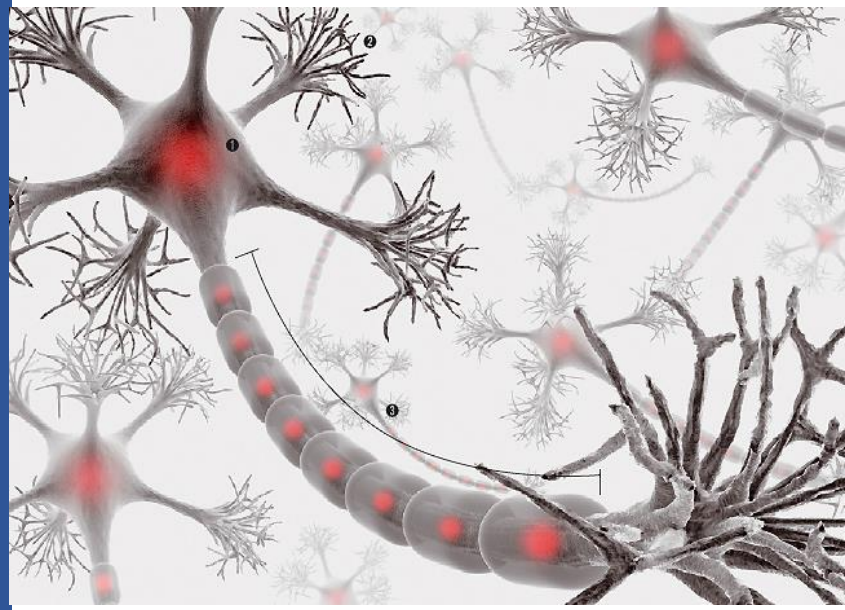
초기화, 정규화

학습 목표

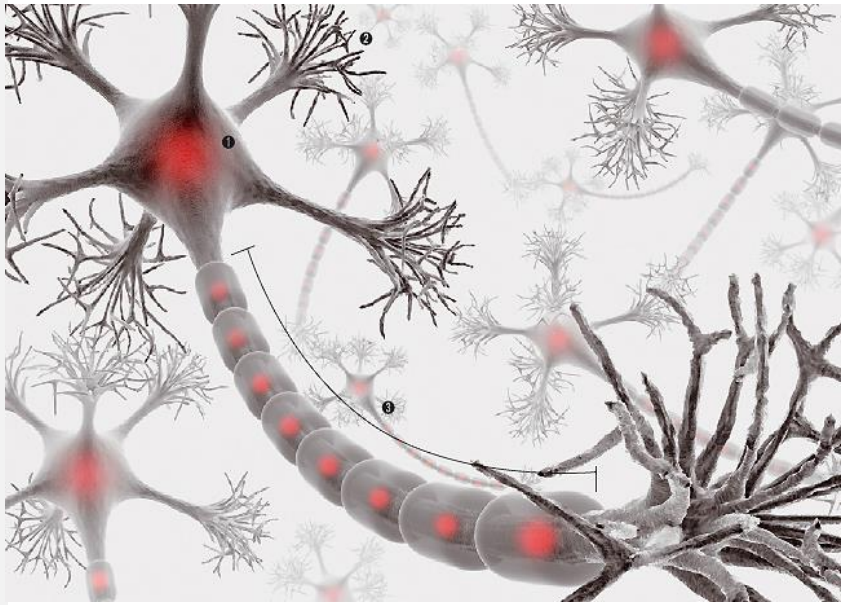
- 신경망을 훈련 단계에 필요한 초기화 및 정규화 기법을 이해한다.

주요 내용

1. 초기화 (Initialization)
2. 정규화 (Regularization)
3. 배치정규화
4. 가중치 감소 (weight decay)
5. 조기 종료 (Early Stopping)
6. 데이터 확장 (Data Augmentation)
7. 앙상블 (Ensemble)
8. 드롭아웃 (Dropout)

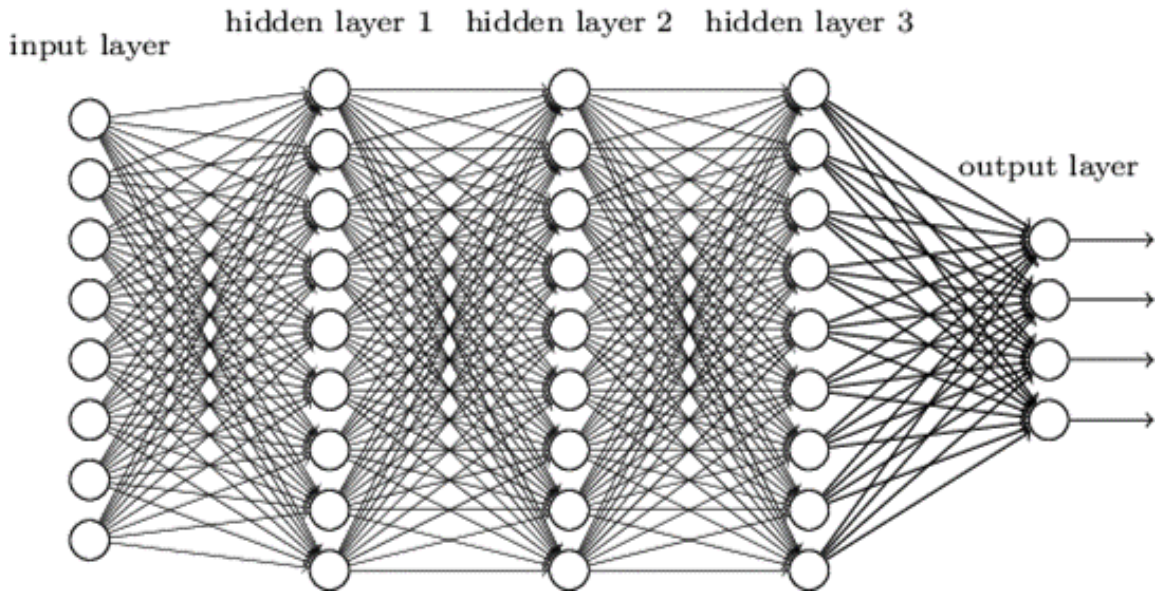


1 초기화 (Initialization)



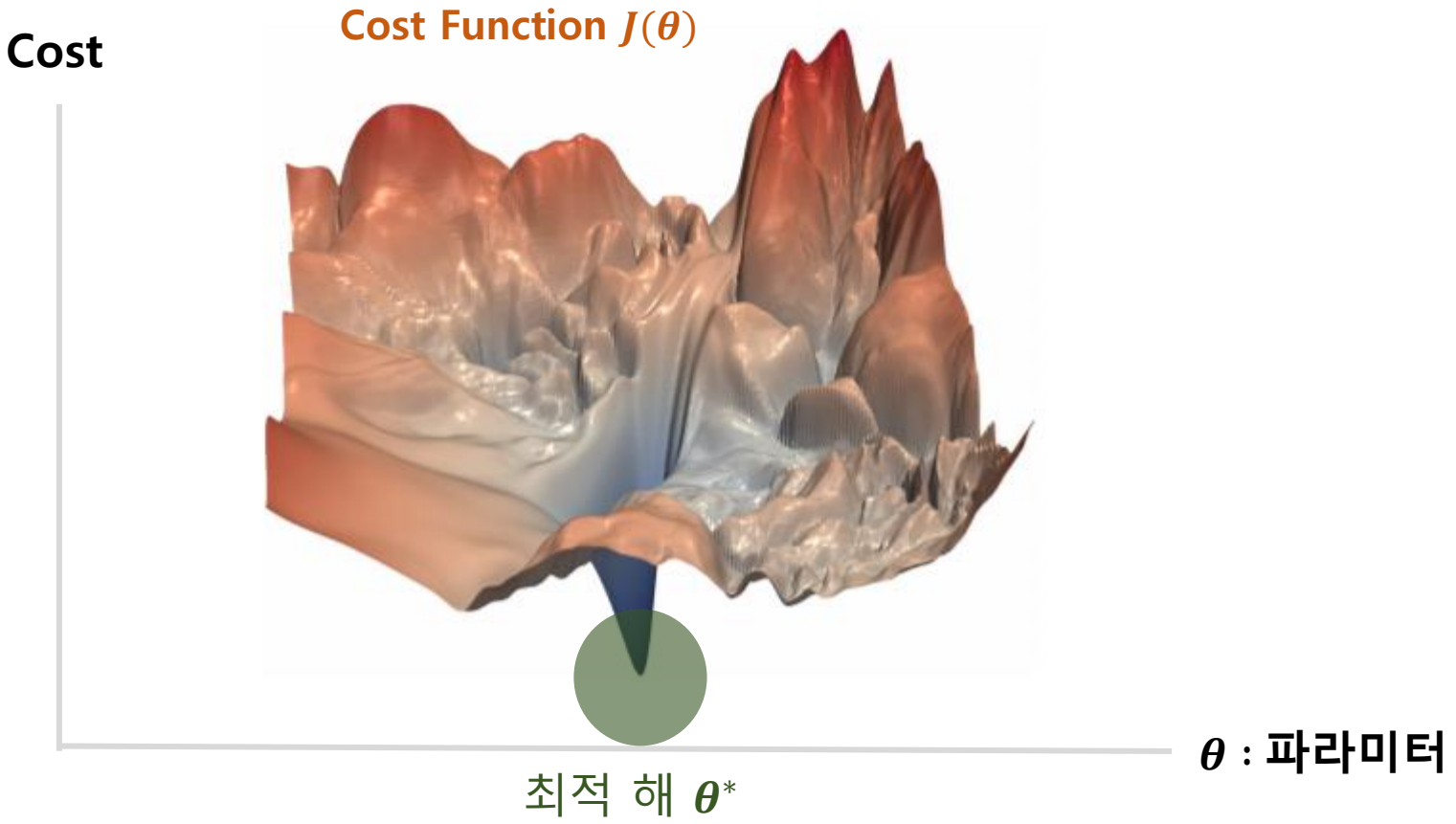
가중치 초기화

가중치를 어떻게 초기화해야 학습이 잘 될까?



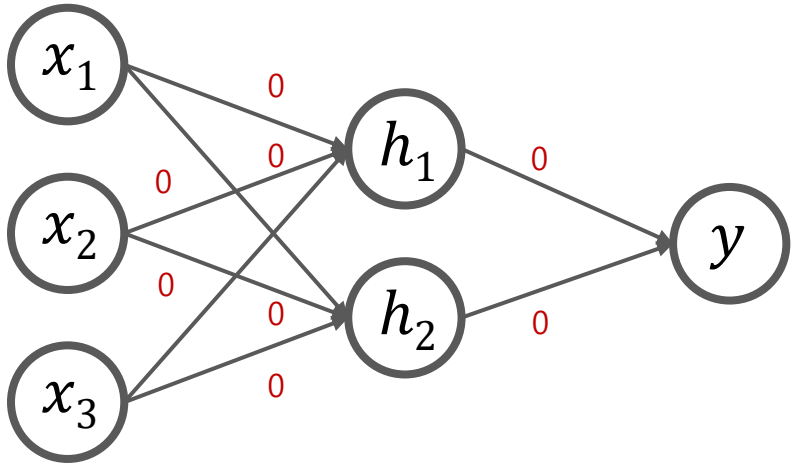
가중치 초기화란?

초기화 문제는 손실 곡면의 어느 위치에서 출발하면 좋은 지의 문제



제로 초기화

가중치를 0으로 초기화하면 어떻게 될까?

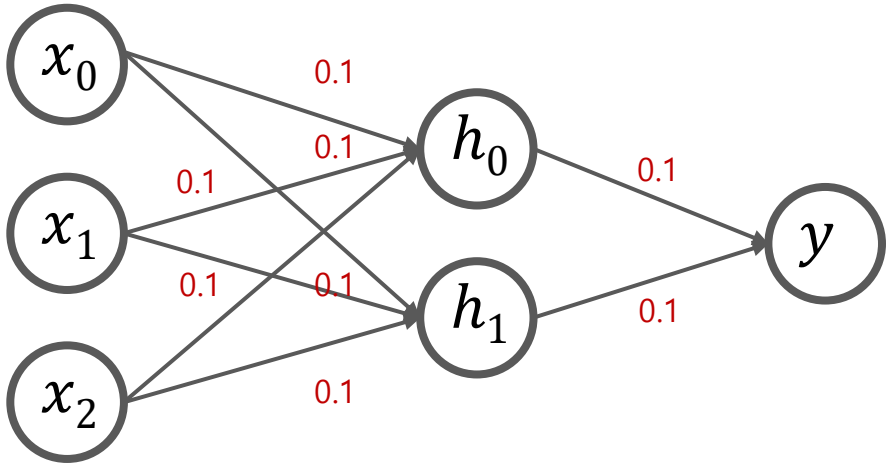


- 모든 뉴런의 가중 합산 결과가 0이 되어 출력이 상수가 됨
- 그래디언트가 0이 되어 학습이 진행되지 않음

$$n = \boldsymbol{w}^T\boldsymbol{x} + b = 0 \qquad \frac{\partial n}{\partial \boldsymbol{x}} = \boldsymbol{w} = 0$$

상수 초기화

가중치를 동일한 상수로 초기화하면 어떻게 될까?



