2022 WINTER ESC

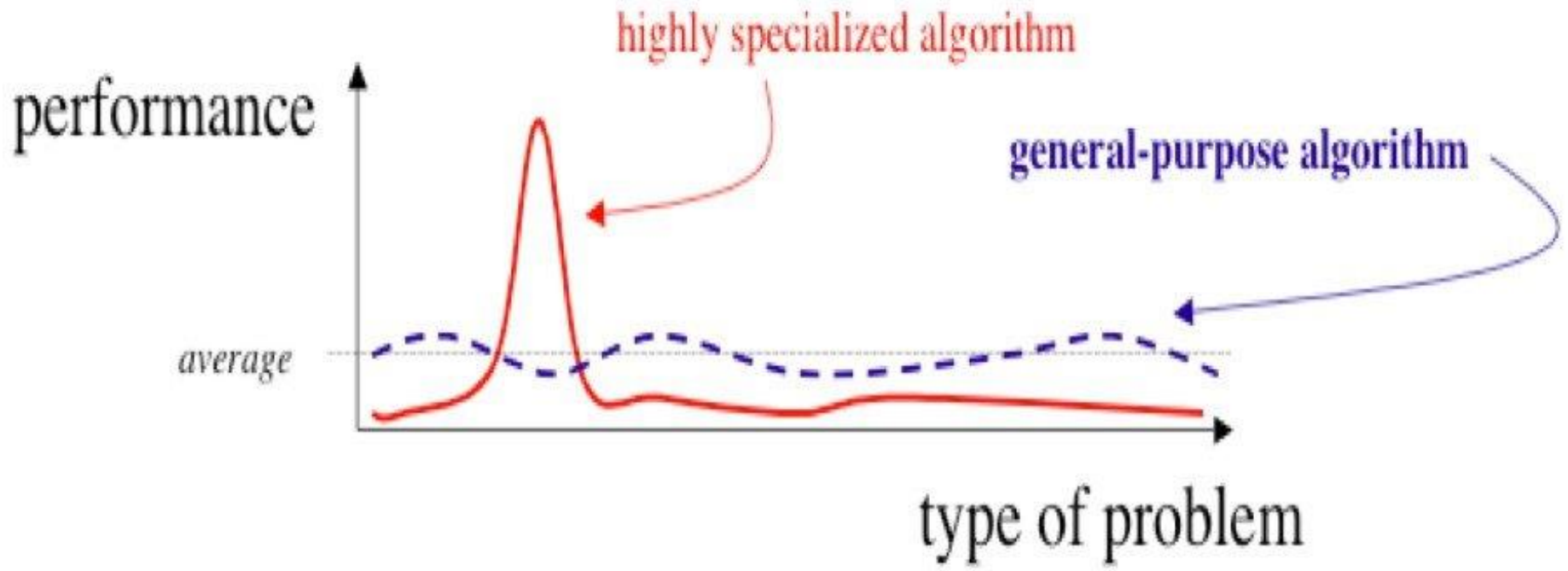
Deep Learning Problem Solving Technique

고정민, 박상용

Week 3

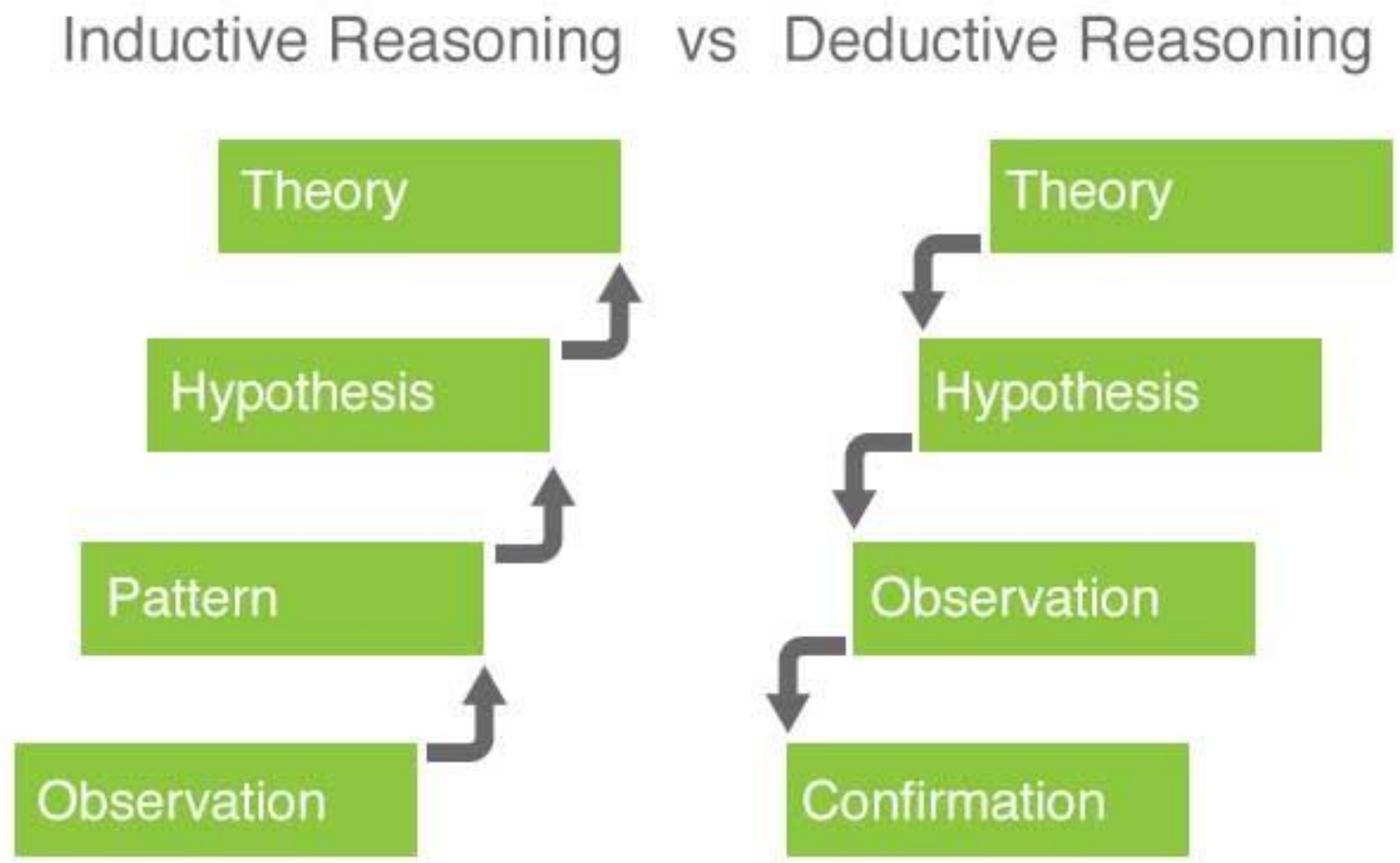
CONTENTS

No Free Lunch Theorem for Optimization



We have dubbed the associated results "No Free Lunch" theorems because they demonstrate that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems.

No Free Lunch Theorem for Optimization



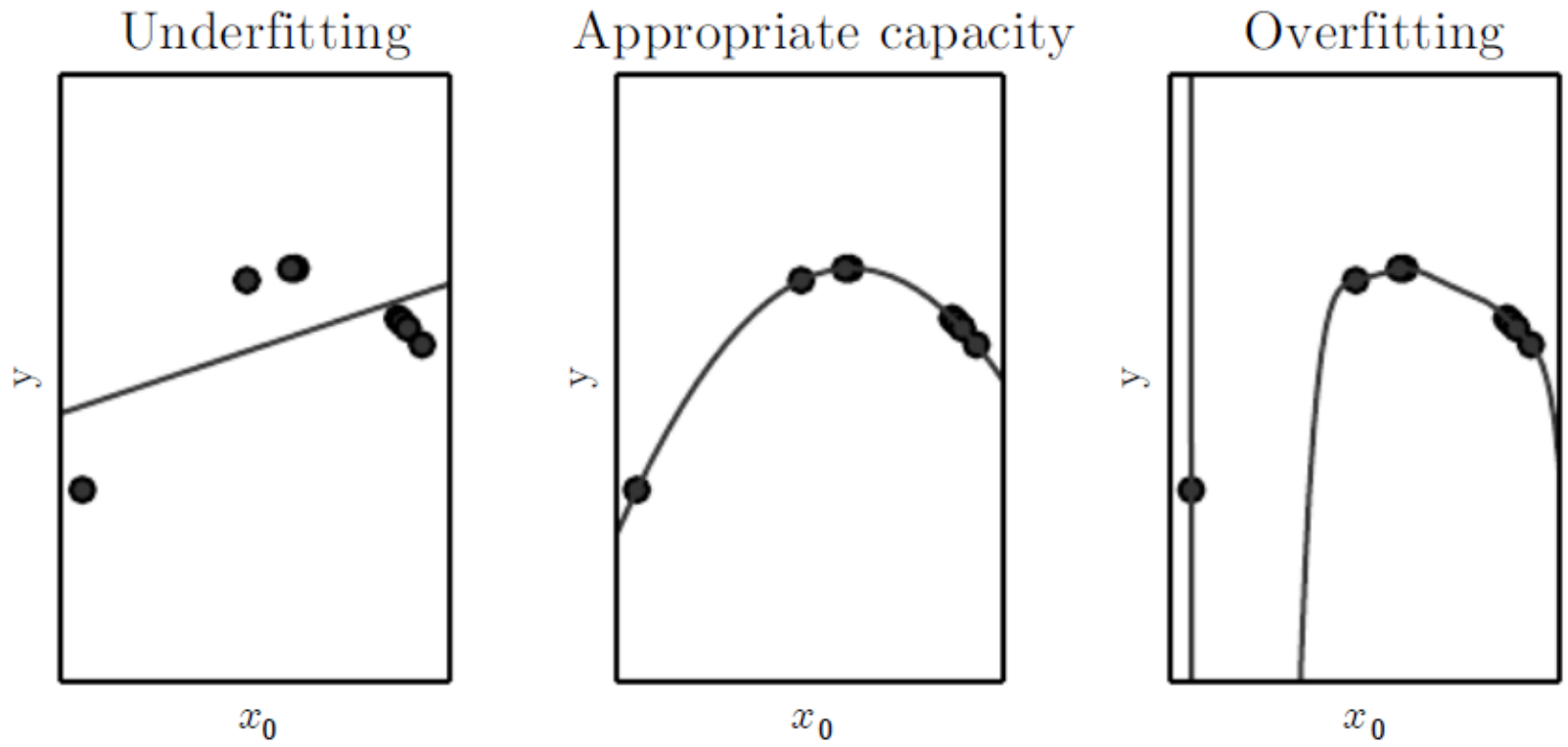
No Free Lunch Theorem for Optimization

The factors determining how well a machine learning algorithm will perform

1. Make the training error small.
2. Make the gap between training and test error small.

No Free Lunch Theorem for Optimization

Need to control the capacity of a learning algorithm



No Free Lunch Theorem for Optimization

Bias - Variance Trade-Off

$\hat{f}(x_0) = \hat{f}$ evaluated at $X = x_0$

$$\mathbb{E}\left[\{\hat{f}(x_0) - y_0\}^2 \mid X = x_0\right]$$

$$= \mathbb{E}\left[\{\hat{f}(x_0) - (f(x_0) + \varepsilon)\}^2 \mid X = x_0\right] \quad (y = f(x) + \varepsilon\,, \mathbb{E}(\varepsilon) = 0\,, Var(\varepsilon) = {\sigma_\varepsilon}^2)$$

$$= \mathbb{E}\left[\left\{\hat{f}(x_0) - (f(x_0) + \varepsilon) + \mathbb{E}\left(\hat{f}(x_0)\right) - \mathbb{E}\left(\hat{f}(x_0)\right)\right\}^2 \mid X = x_0\right]$$

No Free Lunch Theorem for Optimization

Bias - Variance Trade-Off

$$= \mathbb{E}\left[\left\{\varepsilon - \left[\mathbb{E}\left(\hat{f}(x_0)\right) - f(x_0)\right] - \left[\hat{f}(x_0) - \mathbb{E}\left(\hat{f}(x_0)\right)\right]\right\}^2 \middle| X = x_0\right]$$

$$= \mathbb{E}(\varepsilon^2) + \mathbb{E}\left[\left\{\mathbb{E}\left(\hat{f}(x_0)\right) - f(x_0)\right\}^2\right] + \mathbb{E}\left[\left\{\hat{f}(x_0) - \mathbb{E}\left(\hat{f}(x_0)\right)\right\}^2 \mid X = x_0\right]$$

$$= {\sigma_\varepsilon}^2 + \left\{\mathbb{E}\left(\hat{f}(x_0)\right) - f(x_0)\right\}^2 + Var(\hat{f}(x_0))$$

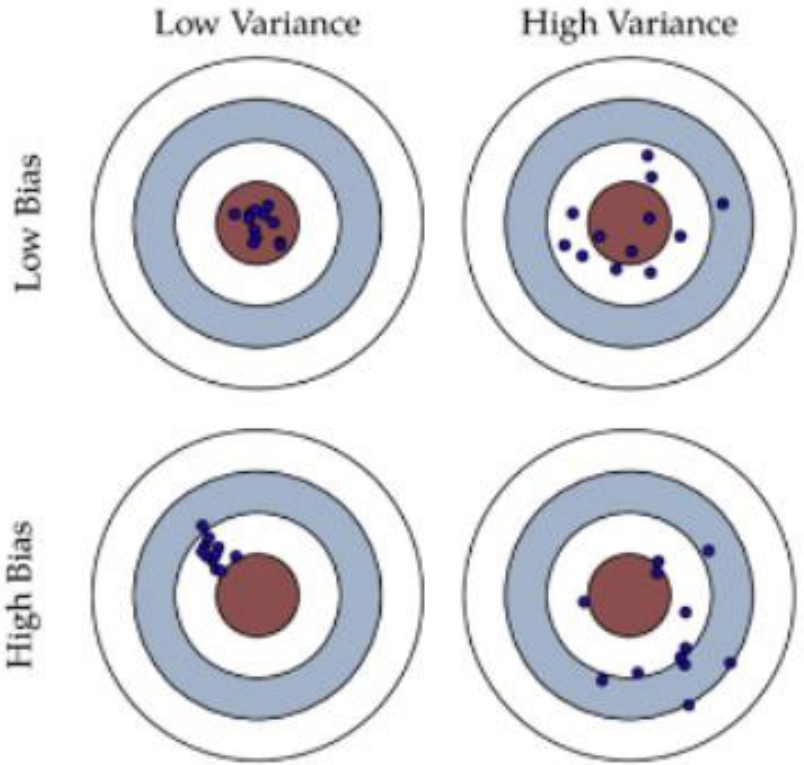
$$= {\sigma_\varepsilon}^2 + Bias^2\left(\hat{f}(x_0)\right) + Var(\hat{f}(x_0))$$

No Free Lunch Theorem for Optimization

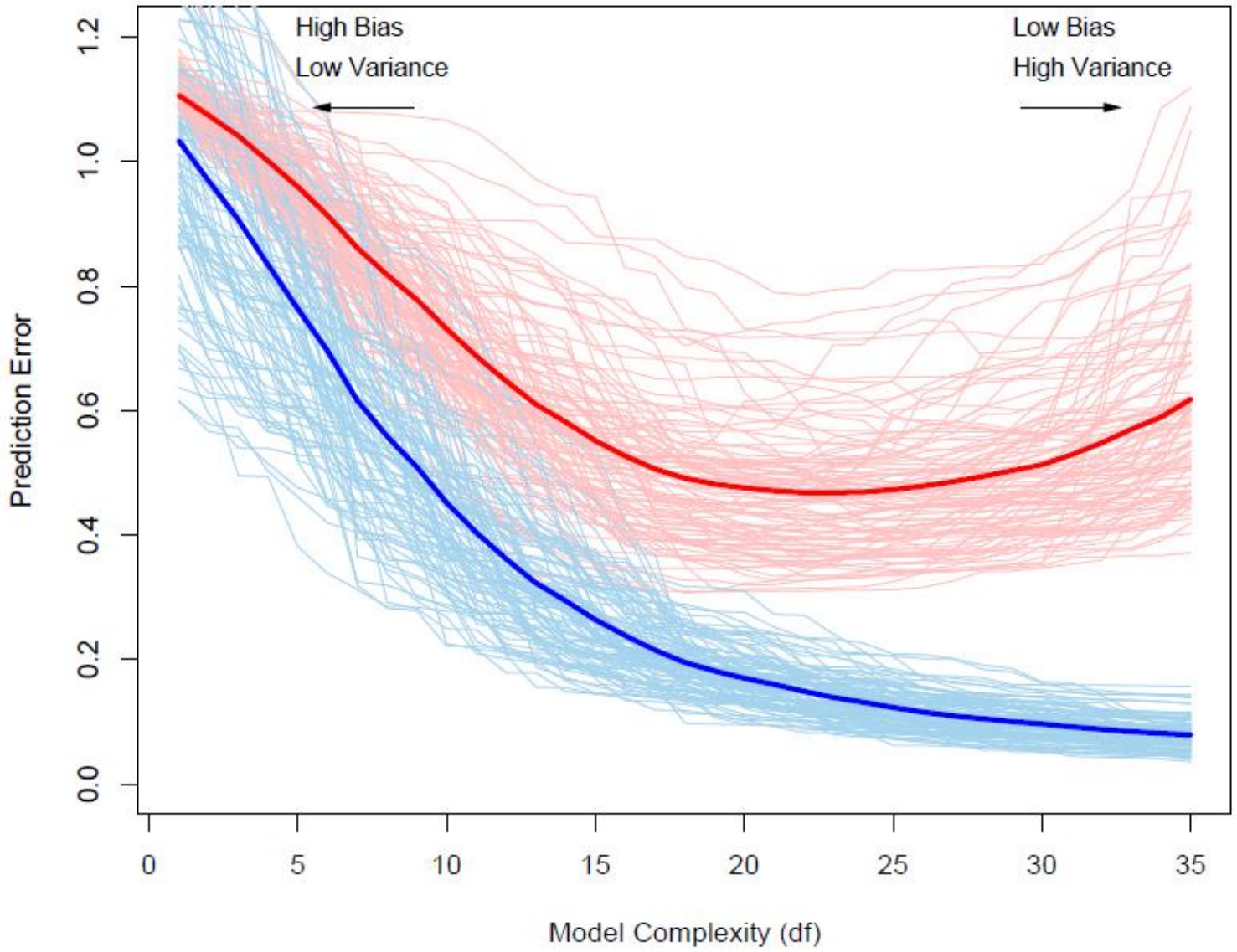
Bias - Variance Trade-Off

$$\sigma_\varepsilon{}^2 + Bias^2\,(\hat{f}(x_0)) + Var(\hat{f}(x_0))$$

$$Irreducible\ Error + Bias^2 + Variance$$



No Free Lunch Theorem for Optimization



Regularization - Weight Decay

출처 : https://youtu.be/edMByVIO8MM

Regularization - Weight Decay

$$f(\boldsymbol{x}\,;\,\boldsymbol{\theta}) \leftarrow \Omega(\boldsymbol{w}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{w} \quad \text{(Regularizer)}$$

$$J(\boldsymbol{w}) = MSE_{train} + \lambda \boldsymbol{w}^{\mathsf{T}}\boldsymbol{w}$$

Minimizing $J(w)$ results in a choice of weights that make a trade-off between fitting the training data and being small.

