

# Optimization & Regularization

모두의연구소 Rubato Lab.  
소준섭



# Optimization

# Dataset and batch size

모델은 큰 데이터셋을 어떻게 학습할까?

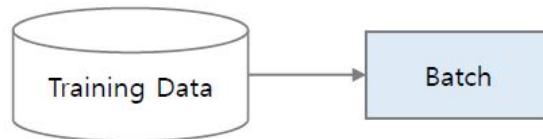


- 하드웨어의 한계 때문에 모든 데이터를 한번에 올리기 힘든 단점 존재
- 데이터를 나눠서 모델에 학습시키는 mini-batch 방식으로 학습

# Dataset and batch size

## Batch

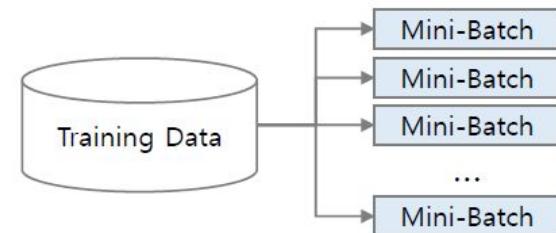
전체 훈련 데이터를 하나의 배치로 만들어 훈련



훈련 집합이 너무 크면 불가능!

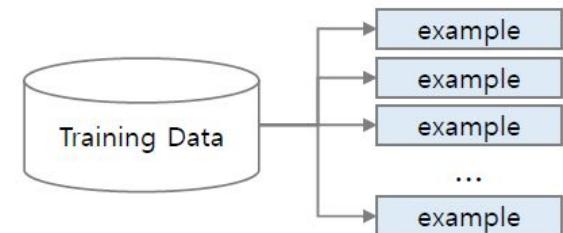
## Mini-Batch

n개 샘플을 묶은 미니배치 단위로 훈련



## Stochastic

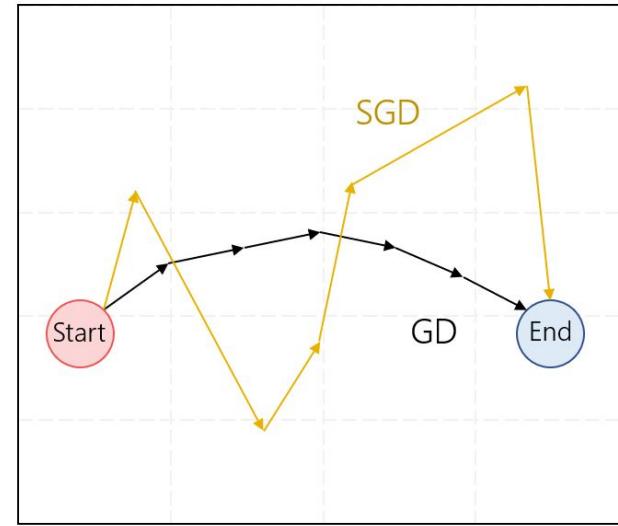
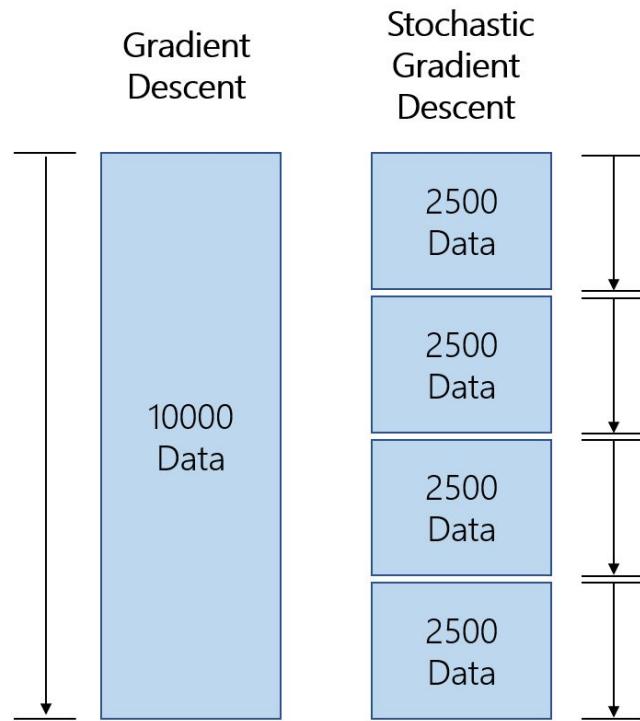
각 example 단위로 훈련



Too Noisy!

거의 유사하다.

# Optimizer



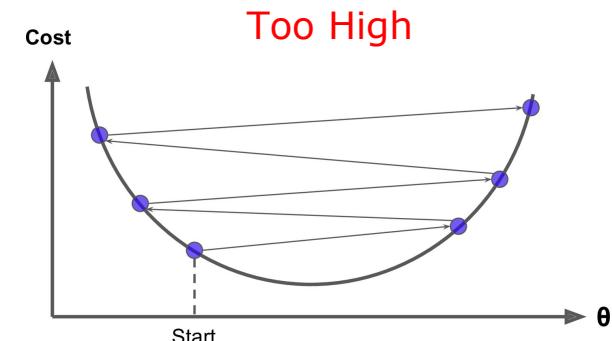
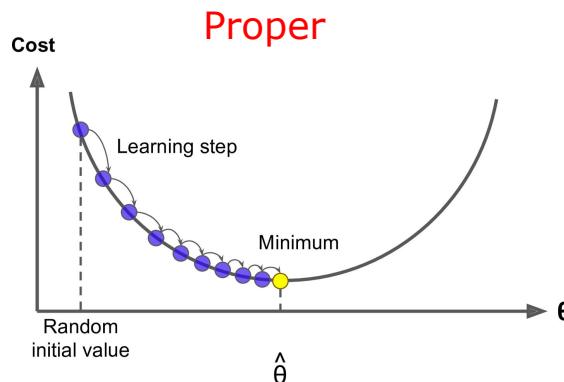
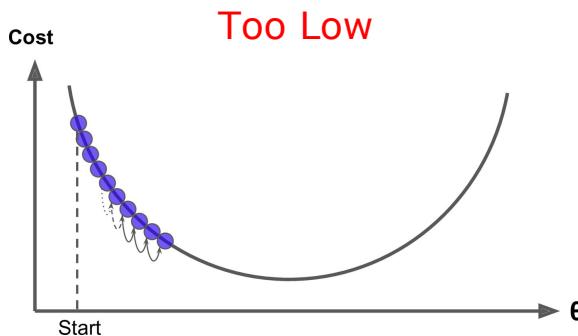
더 비틀비틀 가는 것 같지만  
기존 보다 더 빠르게 학습합니다

# Optimizer

Learning rate에 따라서 모델의 학습 결과가 달라진다.

$$w^+ = w - \eta * \frac{\partial E}{\partial w}$$

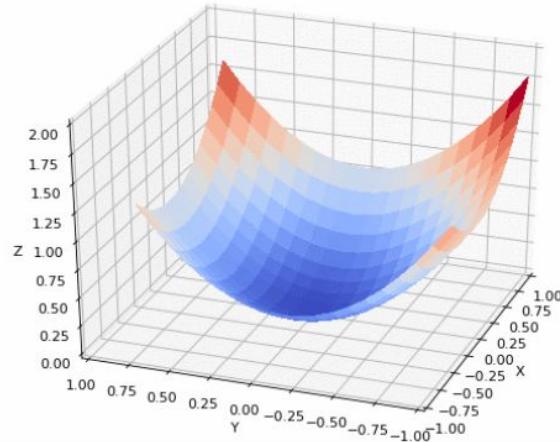
learning rate : 한번에 얼마나 학습할지  
gradient : 어떤 방향으로 학습할지