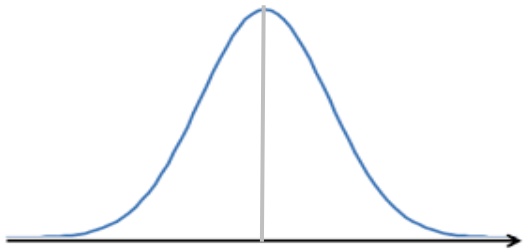
$$h_0 = h_1$$

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

가우시안 분포 초기화

가중치를 아주 작은 난수로 초기화하면 어떻게 될까?



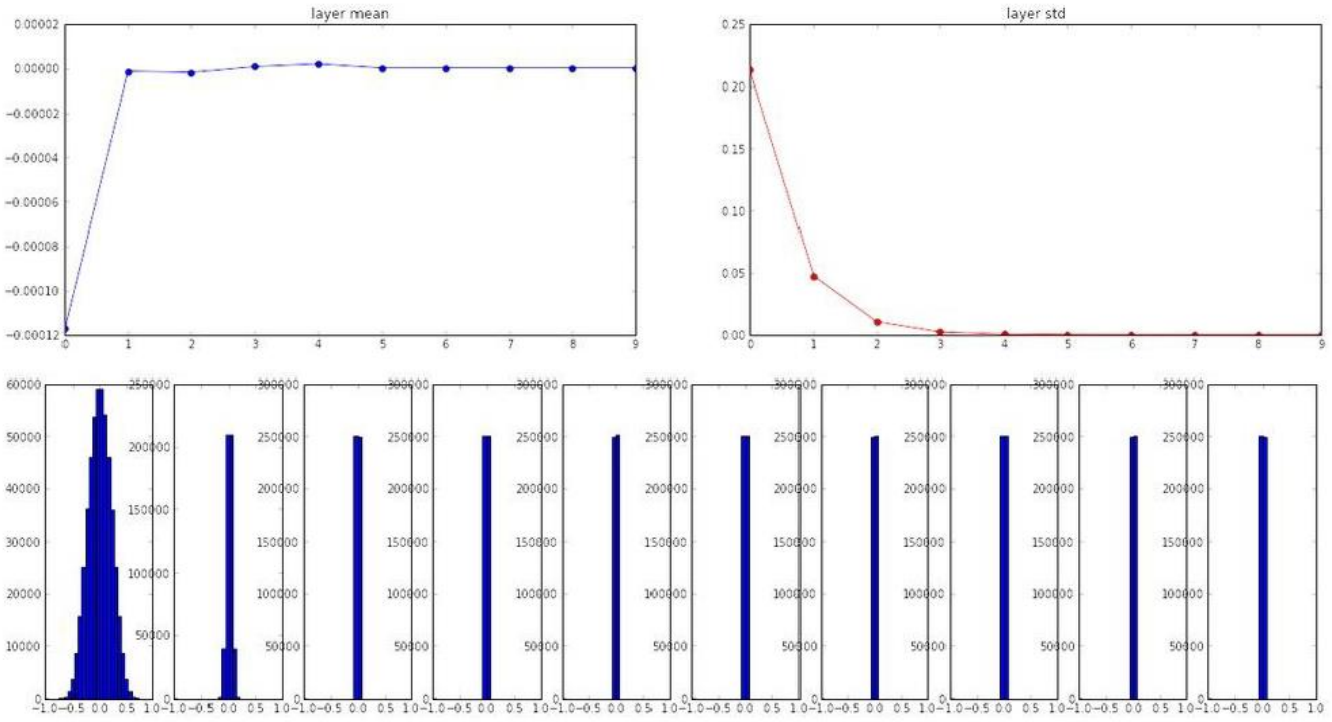

```
W = 0.01 * np.random.randn(D, H)
```

작은 네트워크에는 작동하지만 네트워크가 깊어질수록 문제 발생!

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

가우시안 분포 초기화 – 분산이 작은 경우

Activation 분포



Activation이 점점 0으로 변화

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

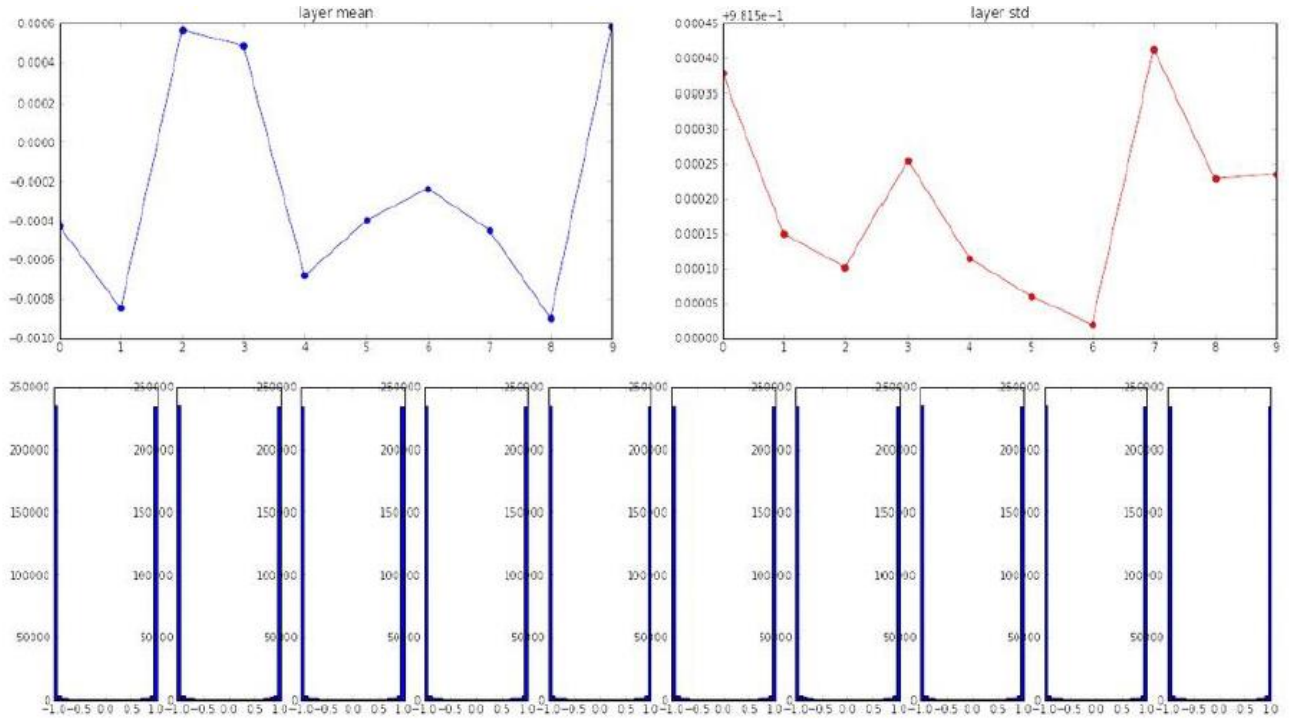
```
W = 0.01 * np.random.randn(D, H)
```

Activation이 0이면 Gradient도 0!
학습이 진행되지 않음!

