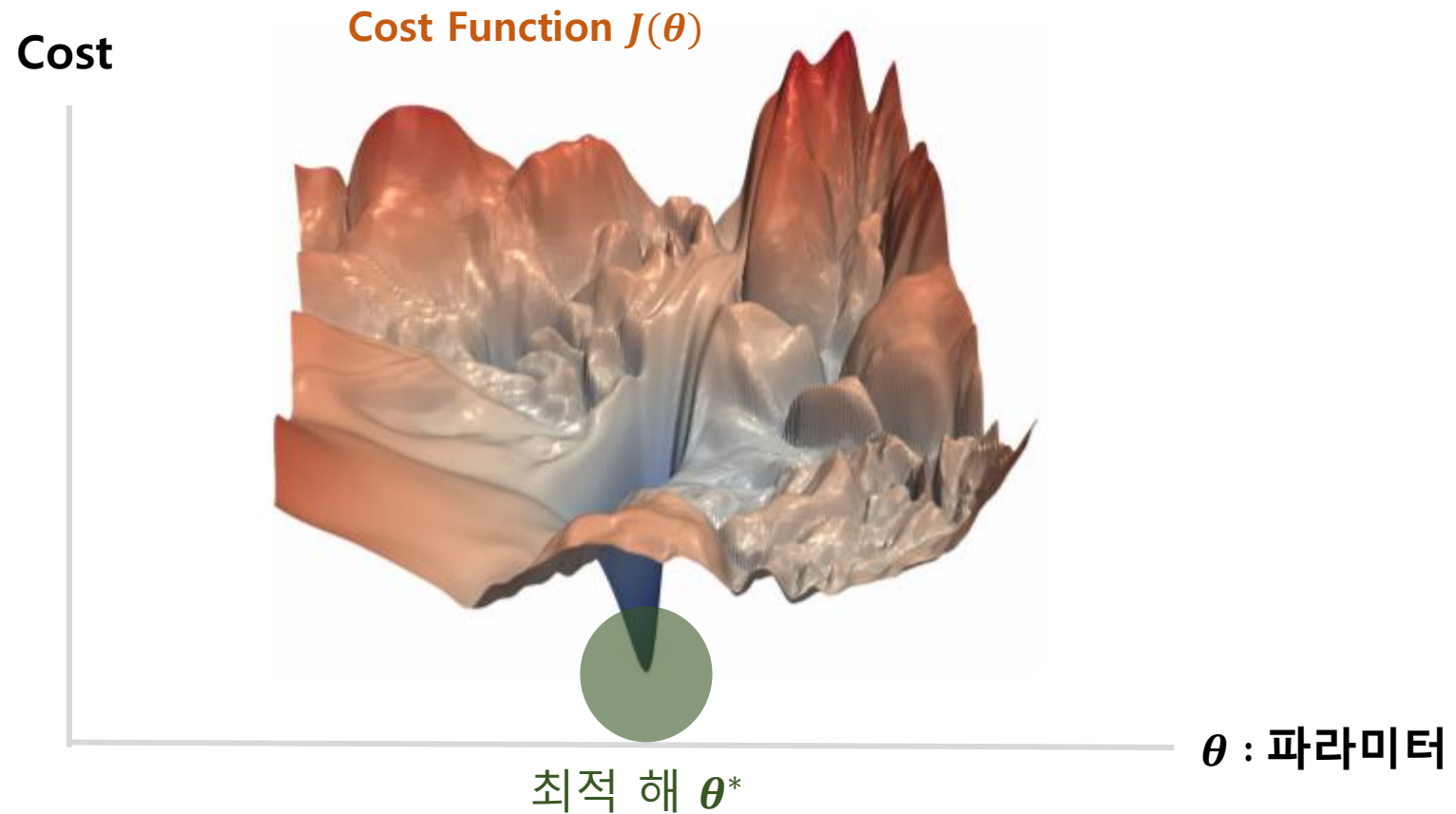
- t가 커질수록 분모항이 1로 수렴하므로 원래의 식으로 복원됨

# 초기화 (Initialization)



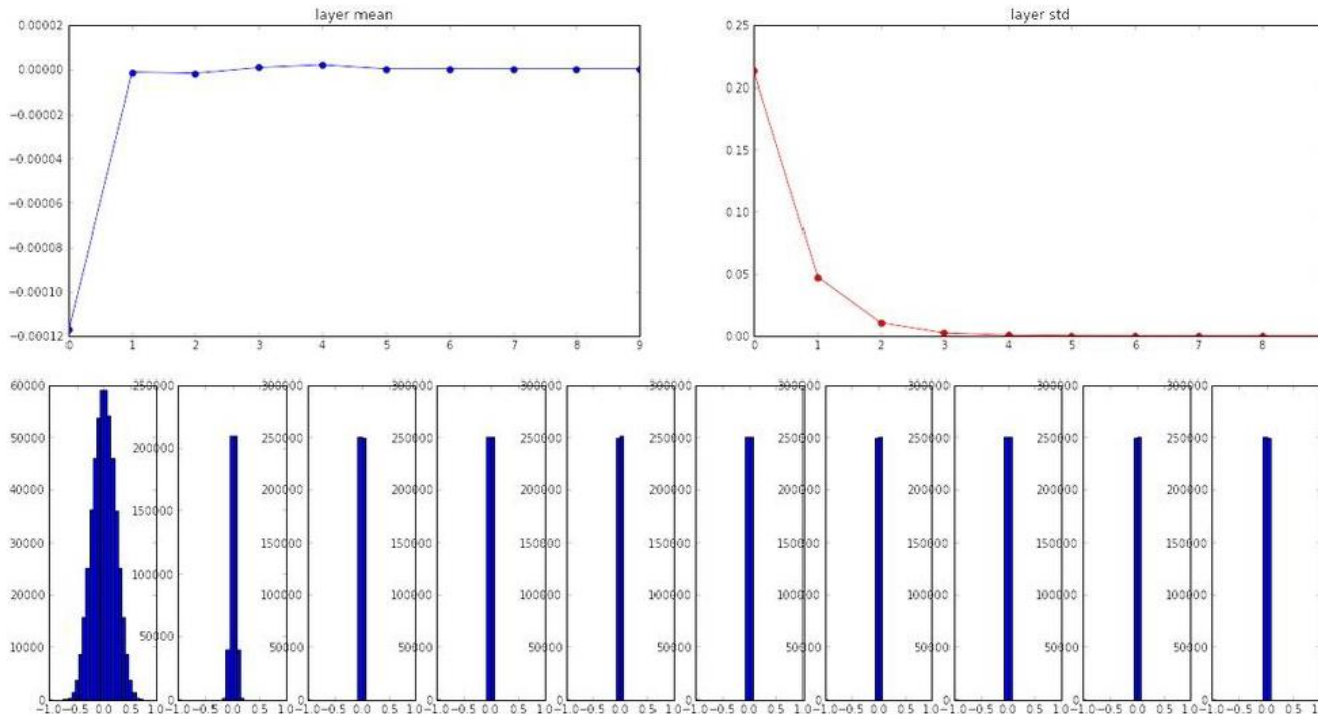
# 가중치 초기화란?