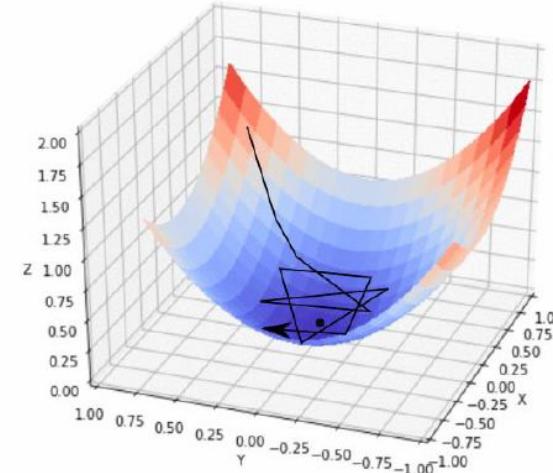
# Optimizer - SGD

SGD는 Gradient에만 의존해 최적화

Gradient Descent 수렴 경로



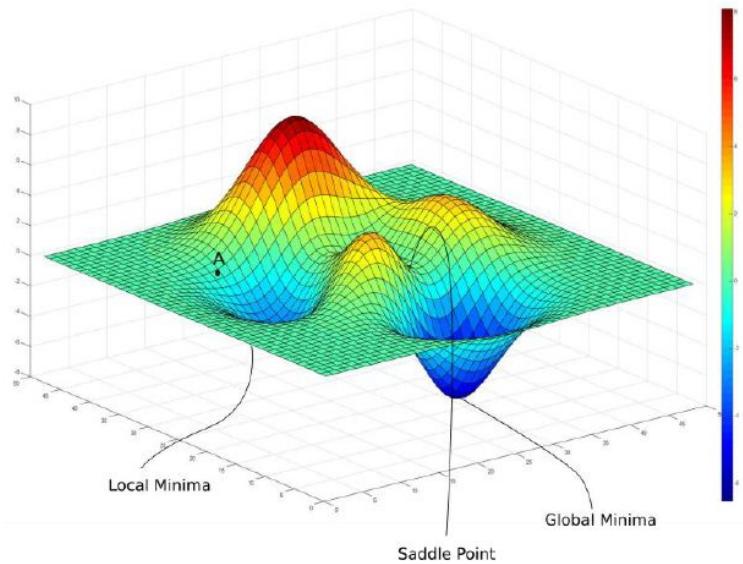
학습률이 큰 경우



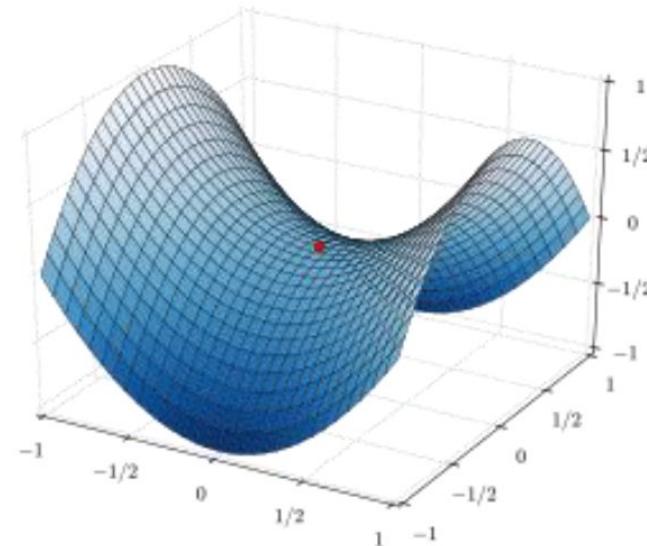
최소점 부근에서 수렴하지 못하고 진동

# Optimizer - SGD

**Saddle Point** 나 **Local Minima**에서 잘 탈출하지 못함



Saddle Point



# Optimizer - SGD + Momentum

## SGD + Momentum

이전 단계의  
Velocity

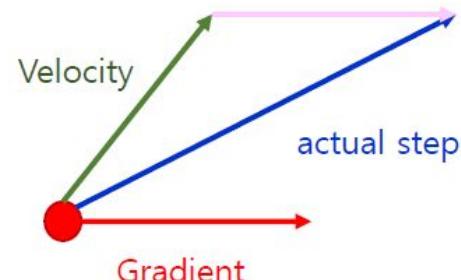
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

현재 Gradient

Velocity = 누적 Gradient

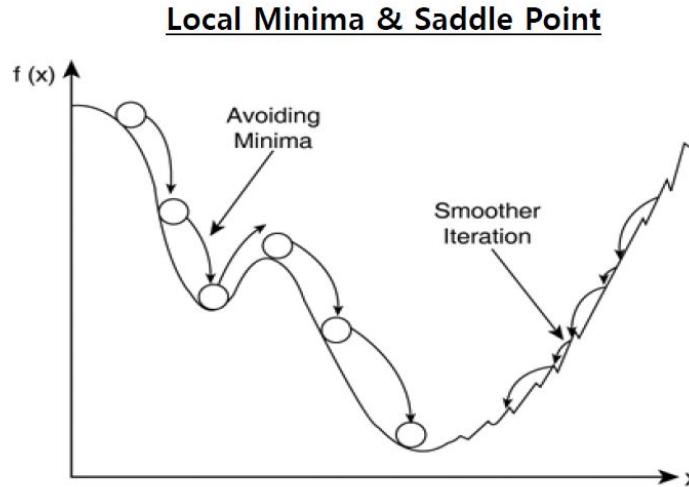
- $\rho$  : “friction” 마찰 계수로 0.90이나 0.99를 사용

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



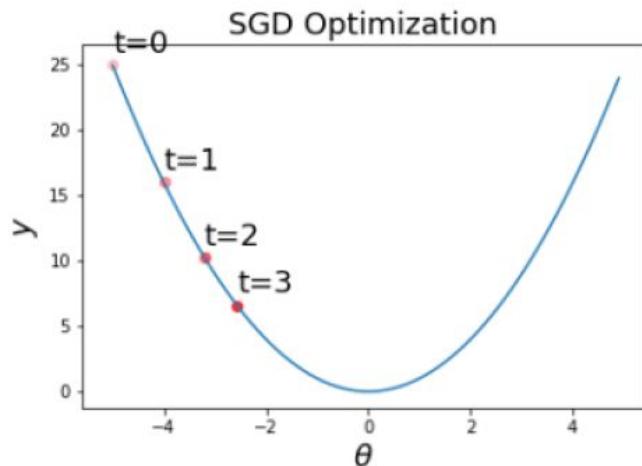
# Optimizer - SGD + Momentum

**Momentum**의 도움으로 조금 더 부드럽게 학습할 수 있다.

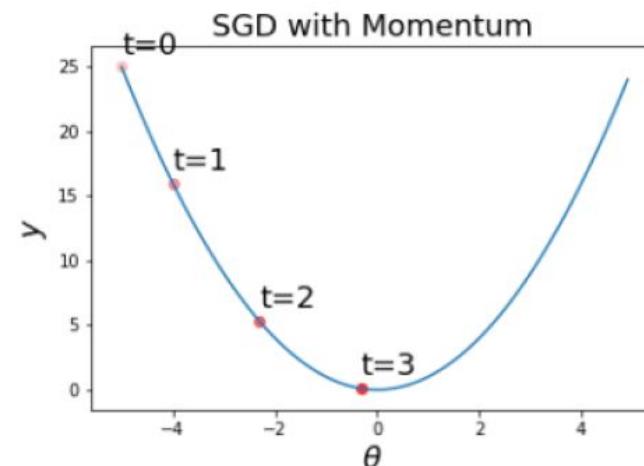


# Optimizer - SGD + Momentum

SGD



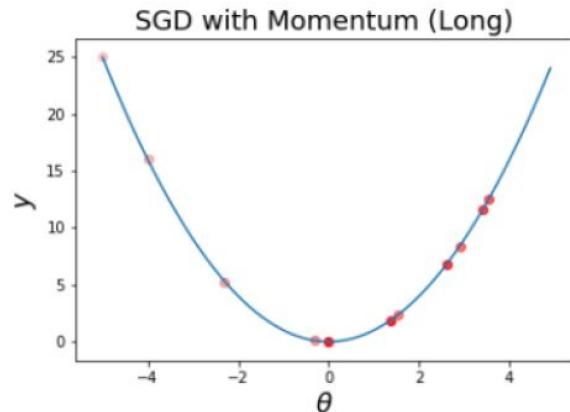
SGD+Momentum



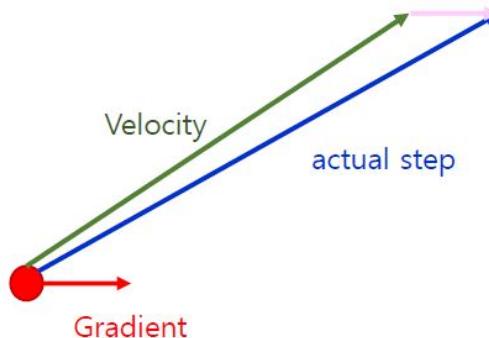
- + Gets to the optimal quicker

# Optimizer - SGD + Momentum

## SGD+Momentum

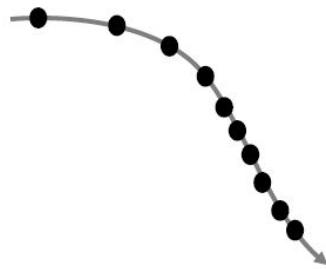


- 최소점에 도달한 이후에도 Overshooting 될 수 있음
- 즉, 현재 Gradient가 작더라도 Velocity가 크면 최적화가 계속 진행됨

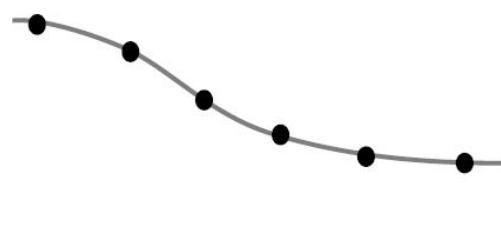


# Optimizer - AdaGrad

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?

# Optimizer - AdaGrad

총 Gradient 크기는 전체 변화량이므로 이것으로 학습률을 정해보자!