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

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)
```

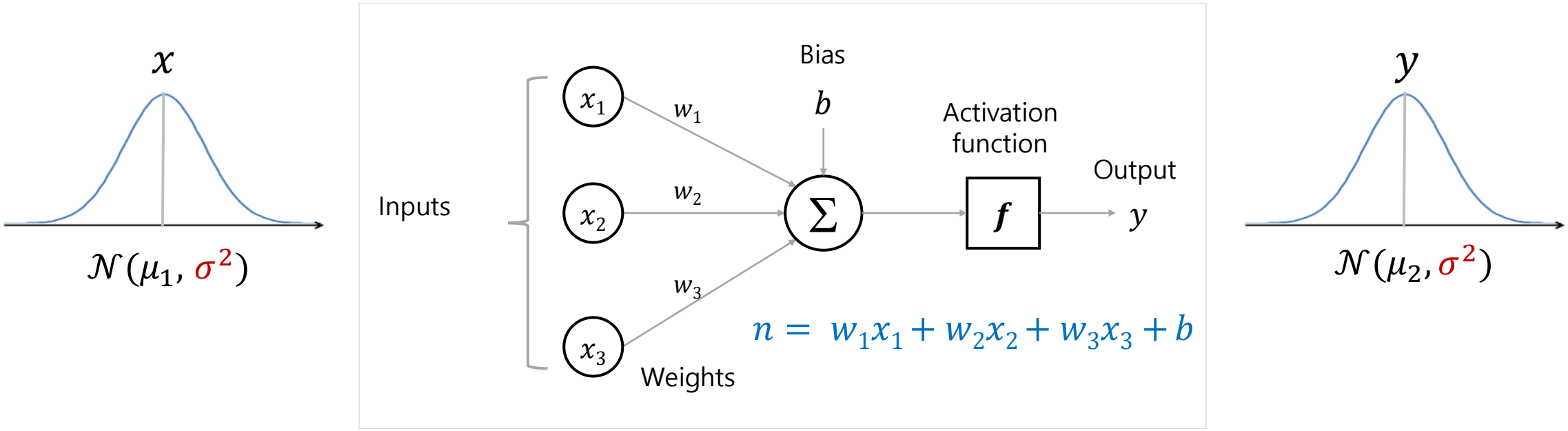
- 가중치가 커지면 tanh 입력 값이 커짐
- 따라서, tanh가 saturation 되어 출력이 1이나 -1로 값이 편향됨

Gradient Saturation이 일어나
학습이 진행되지 않음!

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

Xavier Initialization

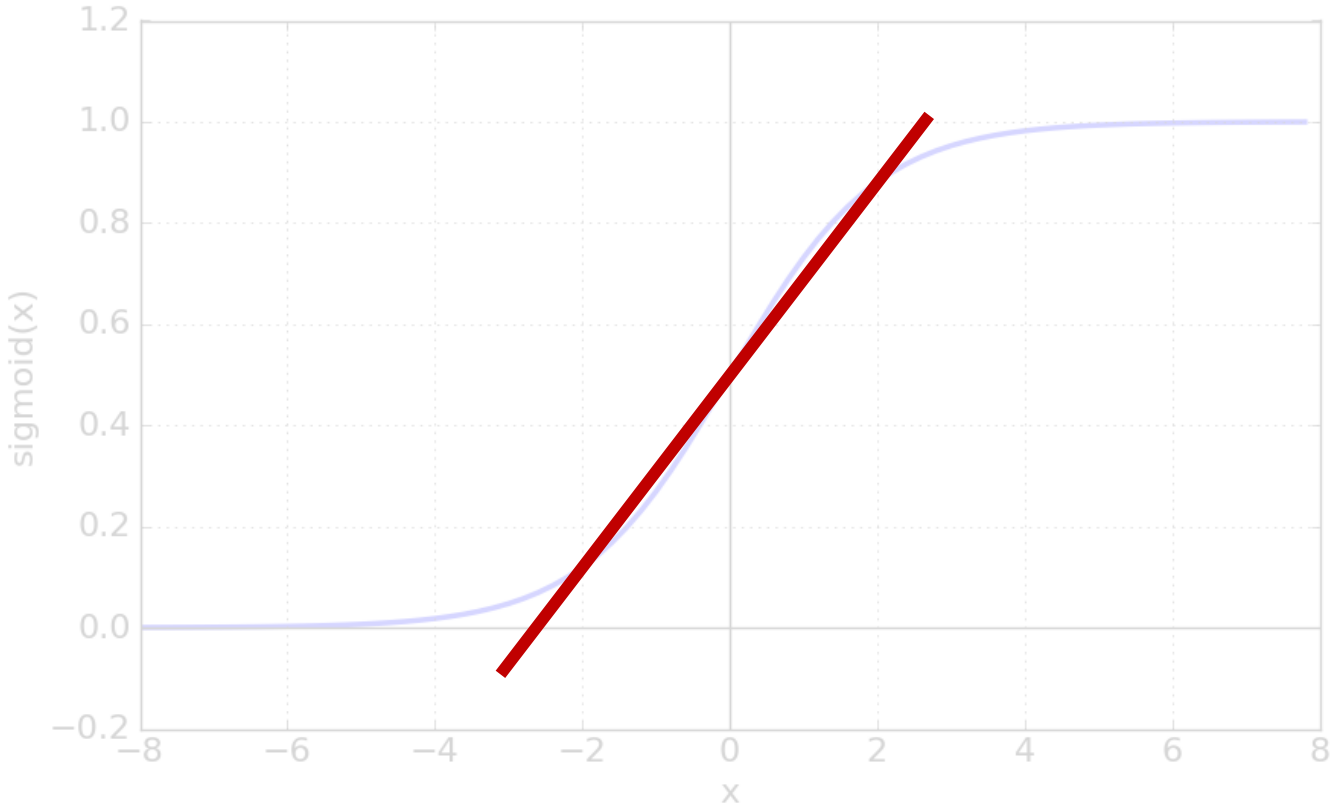
입력과 출력의 분산이 같아지도록 가중치로 초기화하자!



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

Xavier Initialization

시그모이드 계열 함수



- 활성 함수를 선형으로 가정

Sigmoid, tanh와 같은 활성 함수는 가운데 부분이 선형 함수와 같다.

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

Xavier Initialization

x의 분산과 y의 분산을 같게 하려면?

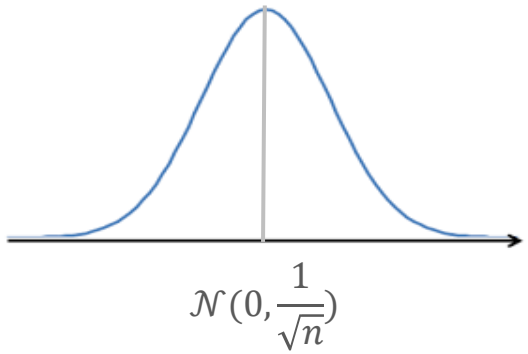
$$y = n = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

x_i 와 w_i는 독립

$$Var(y) = nVar(x_i)Var(w_i)$$

$\mathrm{Var}(x_i) = \mathrm{Var}(y)$

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



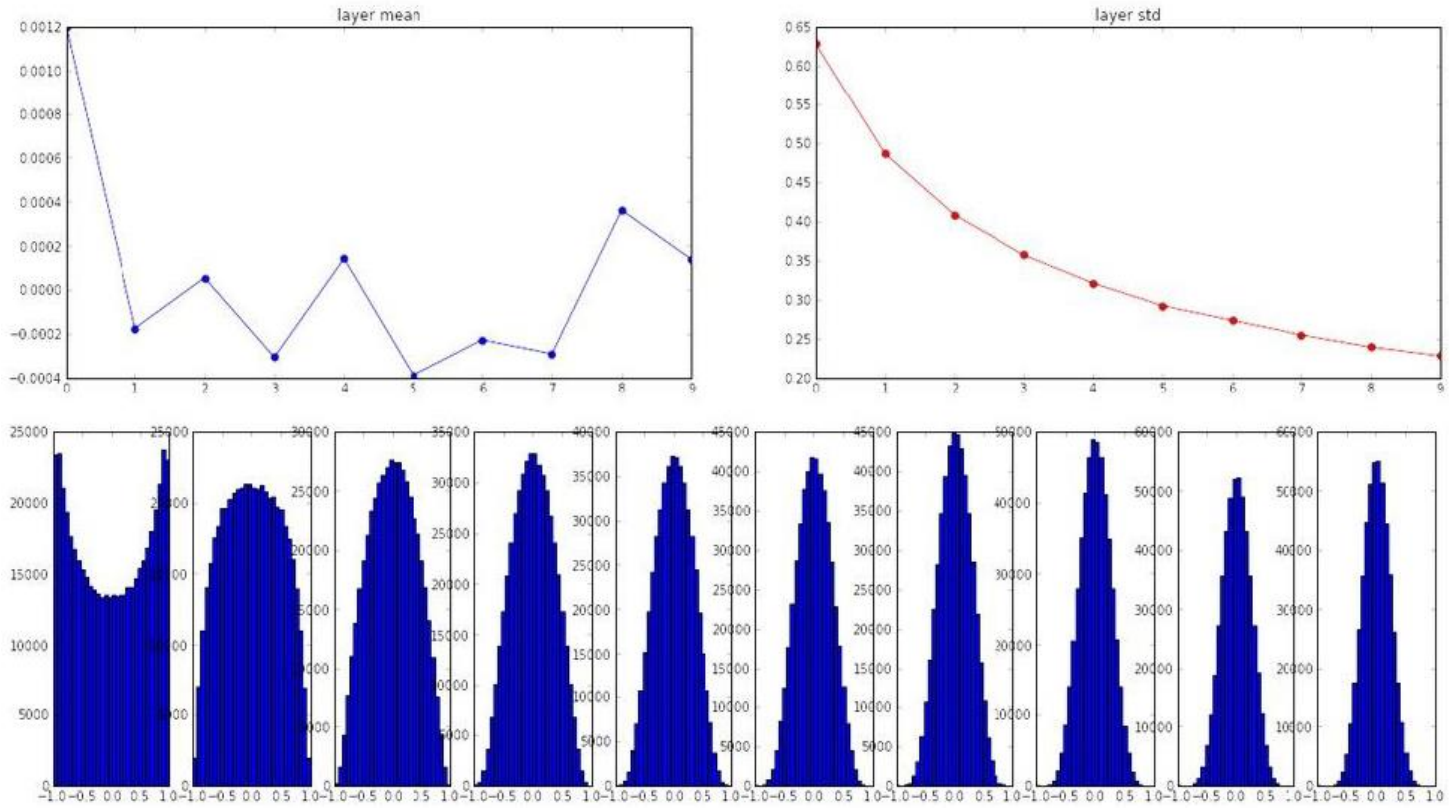
$$\mathbf{Var}(\boldsymbol{W}) = \frac{1}{n}$$

n : # of inputs

가중치의 분산을 입력 개수에 반비례하게 만든다!

Xavier Initialization

Activation 분포

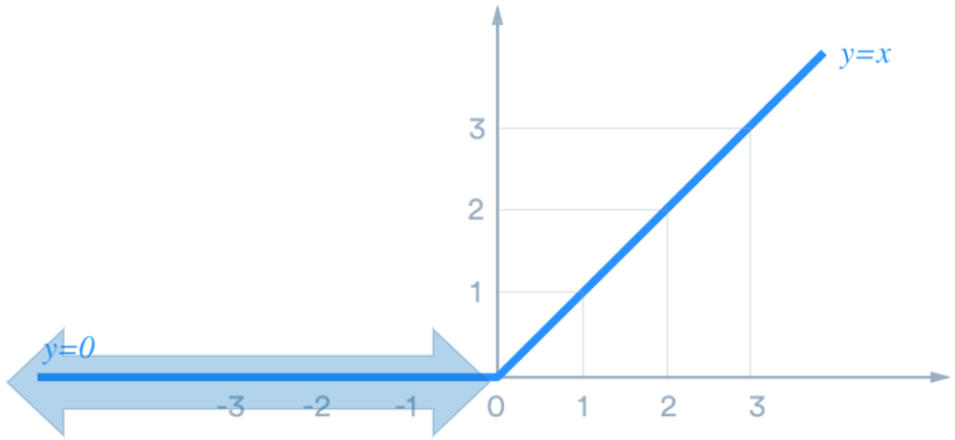


각 Layer 마다 Unit-Gaussian Input 및 Output을 근사하게 됨

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

Xavier Initialization

Non-Linear Activation인 ReLU를 사용하면?

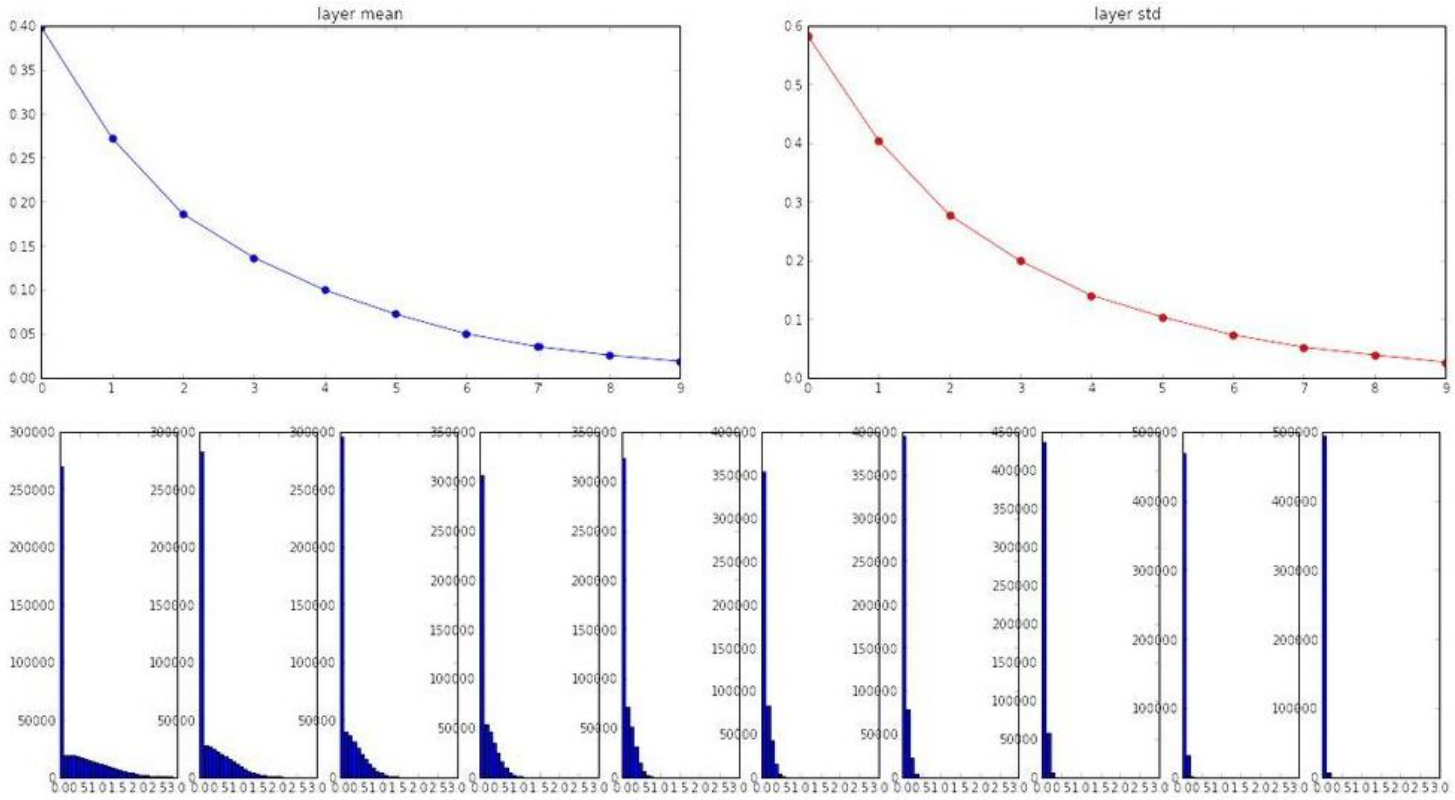


- 정규 분포의 음수 영역이 ReLU Activation 과정에서 0으로 바뀜
- Layer가 깊어질수록 Activation이 점점 0으로 치우치게 됨

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

Xavier Initialization

Activation 분포



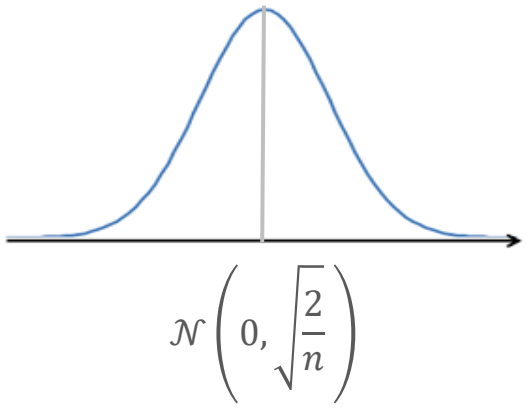
Activation이 점점 0으로 변화

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

He Initialization

가중치의 분산을 2배로 키워서 Activation을 넓게 퍼지게 만듦

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



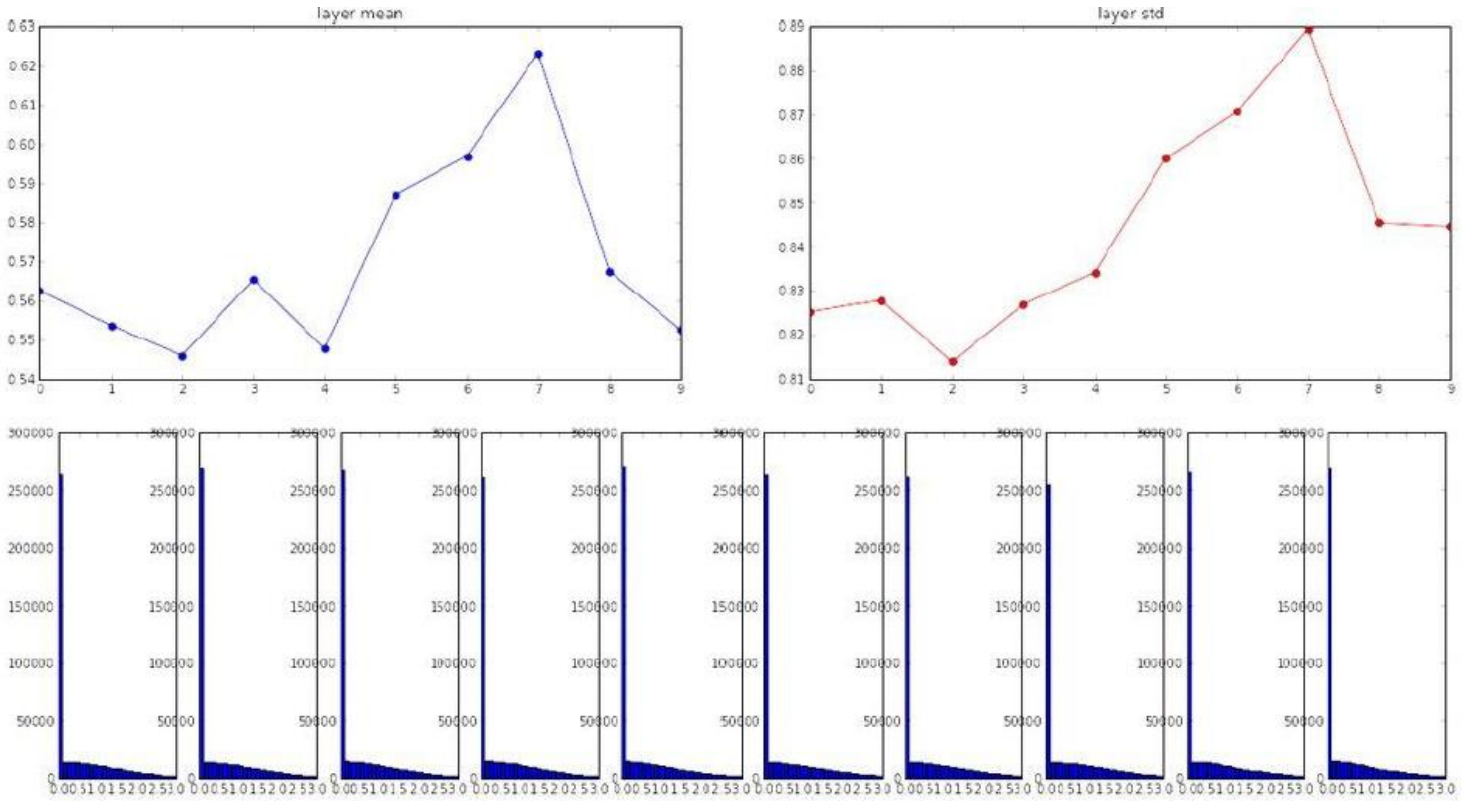