초기화 문제는 손실 곡면의 어느 위치에서 출발하면 좋은 지의 문제



# 가우시안 분포 초기화 - 분산이 작은 경우

## Activation 분포



- Layer가 10개
- Layer 별 뉴런은 500개
- tanh 사용
- 초기화 : 분산이 0.01

$W = 0.01 * \text{np.random.randn}(D, H)$

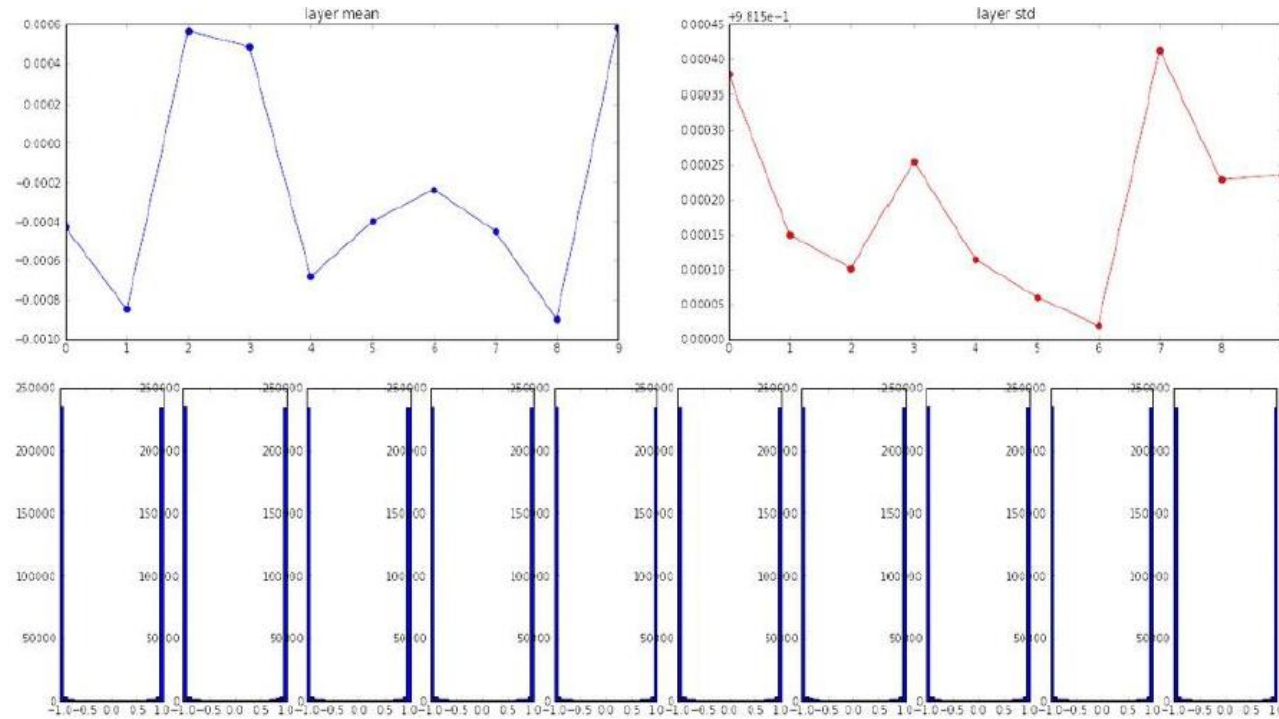
Activation이 0이면 Gradient도 0!  
학습이 진행되지 않음!

Activation이 점점 0으로 변화



# 가우시안 분포 초기화 - 분산이 큰 경우

Activation 분포



Saturated Activation (-1 or 1)

- 초기화 : 분산이 1

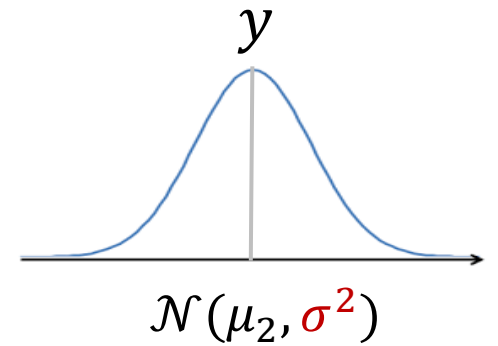
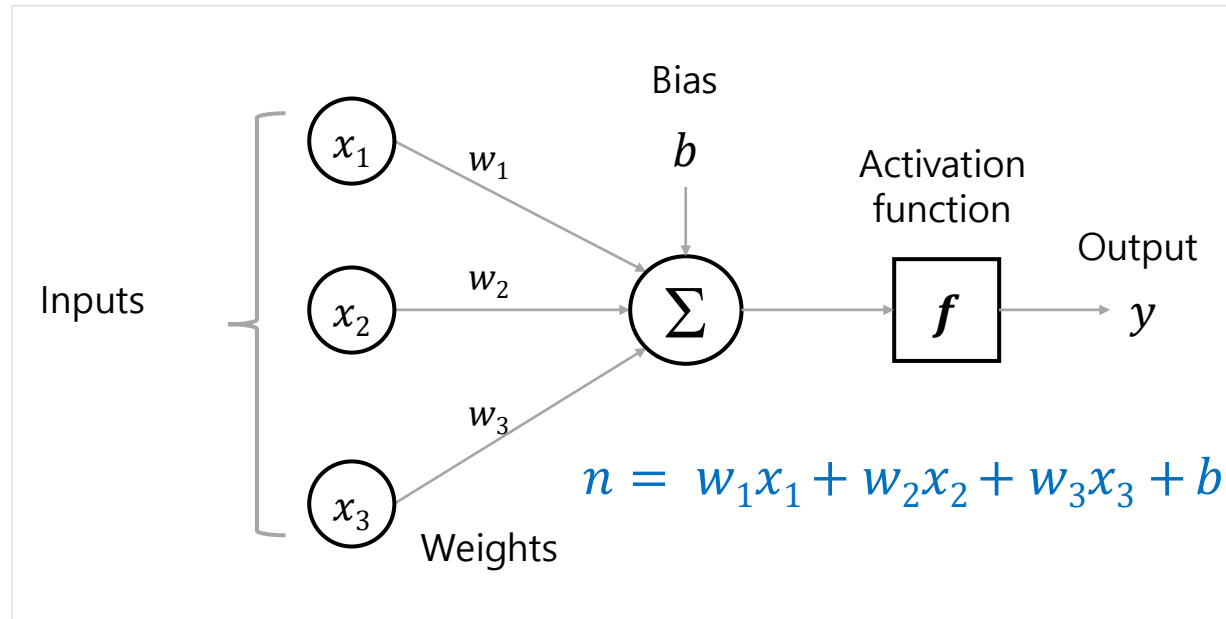
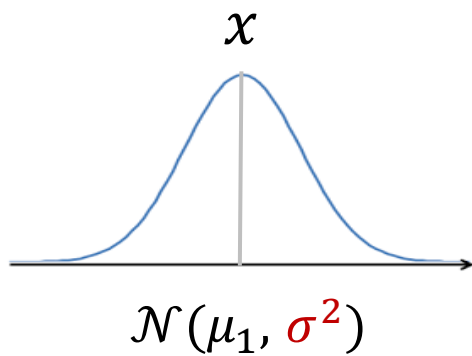
```
W = 1.0 * np.random.randn(fan_in, fan_out)
```

- 가중치가 커지면 tanh 입력 값이 커짐
- 따라서, tanh가 saturation 되어 출력이 1이나 -1로 값이 편향됨

Gradient Saturation이 일어나  
학습이 진행되지 않음!

# Xavier Initialization

입력과 출력의 분산이 같아지도록 가중치로 초기화하자!