Regularization - Weight Decay

Norm

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}, \quad p \in \mathbb{R}, \quad p \geq 1$$

Regularization - Weight Decay

L^2 Norm (Euclidean Norm)

$$\|x\|_2 = \left(\sum_i |x_i|^2\right)^{\frac{1}{2}} \quad \rightarrow \quad \|x\|_2^2 = \sum_i |x_i|^2$$

Squared L^2 Norm

L^1 Norm

$$\|x\|_1 = \sum_i |x_i|$$

Regularization - Weight Decay

Parameter Norm Penalties

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

Different choices for the parameter norm Ω can result in different solutions being preferred.

Regularization - Weight Decay

L^2 Parameter Regularization

(a.k.a. Ridge Regression / Tikhonov Regularization)

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 = \frac{1}{2}\boldsymbol{w}^\intercal\boldsymbol{w}$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\intercal\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Regularization - Weight Decay

L^2 Parameter Regularization

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

with the corresponding parameter gradient,

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Regularization - Weight Decay

L^2 Parameter Regularization

To take a single gradient step to update the weights, we perform this update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \right)$$

Written another way, the update is:

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}))$$

Regularization - Weight Decay

L^1 Parameter Regularization (a.k.a. Lasso Regression)

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Regularization - Weight Decay

L^1 Parameter Regularization

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

with the corresponding gradient (actually, sub-gradient),

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \cdot sign(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Regularization - Weight Decay

Non-differentiable functions and subgradients

Given a convex function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$,
a vector $z \in \mathbb{R}^n$ is said to be a $subgradient$ of f at x_0 if

$$f(x) \geq f(x_0) + \langle z, x - x_0 \rangle \quad \text{for all} \quad x \in \mathbb{R}^n$$

The set of all subgradients of f at x_0 is called the $subdifferential$ and denoted by $\partial f(x_0)$.

When f is differentiable at x_0, then the sub-differential reduces to a single vector - that is $\partial f(x_0) = \{\nabla f(x_0)\}$.

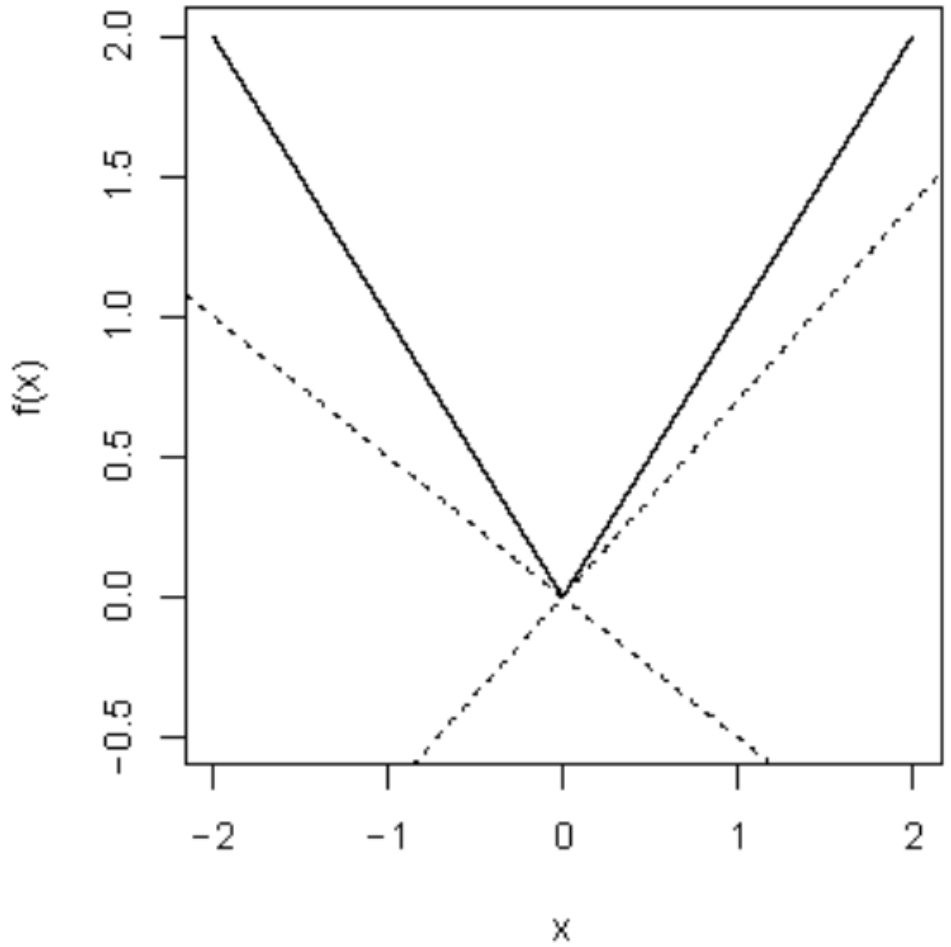
Regularization - Weight Decay

Non-differentiable functions and subgradients

Example 1

$$f : \mathbb{R} \longrightarrow \mathbb{R},$$

$$f(x) = \|x\|_1 = |x|$$



Regularization - Weight Decay

Non-differentiable functions and subgradients

Example 1

$$f : \mathbb{R} \longrightarrow \mathbb{R}, \quad f(x) = \|x\|_1 = |x|$$

$$\partial f(x) = \begin{cases} \{+1\}, & if\ x > 0 \\ \{-1\}, & if\ x < 0 \\ [-1, 1], & if\ x = 0 \end{cases}$$

Which is frequently denoted by $sign(x)$. $z \in sign(x)$: z belongs to the sub-differential of the absolute value function at x.

Regularization - Weight Decay

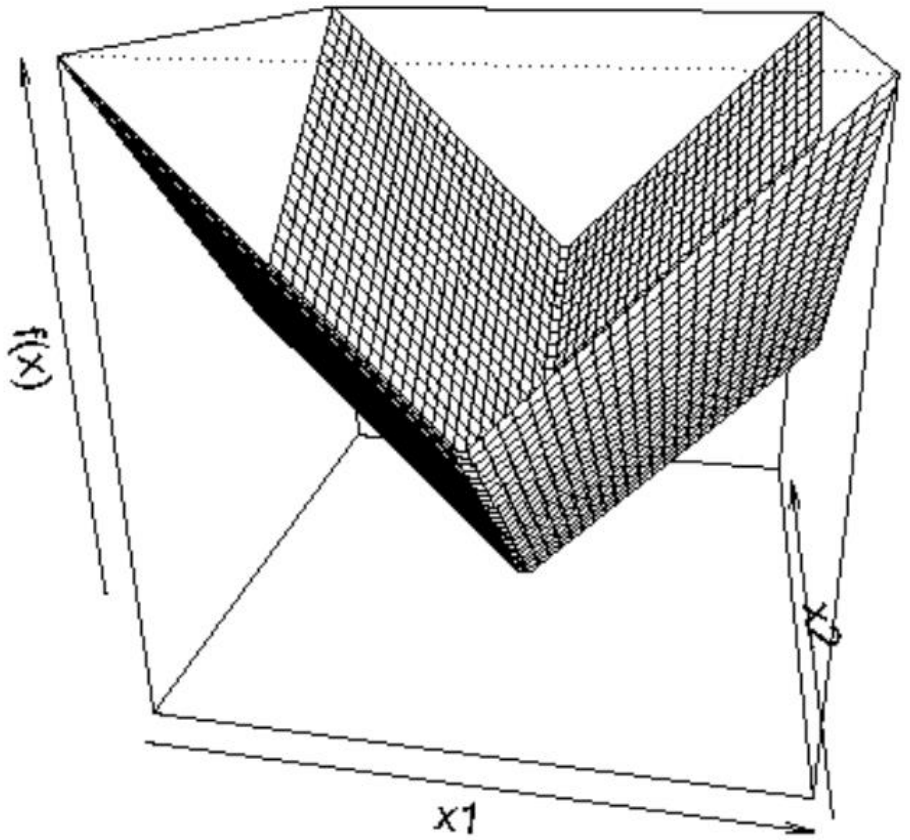
Non-differentiable functions and subgradients

Example 2

$$f : \mathbb{R}^n \longrightarrow \mathbb{R},$$

$$f(x) = \|x\|_1 = |x|$$

$$x = (x_1, x_2, x_3, \ldots, x_n)$$



Regularization - Weight Decay

Non-differentiable functions and subgradients

Example 2

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}, \qquad f(x) = \|x\|_1 = |x|$$

$$\partial f(x_i) = \begin{cases} \{+1\}, & if\ x_i > 0 \\ \{-1\}, & if\ x_i < 0 \\ [-1, 1], & if\ x_i = 0 \end{cases} \quad for\ i \in \{1, 2, \dots, n\}$$

Which is frequently denoted by $sign(x_i)$. $z \in sign(x_i)$: z belongs to the sub-differential of the absolute value function at x_i.

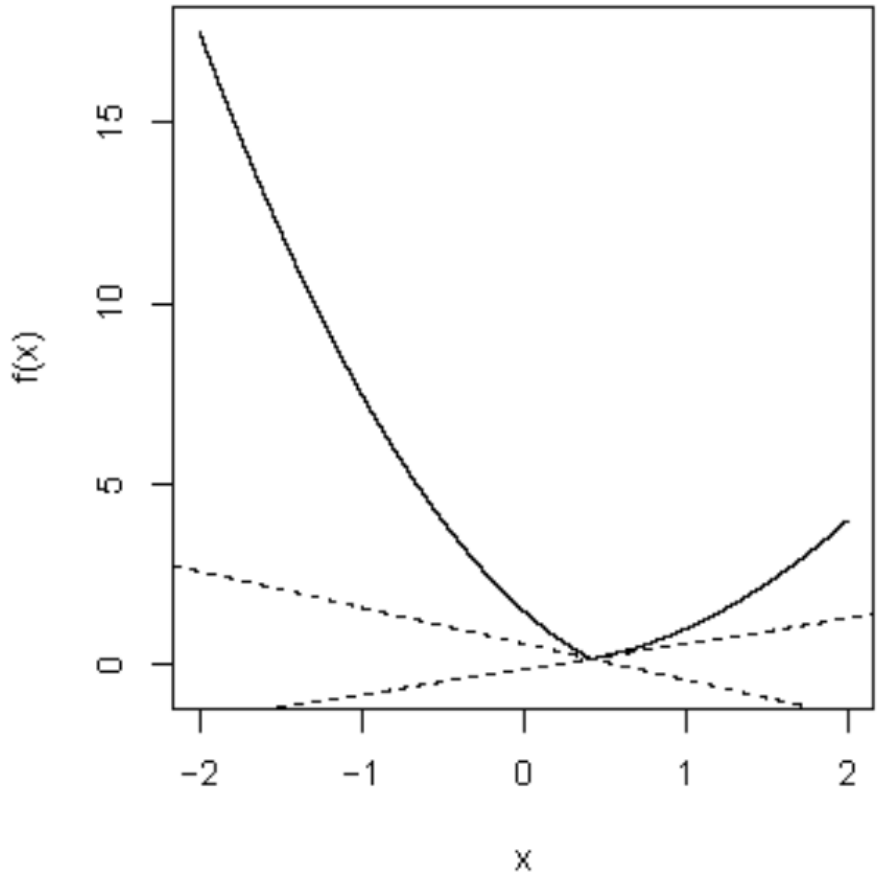
Regularization - Weight Decay

Non-differentiable functions and subgradients

Example 3

$$f(x) = \max\{f(x_1), f(x_2)\}$$

$$f_1, f_2 : \mathbb{R}^n \longrightarrow \mathbb{R}$$



Regularization - Weight Decay

Non-differentiable functions and subgradients

Example 3

$$f(x) = \max\{f(x_1), f(x_2)\}\ , \quad f_1, f_2 : \mathbb{R}^n \longrightarrow \mathbb{R}$$

$$f_1(x) > f_2(x) \rightarrow \partial f(x) = \nabla f_1(x)$$

$$f_1(x) < f_2(x) \rightarrow \partial f(x) = \nabla f_2(x)$$

$$f_1(x) = f_2(x) \rightarrow \partial f(x) \in \{\theta_1 \nabla f_1(x) + \theta_2 \nabla f_2(x) : \theta_1 + \theta_2 = 1, \theta_1 \geq 0, \theta_2 \geq 0\}$$

Regularization - Weight Decay

Elastic Net

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = r\alpha\|\boldsymbol{w}\|_1 + \frac{1-r}{2}\alpha\boldsymbol{w}^\top\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

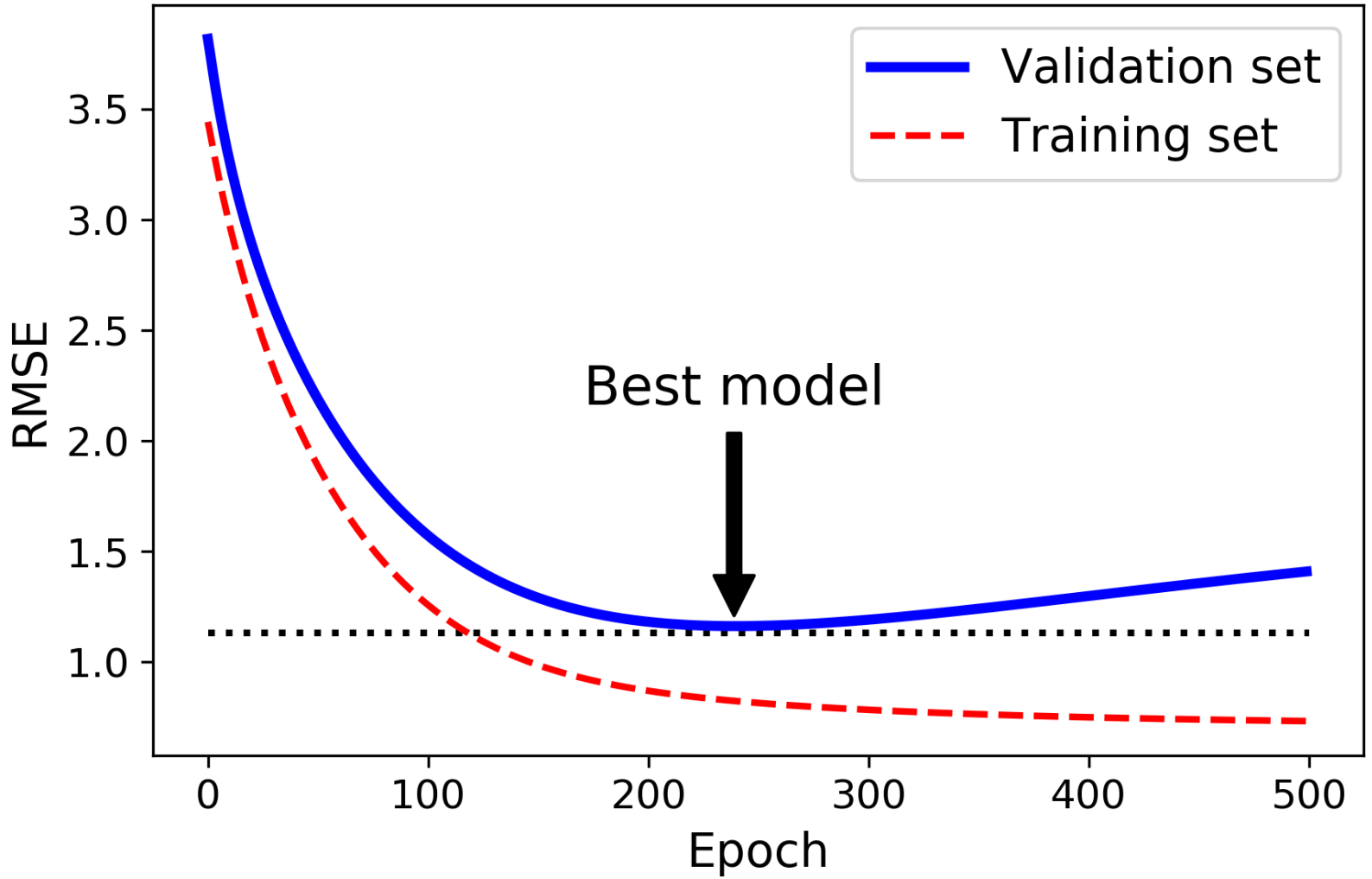
$r = 0 \longrightarrow L^2$ regularization

$r = 1 \longrightarrow L^1$ regularization

Regularization - Weight Decay

Ridge → Lasso → Elastic Net

Regularization - Early Stopping



Regularization - Early Stopping

Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.

Let p be the "patience," the number of times to observe worsening validation set error before giving up.

Let $\boldsymbol{\theta}_o$ be the initial parameters.

Regularization - Early Stopping

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$

while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

"Unobstrusive"

Regularization - Early Stopping

Algorithm 7.2 A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (Algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.
Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for i^* steps.

Regularization - Early Stopping

Algorithm 7.2 A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (Algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.
Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for i^* steps.

Ensemble Method

Wisdom of Crowds

Ensemble Method

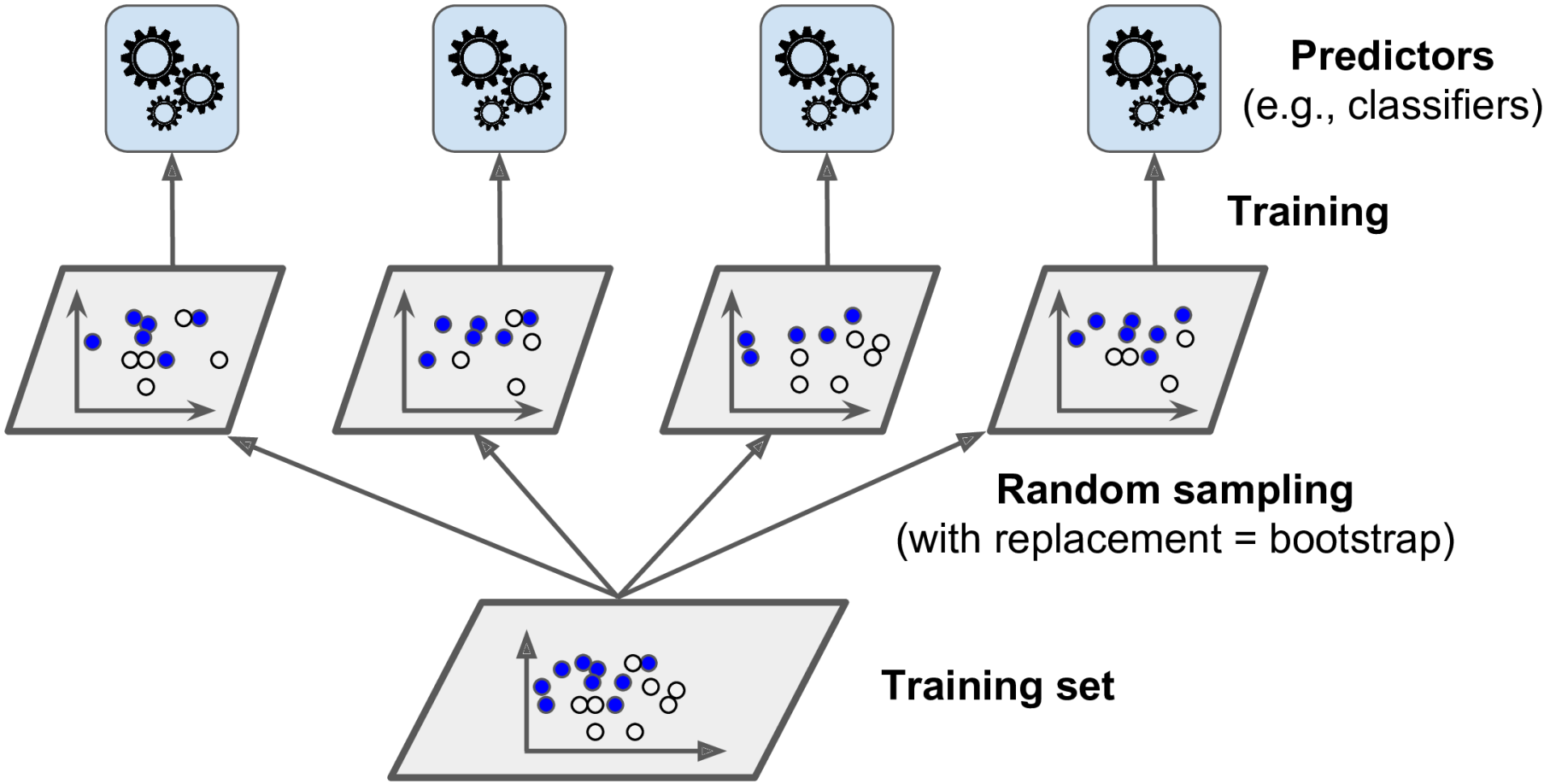
Ensemble Method - Model Averaging

The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models.