$$\mathbf{Var}(\boldsymbol{W}) = \frac{2}{n}$$

n : # of inputs

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

He Initialization

Activation 분포

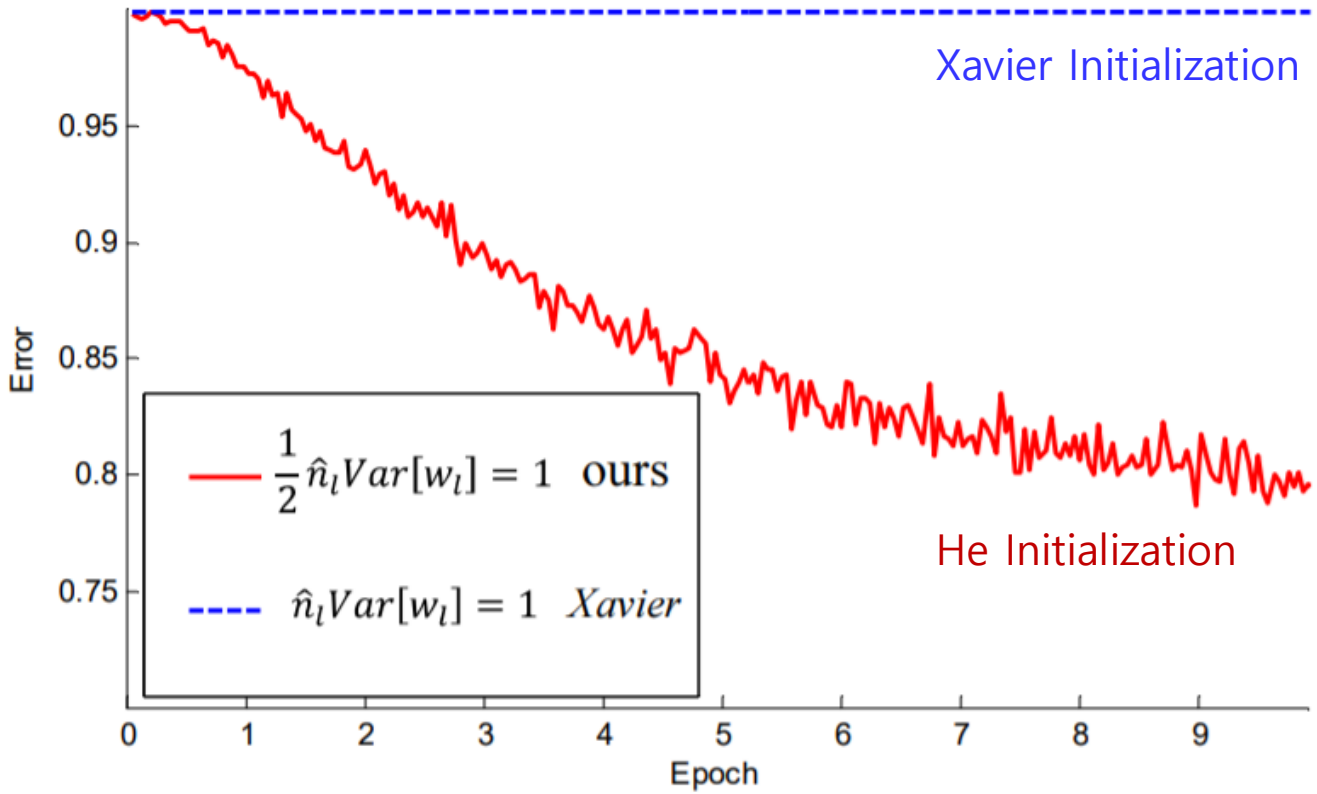


Activation이 균일하게 분포

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

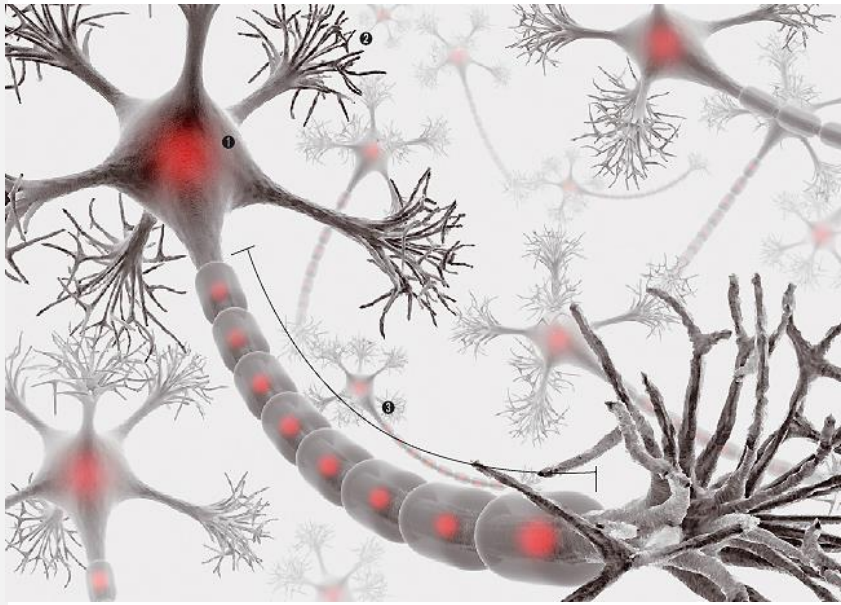
He Initialization

Xavier와 He의 성능 비교

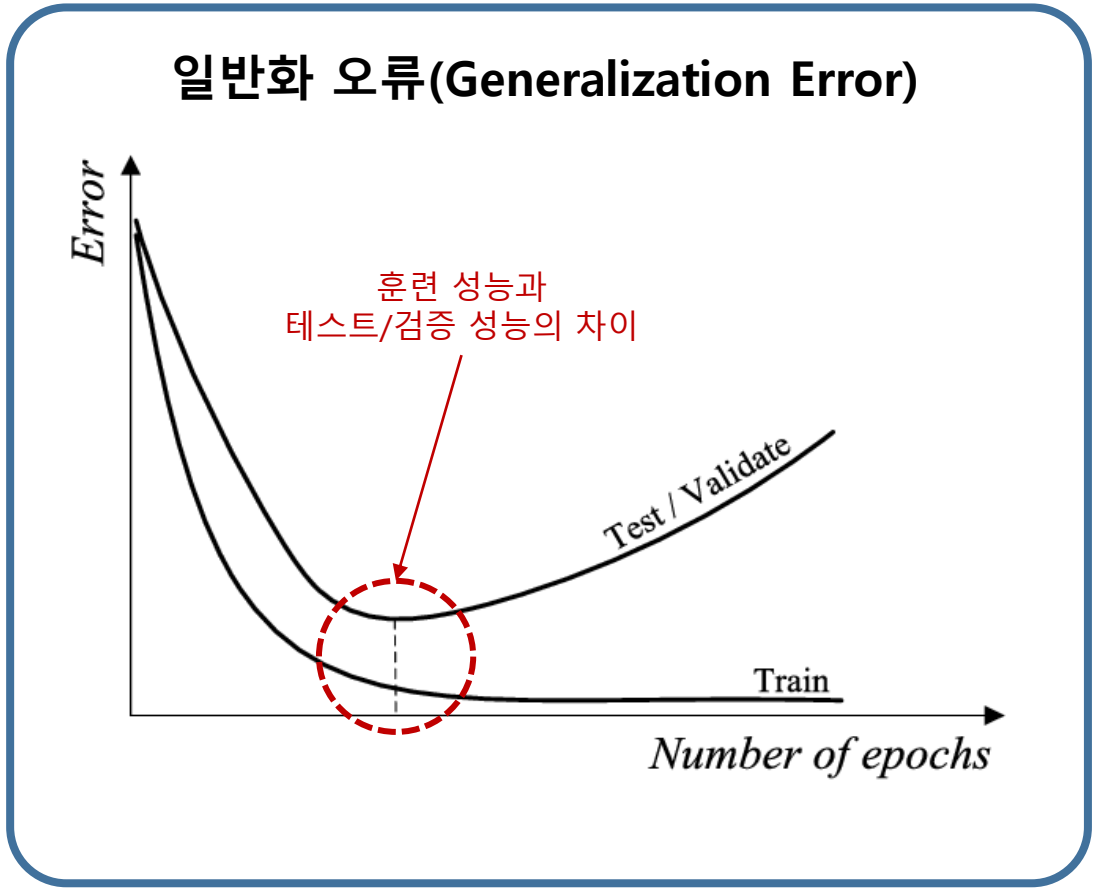


Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He (2015)

2 정규화



정규화 (Regularization)



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

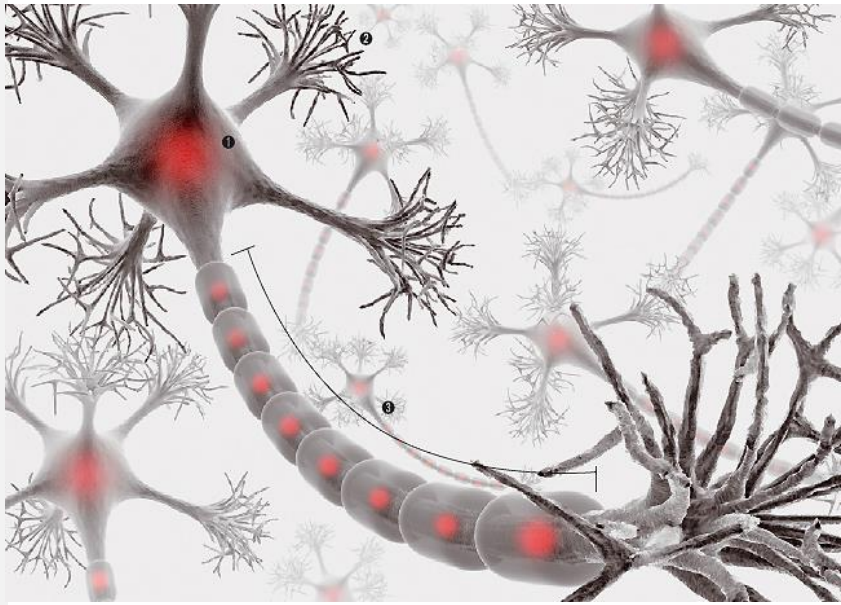
Optimization + **Regularization**

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

3 배치 정규화 (Batch Normalization)



Internal Covariate Shift

Deep Neural Net의 학습이 어려운 이유는?

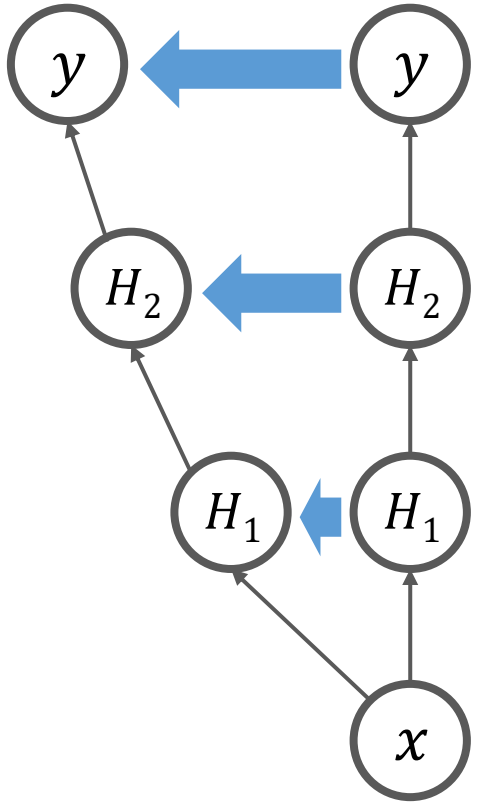
- 훈련을 할 때마다 입력 데이터의 분포가 각 계층에서 Shift 되어 원래 분포에서 멀어지게 됨

"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

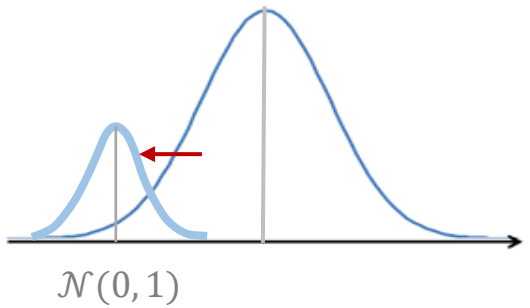
배치 정규화

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

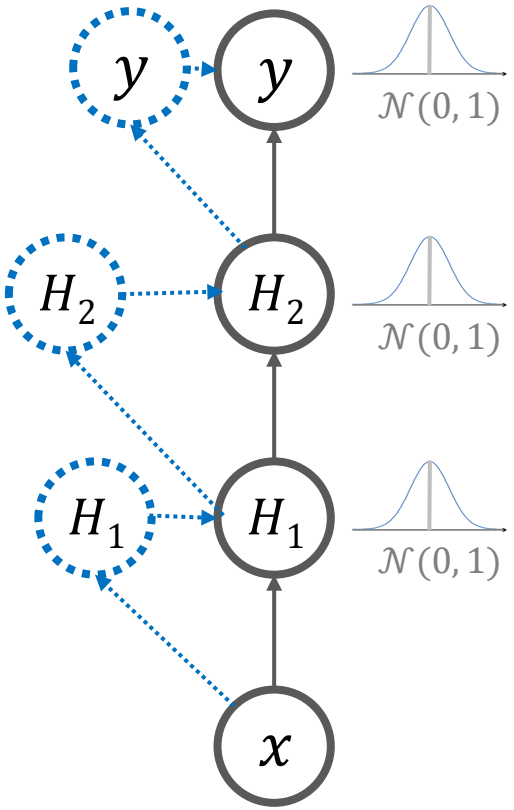
각 뉴런에서 입력 데이터를 정규화

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

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



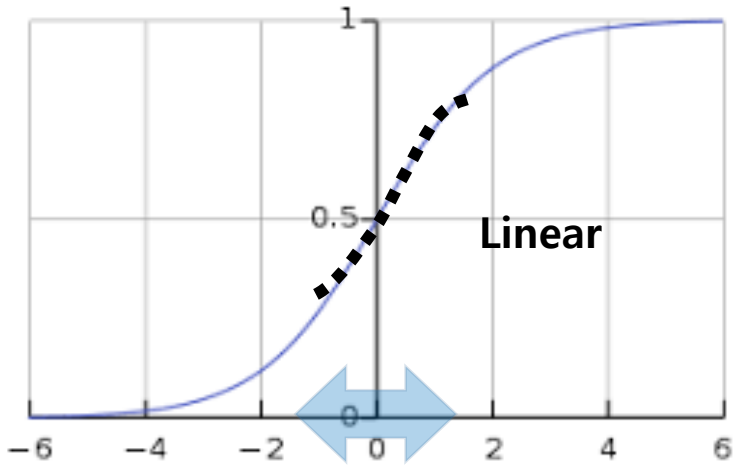
데이터 분포가 동일한 분포로 유지됨



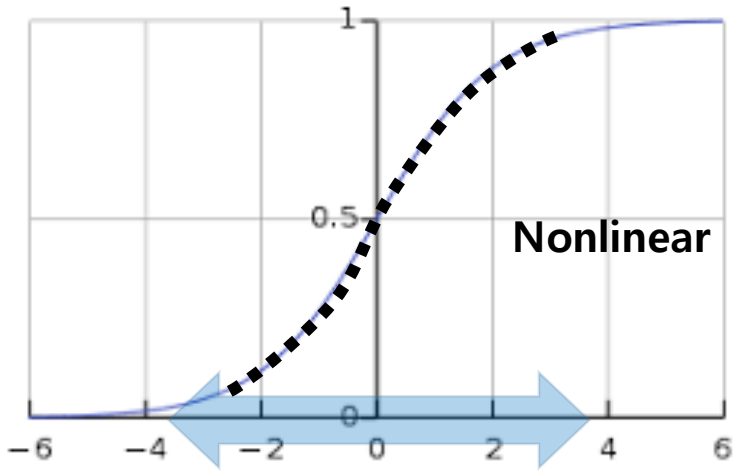
Scale and Shift

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

Zero-Mean Unit-Variance



Scale and Shift



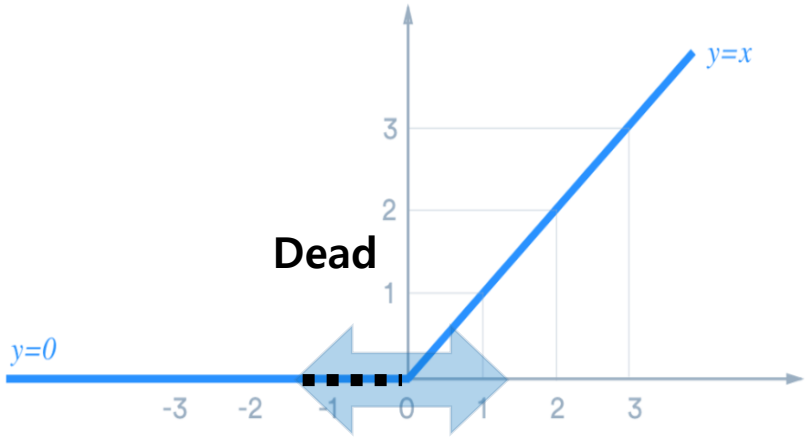
$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

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

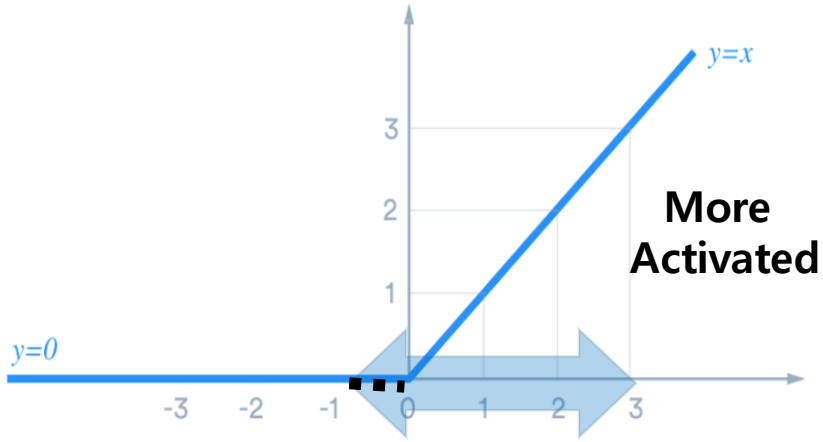
Scale and Shift

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

Zero-Mean Unit-Variance



Scale and Shift



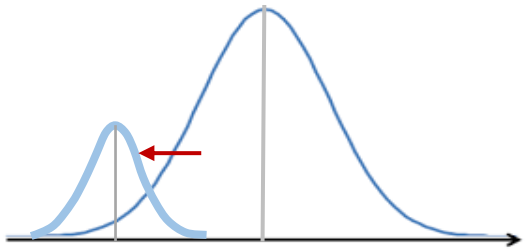
$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