Glorot, Bengio 'Understanding the Difficulty of Training Deep Feedforward Neural Networks', 2010

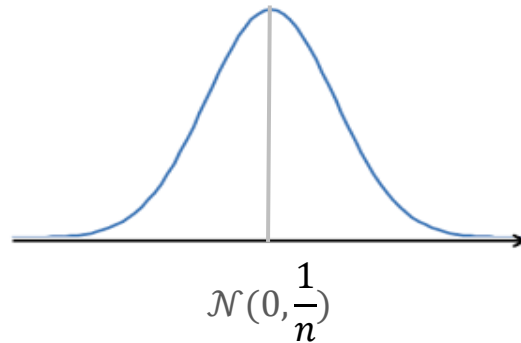
# Xavier Initialization

$x$ 의 분산과  $y$ 의 분산을 같게 하려면?

$$y = n = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \quad x_i \text{ 와 } w_i \text{는 독립이고 iid}$$

$$\text{Var}(x_iw_i) = \text{Var}(x_i)\text{Var}(w_i) \text{ 이므로} \quad \text{Var}(y) = n\text{Var}(x_i)\text{Var}(w_i) \quad \text{Var}(x_i) = \text{Var}(y)$$

$$\text{Var}(w_i) = \frac{1}{n}$$



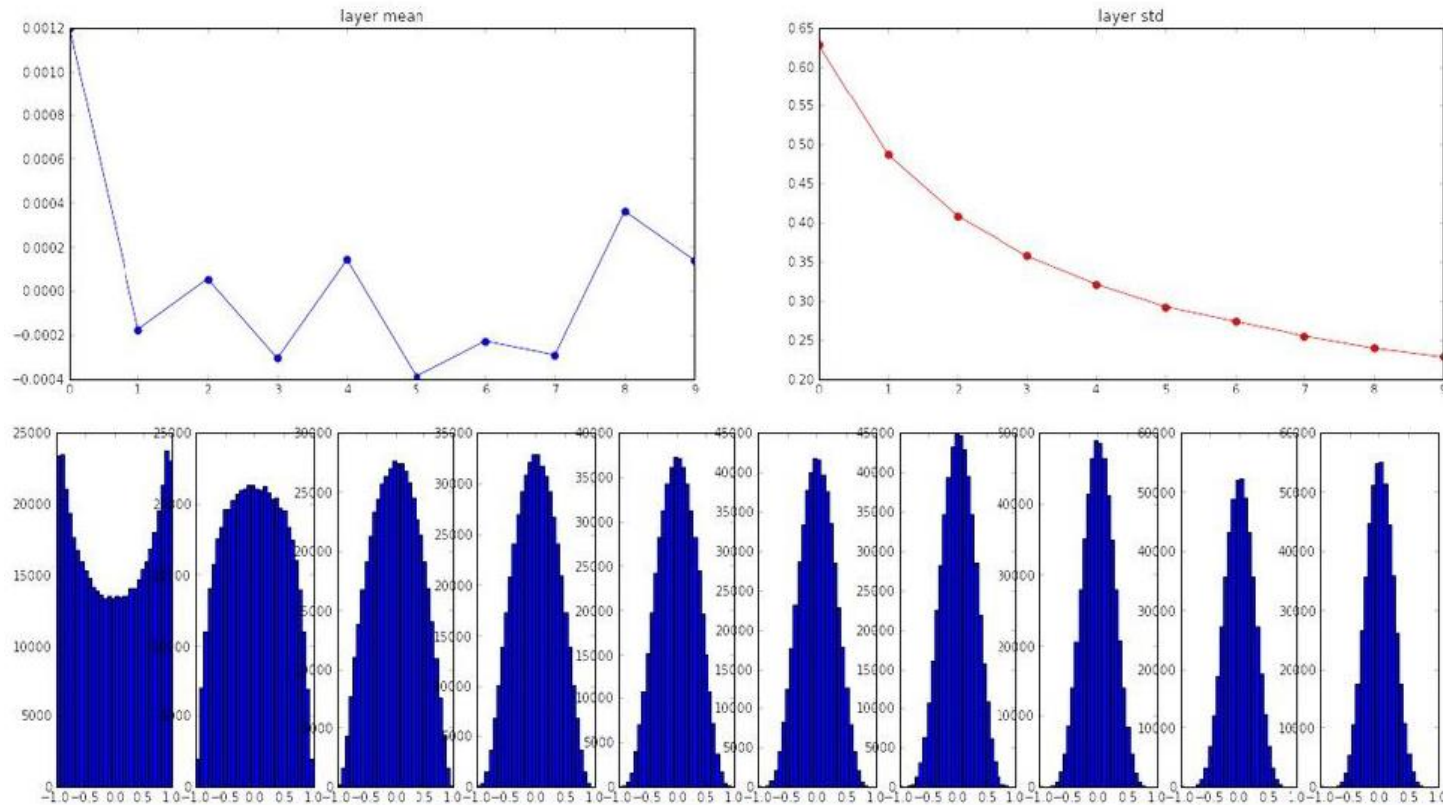
$$\text{Var}(W) = \frac{1}{n}$$

$n$  : # of inputs

가중치의 분산을 입력 개수에 반비례하게 만든다!