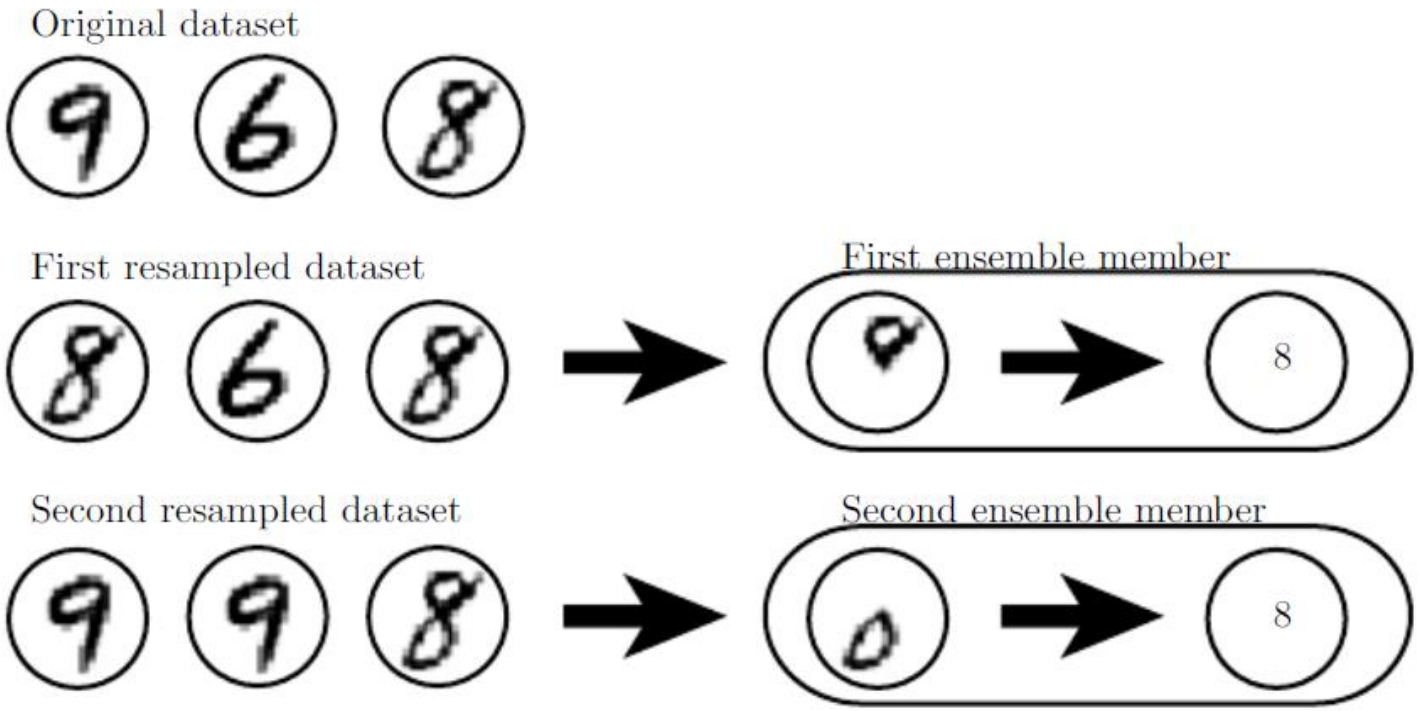
Bagging (Bootstrap aggregation)

The idea is to train several different models separately, then have all of the models vote on the output for test examples.

Ensemble Method - Bagging



Ensemble Method - Bagging



The first dataset omits the '9' and repeats the '8'. On this dataset, the detector learns that a loop on top of the digit corresponds to an '8'. On the second dataset, we repeat the '9' and omit the '6'. In this case, the detector learns that a loop on the bottom of the digit corresponds to an '8'.

Ensemble Method - Bagging

Variance of the Average of Prediction

Consider for example a set of k regression models.
Suppose that each model makes an error ϵ_i on each example.

$$\epsilon_i \sim N(m, \sigma^2), \quad Corr(\epsilon_i, \epsilon_j) = \rho$$

$$\frac{1}{\sigma^2}\,\mathbb{E}\left[(\epsilon_i - m)\,(\epsilon_j - m)\right] = \rho > 0$$

$$\mathbb{E}\left[\epsilon_i \epsilon_j\right] = \rho\sigma^2 + m^2$$

Ensemble Method - Bagging

Variance of the Average of Prediction

$$\mathbb{E}[\epsilon_i] = m \qquad \mathbb{E}[\epsilon_i \epsilon_j] = \begin{cases} \rho\sigma^2 + m^2 & i \neq j \\ \sigma^2 + m^2 & i = j \end{cases}$$

$$Var\left(\frac{1}{k}\sum_i \epsilon_i\right) = \frac{1}{k^2} Var\left(\sum_i \epsilon_i\right) = \frac{1}{k^2}\left[\mathbb{E}\left[\left(\sum_i \epsilon_i\right)^2\right] - \mathbb{E}\left[\sum_i \epsilon_i\right]^2\right]$$

Ensemble Method - Bagging

Variance of the Average of Prediction

$$\mathbb{E}\left[\sum_i \epsilon_i\right] = \sum_i \mathbb{E}[\epsilon_i] = km$$

$$\left(\sum_i \epsilon_i\right)^2 = \sum_{i,j} \epsilon_i \epsilon_j \quad \mathbb{E}\left[\left(\sum_i \epsilon_i\right)^2\right] = \sum_{i,j} \mathbb{E}[\epsilon_i \epsilon_j] = k\mathbb{E}[{\epsilon_i}^2] + (k^2 - k)\mathbb{E}[\epsilon_i \epsilon_j]$$

$$= k(\sigma^2 + m^2) + k(k-1)(\rho\sigma^2 + m^2)$$

Ensemble Method - Bagging

Variance of the Average of Prediction

$$k(\sigma^2 + m^2) + k(k-1)(\rho\sigma^2 + m^2)$$

$$= k\sigma^2 + k^2\rho\sigma^2 - k\rho\sigma^2 + k^2m^2$$

$$Var\left(\frac{1}{k}\sum_i \epsilon_i\right) = \frac{1}{k^2}(k\sigma^2 + k^2\rho\sigma^2 - k\rho\sigma^2) = \frac{1}{k}(\sigma^2 + k\rho\sigma^2 - \rho\sigma^2)$$