ReLU의 경우 Dead ReLU를 줄이려면 양수 영역이 더 넓은 것이 유리하다.

Scale and Shift

내부 공변량이 이동이 제거된 원래의 분포로 만들어 보자!

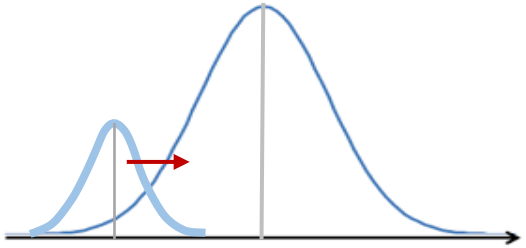
Normalize



$\mathcal{N}(0,1)$

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

Scale and Shift



$\mathcal{N}(\beta^{(k)}, \gamma^{(k)})$

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

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!

훈련 및 테스트 알고리즘

Training

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

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

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

훈련 시점 :
미니 배치 단위로 정규화 수행

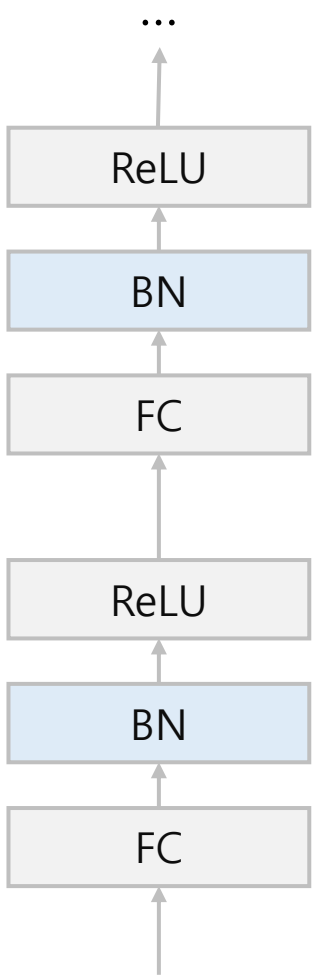
테스트 시점 :
전체 데이터의 평균과 분산으로 정규화

$$\mathrm{E}[x] \leftarrow E_B[\mu_B]$$

$$Var[x] \leftarrow \frac{m}{m-1}E_B[\sigma_B^2]$$

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

배치 정규화 위치



Fully Connected 또는 Convolution 이후에 Activation Function 이전에 실행

Activation 이후에 실행하는 경우도 있음

Summary

- 네트워크 Gradient Flow가 원활해지면서 학습이 잘 됨

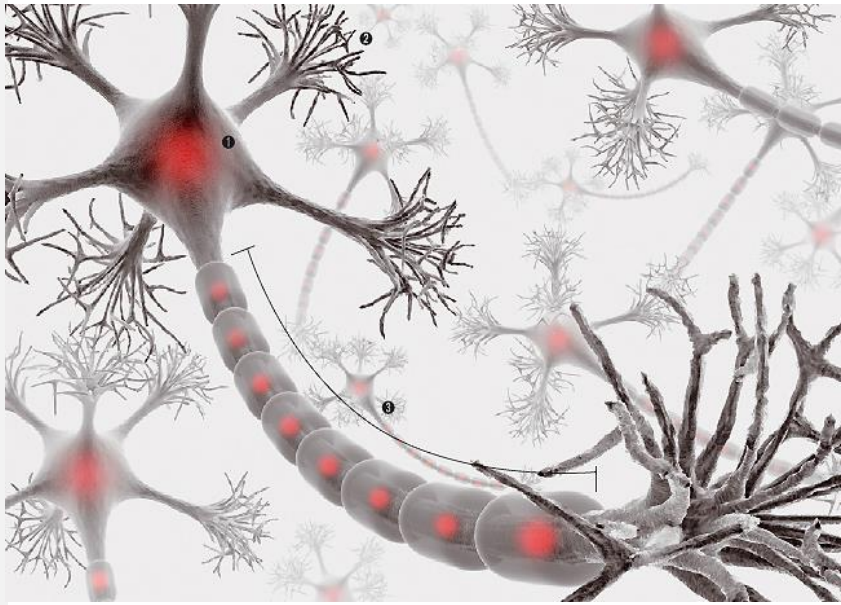
- 높은 학습률이 가능
- 초기화 의존도가 낮아짐

} 훈련이 까다롭지 않게 됨

- 정규화 효과

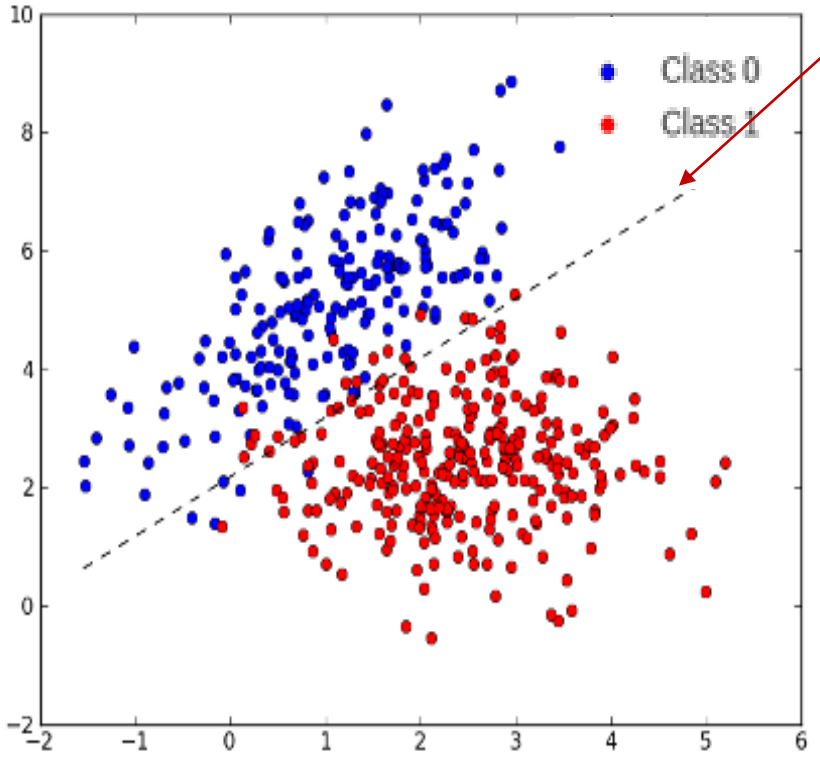
  - 미니 배치 단위의 데이터 분포를 구해 정규화를 하기 때문에 배치구성에 따라 입력 값이 조금씩 달라지는 stochastic한 성격을 갖게 됨
    (즉, 입력 값이 더 이상 deterministic하지 않음)
  - Dropout의 필요성이 감소

4 가중치 감소 (Weight Decay)



어떤 가중치가 좋은가?

Linear Classification



결정 경계 (Decision boundary) :

$$\boldsymbol{wx} = b$$

양 변에 2를 곱하면 해인 $\boldsymbol{x}$가 달라지는가?

$$\boldsymbol{2wx} = 2b$$

Q. w와 $2w$ 중 어떤 것이 좋은 가중치인가?

w가 작을 수록 variance를 줄어들어
최적화가 용이해지고 과적합이 방지된다.

가중치 감소 (Weight Decay)

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

λ : 정규화 상수

모델 오차가 최소화 되도록 예측하게 함

가중치 크기를 작게 만들어 모델이 과적합 되지 않게 함

Regularizer 종류

L_2 Regularizer

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

리지 회귀 Ridge regression

L_1 Regularizer