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

총 Gradient 값 이용 각각 Learning rate를 가지는 효과

$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\|\nabla f(x)\|_2}$$



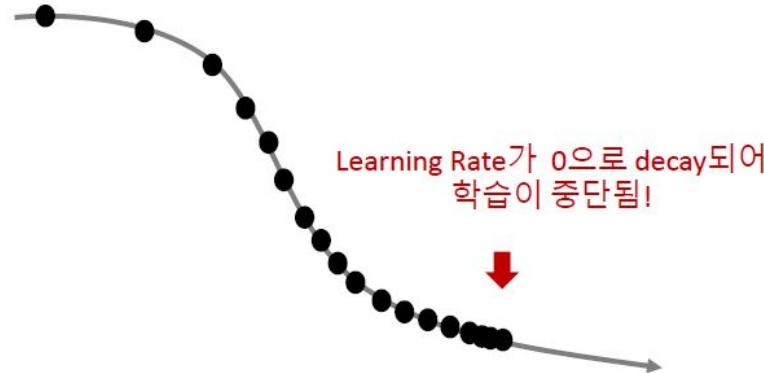
$$r_{t+1} = r_t + \nabla f(x_t)^2$$
$$x_{t+1} = x_t - \frac{\alpha \nabla f(x_t)}{\sqrt{r_{t+1}} + \epsilon}$$

$\epsilon$  : 분모가 0이 되기 않게 더해주는 상수

AdaGrad : adaptive gradient algorithm

# Optimizer - AdaGrad

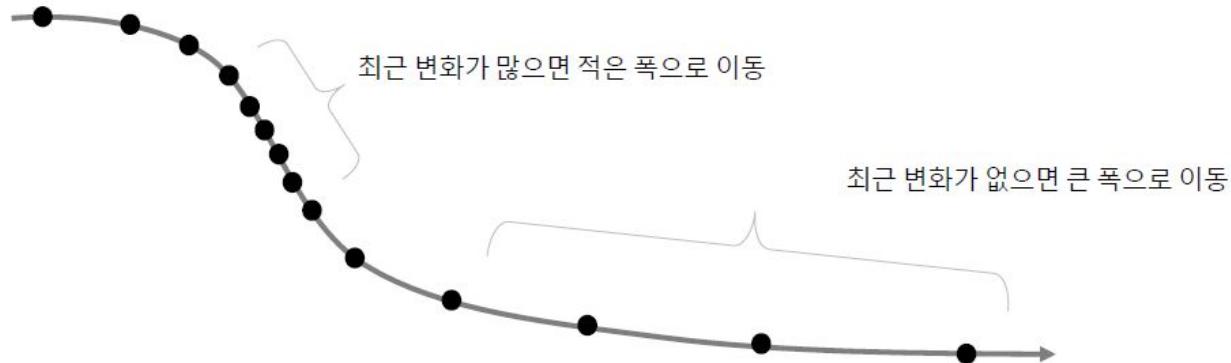
경로가 길어질수록 총 Gradient 크기가 점점 커지는 문제 발생



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

# Optimizer - RMSProp

최근 변화량을 중심으로 이동 폭을 정해보자



# Optimizer - 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)

RMSProp : Root Mean Square Propagation

# Optimizer - RMSProp

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

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

$$r_{t+1} = \alpha r_t + (1 - \alpha) \nabla f(x_t)^2 \quad r_t = \alpha r_{t-1} + (1 - \alpha) \nabla f(x_{t-1})^2 \text{ 를 대입}$$

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

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

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

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

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

최근 변화(Gradient)는 많이 반영됨

오래전 변화(Gradient)는 적게 반영됨

# Optimizer - Adam

Adaptive Moments 전략

Momentum

+

Adaptive  
Learning Rate

=

Adam

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

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

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

$\epsilon$  : 분모가 0이 되기 않게 더해주는 상수

# Optimizer - Adam

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

$$v_0 = 0, r_0 = 0$$

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

# Optimizer - Adam

$$v_0 = 0, r_0 = 0$$

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

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

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

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

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

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

- 현재 **Adam**이 기본적으로 사용되는 Optimizer

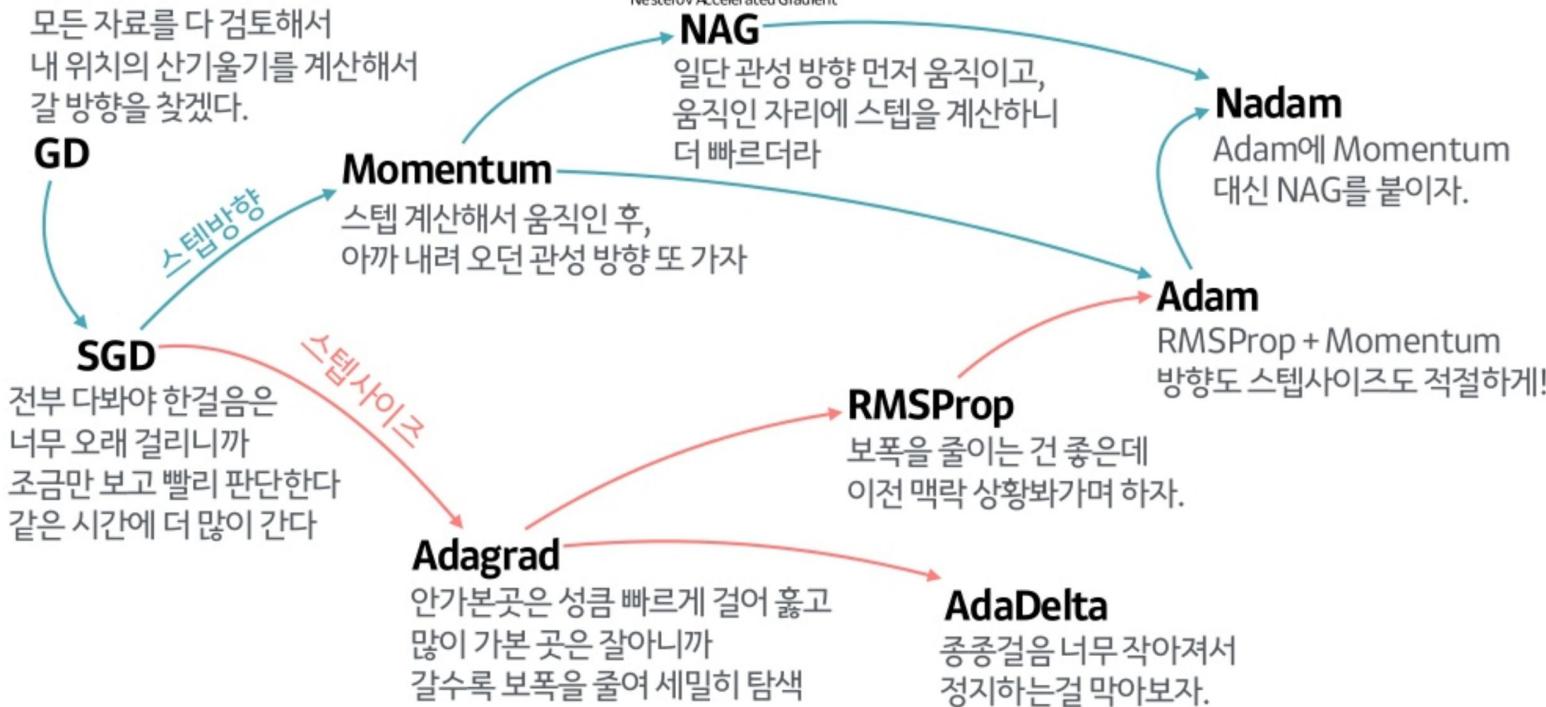
훈련 초반에 발생하는 편향을 제거

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

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

- t가 커질수록 분모항이 1로 수렴하므로 원래의 식으로 복원됨

# Optimizer



# Regularization

# Initialization

A portrait photograph of Geoffrey Hinton, a middle-aged man with grey hair, wearing a dark grey V-neck sweater over a white collared shirt. He is looking directly at the camera with a neutral expression. The background is blurred green foliage.