$$= \rho\sigma^2 + \frac{1-\rho}{k}\sigma^2$$


Break Time

Session Flow

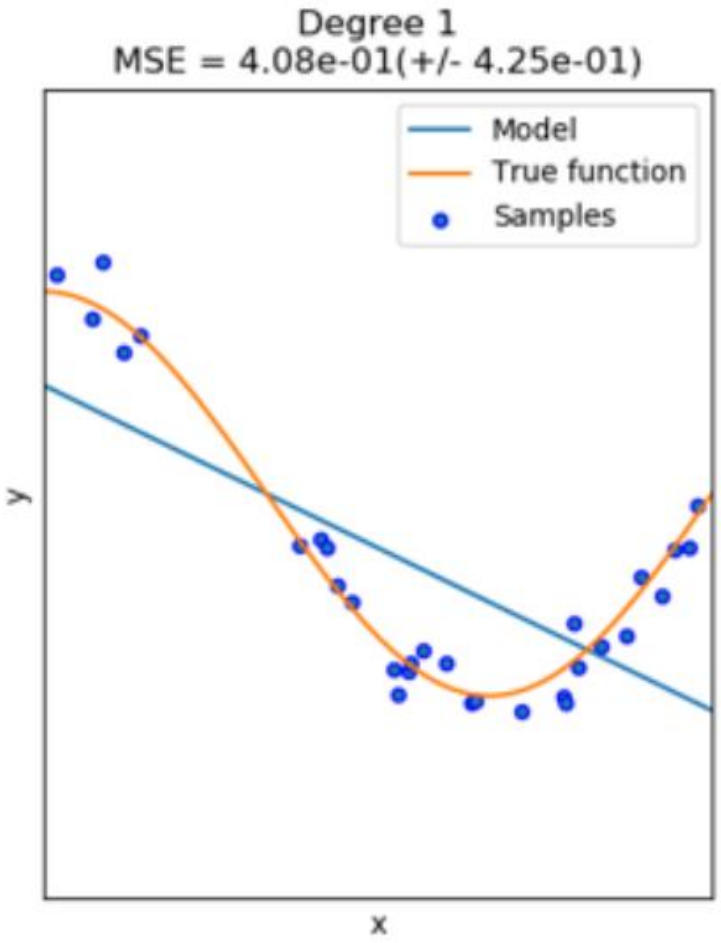
01 Curse of Dimensionality

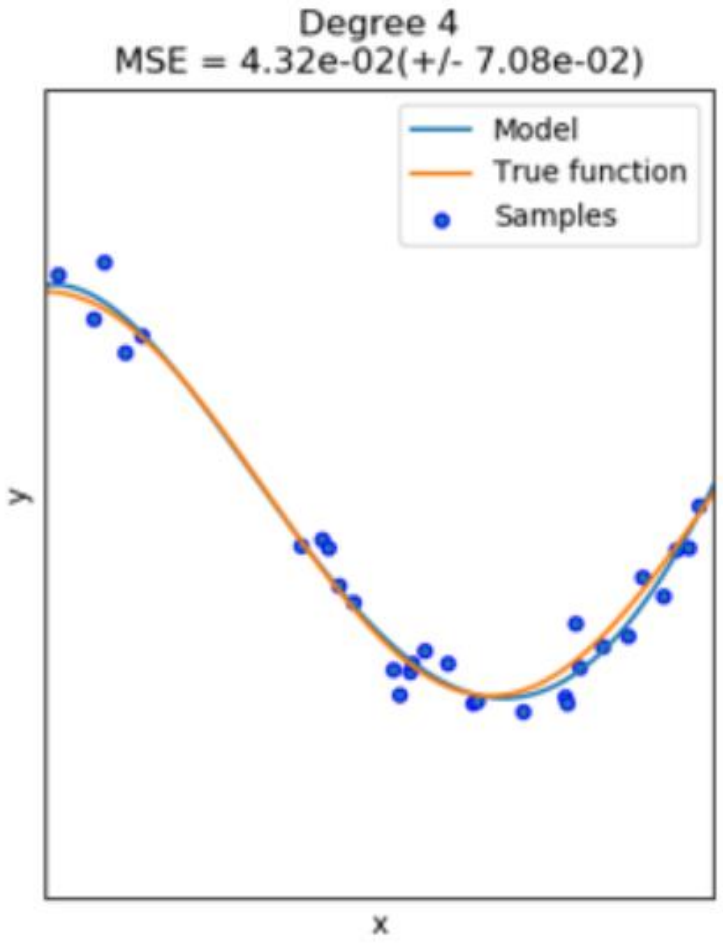
02 Dropout & Drop-Connect

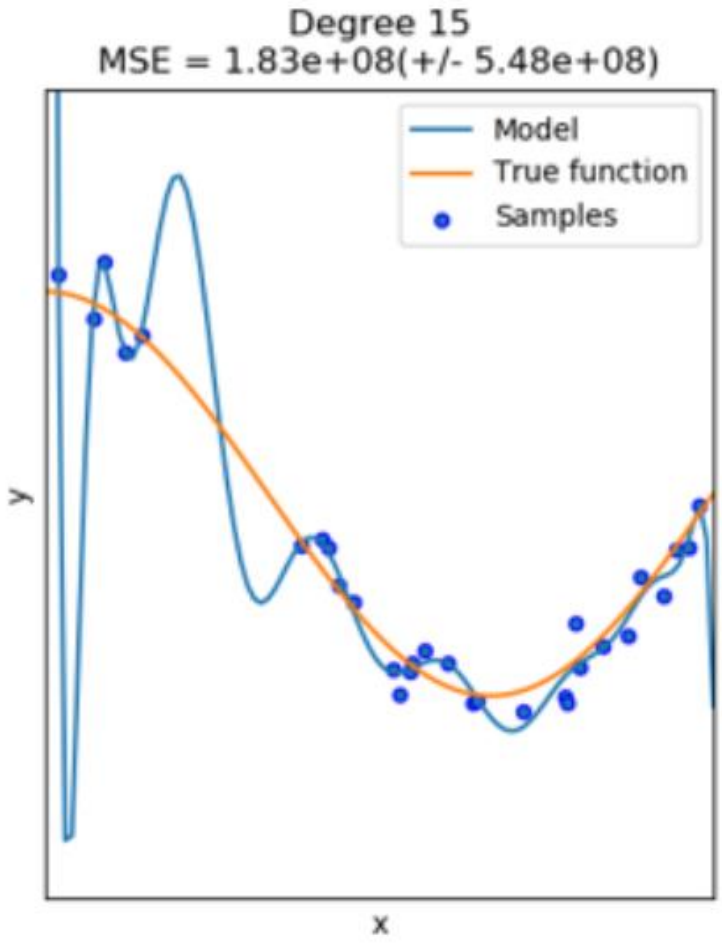
03 Batch-Normalization

04 Q & A

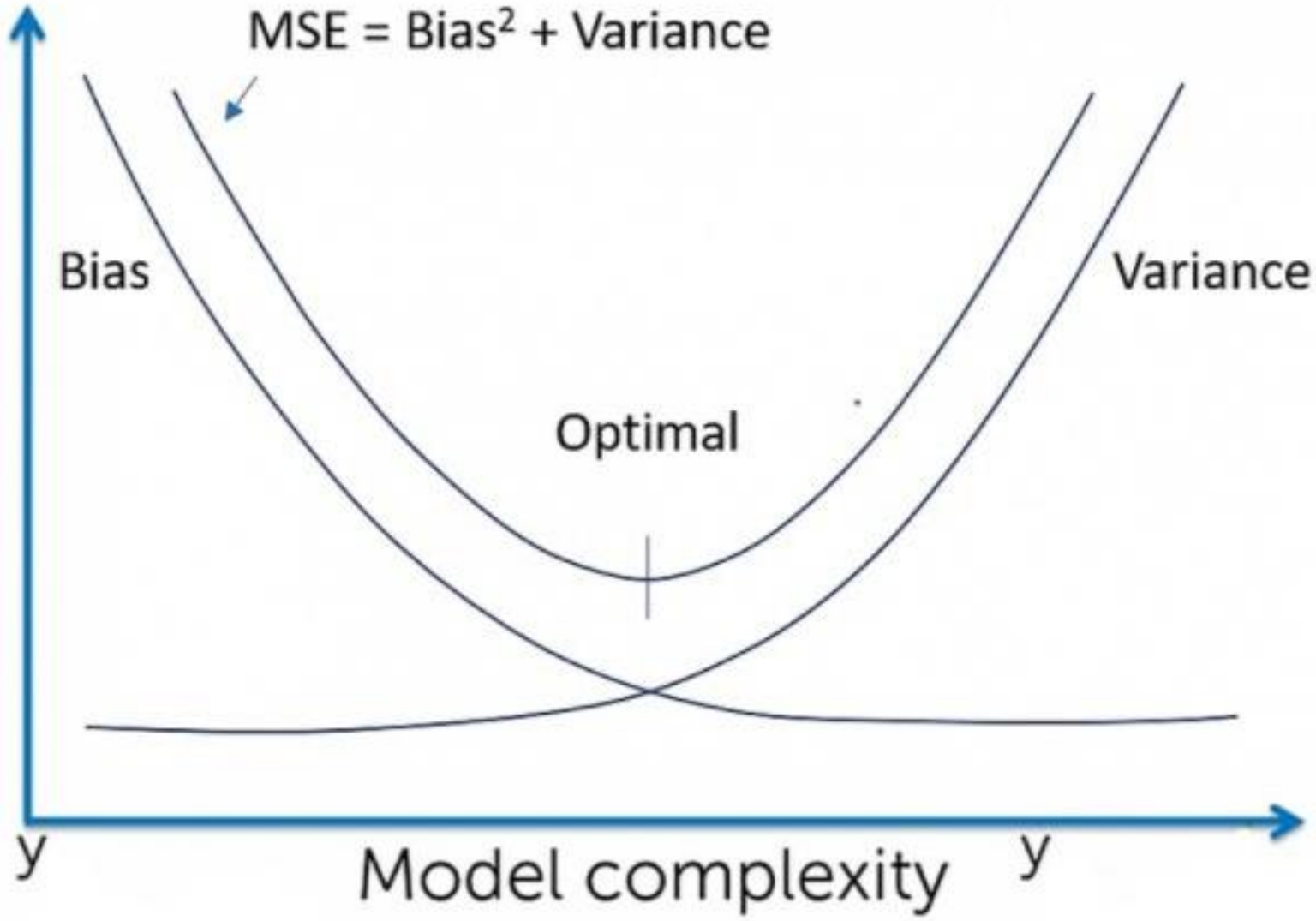
1. Over-fitting



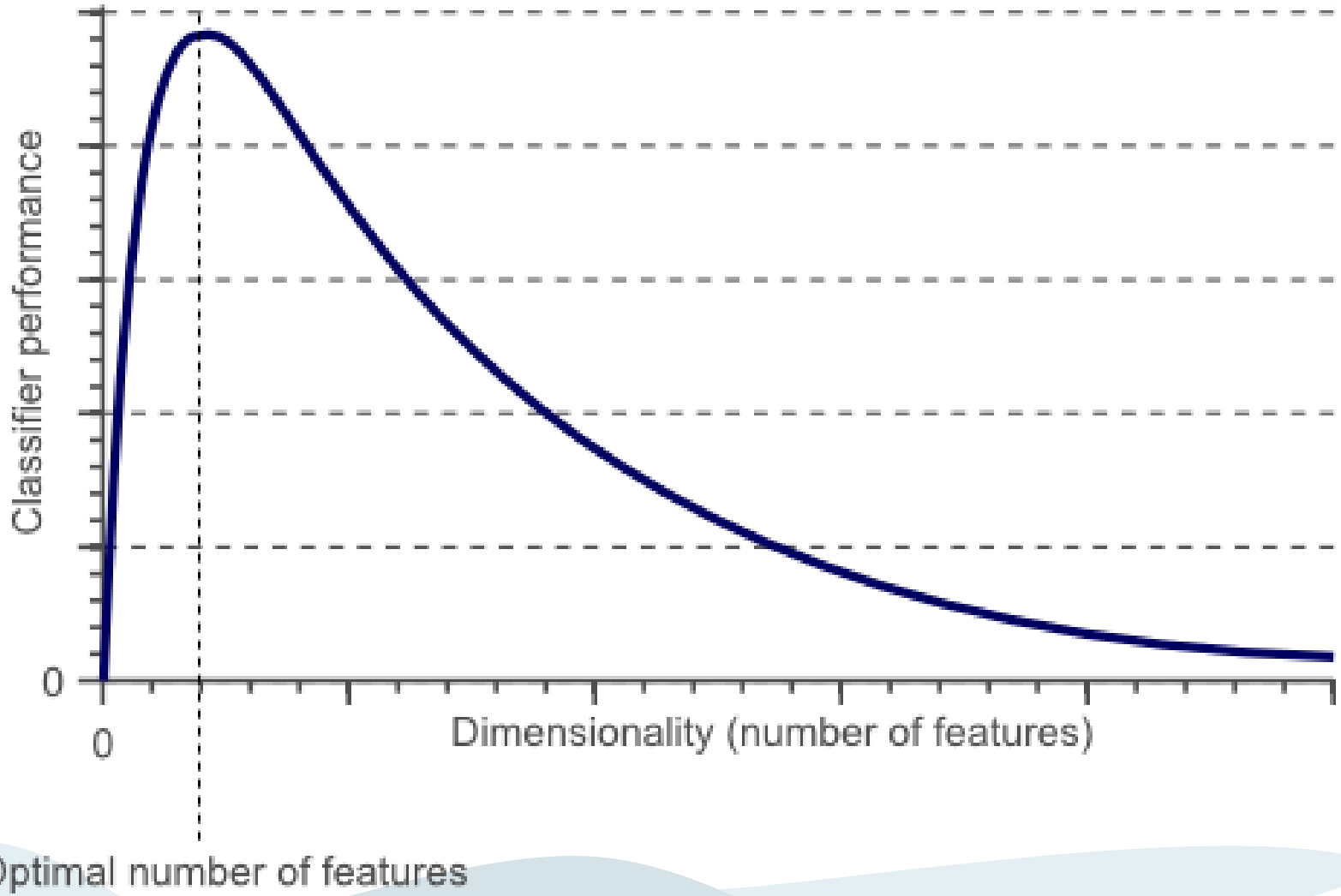




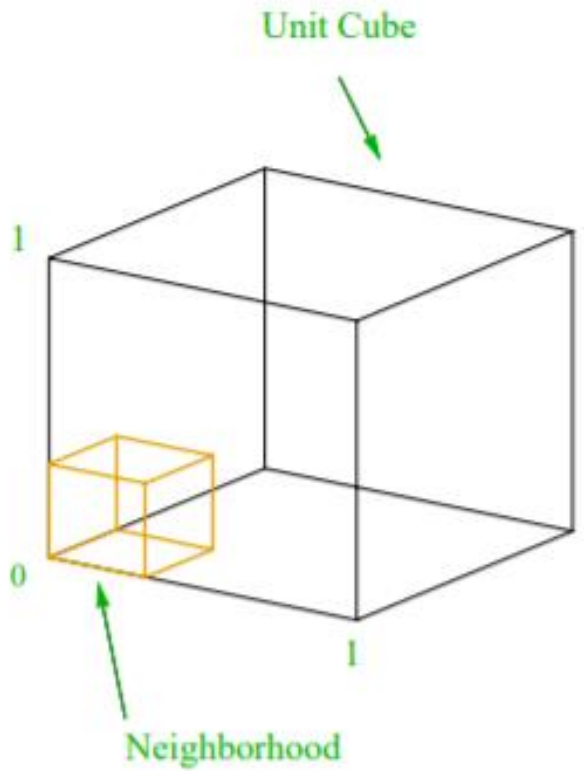
Bias-Variance Trade-off



Curse of Dimensionality



Curse of Dimensionality



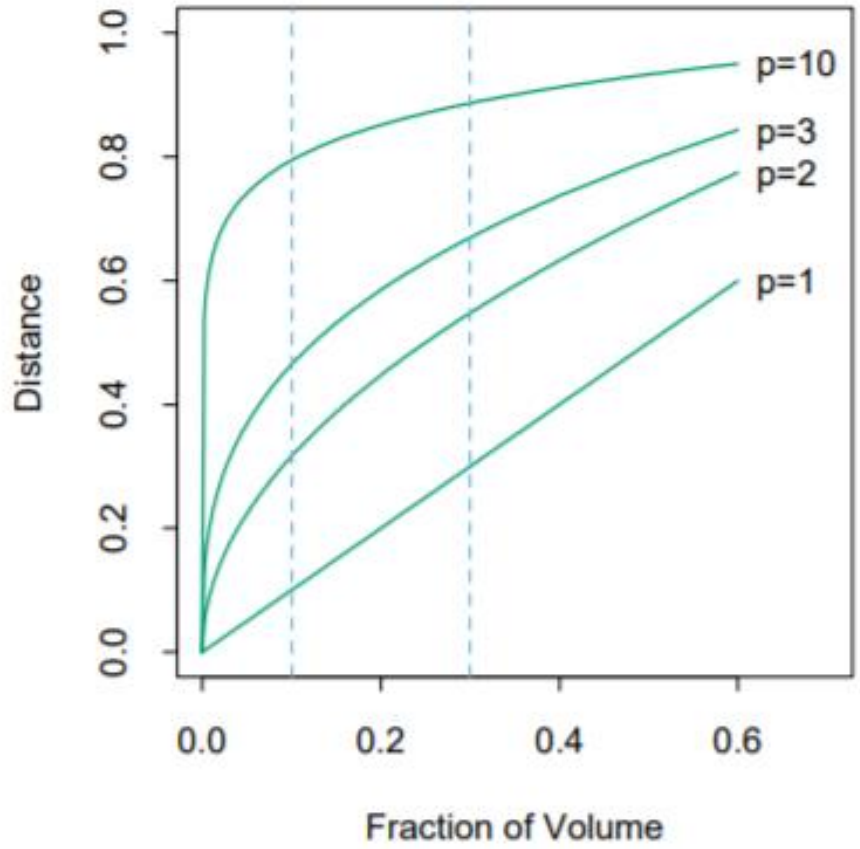
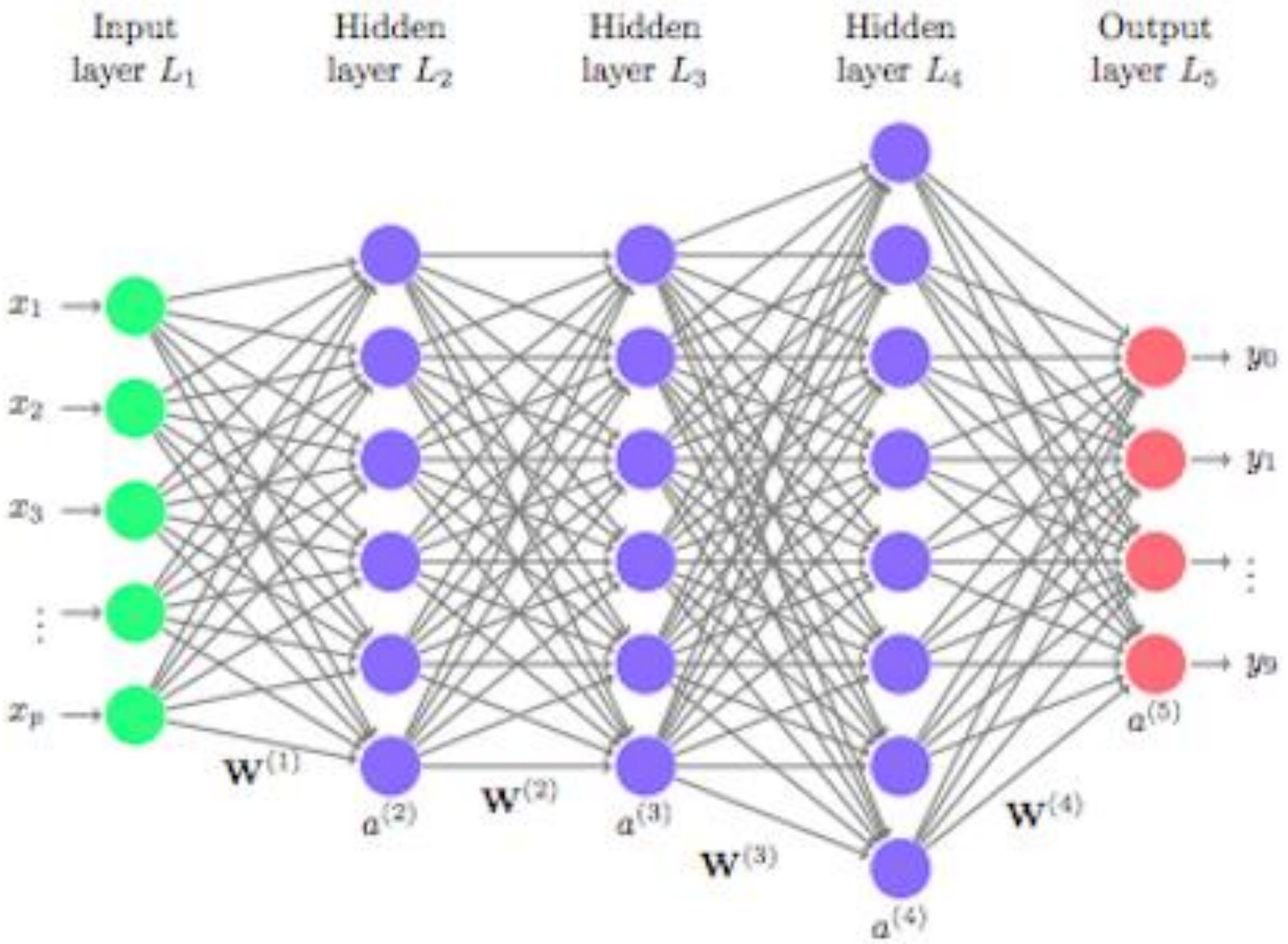
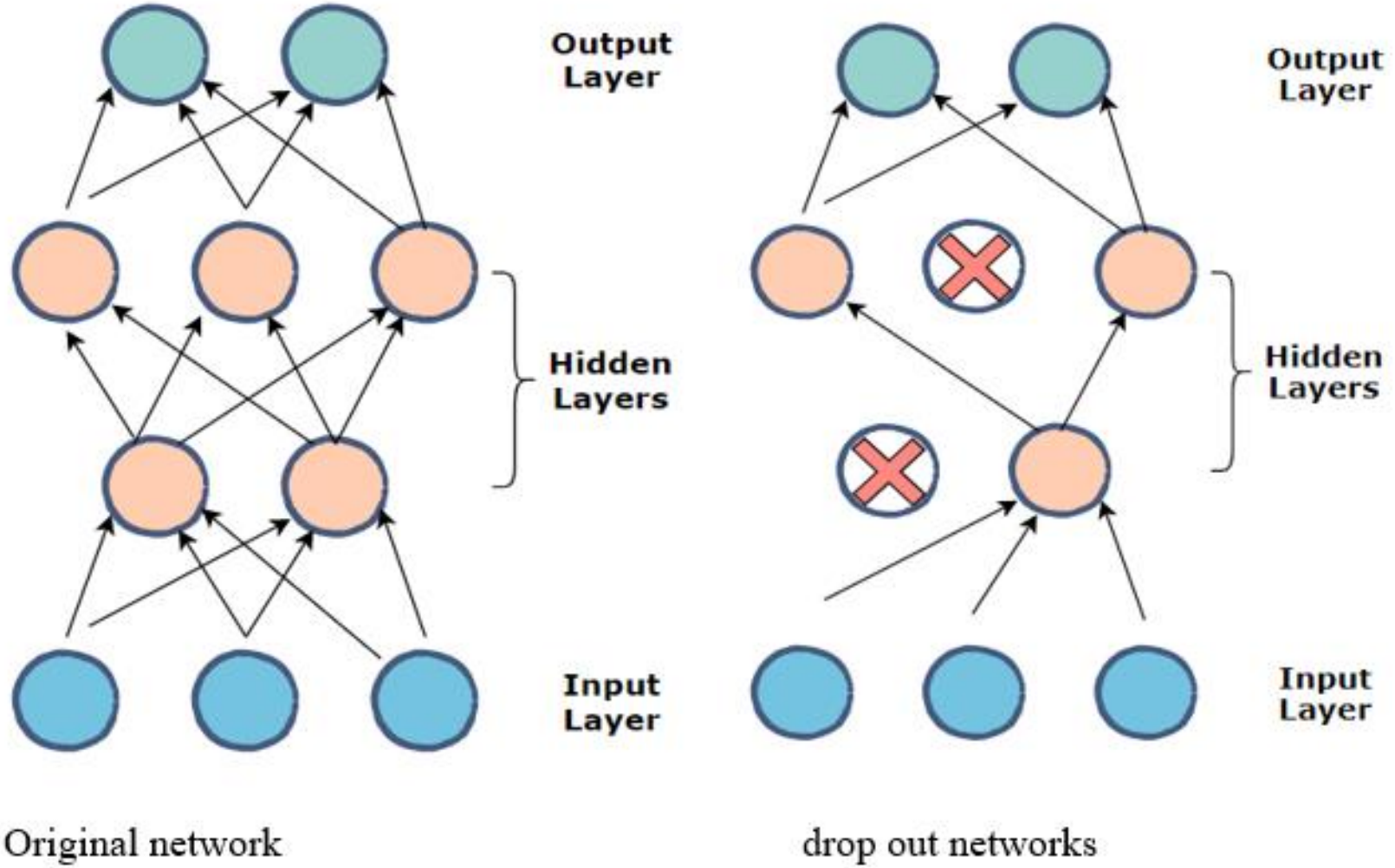


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.*

Dimensionality in DL



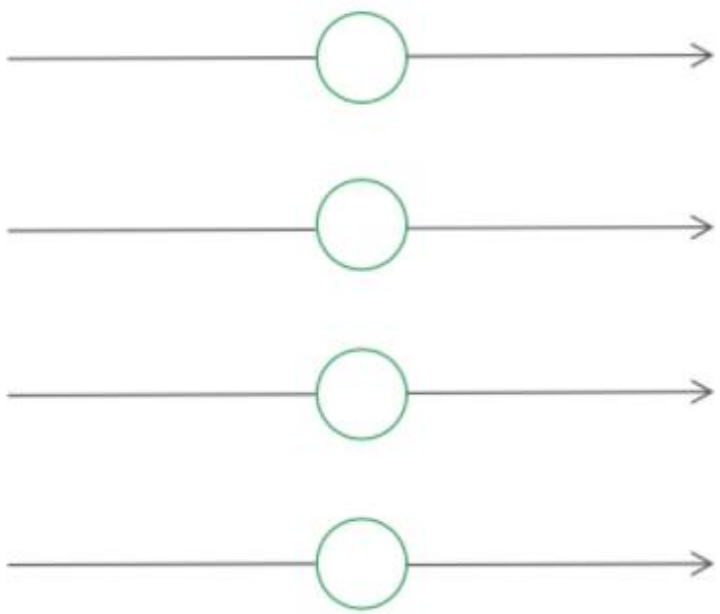
2. What is Dropout?