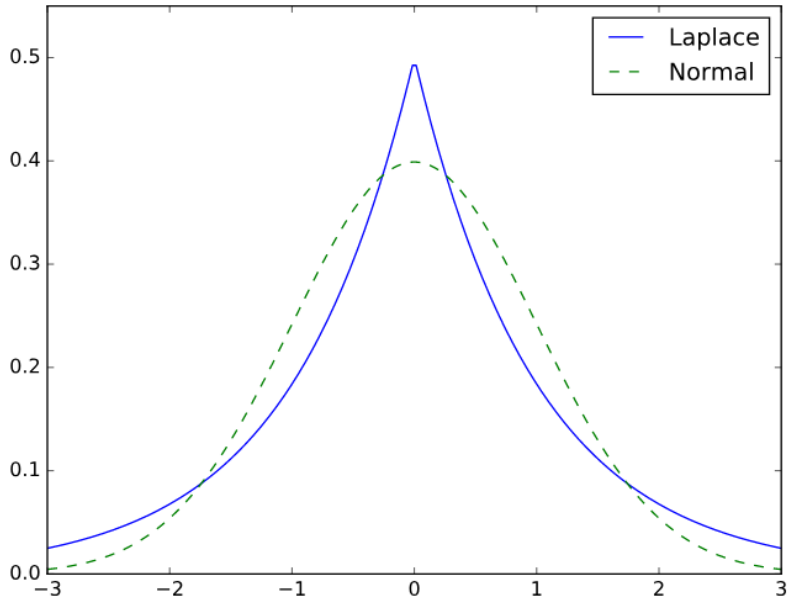
$$\tilde{J}(\boldsymbol{W};\boldsymbol{X},\boldsymbol{y}) = J(\boldsymbol{W};\boldsymbol{X},\boldsymbol{y}) + \lambda \| \boldsymbol{W} \|_1^2$$

라소 회귀 Rasso regression

Parameter Norm Penalty 형태로 Prior에 따라 다양한 Norm을 사용할 수 있음

Regularizer 종류

가중치 분포에 따라 L_1 또는 L_2를 선택



L_2 Regularizer

W가 정규 분포일 때 사용

$$N(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

* 정규 분포의 Log Likelihood는 **L^2** Norm

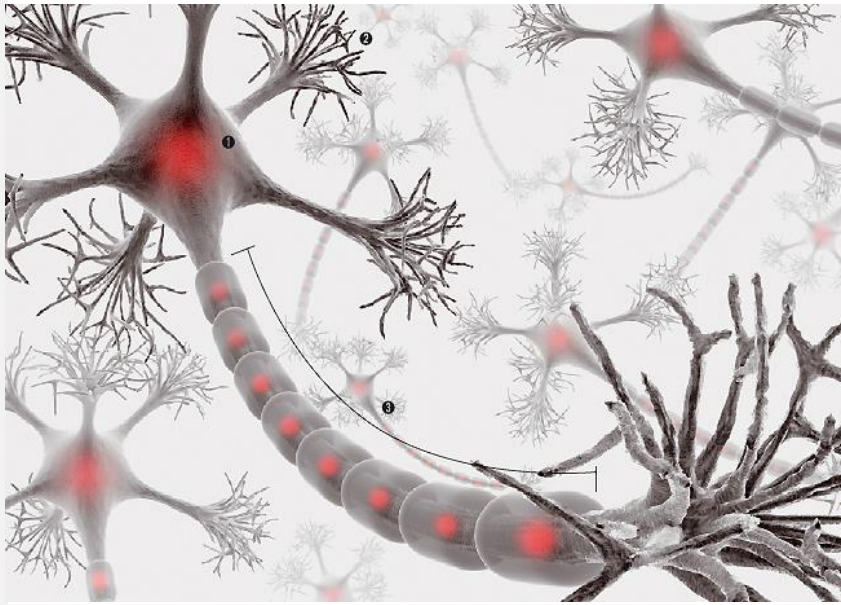
L_1 Regularizer

W가 라플라스 분포일 때 사용

$$f(x \mid \mu, b) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$$

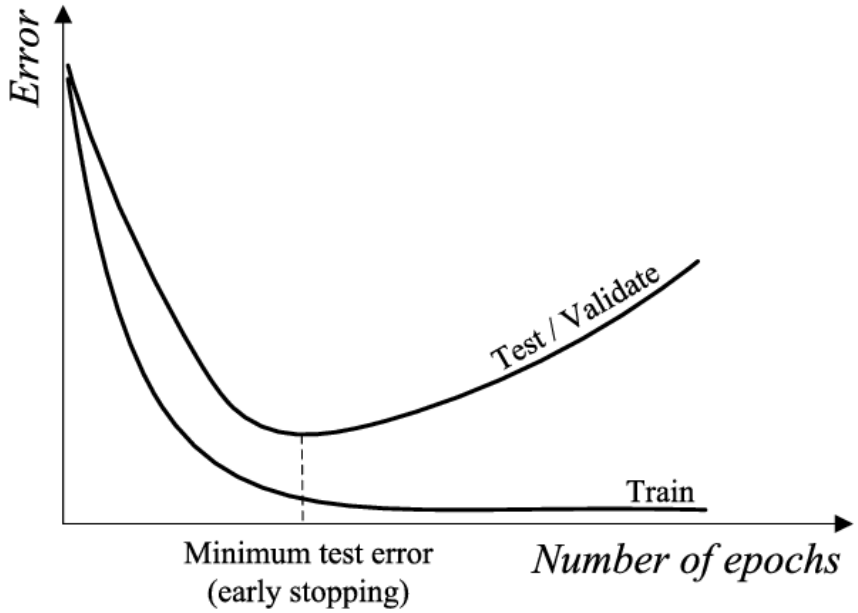
* 라플라스 분포의 Log Likelihood는 **L^1** Norm

5 조기 종료 (Early Stopping)

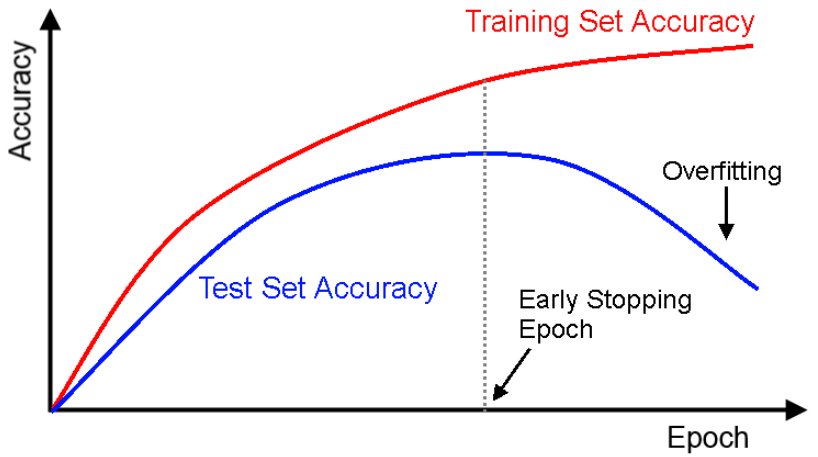


조기 종료 (Early Stopping)

Error/Loss



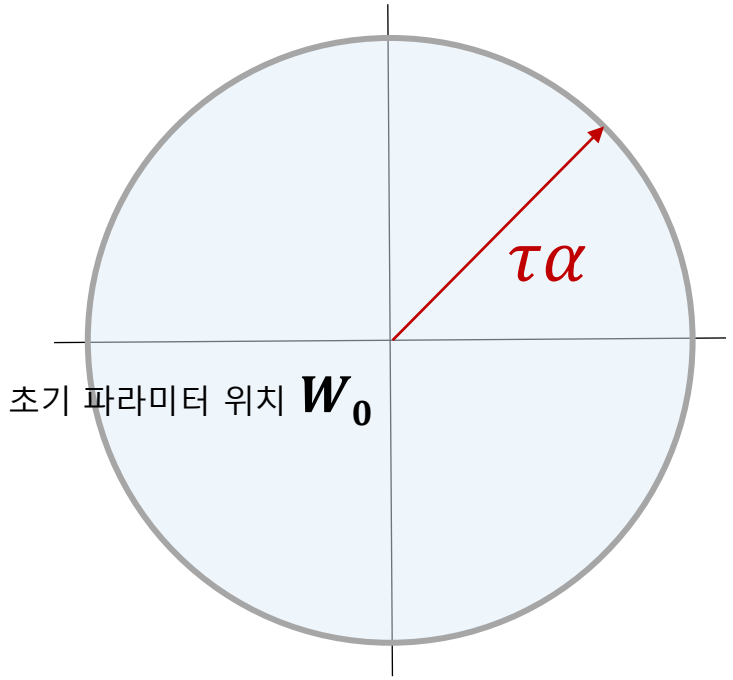
Accuracy



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

조기 종료가 정규화 기법인 이유

파라미터 공간에서 조기 종료의 영향



$\tau\alpha$: 파라미터 공간의 반경

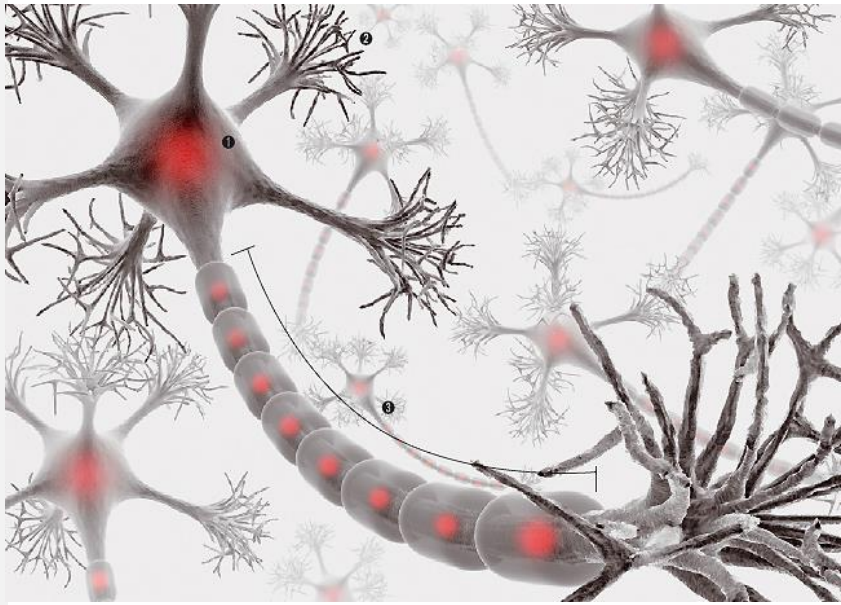
τ : optimization steps

α : learning rate

최적화 단계와 학습률에 의해 파라미터 공간의 크기가 정해짐

파라미터 공간을 작게 만들어서
최적화를 수행함으로써 과적합을 방지

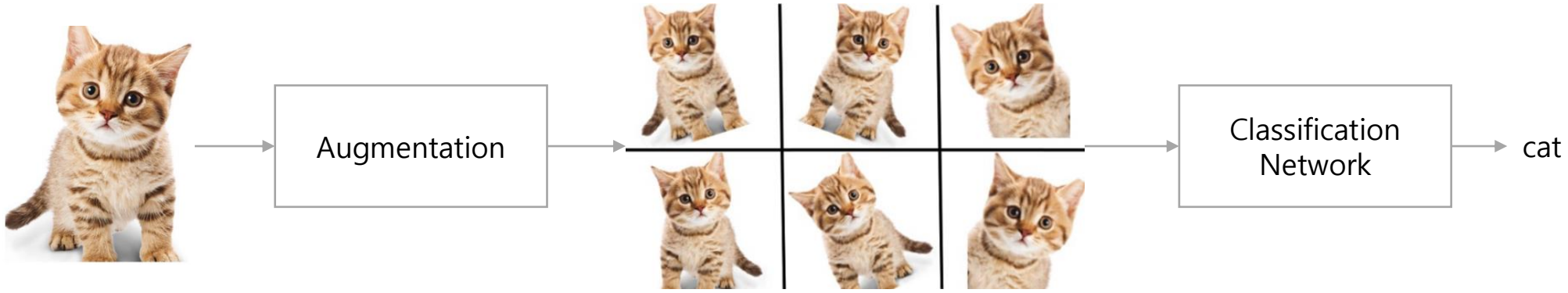
6 데이터 확장 (Data Augmentation)



데이터 확장 (Data Augmentation)

일반화를 위해 가장 좋은 방법은 많은 데이터로 훈련시키는 것!

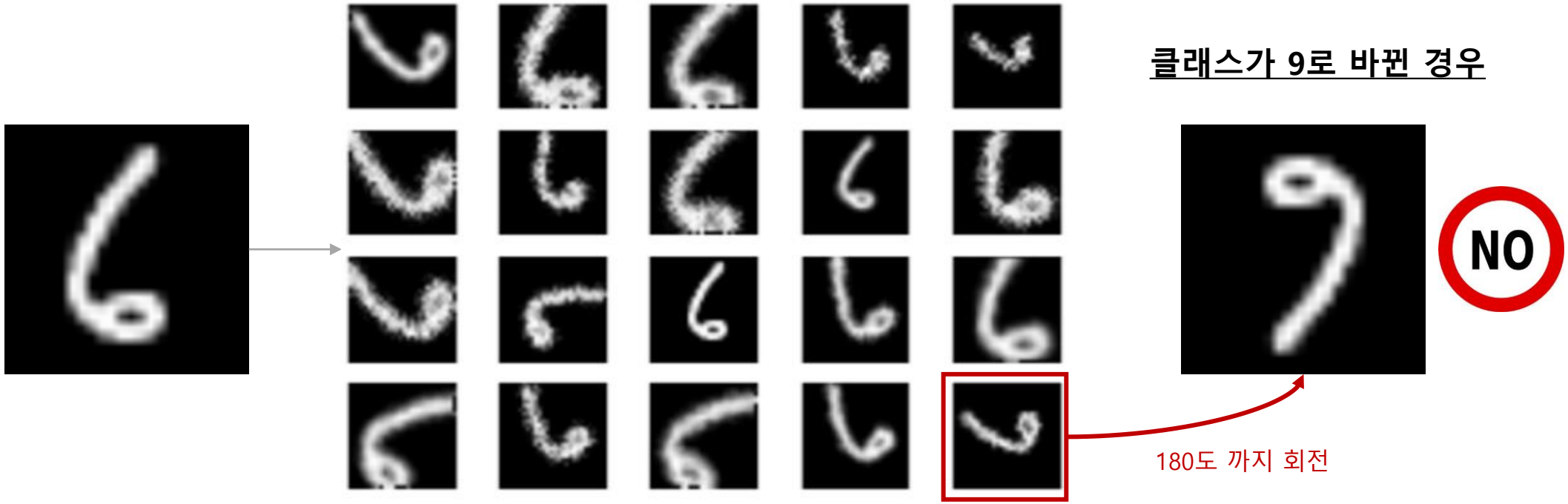
데이터 확장이 결합된 학습 과정



Classification과 같은
transformation invariant task에 매우 적합

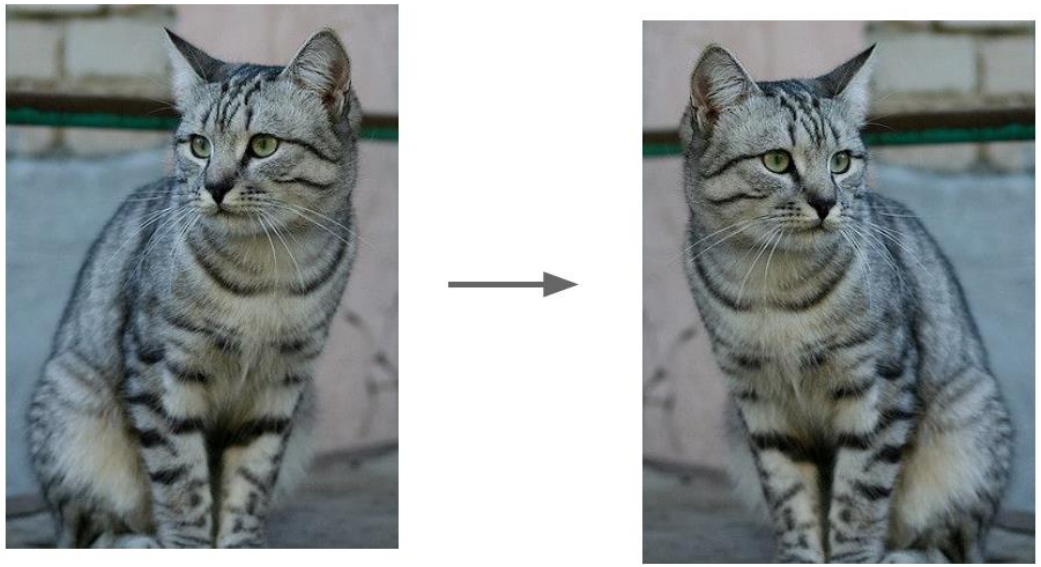
Class-Invariance Assumption

클래스를 바꾸지 않아야 한다!



이미지 데이터 확장 기법

Flips



- Horizontal flip, vertical flip

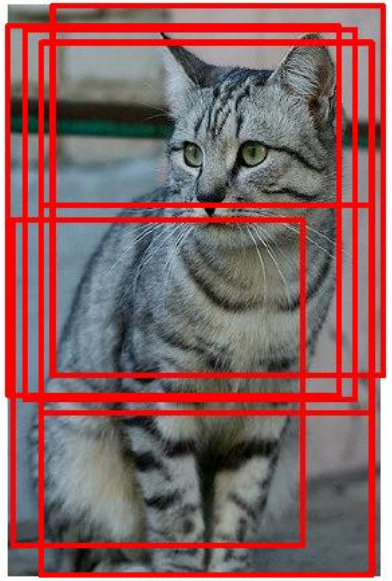
Color Jitter

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)

이미지 데이터 확장 기법

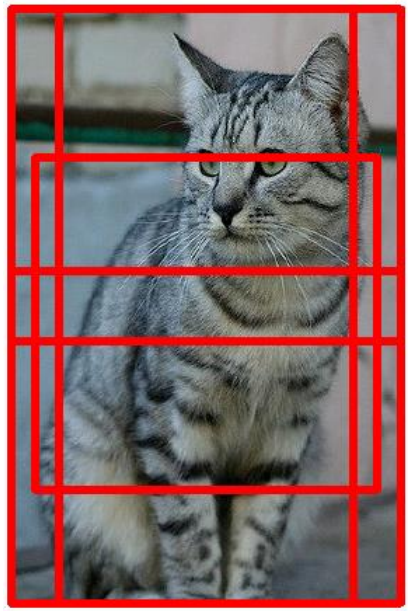
Random crops and scales



훈련 : 랜덤하게 확대 및 자르기

ResNet:

1. [256, 480] 범위에서 L을 임의로 선택
2. 이미지 크기를 가로 세로 중 짧은 쪽 길이가 L이 되게 스케일링