deeplearning.ai presents  
Heroes of Deep Learning

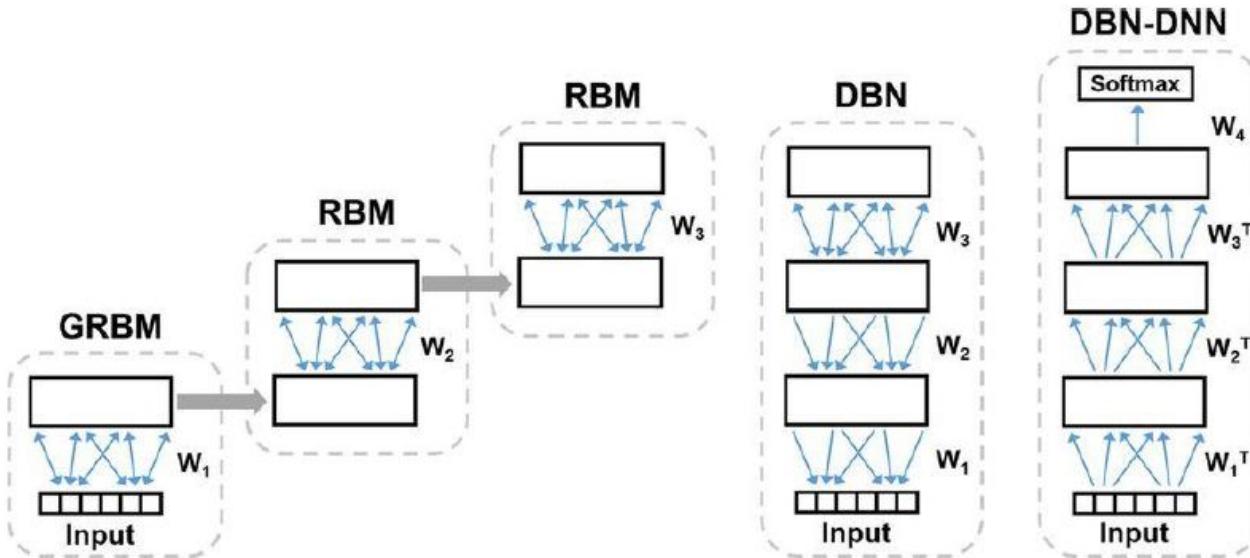
## Geoffrey Hinton

Professor at the University of Toronto  
& Researcher at Google Brain

**Deep Neural Network**를 잘 학습시킬 수 있는 방법 제시



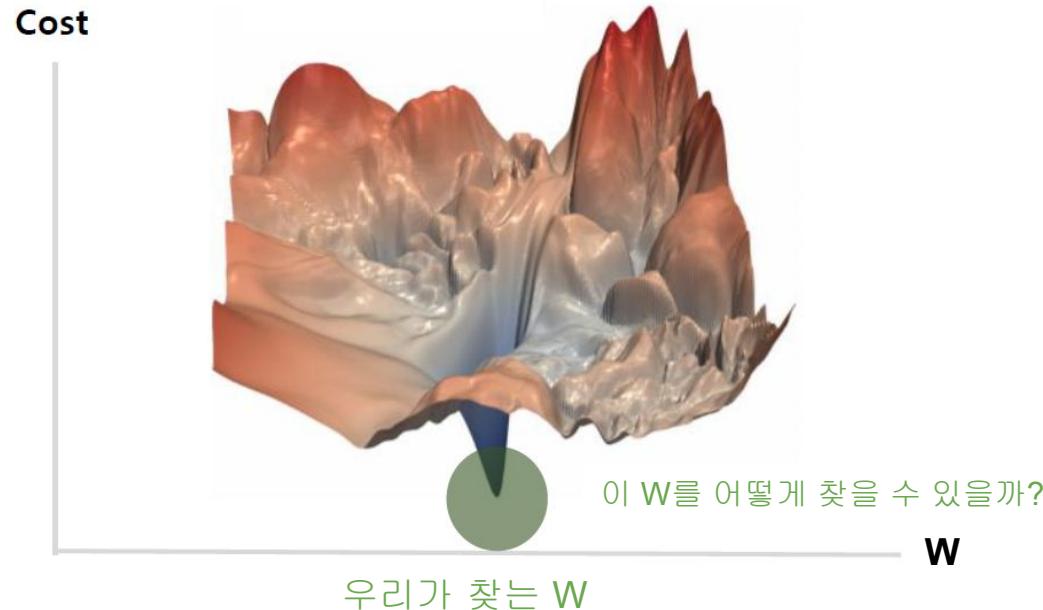
# Initialization



- Restricted Boltzmann machine (RBM) 을 이용한 weight 초기화
- 위 방법으로 모델을 잘 학습할 수 있음을 증명, 이후 다양한 초기화 방법 등장

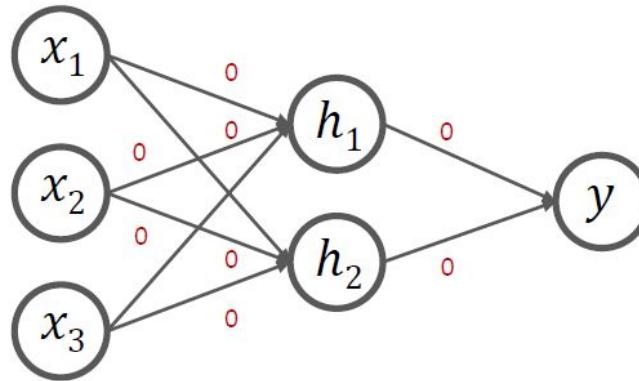
# Initialization

**Starting point**를 어디로 잡을지의 문제



# Initialization

가중치를 0으로 초기화하면 어떻게 될까?

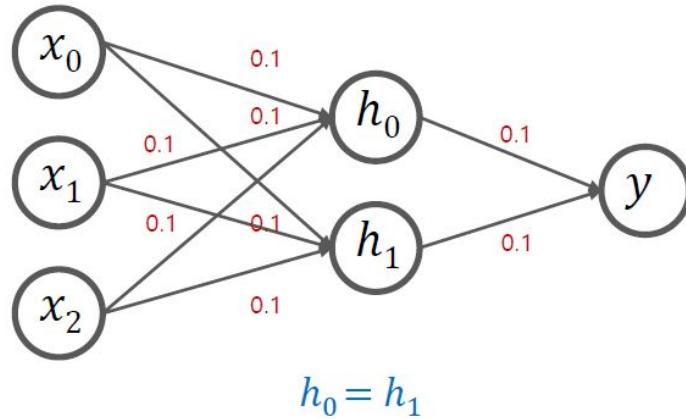


- 모든 뉴런의 가중 합산 결과가 0이 되어 출력이 상수가 됨
- 그래디언트가 0이 되어 학습이 진행되지 않음

$$n = \mathbf{w}^T \mathbf{x} + b = 0 \quad \frac{\partial n}{\partial \mathbf{x}} = \mathbf{w} = 0$$

# Initialization

가중치를 동일한 상수로 초기화하면 어떻게 될까?

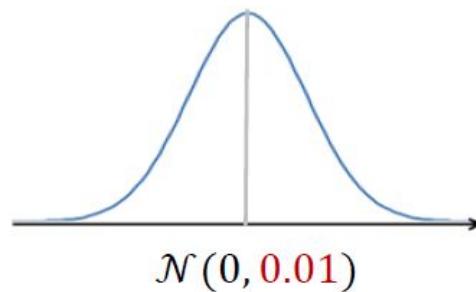


Network에 Symmetry가 생겨서 모든 뉴런이 똑같은 방식으로 작동

- Layer에 있는 모든 뉴런이 동일한 출력과 동일한 그래디언트를 갖게 됨
- 여러 뉴런을 사용하는 효과가 없어지고 하나의 뉴런만 있는 것처럼 작동
- 학습이 적절히 진행되지 않음

# Initialization

가중치를 아주 작은 난수로 초기화하면 어떻게 될까?

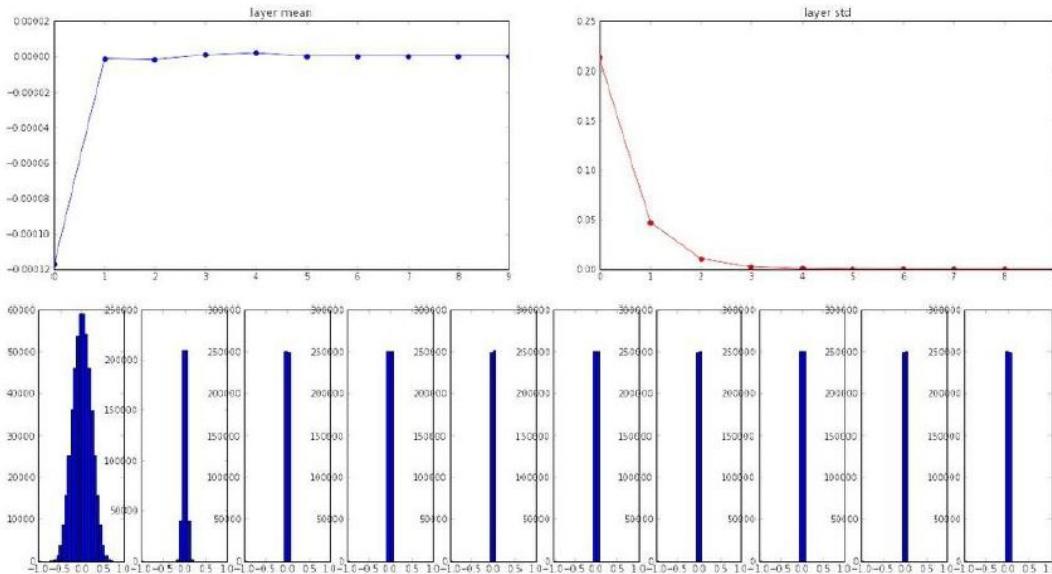


$$W = 0.01 * \text{np.random.randn}(D, H)$$

작은 네트워크에는 작동하지만 네트워크가 깊어질수록 문제 발생!

# Initialization

## Activation 분포



Activation이 점점 0으로 변화

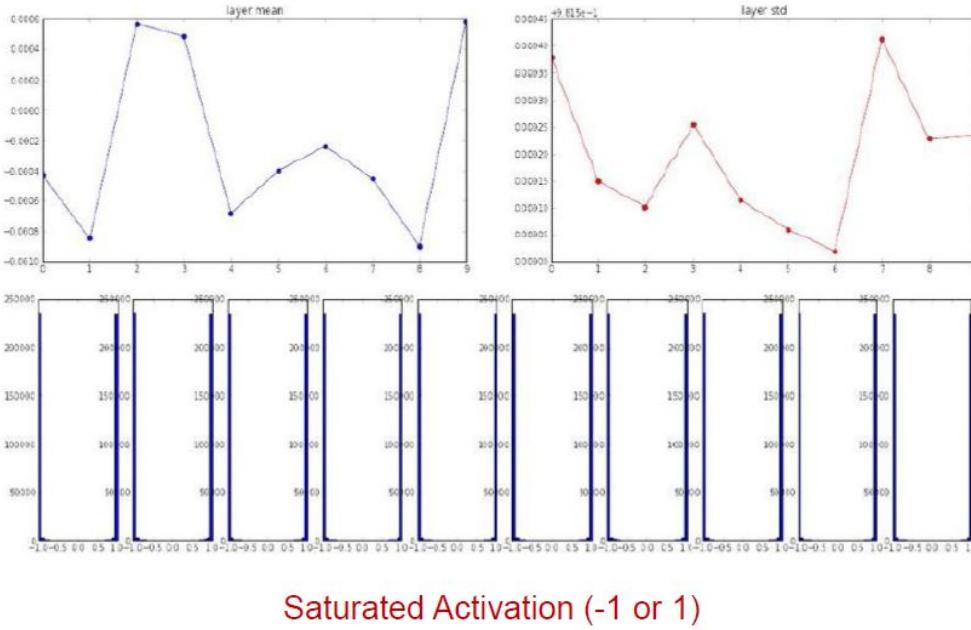
- Layer가 10개
- Layer 별 뉴런은 500개
- tanh 사용
- 초기화 : **분산이 0.01**

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

Activation이 0이면 Gradient도 0!  
학습이 진행되지 않음!