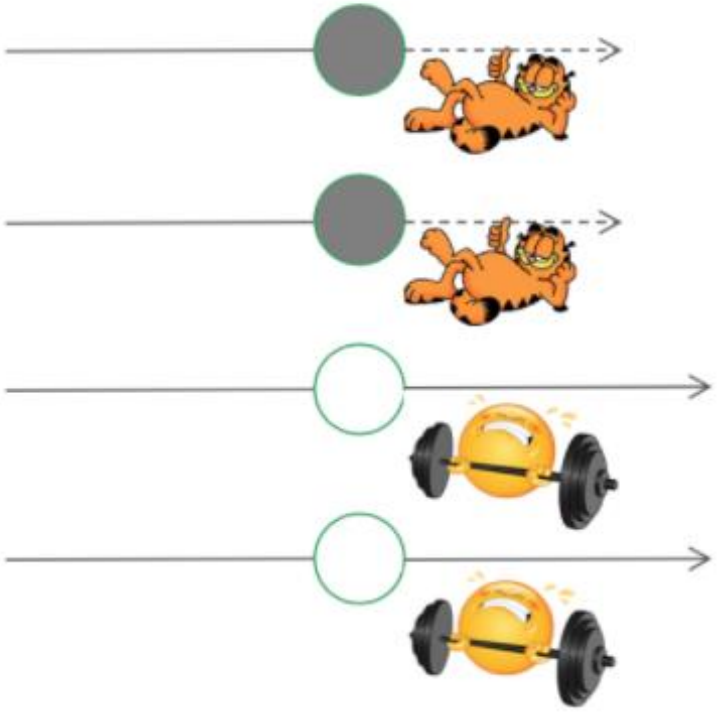


Original network

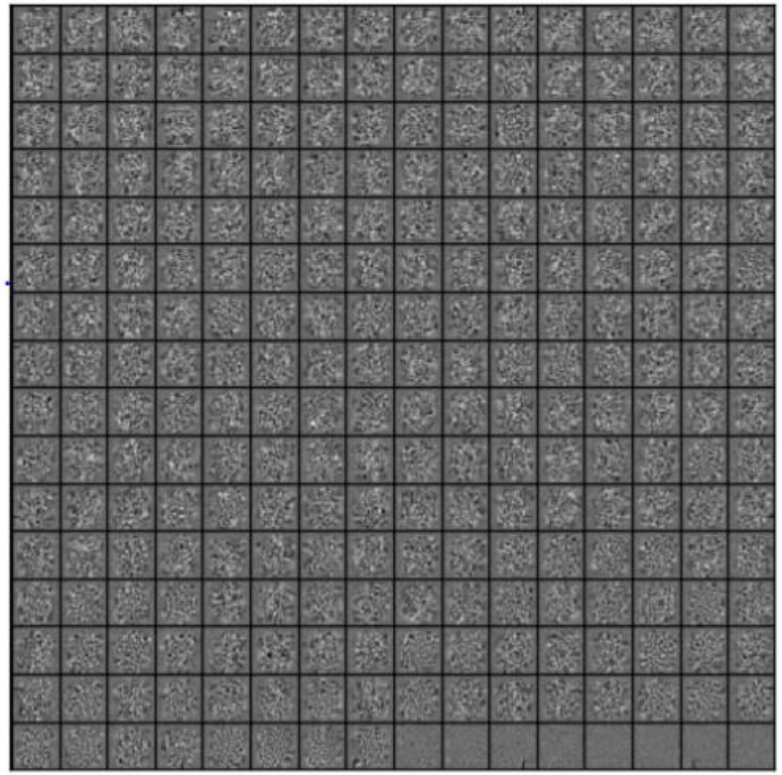
drop out networks

Co-adaptation

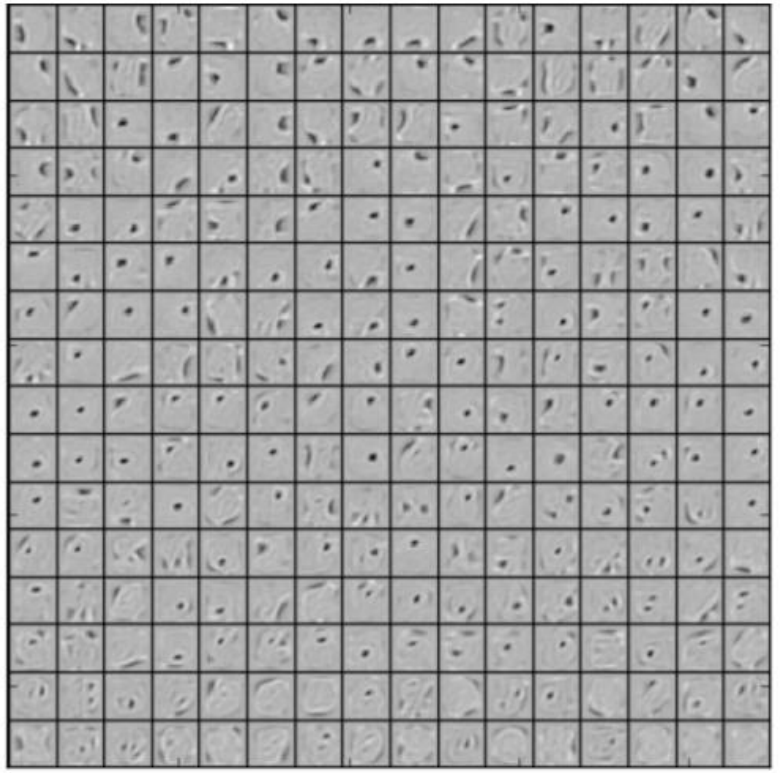




Co-adaptation



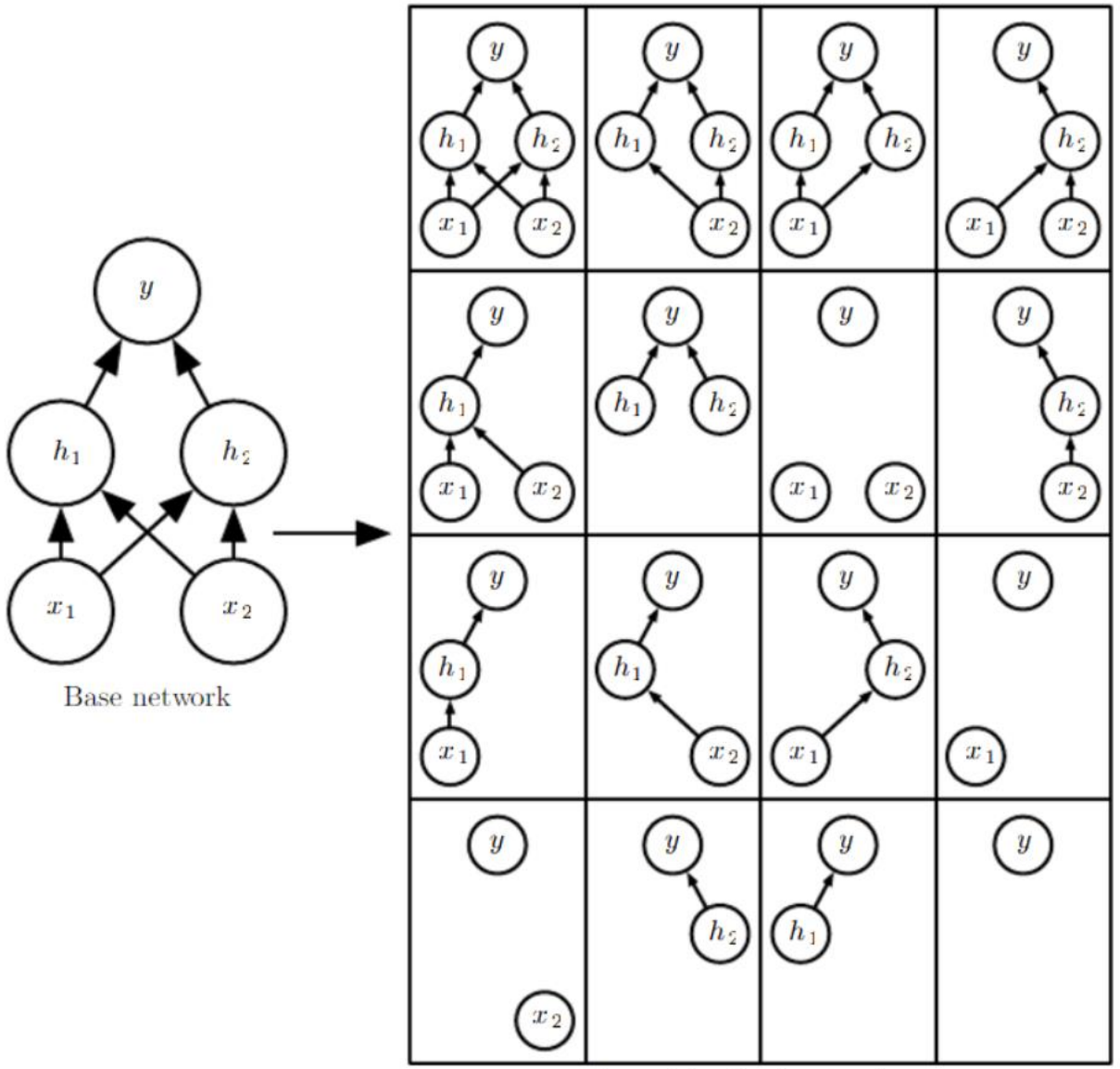
(a) Without dropout



(b) Dropout with $p = 0.5$.

Co-adaptation

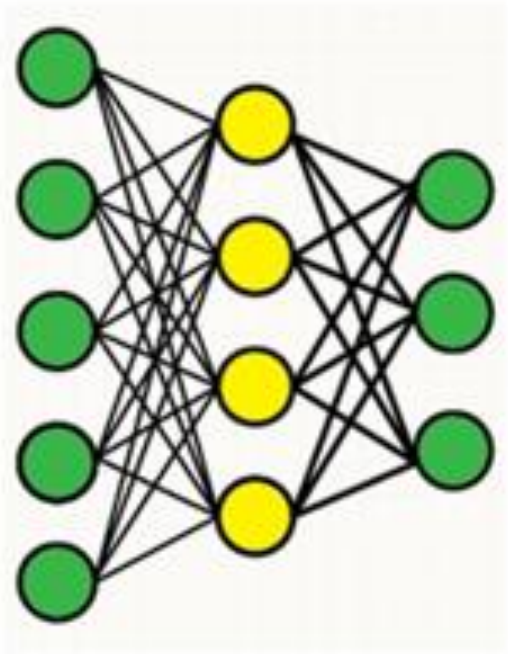
Bagging ~ Dropout



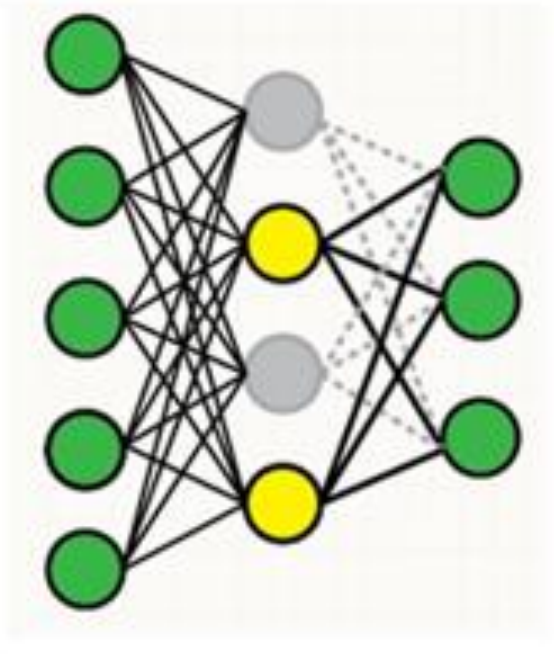
Base network

Ensemble of subnetworks

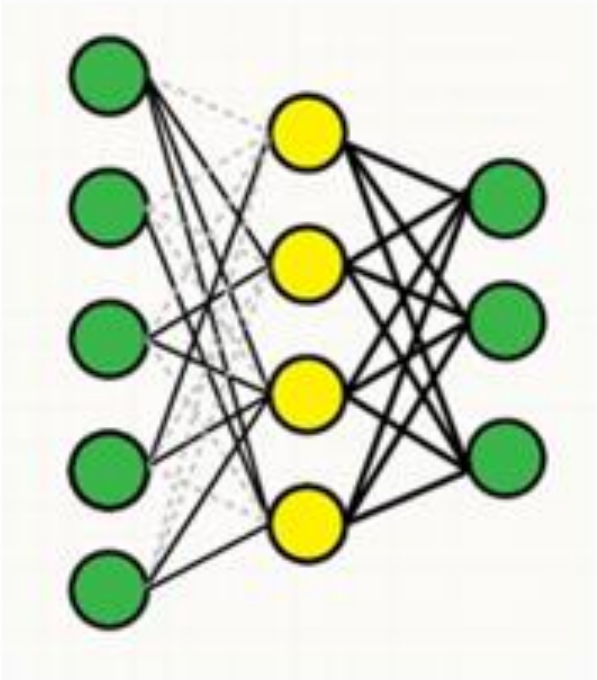
Generalization : Drop-connect



Original

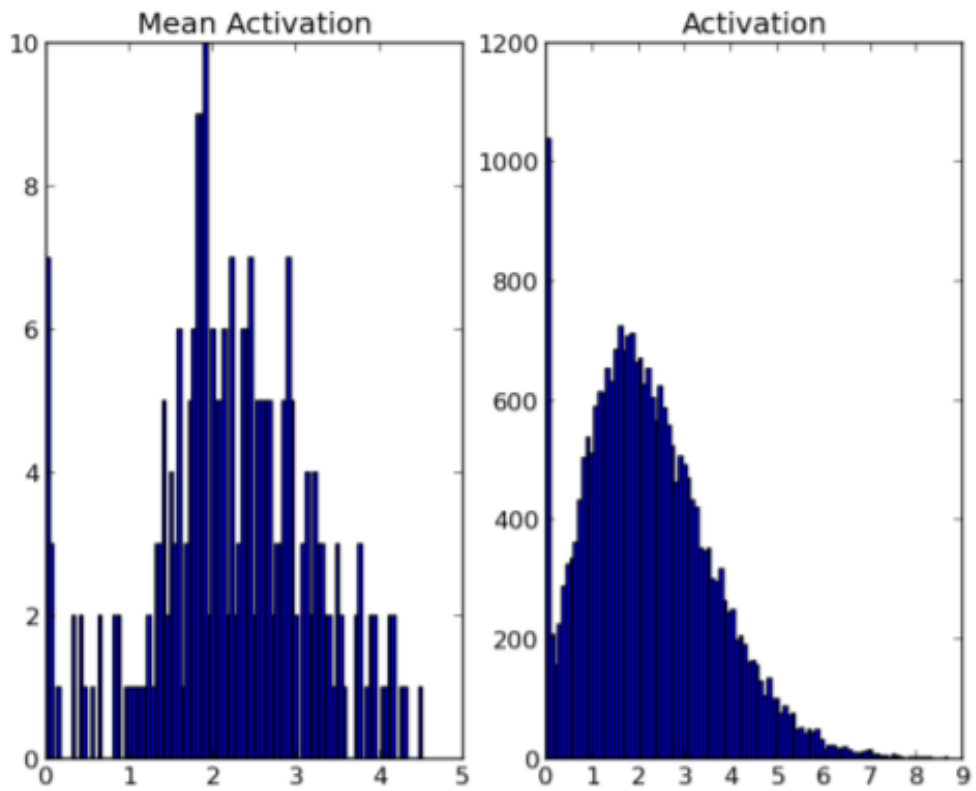


Dropout

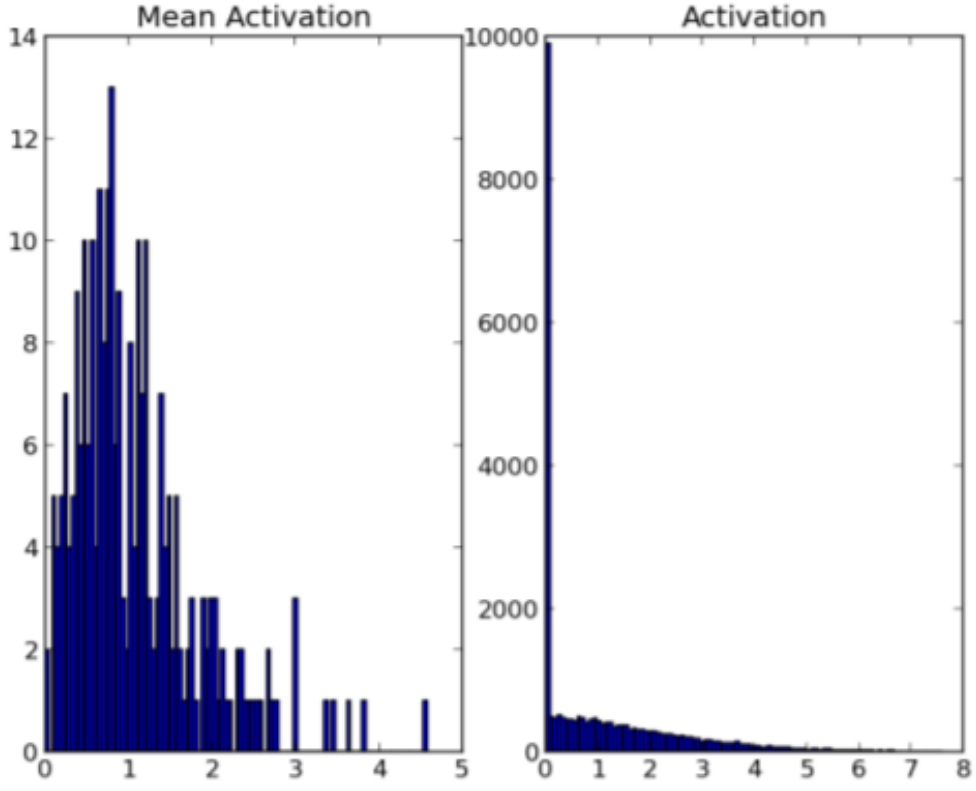


DropConnect

Effect of Drop-connect

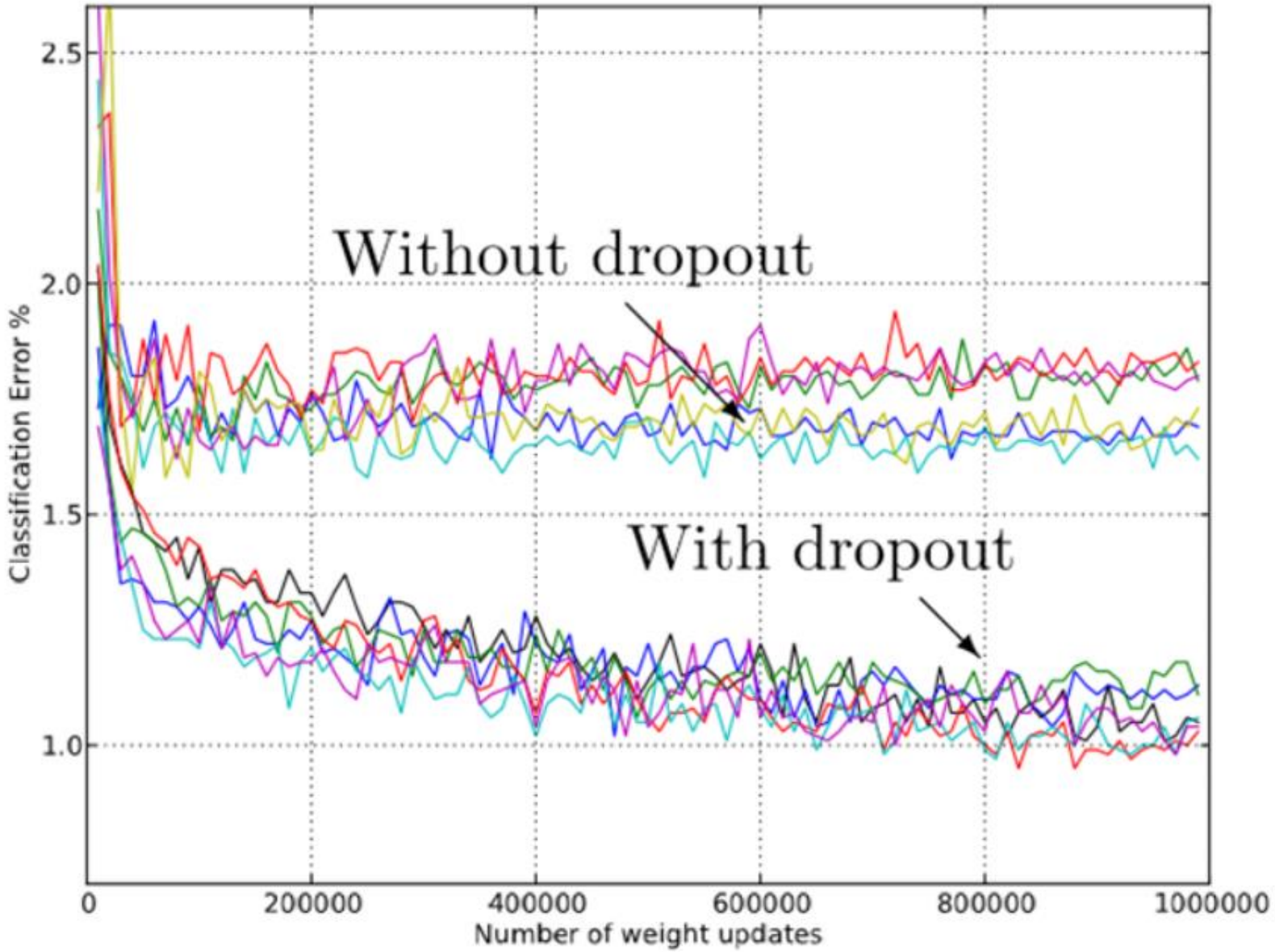


(a) Without dropout

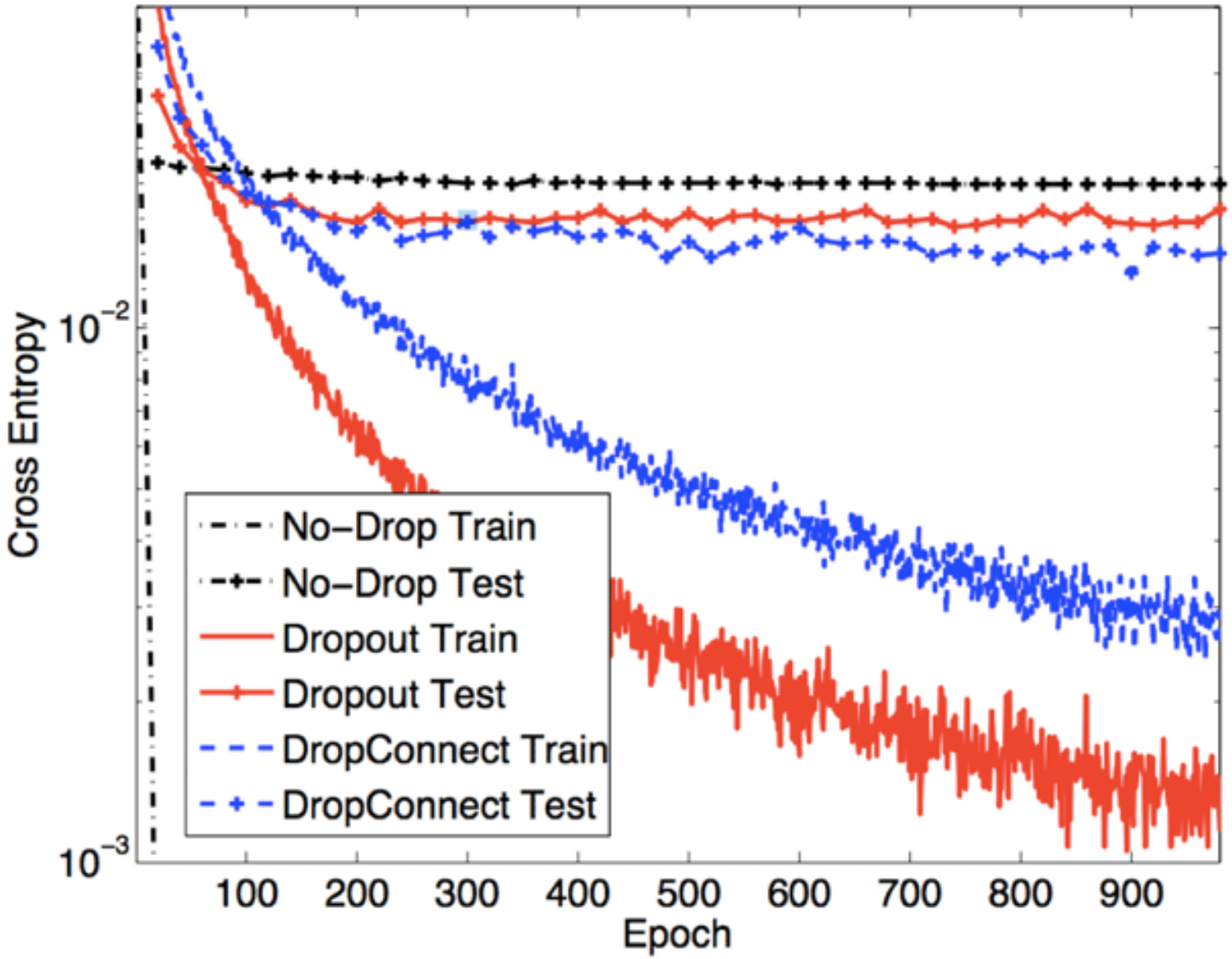


(b) Dropout with $p = 0.5$.