# Xavier Initialization

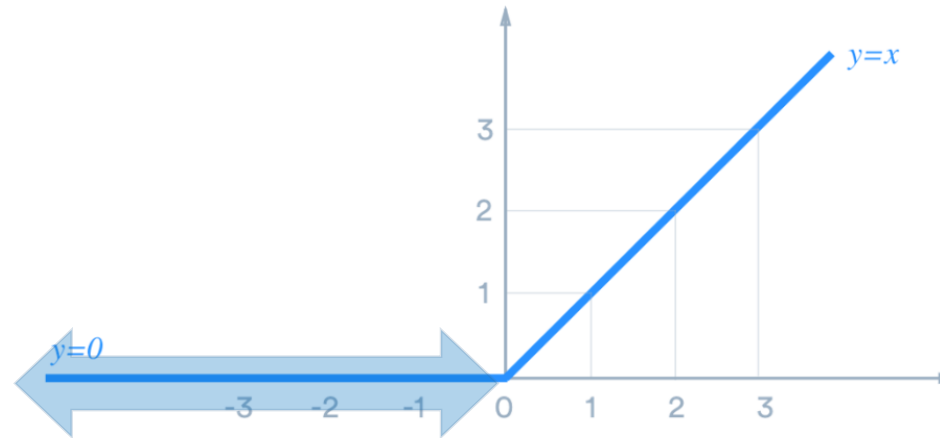
Activation 분포



각 Layer마다 Unit-Gaussian Input 및 Output을 근사하게 됨

# Xavier Initialization

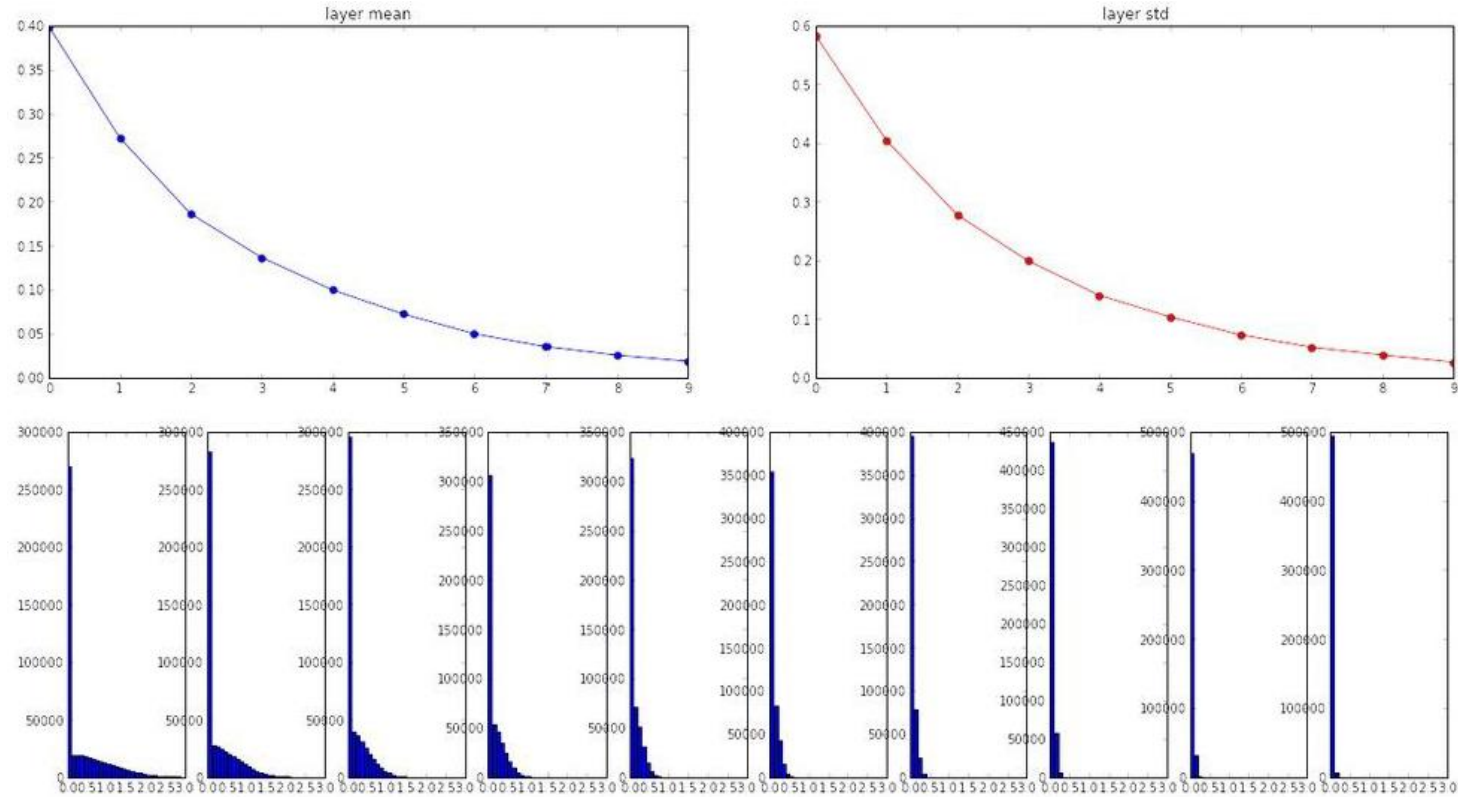
## Non-Linear Activation인 ReLU를 사용하면?



- 정규 분포의 음수 영역이 ReLU Activation 과정에서 0으로 바뀜
- Layer가 깊어질수록 Activation이 점점 0으로 치우치게 됨

# Xavier Initialization

Activation 분포

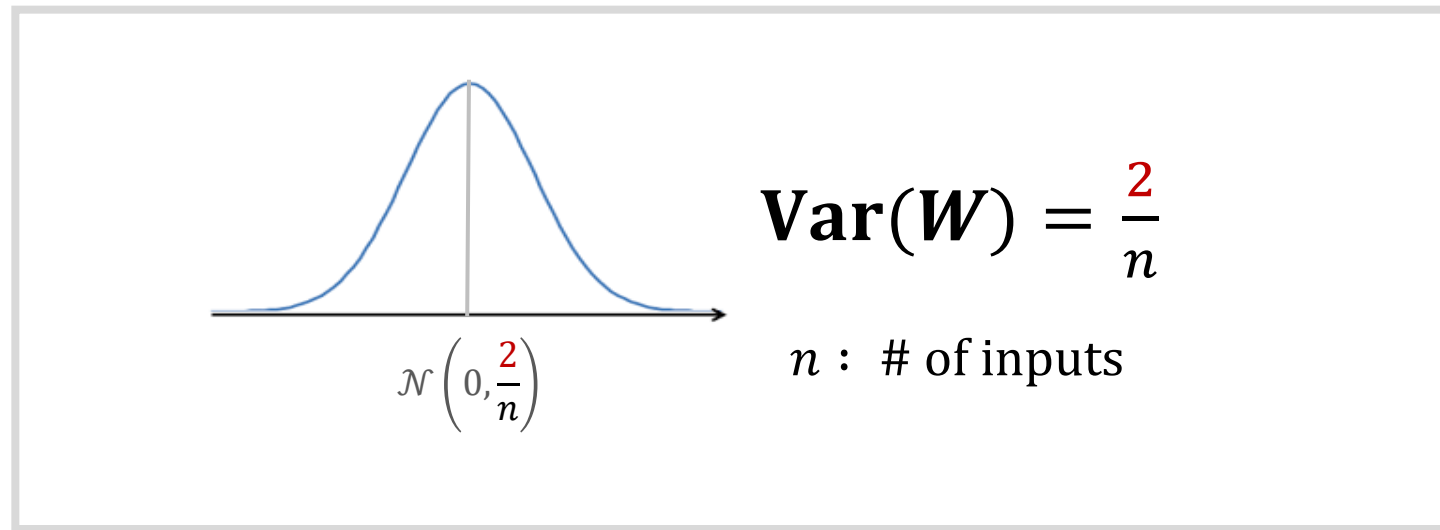


Activation이 점점 0으로 변화

# He Initialization

가중치의 분산을 2배로 키워서 Activation을 넓게 퍼지게 만듦

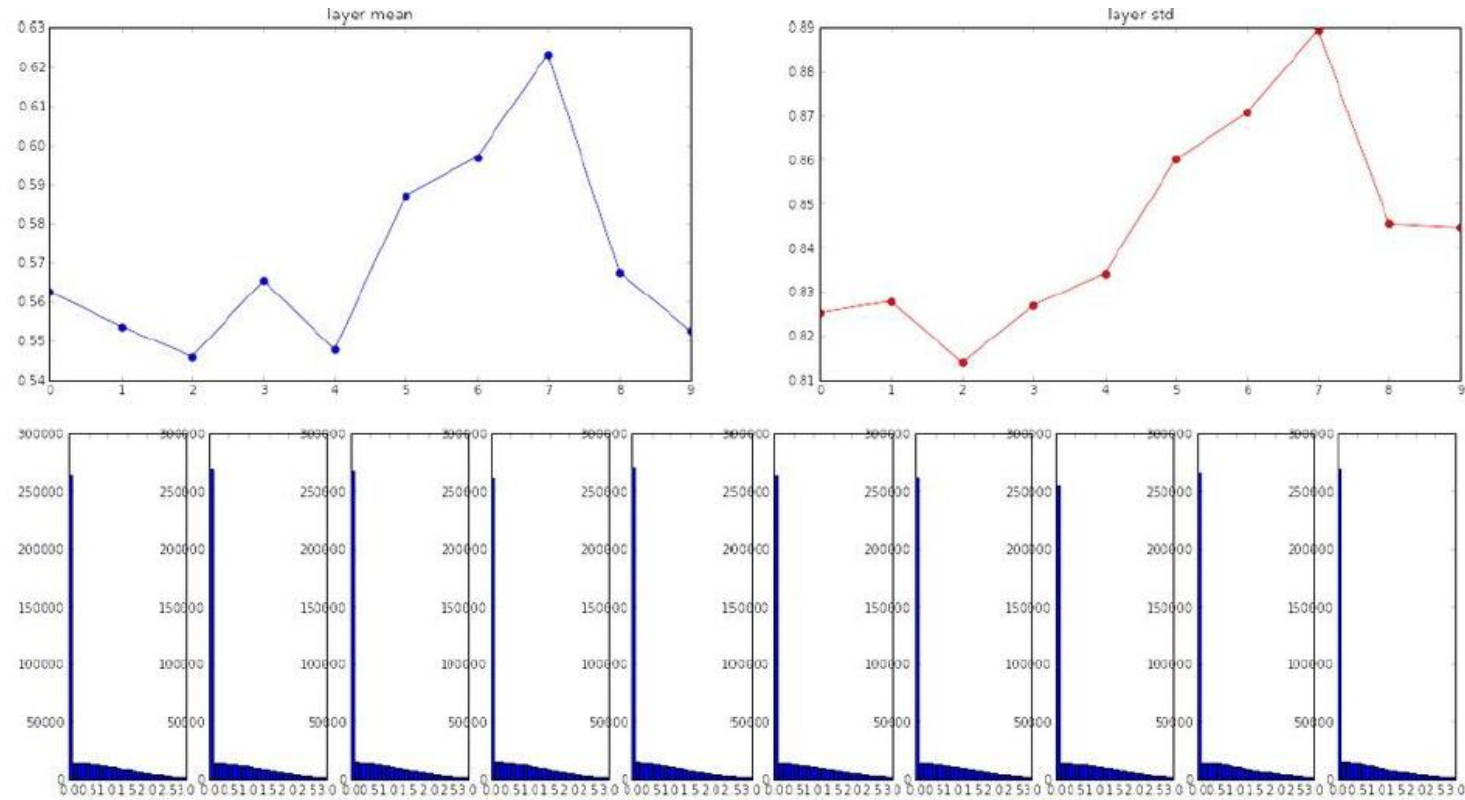
$$y = w_0x_0 + w_1x_1 + \cdots + w_{n-1}x_{n-1} + b$$



Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He (2015)

# He Initialization

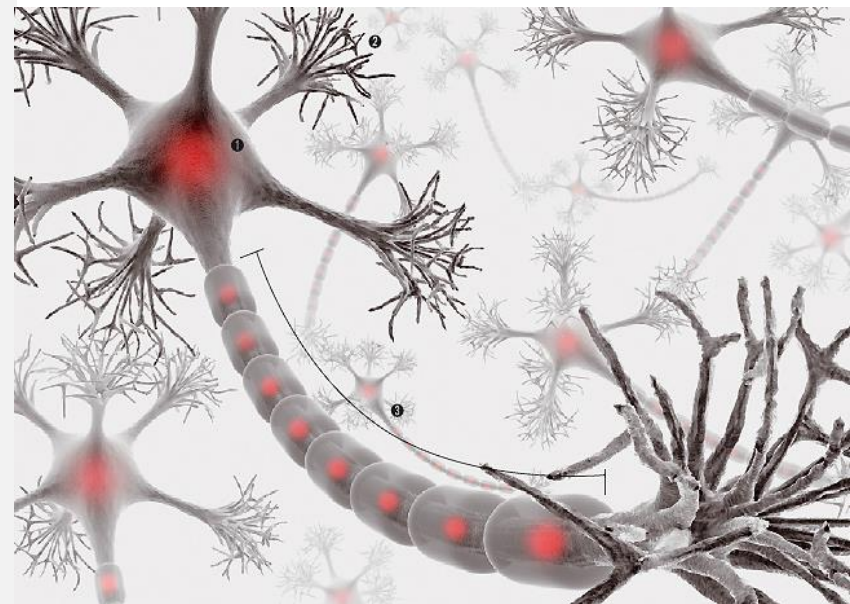
Activation 분포



Activation이 균일하게 분포

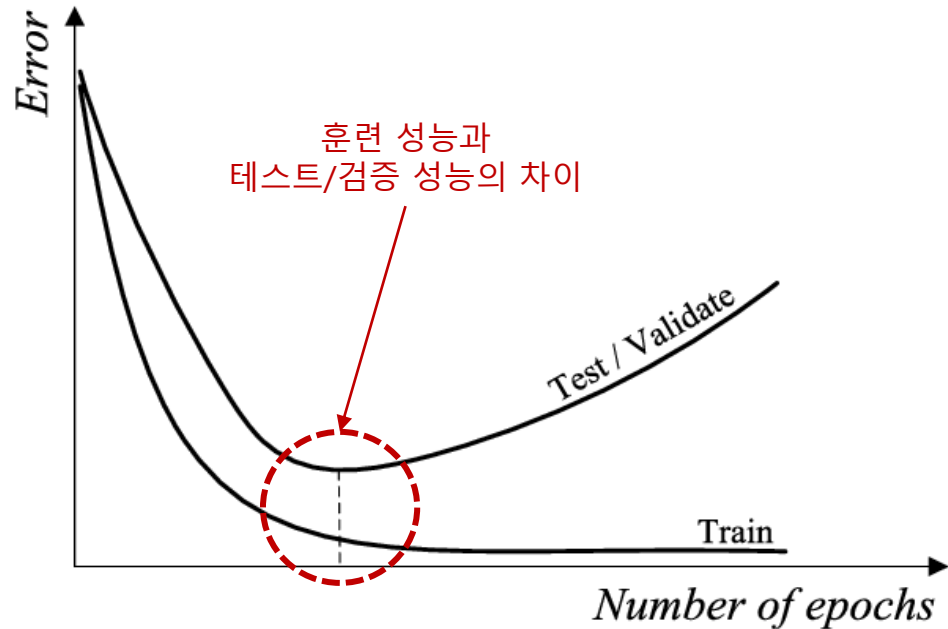


# 정규화



# 정규화 (Regularization)

일반화 오류 (Generalization Error)



## 정규화란?

- 일반화 오류가 최소화 되도록 학습 알고리즘을 보완하는 기법
- 일반화 오류가 작다는 것은 새로운 입력에 대해 얼마나 잘 예측을 잘 하는가를 의미
- Underfitting이나 Overfitting을 막는 방법

Optimization

+

Regularization

# 정규화 기법

✓ 배치 정규화 (Batch Normalization)

✓ 가중치 감소 (Weight Decay)

✓ 학습 조기 종료 (Early Stopping)

✓ 데이터 확장 (Data Augmentation)

✓ 드롭아웃 (Dropout)

✓ 앙상블 (Ensemble)

노이즈 추가 (Noise Robustness)

다중 태스크 학습 (Multi-task learning)

파라미터 공유 (Parameter Sharing)

다양한 연구가 활발히 진행 중

# Internal Covariate Shift

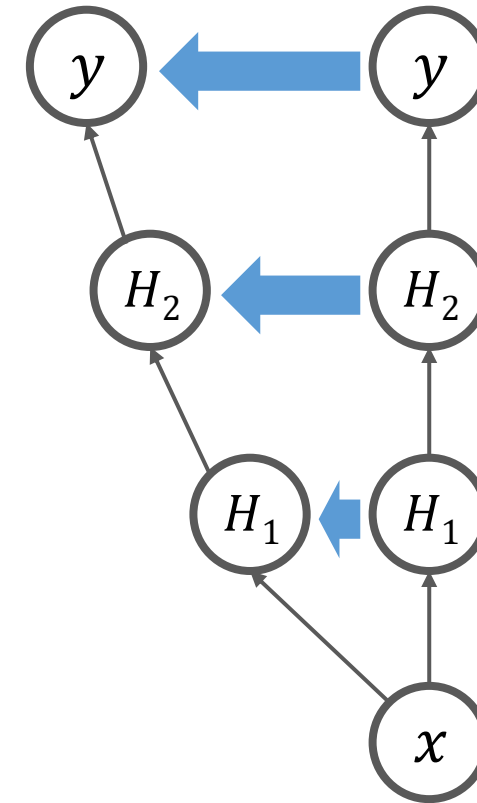
## Deep Neural Net의 학습이 어려운 이유는?

- 훈련을 할 때마다 입력 데이터의 분포가 각 계층에서 Shift 되어 원래 분포에서 멀어지게 됨

“Internal covariate shift”

작은 학습률 사용  
가중치를 신중하게 초기화

But, 학습 속도가 느리고 어렵다!



하위 계층의 작은 오차가 상위 계층으로 갈수록 큰 영향을 주게 됨

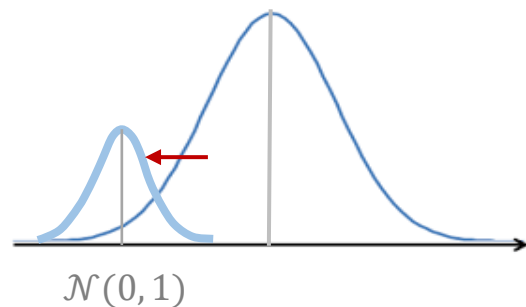
# 배치 정규화

모든 계층의 데이터 분포가 Zero-Mean Unit-Variance가 되게 해보자!