# Initialization

## Activation 분포



- 초기화 : 분산이 1

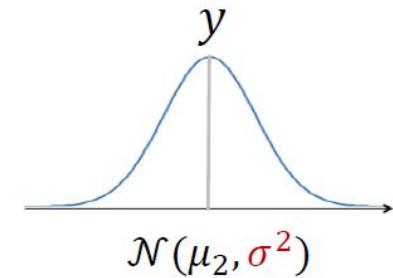
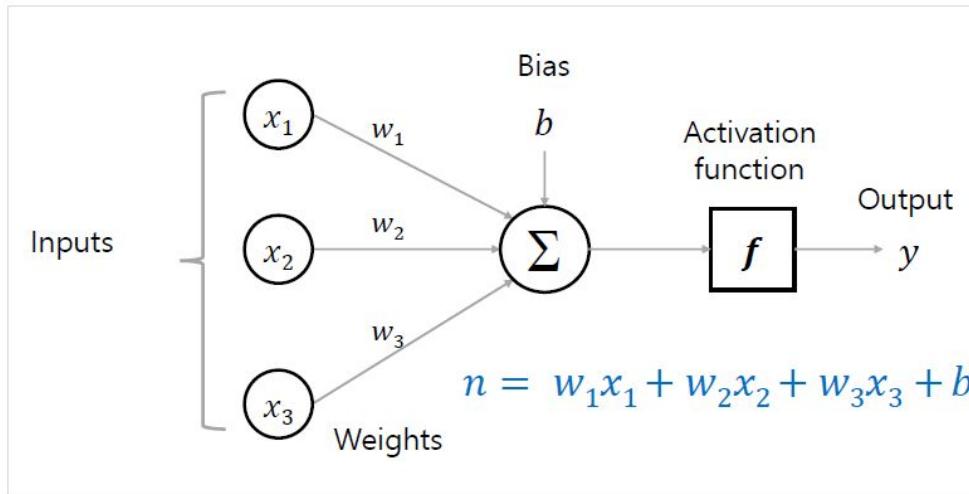
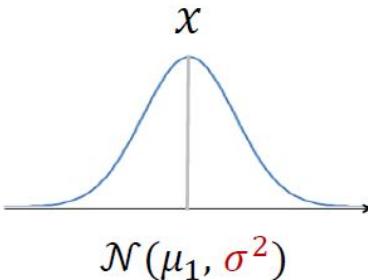
$$W = 1.0 * np.random.randn(fan\_in, fan\_out)$$

- 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!



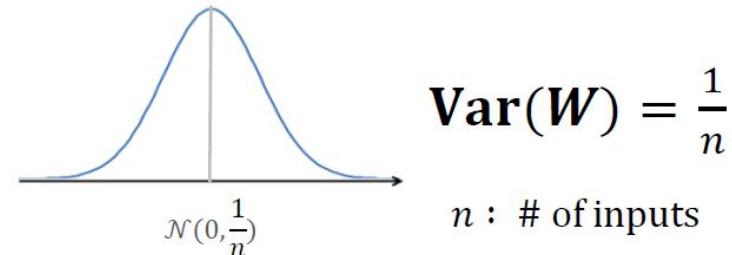
# Xavier Initialization

$x$ 의 분산과  $y$ 의 분산을 같게 하려면?

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

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

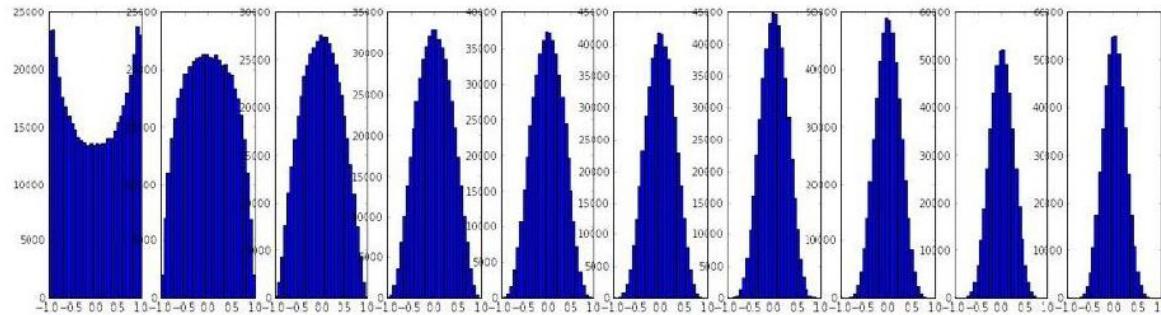
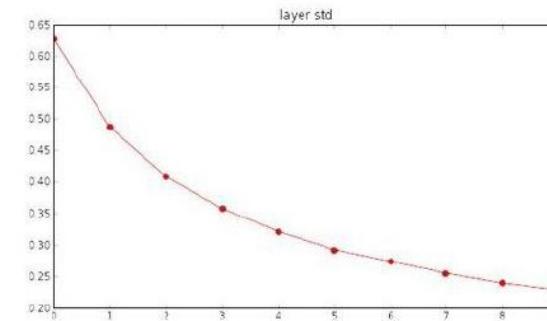
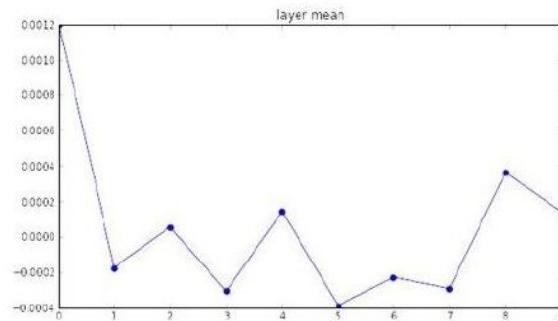
$$\text{Var}(w_i) = \frac{1}{n}$$



가중치의 분산을 입력 개수에 반비례하게 만든다!