Effect of Drop-connect

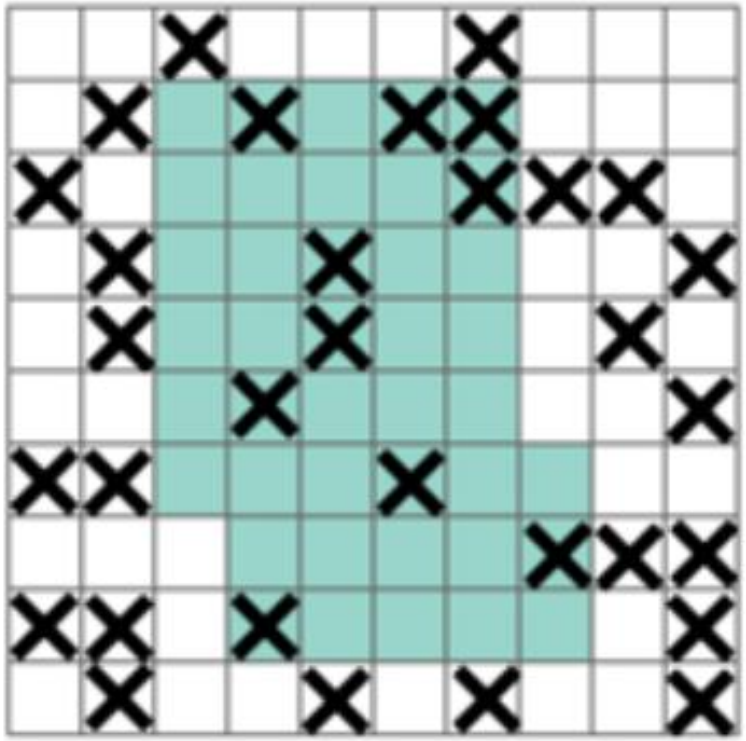


Effect of Drop-connect

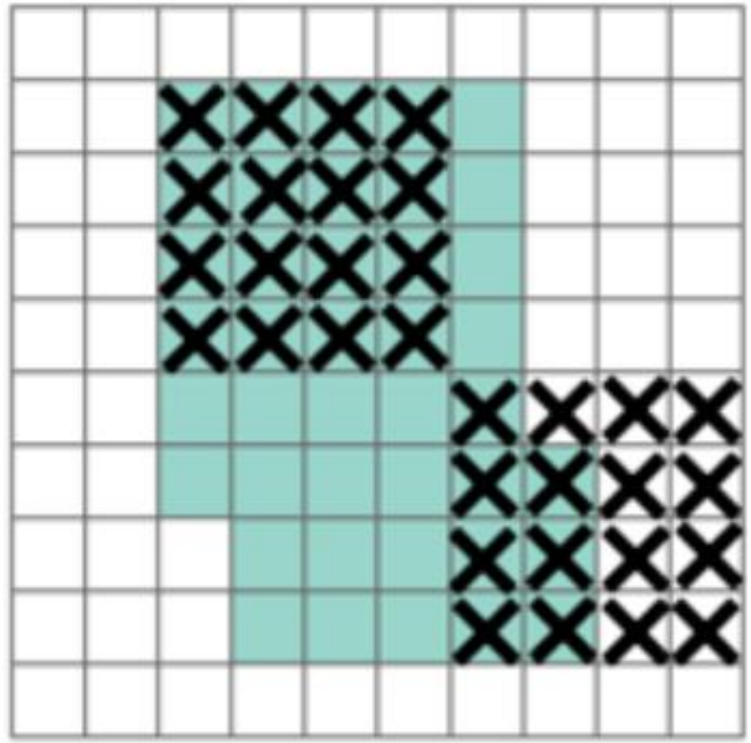


Dropout in CNN

(a)



(b)



(c)

Dropout 사용시 유의점

Under-fitting Model에 사용 X

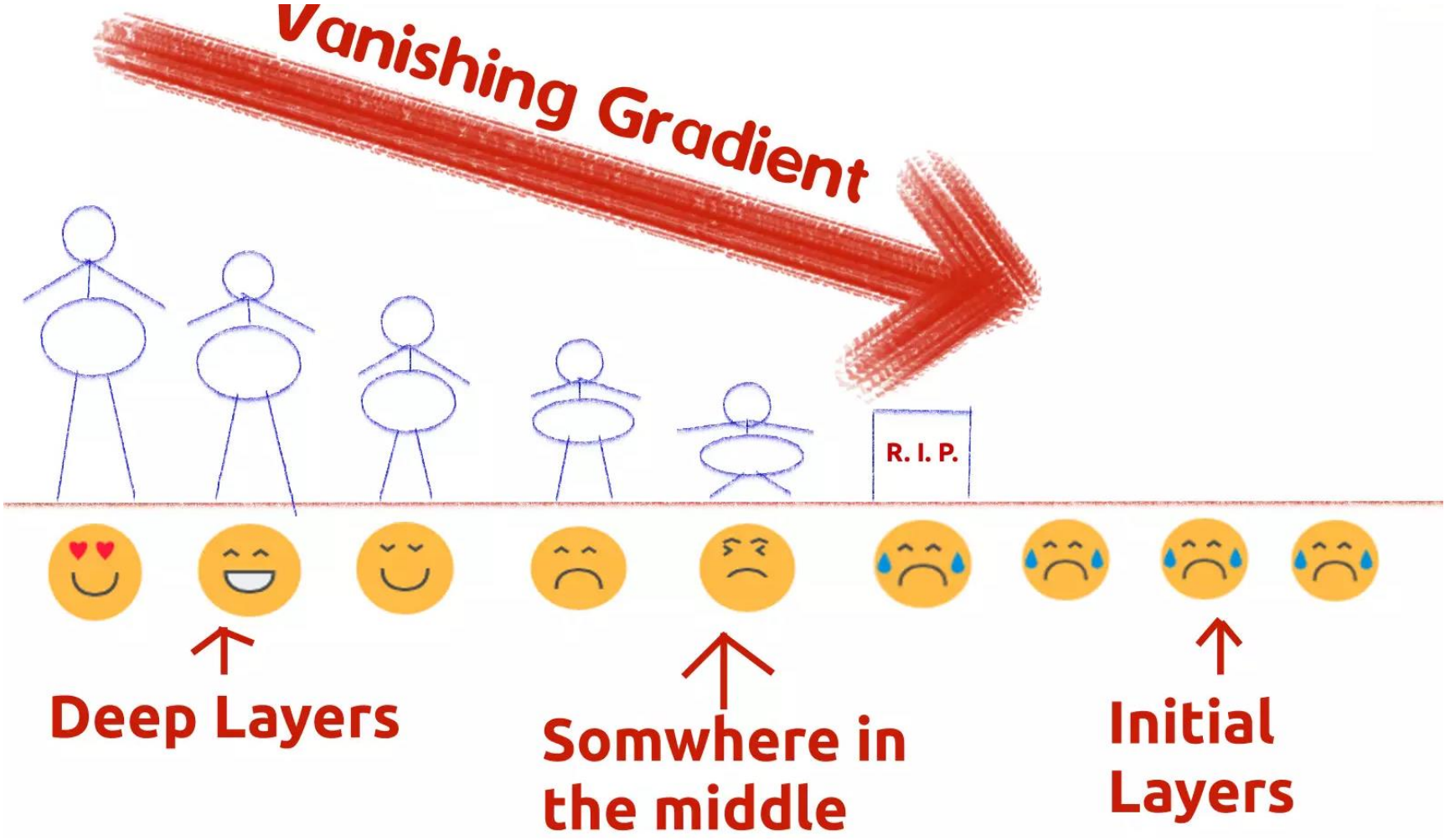
1. Neuron을 확률적으로 누락시키는 기법

2. 만약, under-fitting model에 사용한다면, 성능 악화

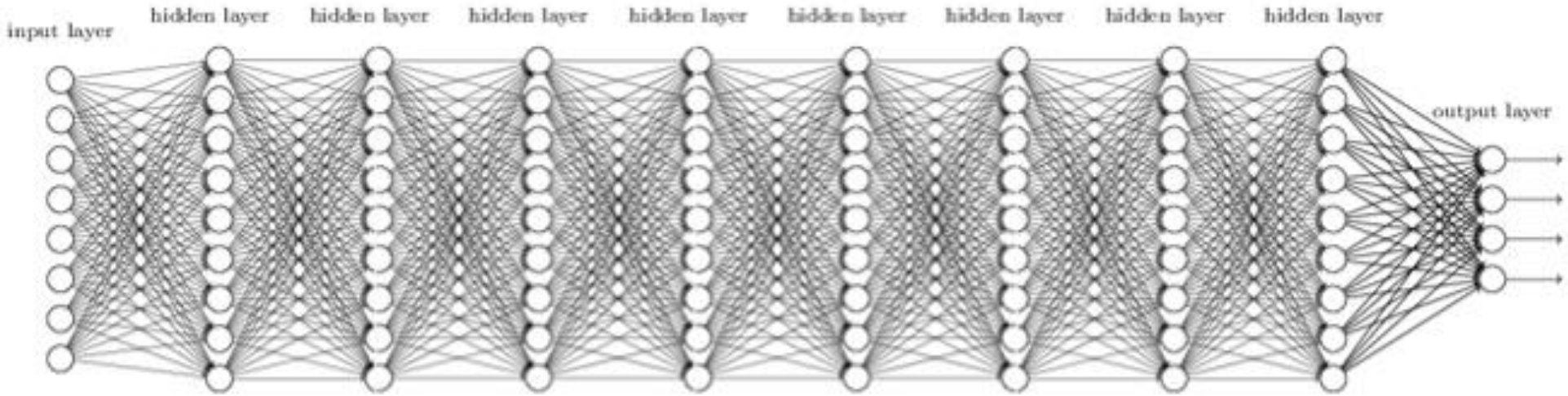
3. CNN Model에서 사용의 제한점

4. 수렴 속도 저하의 문제 & Training error는 증가할 수 있음

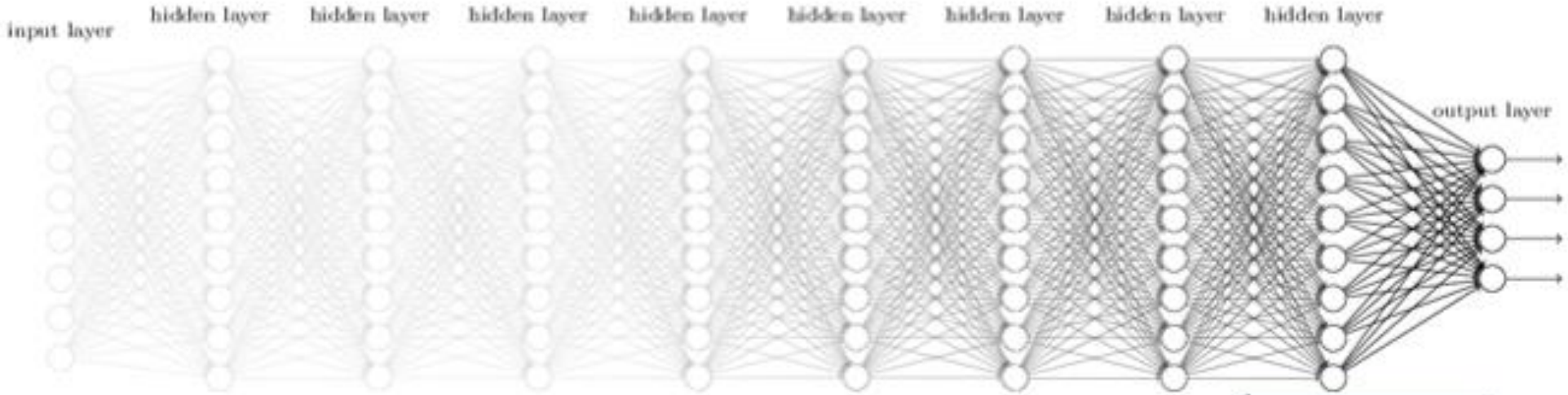
3. Vanishing / Exploding Gradient



3. Vanishing / Exploding Gradient

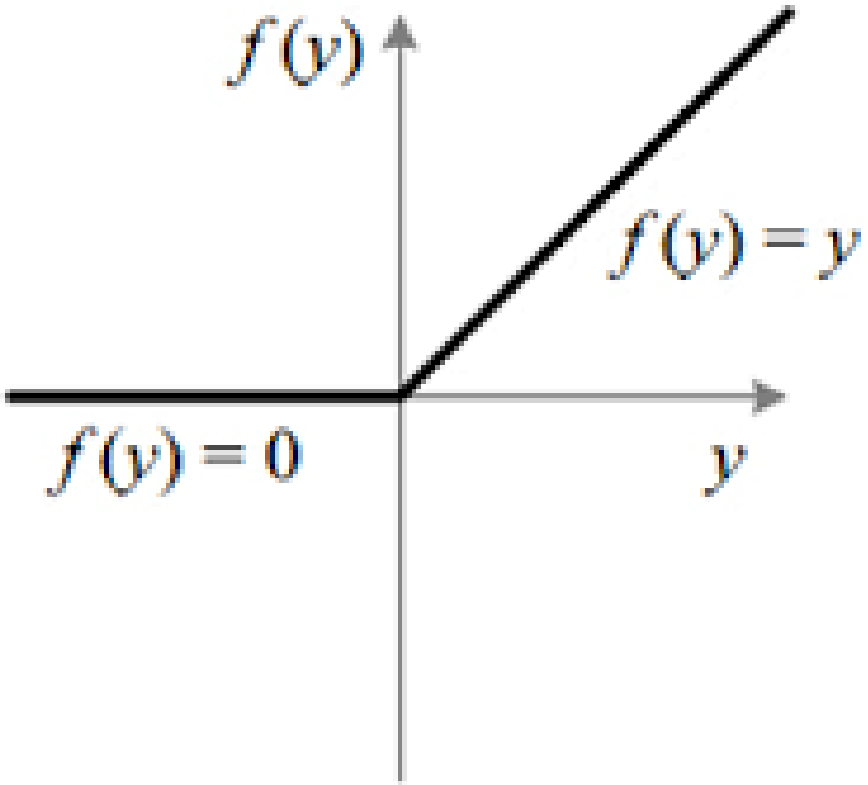


Deep Neural Network



Vanishing Gradient

3. Vanishing / Exploding Gradient



Weight initialization method

- The right weight initialization method can speed up time-to-convergence considerably. The choice of your initialization method depends on your activation function. Some things to try:
- When using ReLU or leaky RELU, use **He initialization**
- When using SELU or ELU, use **LeCun initialization**
- When using softmax, logistic, or tanh, use **Glorot initialization**
- Most initialization methods come in uniform and normal distribution flavors.