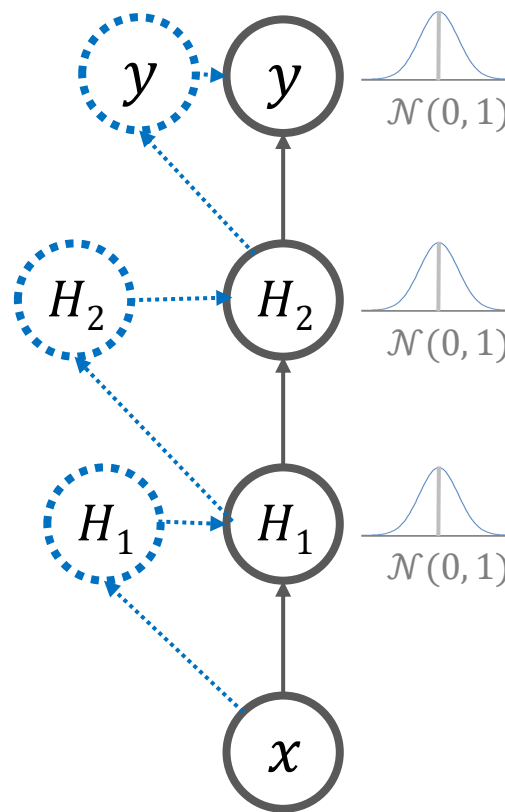
각 뉴런에서 입력 데이터를 정규화

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

입력 데이터의 각 차원 별로 평균과 분산을 구해서  $\mathcal{N}(0, 1)$  정규화



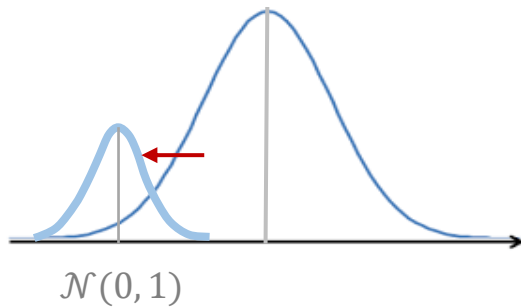
데이터 분포가 동일한 분포로 유지됨



# Scale and Shift

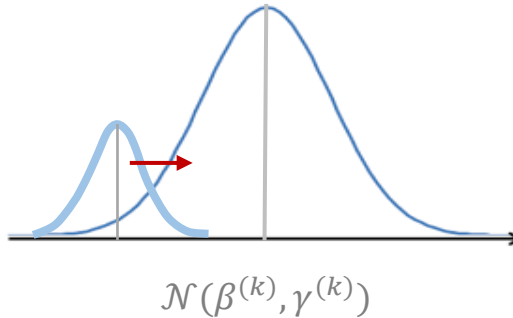
내부 공변량이 이동이 제거된 원래의 분포로 만들어 보자!

## Normalize



$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

## Scale and Shift



$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

## Mean & Variance Learning

+

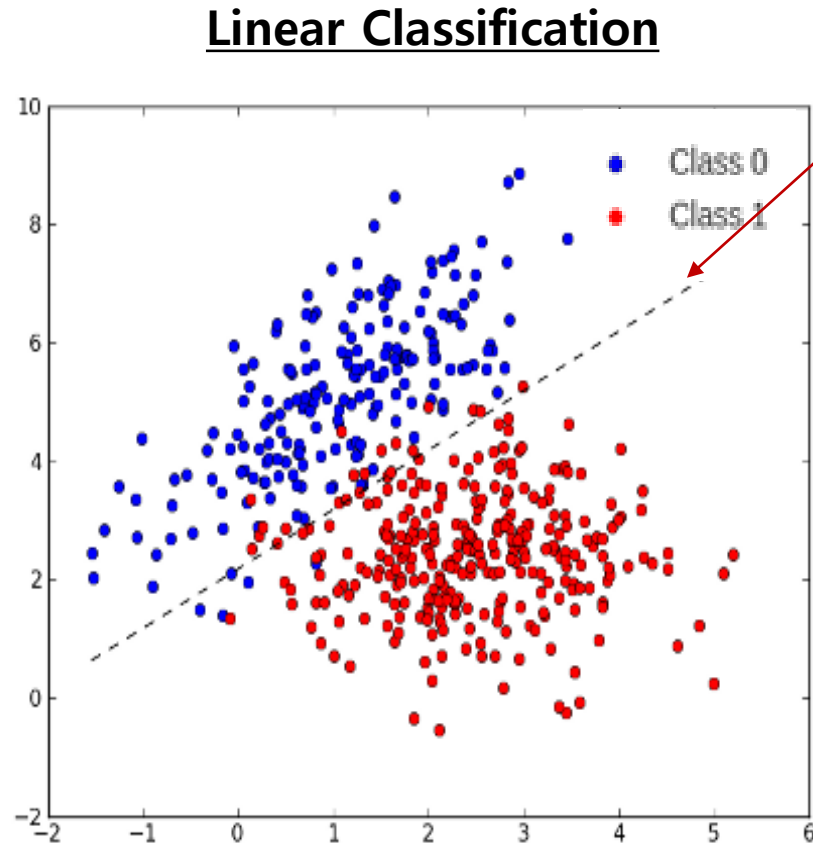
$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathbb{E}[x^{(k)}]$$

데이터 분포를 가장 잘 표현하는  
평균과 분산을 학습

**Extra Flexibility!**

# 어떤 가중치가 좋은가?



결정 경계 (Decision boundary) :

$$wx = b$$



양 변에 2를 곱하면 해인  $x$ 가 달라지는가?

$$2wx = 2b$$

Q.  $w$ 와  $2w$  중 어떤 것이 좋은 가중치인가?

$w$ 가 작을 수록 variance를 줄어들어  
최적화가 용이해지고 과적합이 방지된다.

# 가중치 감소 (Weight Decay)

$$\tilde{J}(W; X, y) = \underbrace{J(W; X, y)}_{\text{Data Loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}} \quad \lambda : \text{정규화 상수}$$

**Data Loss**

모델 오차가 최소화  
되도록 예측하게 함

**Regularization**

가중치 크기를 작게 만들어  
모델이 과적합 되지 않게 함



# Regularizer 종류

## $L_2$ Regularizer

$$\tilde{J}(\mathbf{W}; \mathbf{X}, \mathbf{y}) = J(\mathbf{W}; \mathbf{X}, \mathbf{y}) + \frac{\lambda}{2} \|\mathbf{W}\|_2^2$$