# Xavier Initialization

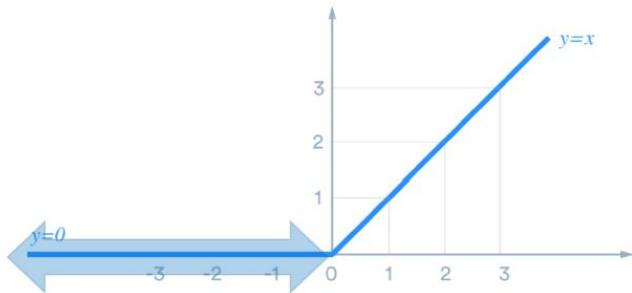
Activation 분포



각 Layer마다 Unit-Gaussian Input 및 Output을 근사하게 됨

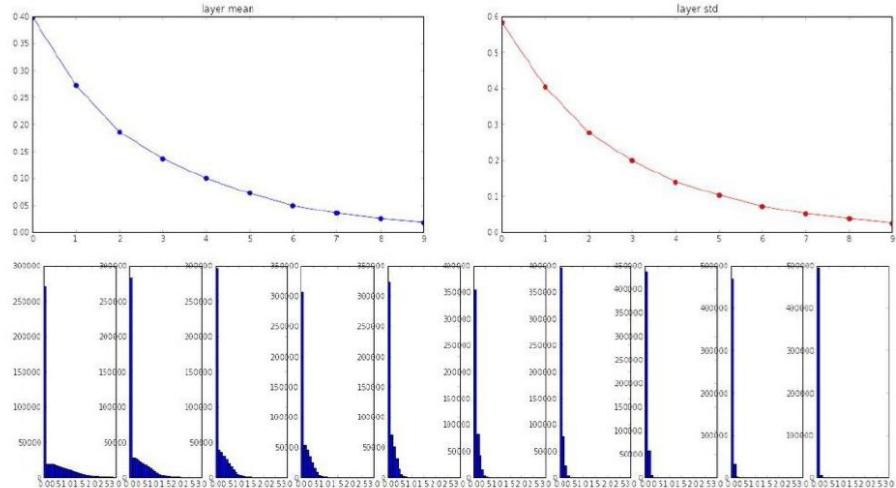
# Xavier Initialization

## Non-Linear Activation인 ReLU를 사용하면?



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

Activation 분포

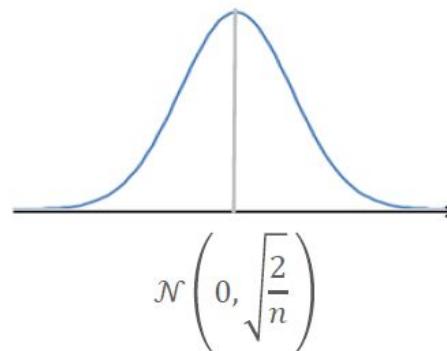


Activation이 점점 0으로 수렴

# He Initialization

가중치의 분산을 2배로 키워서 Activation을 넓게 퍼지게 만듬

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

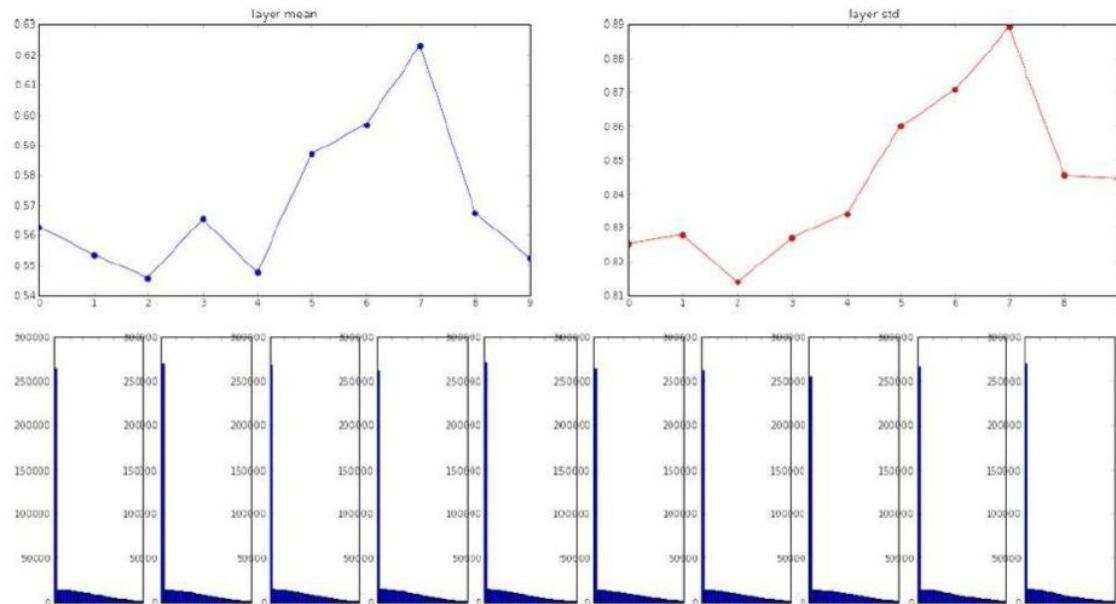


$$\text{Var}(W) = \frac{2}{n}$$

$n$  : # of inputs

# He Initialization

Activation 분포

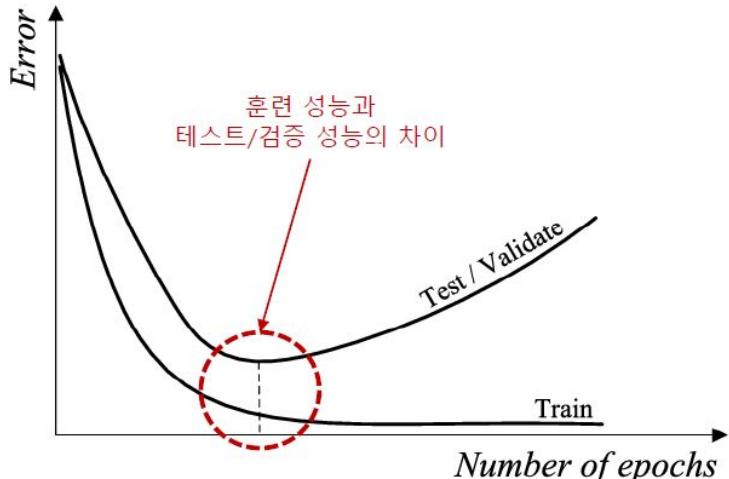


Activation이 균일하게 분포



# 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

# Regularization

- Batch Normalization
- Weight Decay
- Early Stopping
- Data Augmentation
- Dropout
- Ensemble

# Batch Normalization

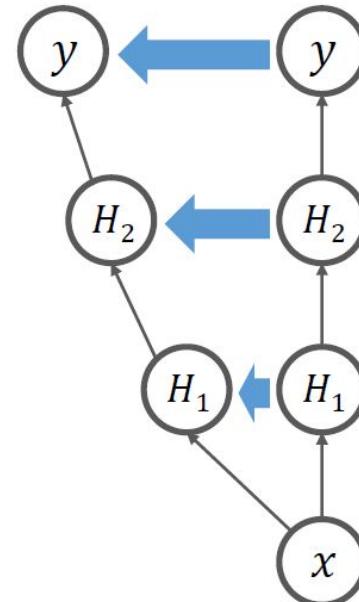
## Deep Neural Net의 학습이 어려운 이유는?

- 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!

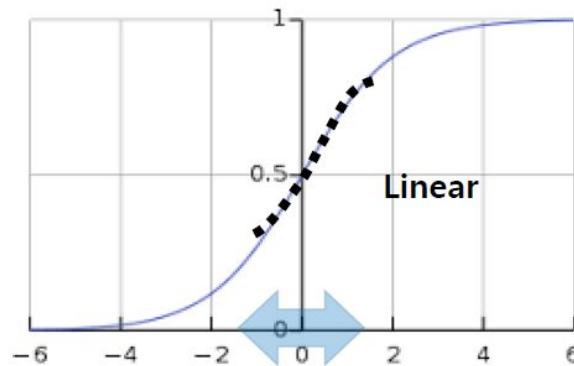


하위 계층의 작은 오차가 상위 계층  
으로 갈수록 큰 영향을 주게 됨

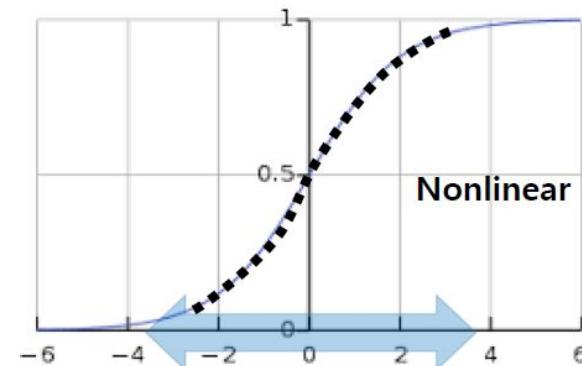
# Batch Normalization

입력 분포를 Zero-Mean Unit-Variance로 만드는 것이 최선일까?

Zero-Mean Unit-Variance



Scale and Shift



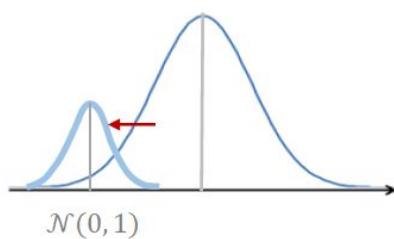
$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Sigmoid는 Unit-Gaussian에서 비성형성을 잃어버리므로 분산이 좀 더 큰 것이 좋다.

# Batch Normalization

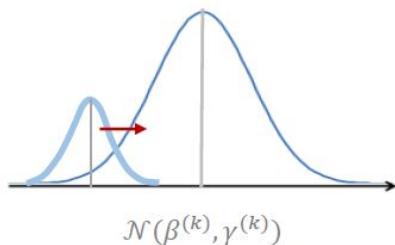
각 Batch data를 Normalize 후 Scale and Shfit

## Normalize



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

## Scale and Shift



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

## Mean & Variance Learning

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

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



데이터 분포를 가장 잘 표현하는  
평균과 분산을 학습

Extra Flexibility!

# Batch Normalization

## Training

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;  
Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

**훈련 시점 :**  
미니 배치 단위로 정규화 수행

**테스트 시점 :**  
전체 데이터의 평균과 분산으로 정규화

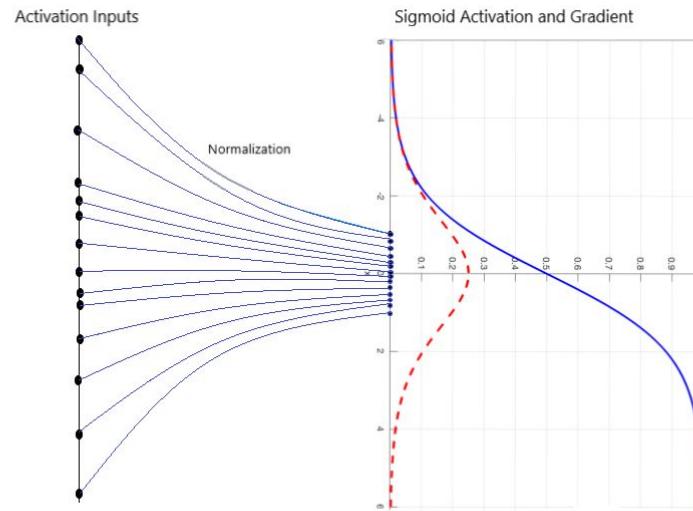
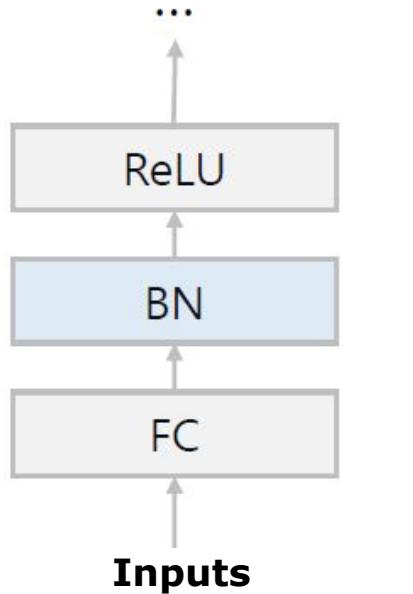
$$E[x] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$Var[x] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

- 평균 : 미니 배치 평균들의 평균
- 분산 : 미니 배치 분산들의 평균 \*  $\frac{m}{m-1}$

# Batch Normalization

Layer와 Activation 사이에 위치

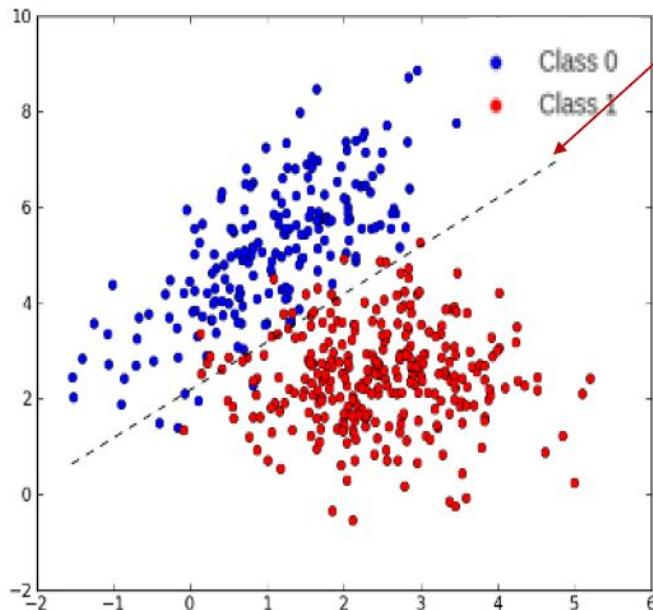


# Regularization

- 초기화 의존도가 낮아짐
  - Weight가 layer들을 지나도 살아 있다.
  - 정규화 효과
    - Mini-batch data의 distribution을 이용해 정규화
    - batch의 구성에 따라서 입력 값이 stochastic 해진다.
    - 입력 값이 더 이상 deterministic하지 않다.
- 
- 학습이 잘된다.**

# Weight decay

## Linear Classification



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.