3. 224 x 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 x 224 10개 자르기 : (4 모퉁이 + 가운데) x 2 (flip)

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

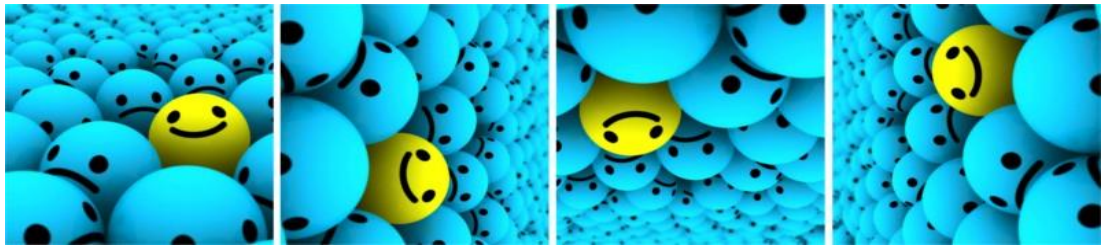
이미지 데이터 확장 기법

여러 방법들을 혼합해서 사용

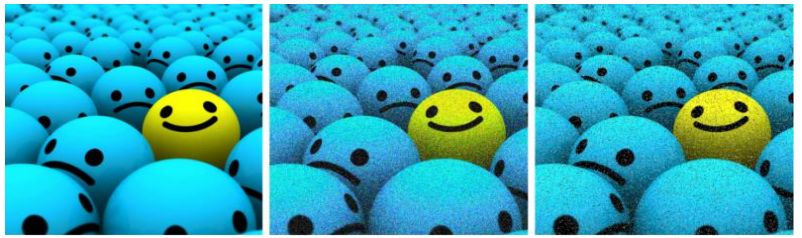
- translation
- rotation
- stretching
- shearing
- lens distortions (warping)

Translation

Rotation

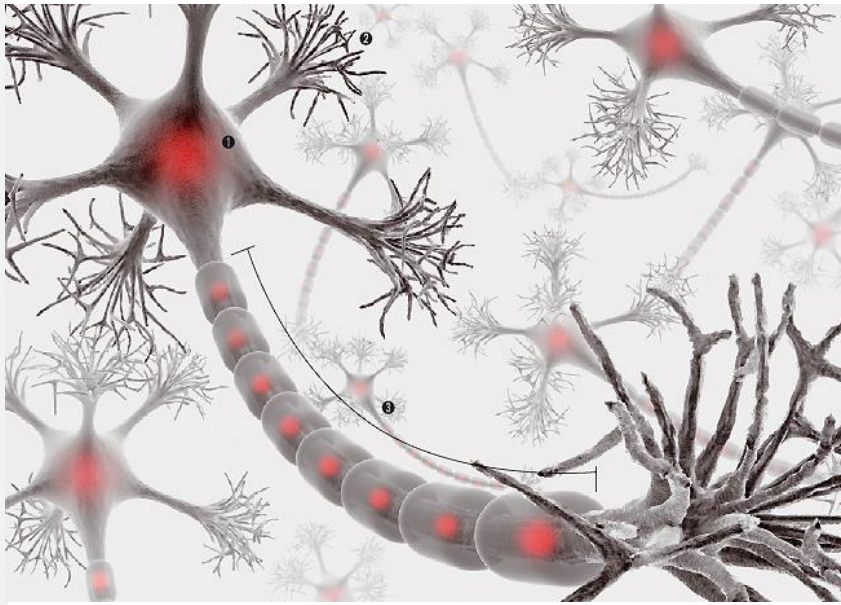


Gaussian Noise Injection



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

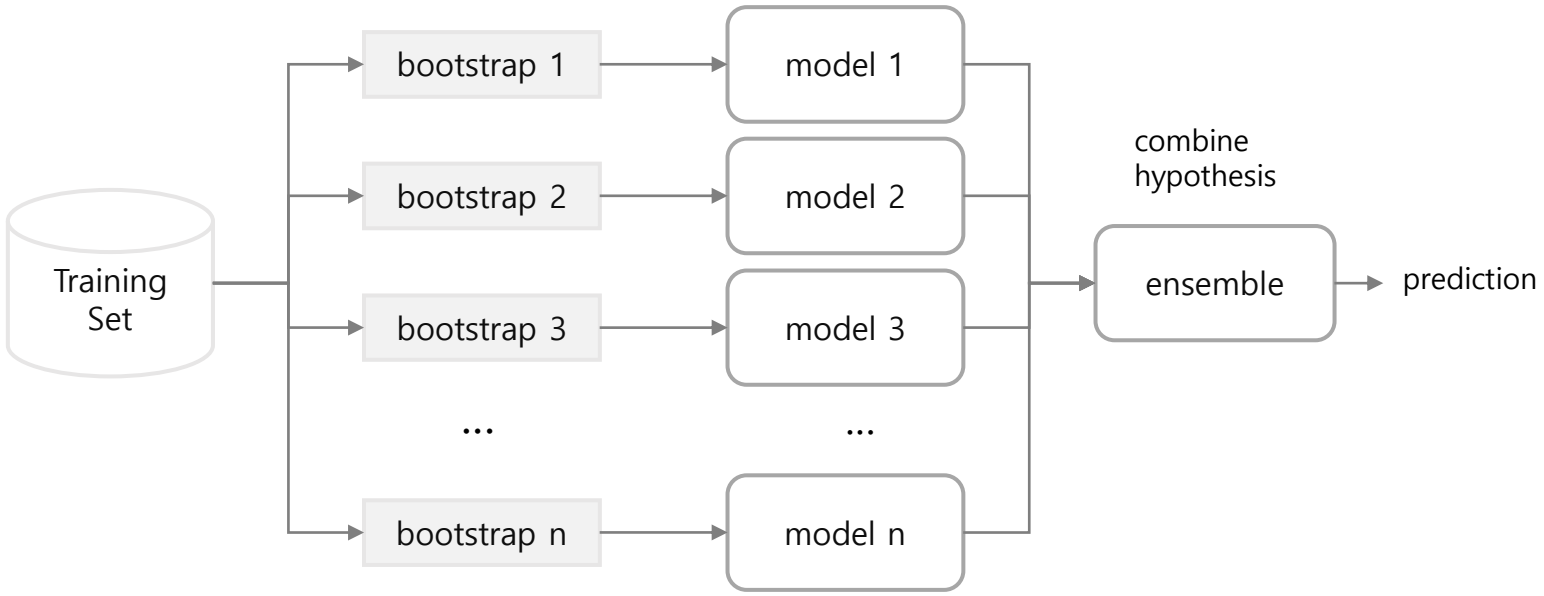
7 앙상블 (Ensemble)



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

정규화 효과

앙상블 오차에 대한 분산

(Expected squared error of the ensemble predictor)

k : Regression model 개수

$\epsilon_i \sim N(x \mid 0, \Sigma)$: Model error $(i = 1, 2, \ldots, k)$

Variance : $\mathbb{E}[\epsilon_i^2] = v$ covariance : $\mathbb{E}[\epsilon_i \epsilon_j] = c$

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_{i=1}^{k}\epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_{i=1}^{k}\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right]$$

앙상블 예측에 대한 오차의 분산

$$= \frac{1}{k^2}\left[\sum_{i=1}^{k}\mathbb{E}(\epsilon_i^2) + \sum_{i=1}^{k}\sum_{j\neq i}\mathbb{E}(\epsilon_i\epsilon_j)\right]$$

$$= \frac{1}{k^2}kv + \frac{1}{k^2}k(k-1)c$$

$$= \frac{1}{k}v + \frac{k-1}{k}c$$

모델 오차가 완전 상관되어 있고 $c = v$ 라면

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_{i}\epsilon_i\right)^2\right] = v$$

각 모델의 오차 분포를 그대로 유지

모델 오차가 서로 독립이면?

$c = 0$ 이므로

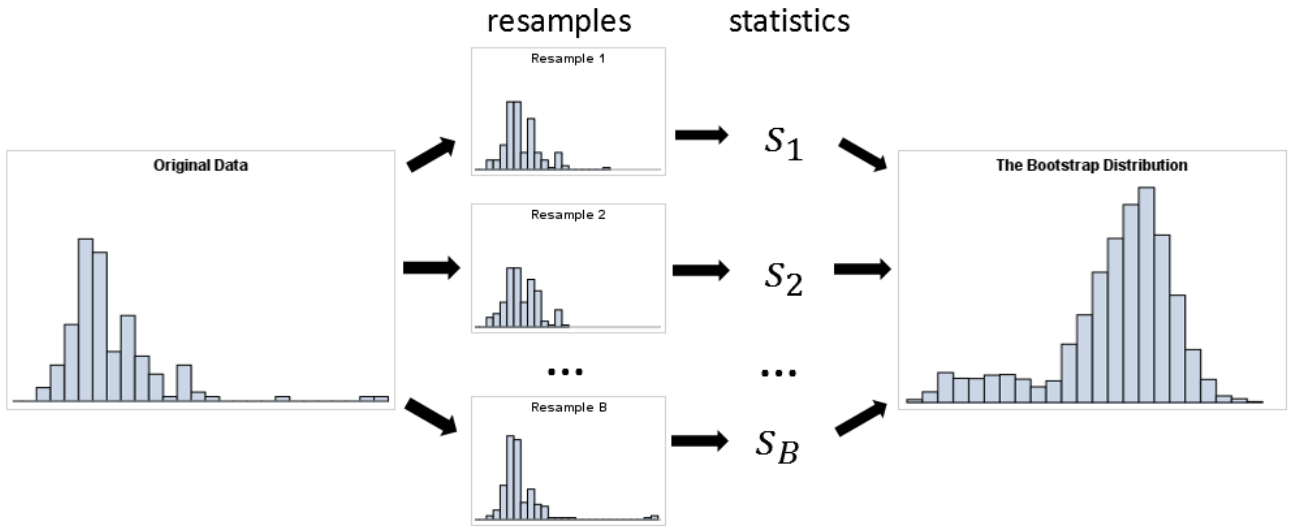
$$\mathbb{E}\left[\left(\frac{1}{k}\sum_{i}\epsilon_i\right)^2\right] = \frac{1}{k}v$$

모델 수에 비례해서 오차를 줄어든다!

앙상블은 독립된 모델을 사용해서 오차를 줄이는 모델

참고 부트스트랩 (Bootstrapping)

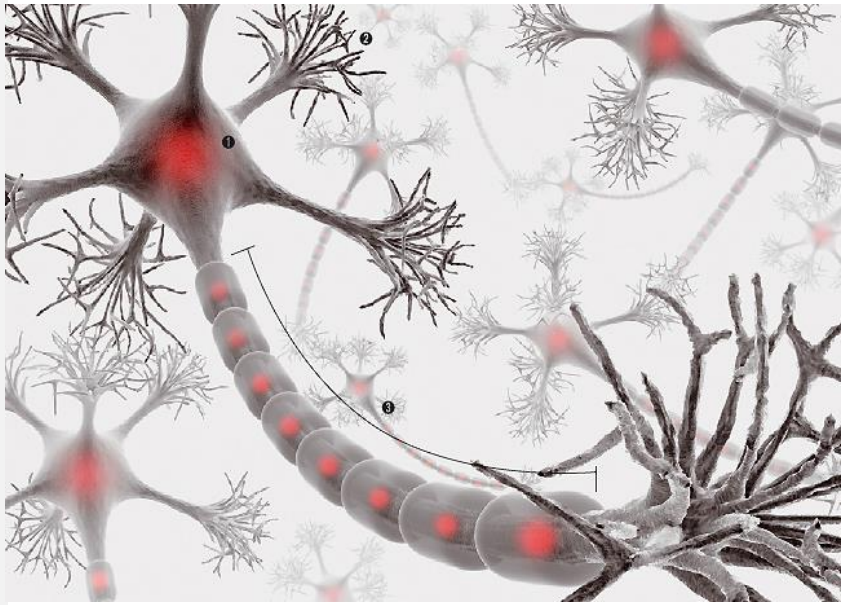
"통계량이나 신뢰구간 추정하기 위해 여러 번 복원 추출을 해서 표본 통계량을 측정"



복원 추출을 하면 30%정도는 재사용 되지 않음