리지 회귀 Ridge regression

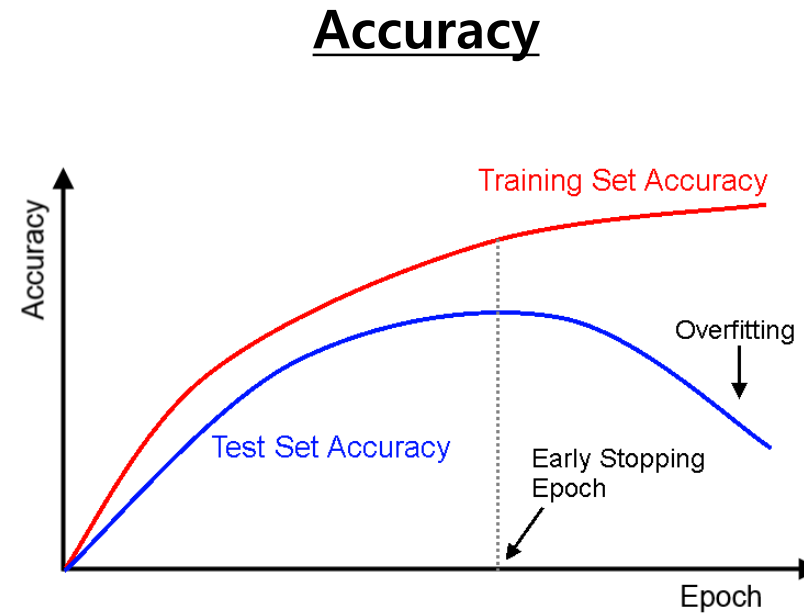
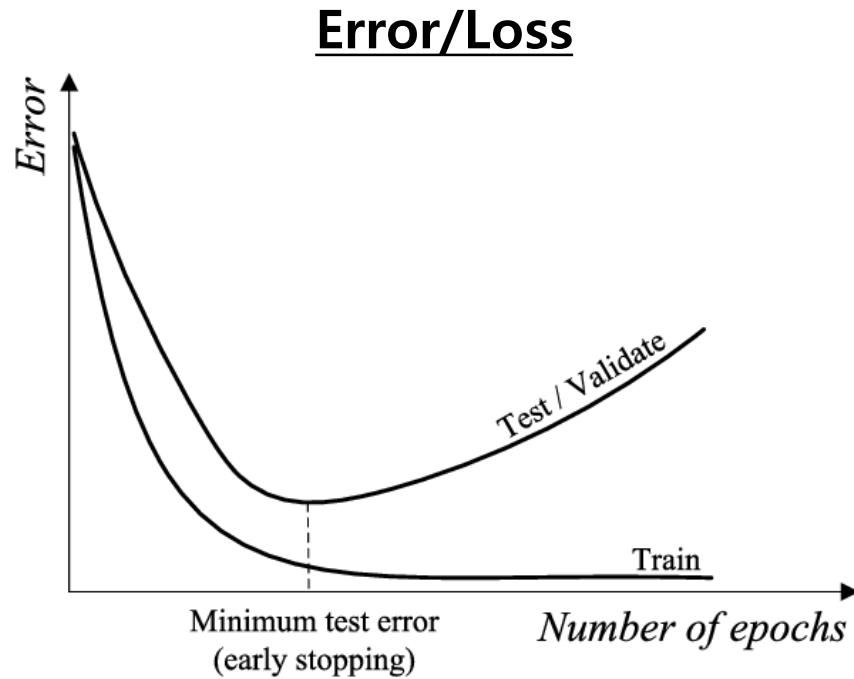
## $L_1$ Regularizer

$$\tilde{J}(\mathbf{W}; \mathbf{X}, \mathbf{y}) = J(\mathbf{W}; \mathbf{X}, \mathbf{y}) + \lambda \|\mathbf{W}\|_1$$

라소 회귀 Rasso regression

Parameter Norm Penalty 형태로 Prior에 따라 다양한 Norm을 사용할 수 있음

# 조기 종료 (Early Stopping)



- 훈련을 하면서 주기적으로 검증을 해서 오차가 높아지거나 정확도가 내려가면 종료
- 가장 성능이 좋은 모델의 스냅샷을 저장해 두었다가 사용

# 데이터 확장 (Data Augmentation)

일반화를 위해 가장 좋은 방법은 많은 데이터로 훈련시키는 것!

## 데이터 확장이 결합된 학습 과정



Classification과 같은  
transformation invariant task에 매우 적합

# 이미지 데이터 확장 기법

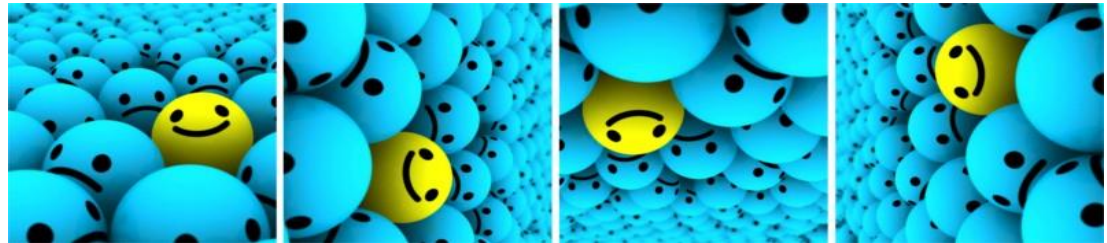
여러 방법들을 혼합해서 사용

- translation
- rotation
- stretching
- shearing
- lens distortions (warping)

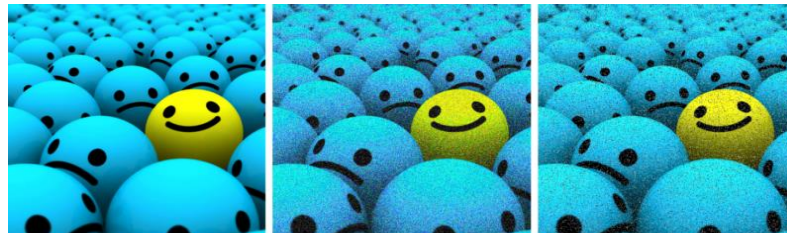
Translation



Rotation



Gaussian Noise Injection

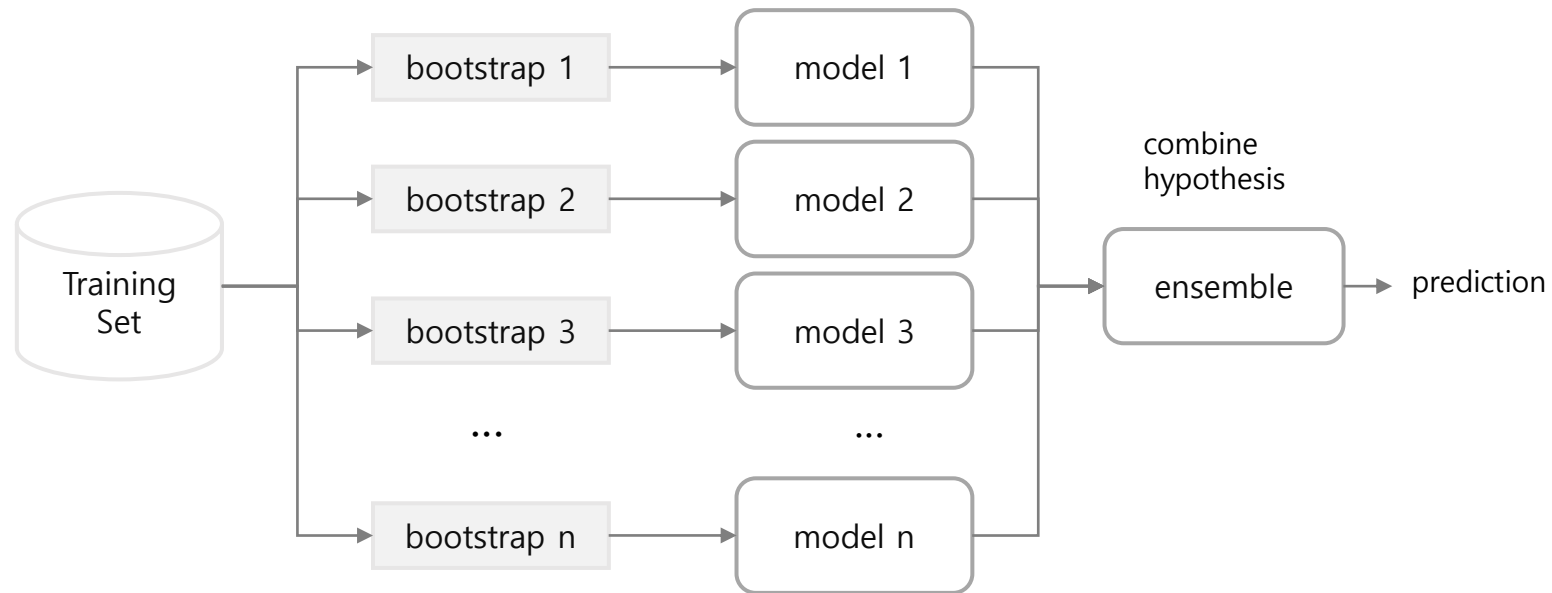


<https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced>

# 앙상블 (Ensemble)

“여러 단순한 모델을 합쳐서 예측 정확도를 높이는 방법”