Activation Function

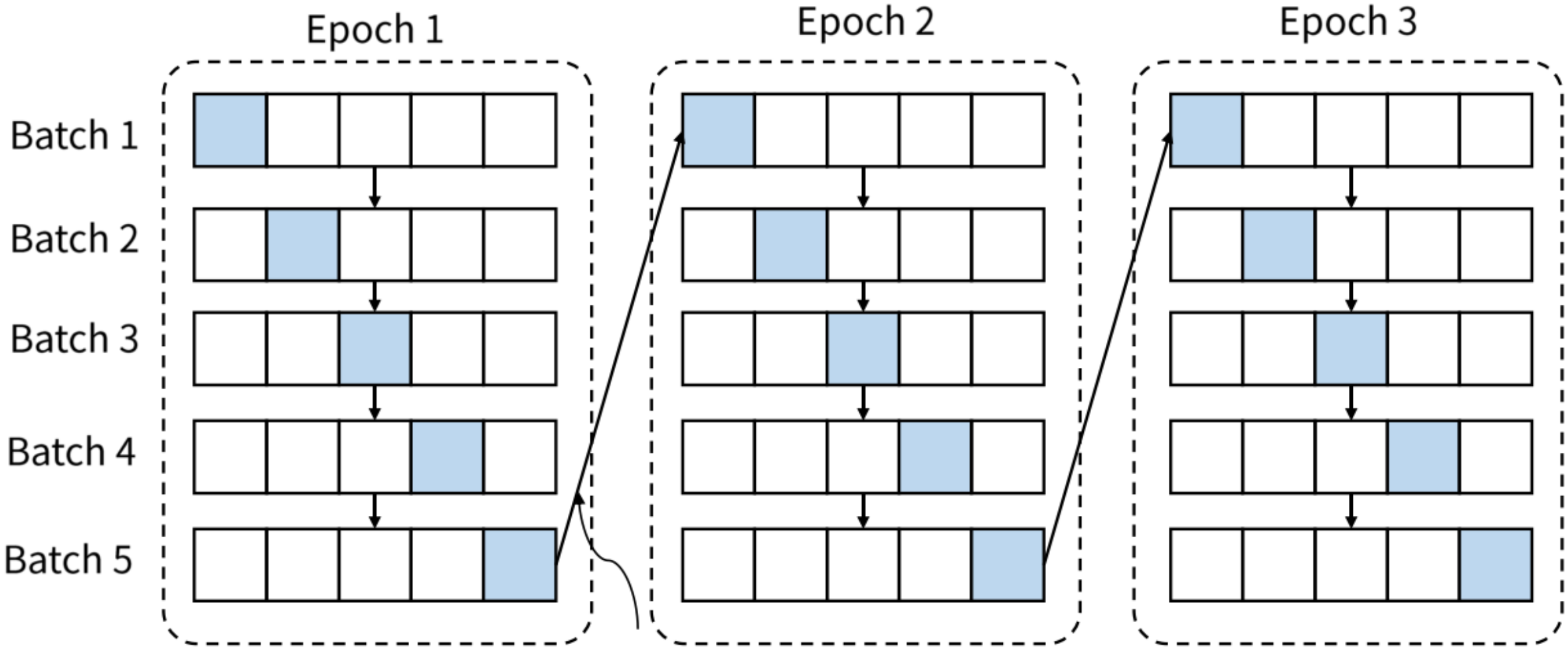
1. ReLu의 문제점에 대해

2.ReLu를 개선 or 응용한 activation function

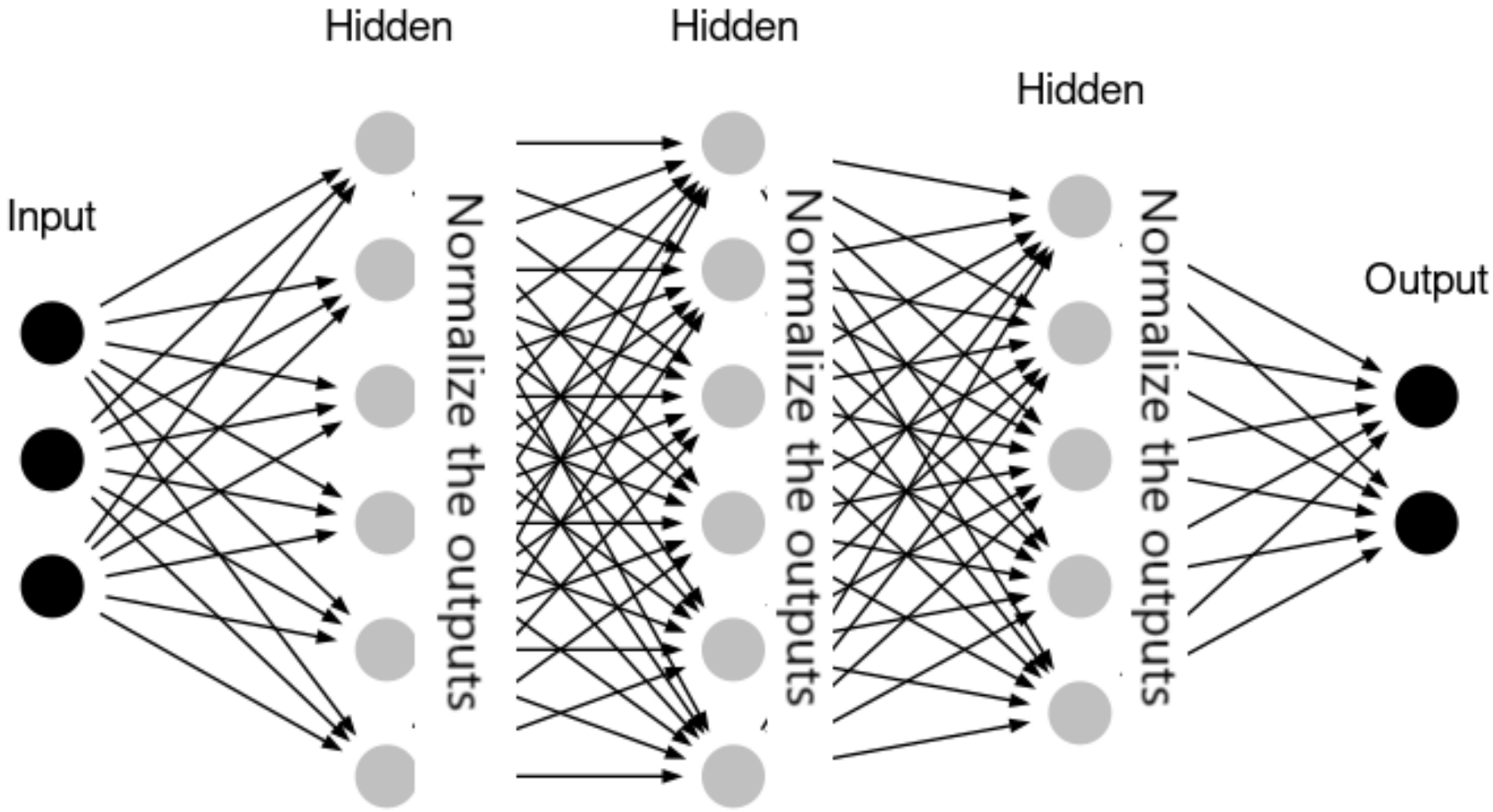
LIST

Leaky ReLu / SELU / ELU /
Swish / GELU

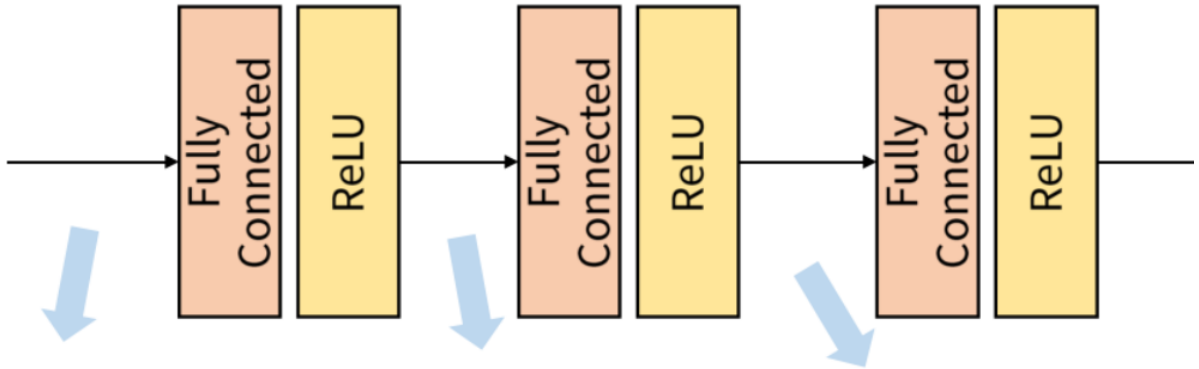
Review of Batch & Epoch

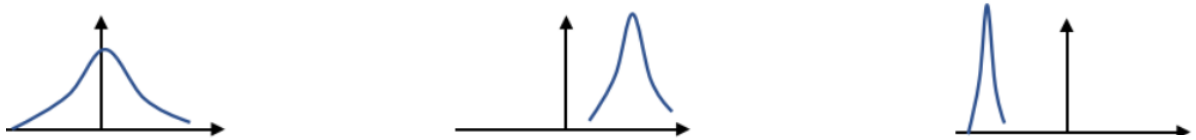


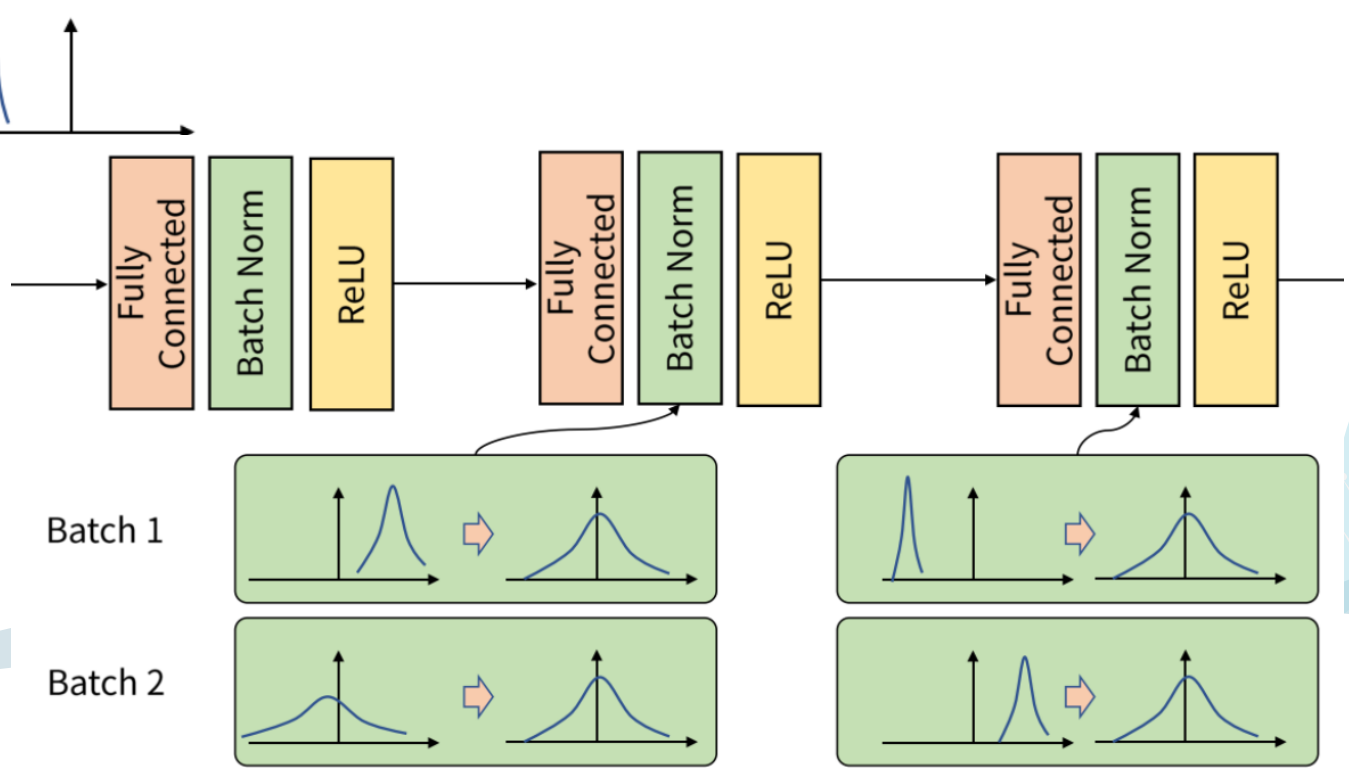
3. Batch Normalization



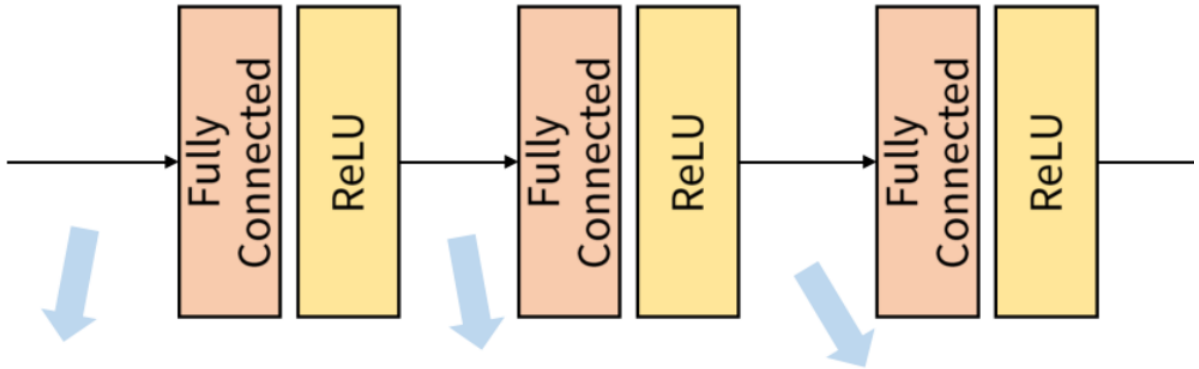
Internal Covariance Shift

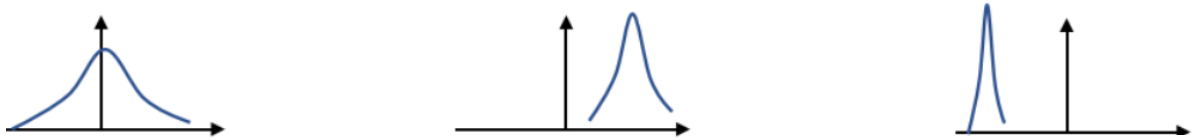


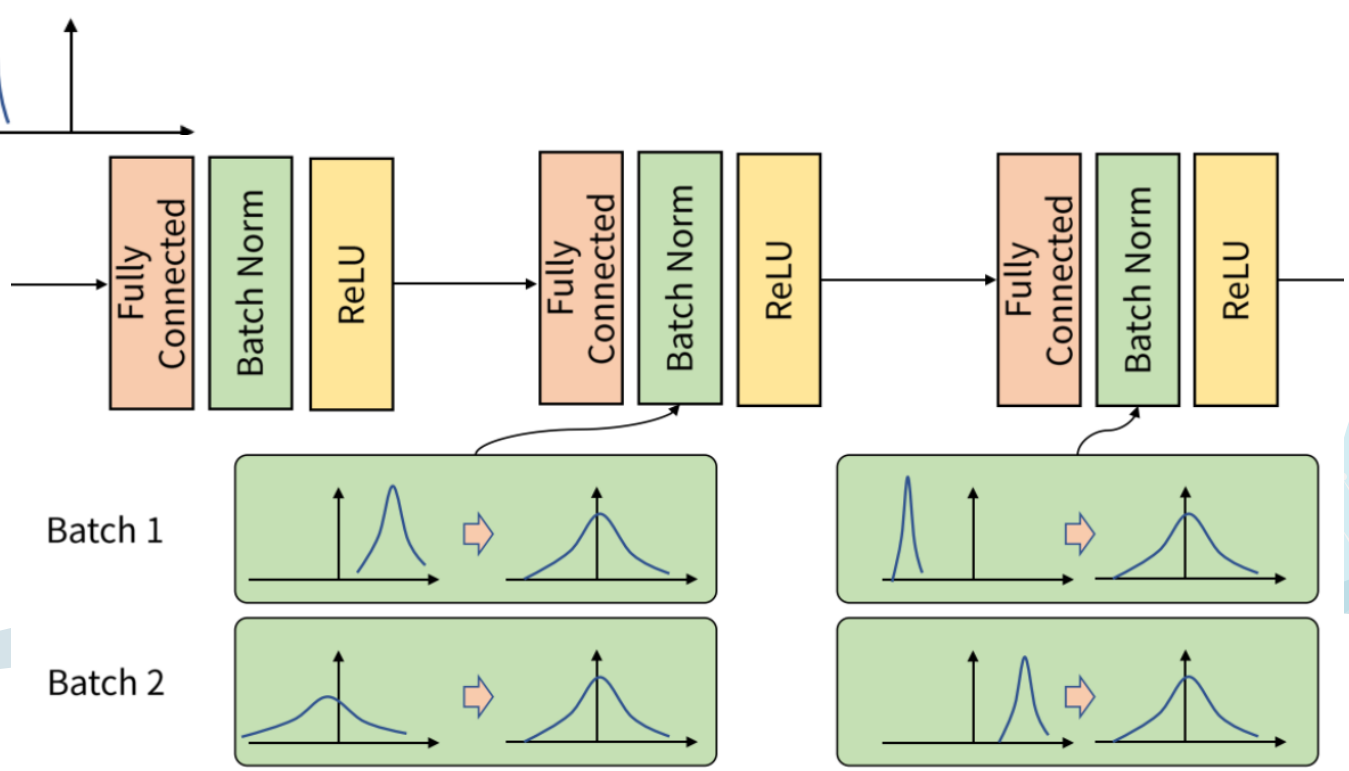




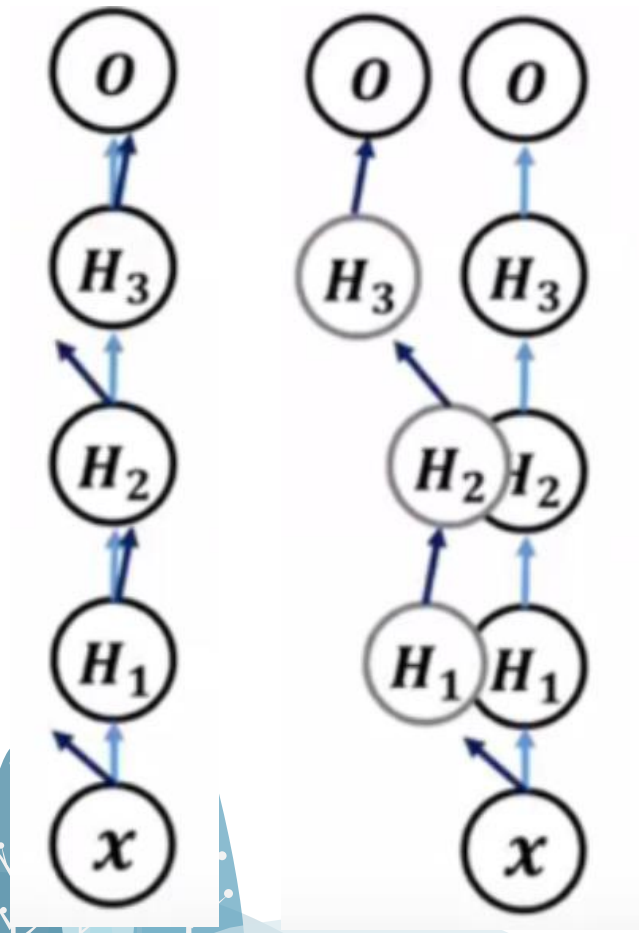
Internal Covariance Shift

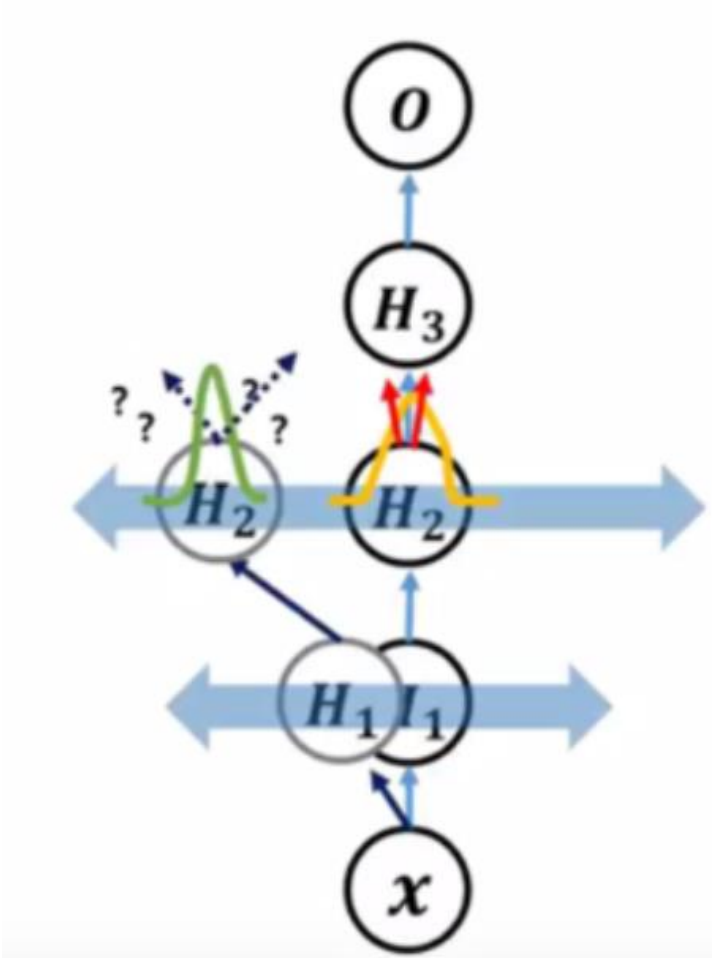