# Weight decay

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

모델 오차가 최소화  
되도록 예측하게 함

Data Loss

Regularization

가중치 크기를 작게 만들어  
모델이 과적합 되지 않게 함

$L_2$  Regularizer

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

리지 회귀 Ridge regression

$L_1$  Regularizer

$$\tilde{J}(W; X, y) = J(W; X, y) + \lambda \|W\|_1^2$$

라소 회귀 Rasso regression

Parameter Norm Penalty 형태로 Prior에 따라 다양한 Norm을 사용할 수 있음

# Weight decay - l2 example

전체 Loss

$$\mathcal{L} = \mathcal{L}_0 + \lambda \sum_w w^2$$

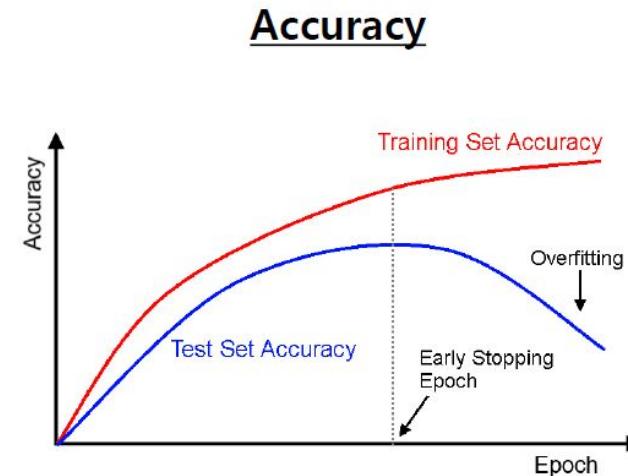
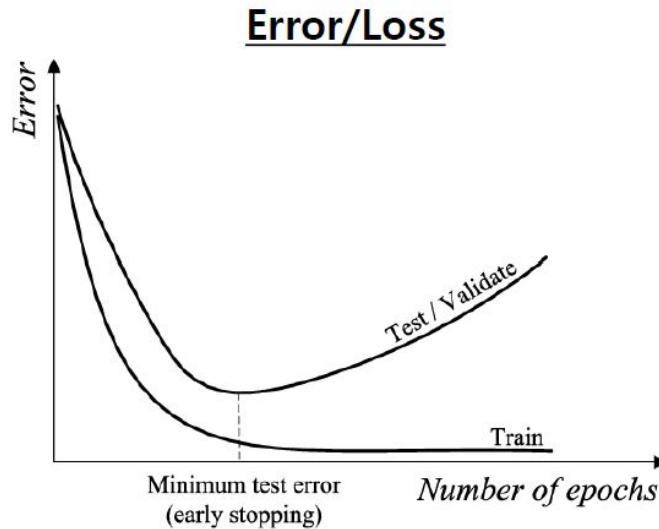
전체 Loss의 미분

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}_0}{\partial w} + 2\lambda w$$

가중치 업데이트

$$\begin{aligned} w &\rightarrow w - \eta \frac{\partial \mathcal{L}_0}{\partial w} - 2\eta \lambda w \quad \bullet \text{ Weight update 시} \\ &= \underline{(1 - 2\eta \lambda) w} - \eta \frac{\partial \mathcal{L}_0}{\partial w} \quad \text{Decay 적용} \\ &\quad \text{Weight Decay} \end{aligned}$$

# Early Stopping

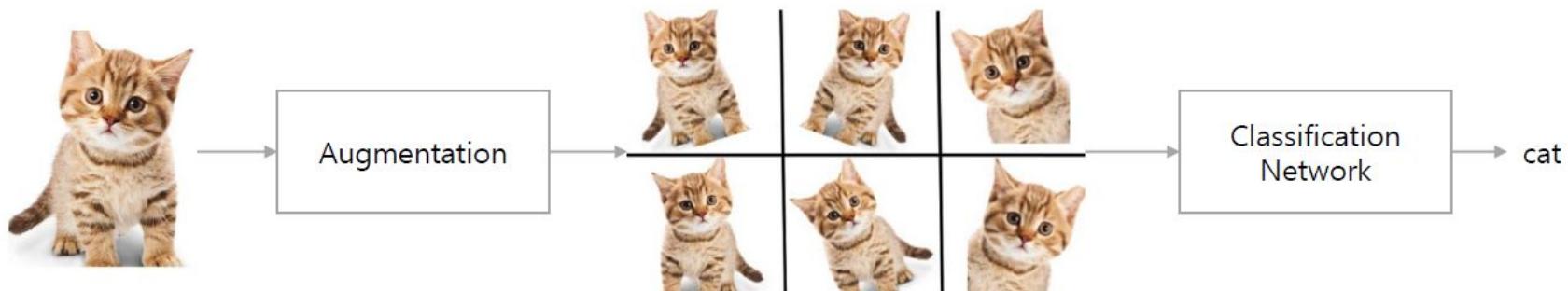


- Test(Validate) loss와 train loss가 차이가 나는 지점이 overfitting이 발생하는 지점
- 학습을 종료하는 방법도 있지만, 그 순간 모델을 저장하는 방법을 많이 쓴다.

# Data Augmentation

일반화를 위해 가장 좋은 방법은 많은 데이터로 훈련시키는 것!

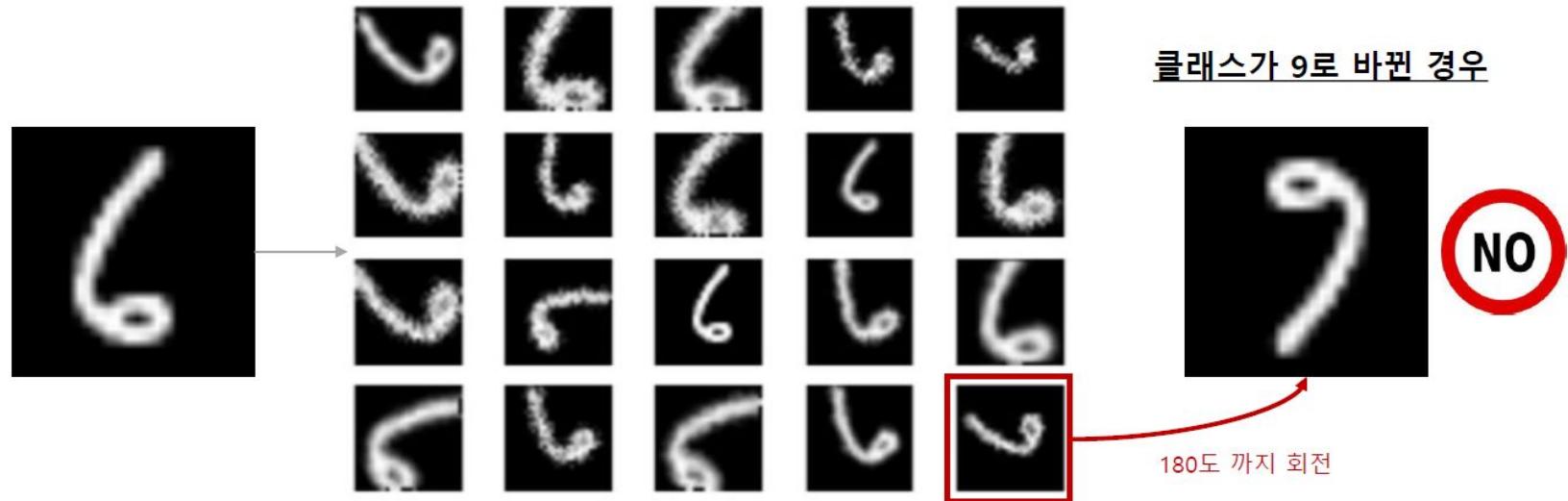
## 데이터 확장이 결합된 학습 과정



Classification과 같은  
transformation invariant task에 매우 적합

# Data Augmentation

클래스를 바꾸지 않아야 한다!