## 배깅 (Bagging)

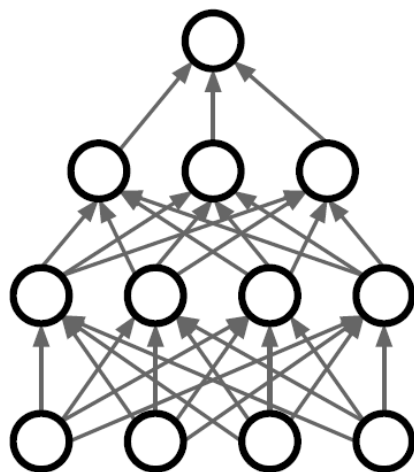


- 일종의 Model Averaging 기법
- 모델의 종류는 상관 없음
- 데이터는 부트스트랩 방식으로 모델의 개수만큼 생성해서 병렬 실행
- 각 모델의 결과는 Voting을 거쳐서 결정

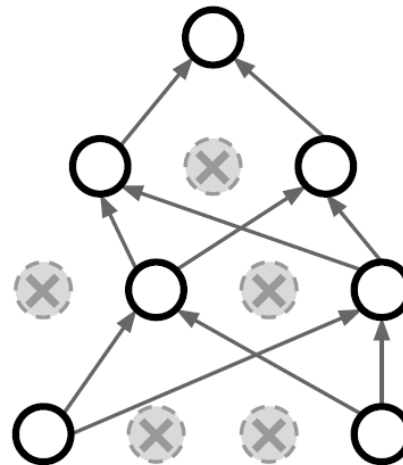
# 드롭아웃 (Dropout)

“한 신경망 모델에서 무한히 많은 모델을 생성하는 일종의 배깅(Bagging) 방법”

신경망



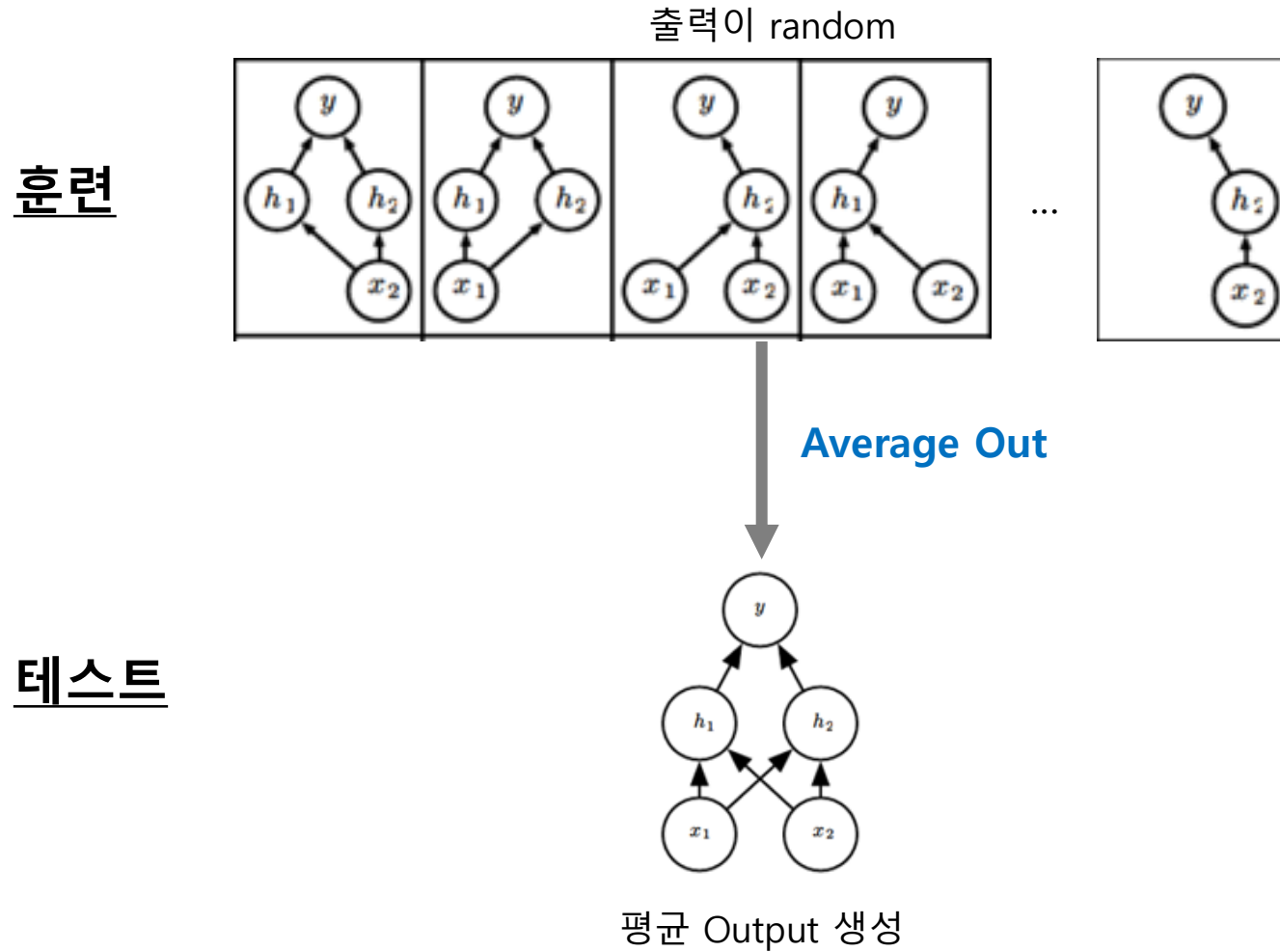
드롭아웃 적용 신경망



- 미니 배치 실행 시마다 뉴런을 랜덤하게 dropout해서 새로운 모델을 만드는 방법
- 계산 시간이 거의 들지 않고 다양한 모델에 대해 정규화 할 수 있는 강력한 방법
- 배깅과 다른 점은 파라미터를 모든 모델이 공유한다는 점

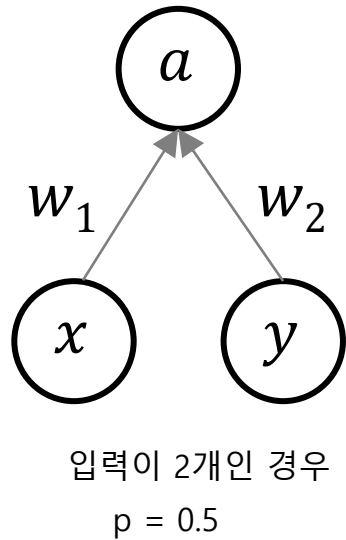
“Dropout: A simple way to prevent neural networks from overfitting”, Srivastava et al, JMLR 2014

# 추론 방식



**Output at test time = Expected output at training time**

# 추론 방식



## 훈련

$$\begin{aligned}\mathbb{E}[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \boxed{\frac{1}{2}}(w_1x + w_2y)\end{aligned}$$

평균을 구하게 되면 Dropout의 뉴런 유지 확률과 동일

## 테스트

$$\mathbb{E}[a] = w_1x + w_2y \longrightarrow \mathbb{E}[a] = \boxed{\frac{1}{2}}(w_1x + w_2y)$$

가중치에 Dropout의 확률을 곱해 줌

**Weight scaling inference rule**



**Thank you!**