그림 출처 : https://blogs.sas.com/content/iml/2018/12/12/essential-guide-bootstrapping-sas.html

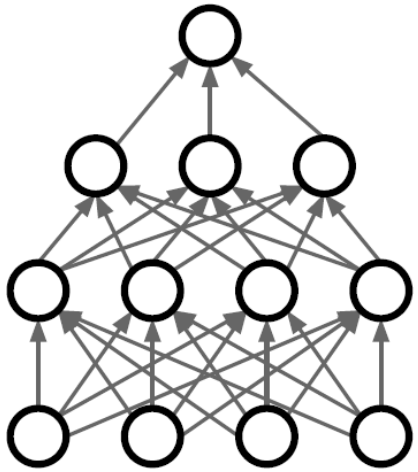
8 드롭아웃 (Dropout)



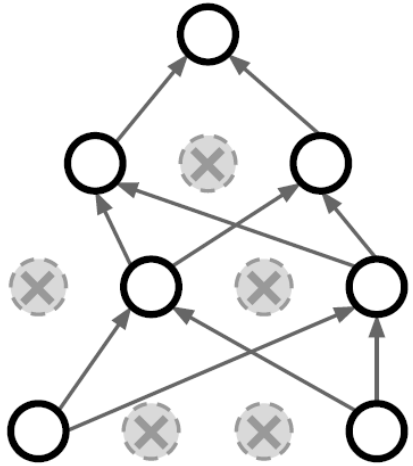
드롭아웃 (Dropout)

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

신경망



드롭아웃 적용 신경망

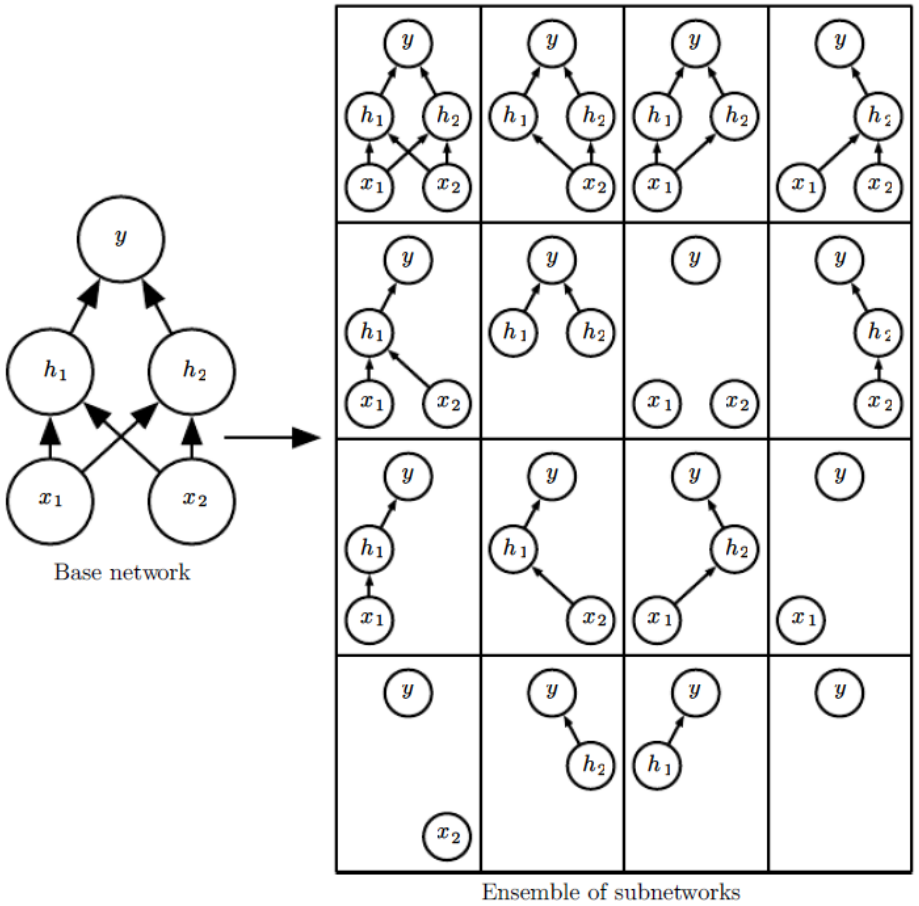


- 미니 배치 실행 시마다 뉴런을 랜덤하게 dropout해서 새로운 모델을 만드는 방법
- 계산 시간이 거의 들지 않고 다양한 모델에 대해 정규화 할 수 있는 강력한 방법
- 배깅과 다른 점은 파라미터를 모든 모델이 공유한다는 점

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

훈련 방식

입력 뉴런 2개, 은닉 뉴런 2개인 경우 예시



[그림] Deep Learning, Ian Goodfellow, Yoshua Bengio, Aaron Courville

- Dropout 대상 : 입력 뉴런과 은닉 뉴런
- 뉴런을 유지할 확률은 하이퍼파라미터
  - 입력 뉴런 0.8
  - 은닉 뉴런 0.5

훈련 방식

미니 배치 실행 시마다

- 입력 및 은닉 뉴런에 대한 Binary Mask 생성
  - Mask 값이 1이면 유지
  - Mask 값이 0이면 Dropout
- 뉴런의 출력에 Binary Mask 곱함

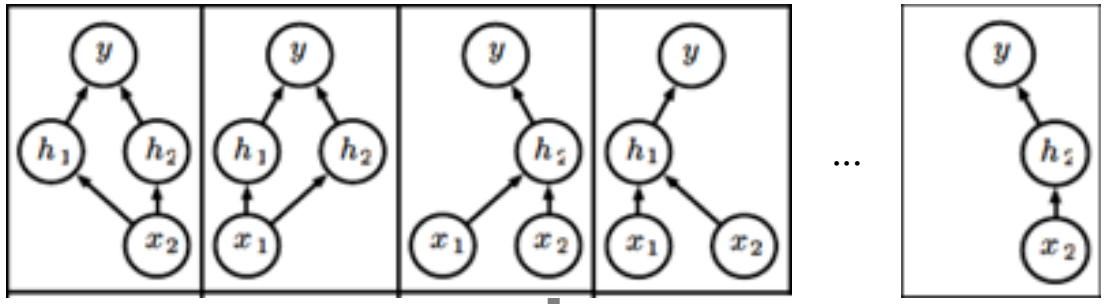
3계층 네트워크 훈련 코드 예시

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

추론 방식

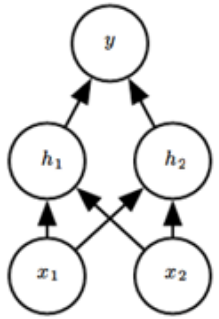
출력이 random

훈련



Average Out

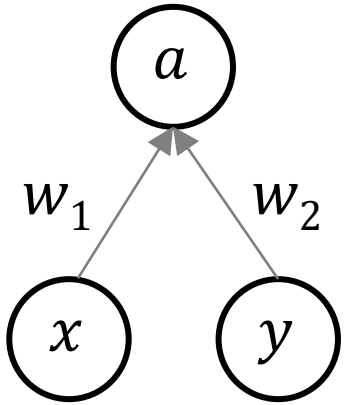
테스트



평균 Output 생성

Output at test time = Expected output at training time

추론 방식



입력이 2개인 경우

p = 0.5

훈련

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

$$= \frac{1}{2}(w_1x + w_2y)$$

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

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

추론 방식

드롭아웃 추론 코드 (3계층 네트워크)

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Output at test time = Expected output at training time

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

추론 방식

Inverted Dropout

```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

Inverted Dropout

- 훈련 시점에 배율을 미리 나눠둠
- 테스트 코드에 변경사항이 없음
- 일반적으로 가장 많이 사용함

test time is unchanged!

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

Thank you!