# Data Augmentation

Flips



Color Jitter



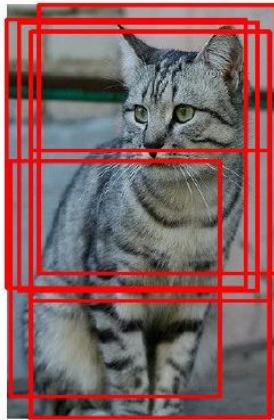
- Horizontal flip, vertical flip

1. 훈련 데이터 셋의 [R, G, B] 픽셀에 PCA 적용
2. 주축 방향으로 "color offset"을 샘플링
3. 모든 픽셀에 offset 더하기

(As seen in [Krizhevsky et al. 2012], ResNet, etc)

# Data Augmentation

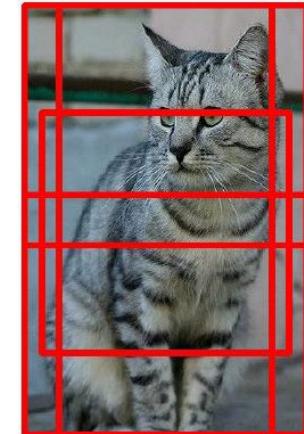
## Random crops and scales



**훈련** : 랜덤하게 확대 및 자르기

ResNet:

1. [256, 480] 범위에서  $L$ 을 임의로 선택
2. 이미지 크기를 가로 세로 중 짧은 쪽 길이가  $L$ 이 되게 스케일링
3.  $224 \times 224$  크기로 랜덤하게 자르기, horizontal flip도 랜덤하게 선택



**테스트** : 고정 크기로 확대해서 고정 위치에서 자르기  
자른 이미지 셋의 출력 값을 평균해서 사용

ResNet:

1. 이미지를 다섯 가지 크기로 스케일링 :  $\{224, 256, 384, 480, 640\}$
2. 각 이미지 크기 별로  $224 \times 224$  10개 자르기 : (4 모퉁이 + 가운데)  $\times 2$  (flip)

# Data Augmentation

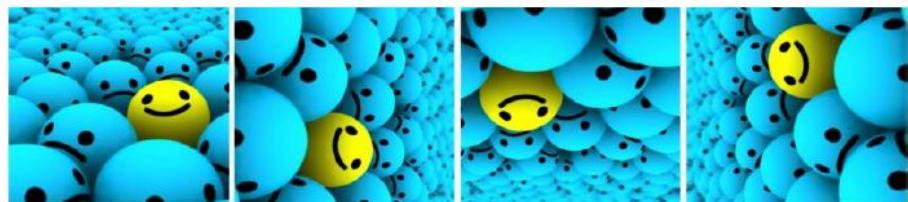
여러 방법들을 혼합해서 사용

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

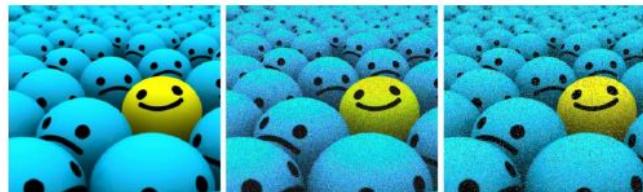
Translation



Rotation

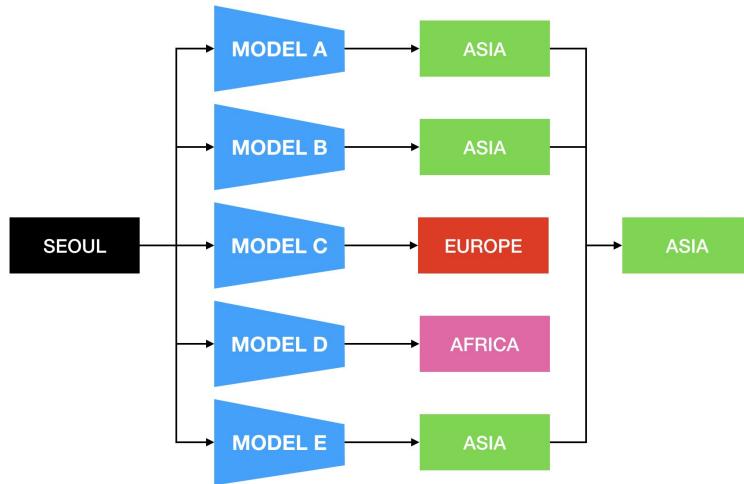


Gaussian Noise Injection



# Ensemble

Simple-Voting-based Ensemble

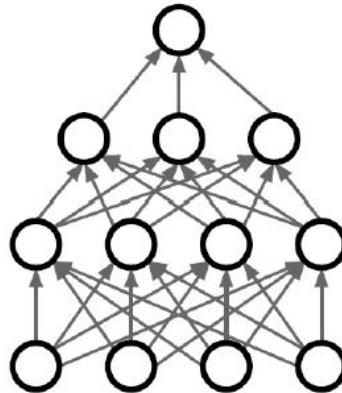


- 단순한 여러 모델을 합쳐 정확도 높이는 방법
- 각 모델의 결과 중 다수 결과를 Output으로 결정

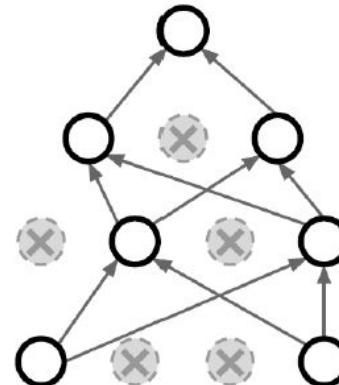
# Dropout

“큰 네트워크를 일종의 **Sub network**로 구성해 정규화를 진행”

신경망



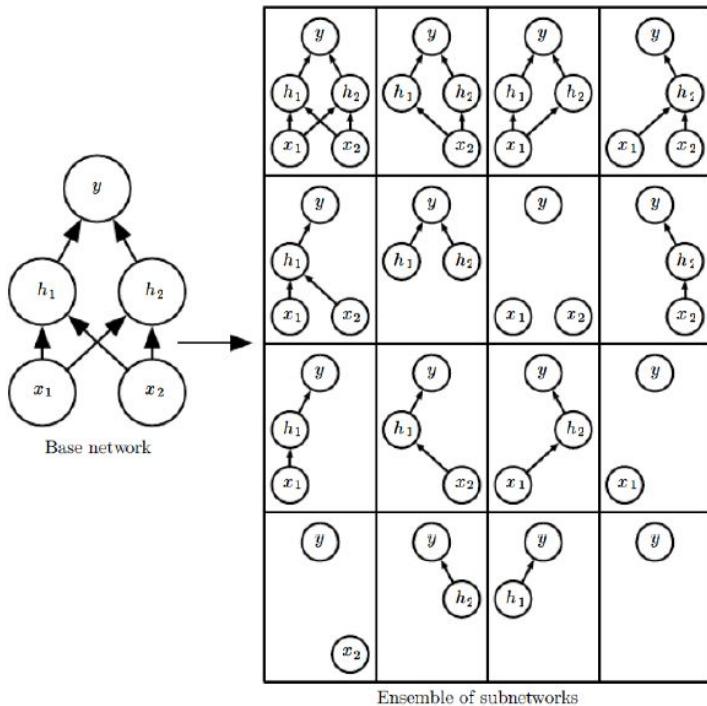
드롭아웃 적용 신경망



- 매 Batch마다 랜덤하게 출력을 0으로 만들어 학습을 누락시킨다.
- 모든 파라미터를 공유하며, 하나의 모델이지만 작은 여러 네트워크가 있는 효과를 발휘

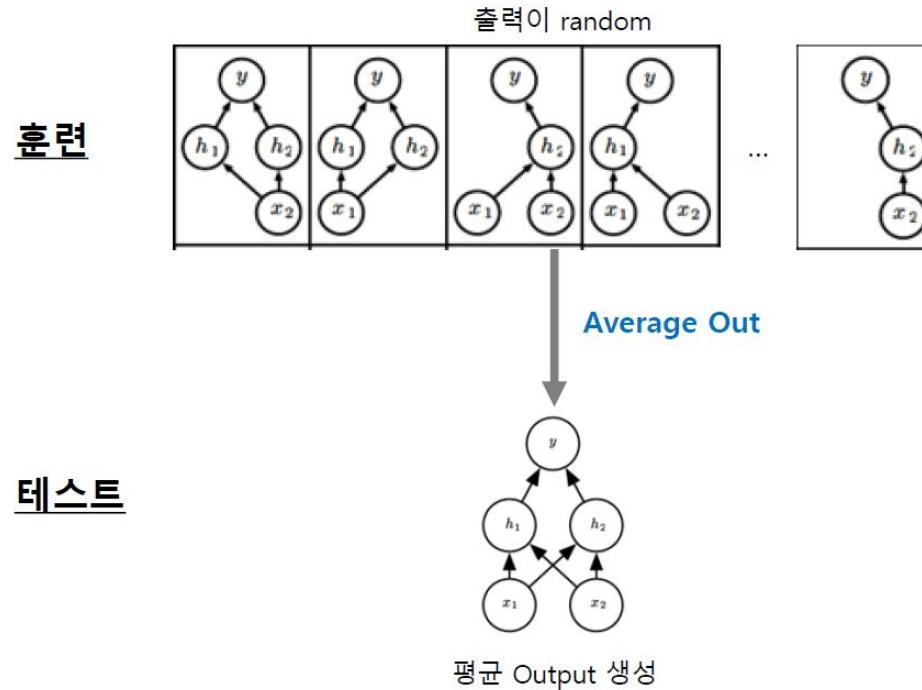
# Dropout

입력 뉴런 2개, 은닉 뉴런 2개인 경우 예시



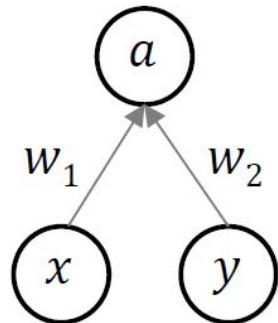
- Base network의 Subset을 옆과 같이 볼수 있다.
- Dropout 동작 방식 - Training
  - Dropout layer가 있는 곳의 출력단에 일정 확률로 0을 곱함

# Dropout



Output at test time = Expected output at training time

# Dropout



입력이 2개인 경우

$$p = 0.5$$

## 훈련

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

평균을 구하게 되면 Dropout의 뉴런 유지 확률과 동일

## 테스트

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

가중치에 Dropout의 확률을 곱해 줌

Weight scaling inference rule

# 참고자료

- 밑바닥부터 시작하는 딥러닝 1, 2  
<http://www.yes24.com/Product/Goods/34970929?Acode=101>  
<http://www.yes24.com/Product/Goods/72173703>
- 모두를 위한 딥러닝 시즌2  
<https://www.edwith.org/boostcourse-dl-tensorflow/joinLectures/22150>
- 모두의 연구소 이일구, 윤성진님(CRAS Lab) 강의 자료  
<https://github.com/ilguyi>