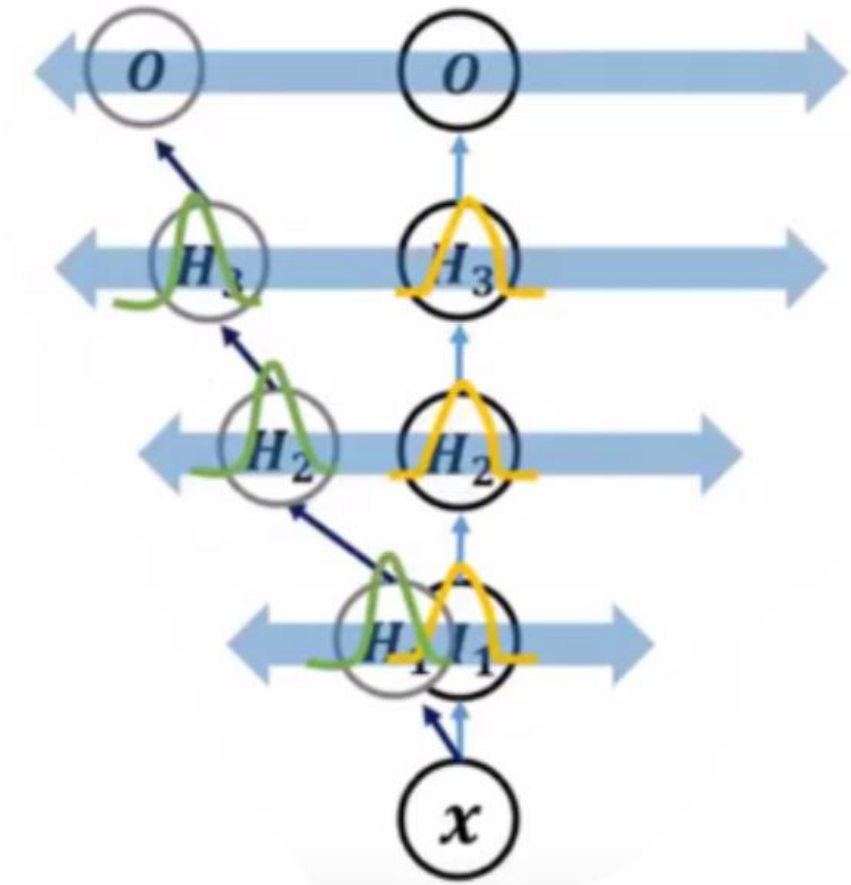




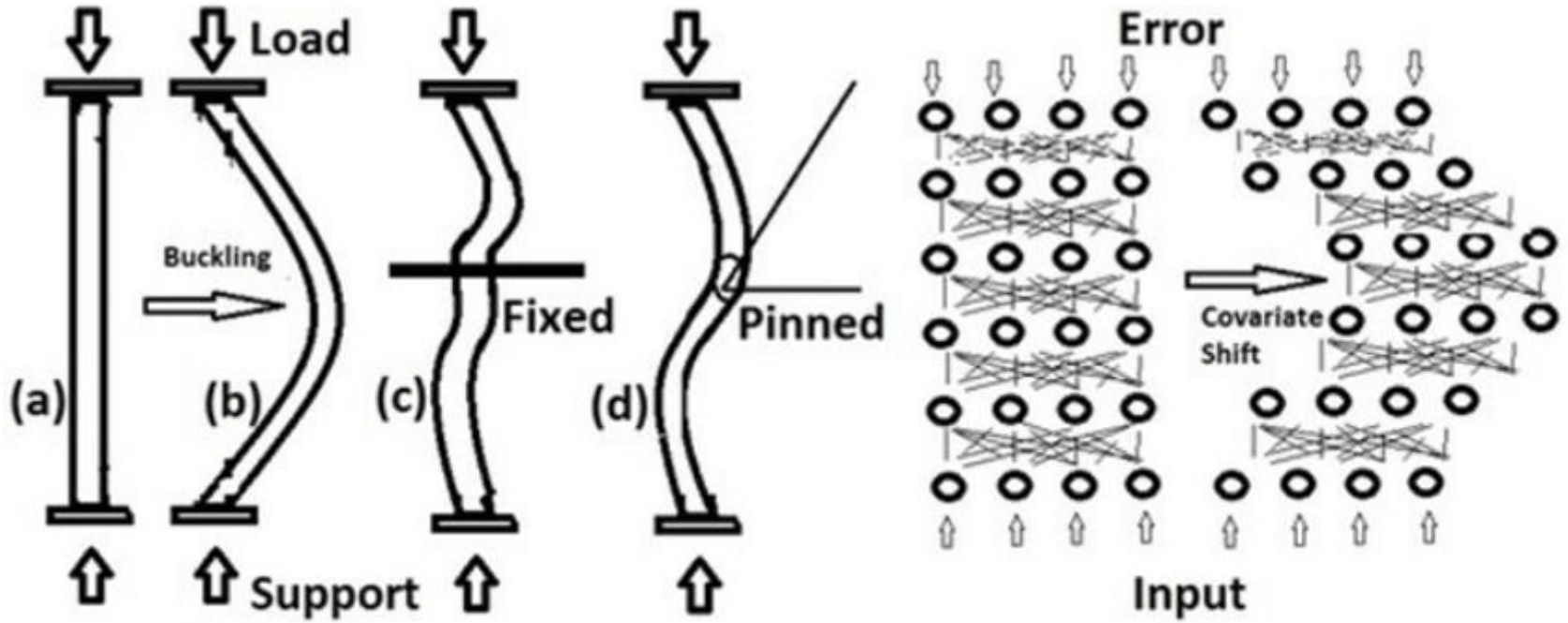
Internal Covariance Shift







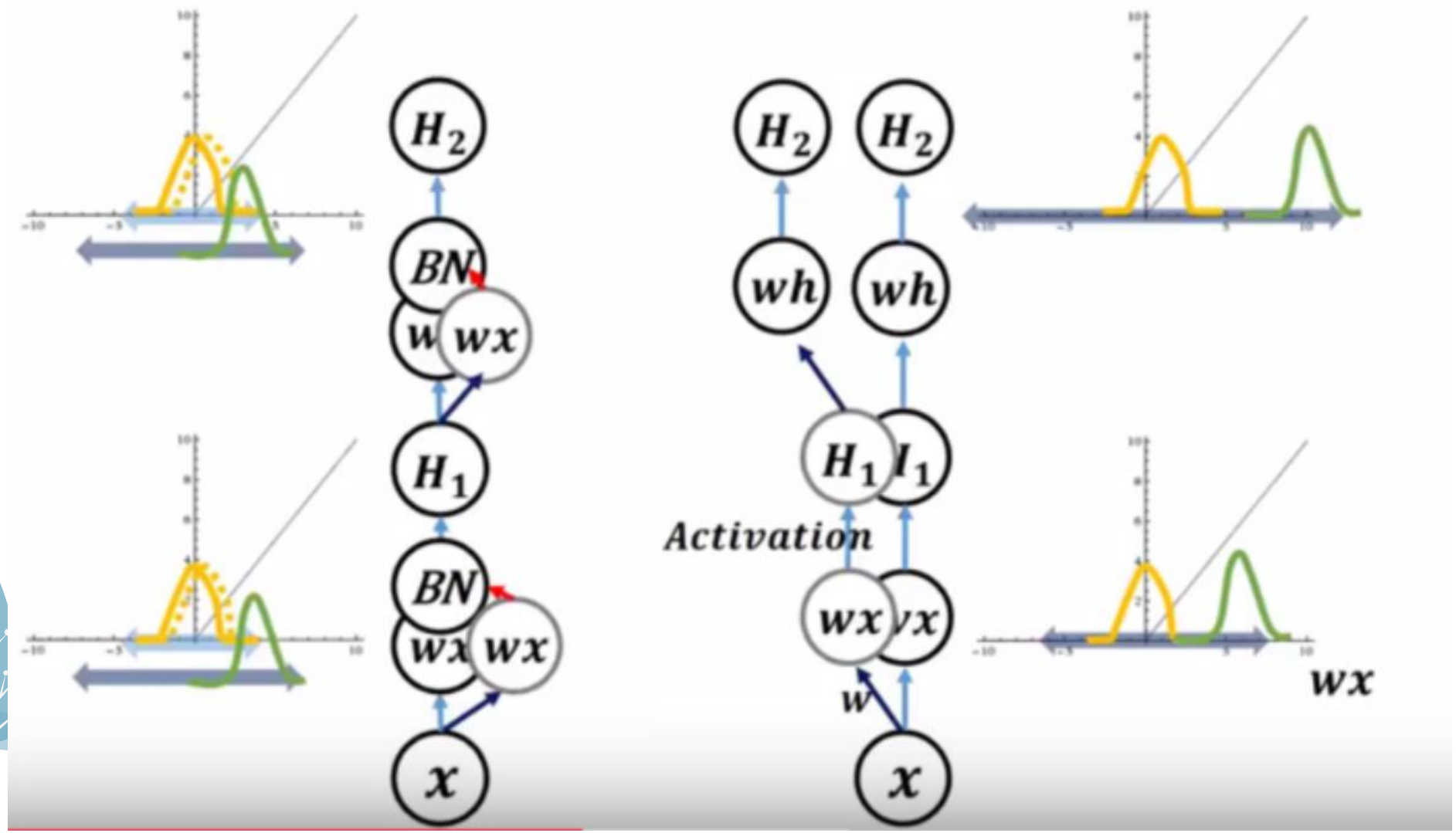
Internal Covariance Shift



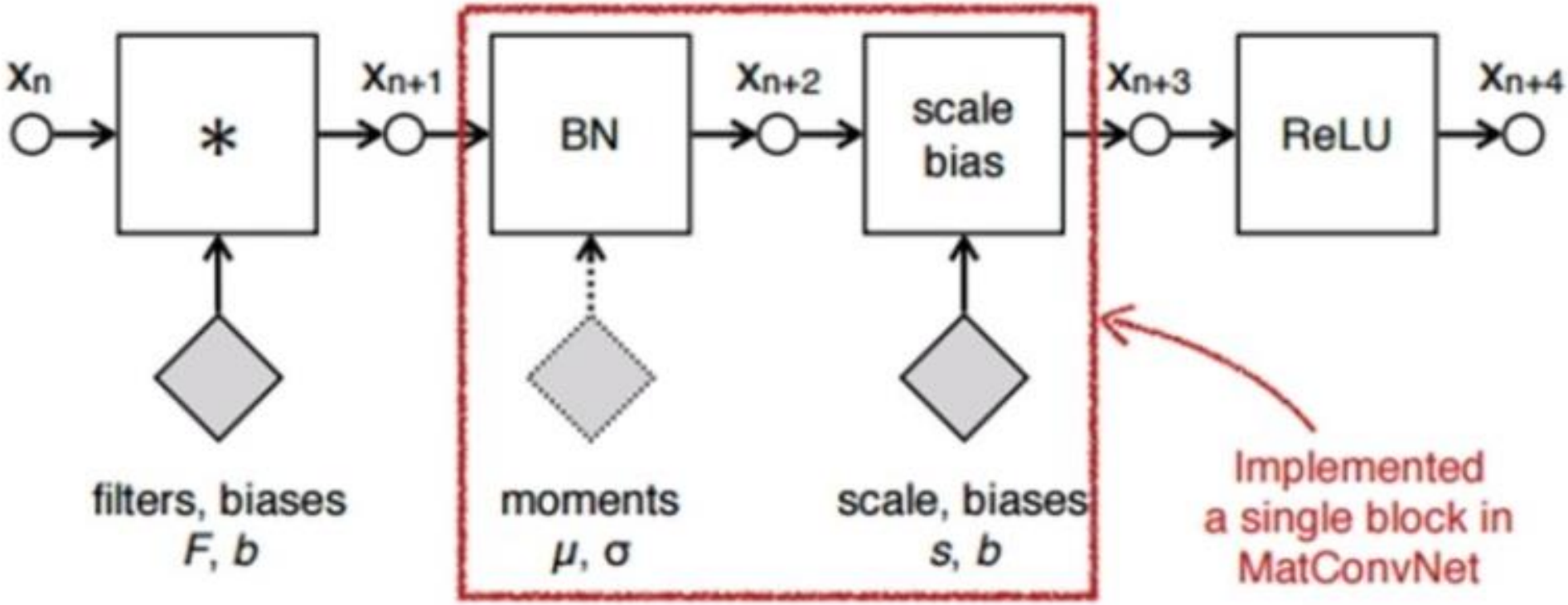
For both, Buckling or Co-Variate Shift a small perturbation leads to a large change in the later.

Debiprasad Ghosh, PhD, Uses AI in Mechanics

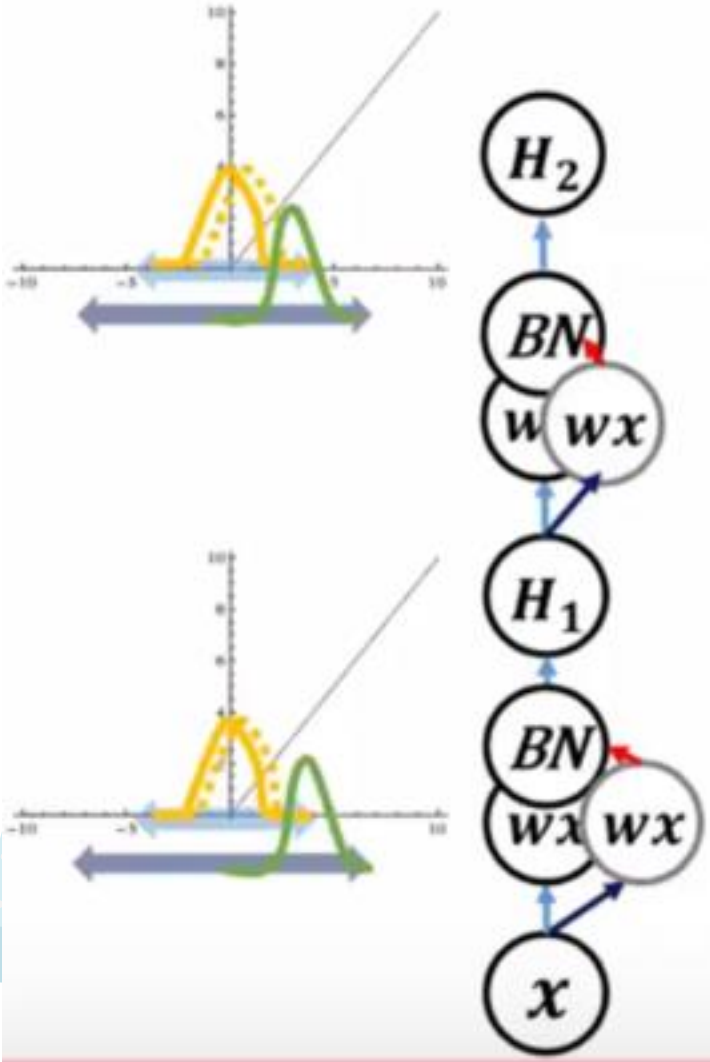
Internal Covariance Shift



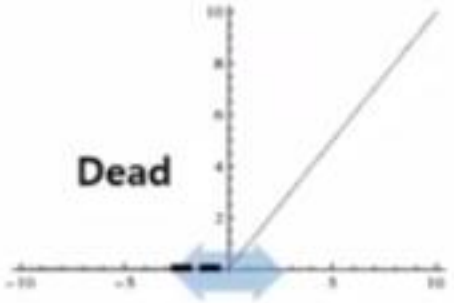
Internal Covariance Shift

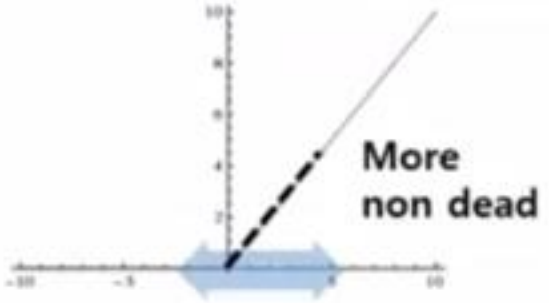


Scale & Shifting Parameters



0 mean and 1 variance is preferred
But it is not best

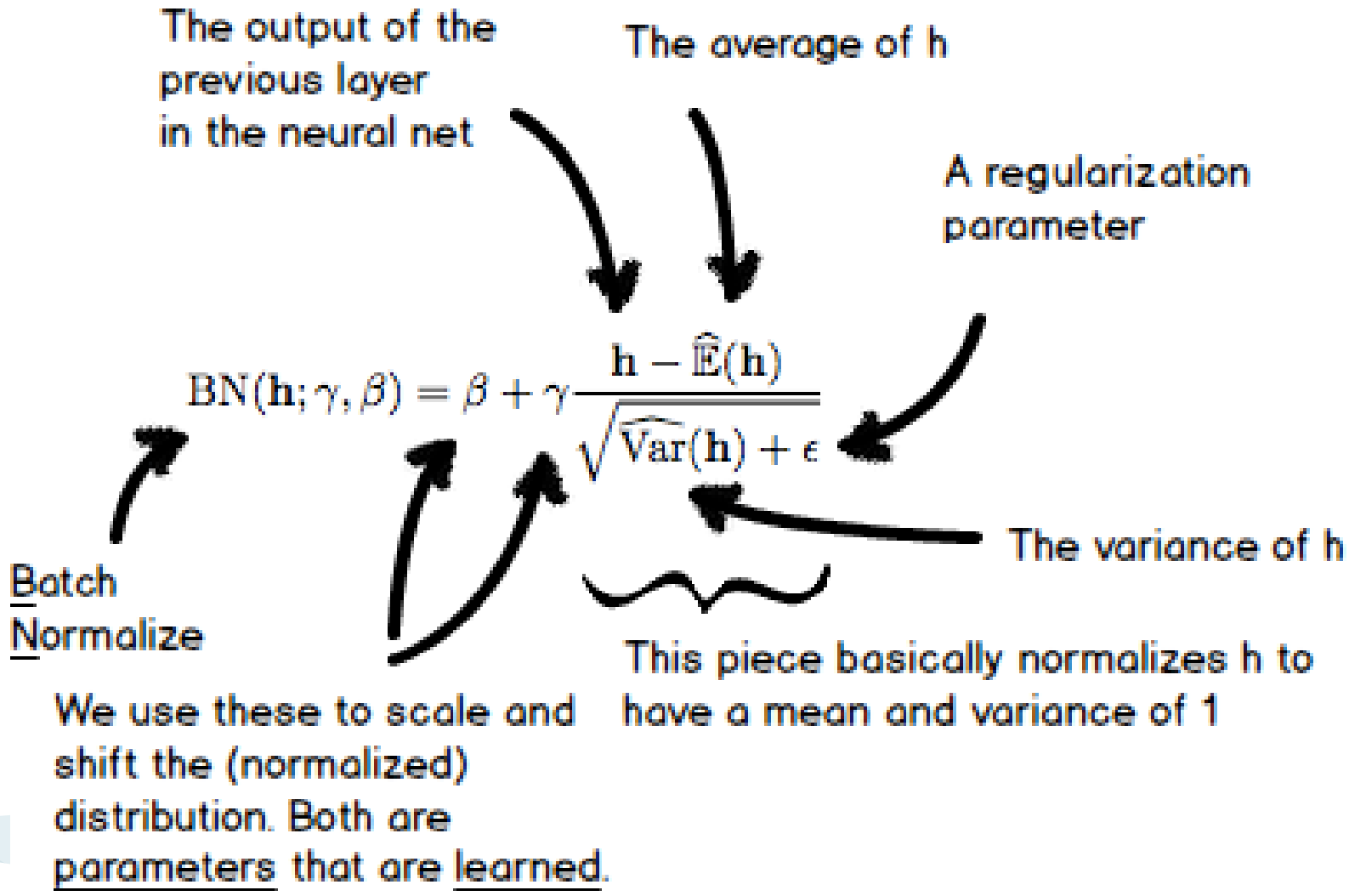




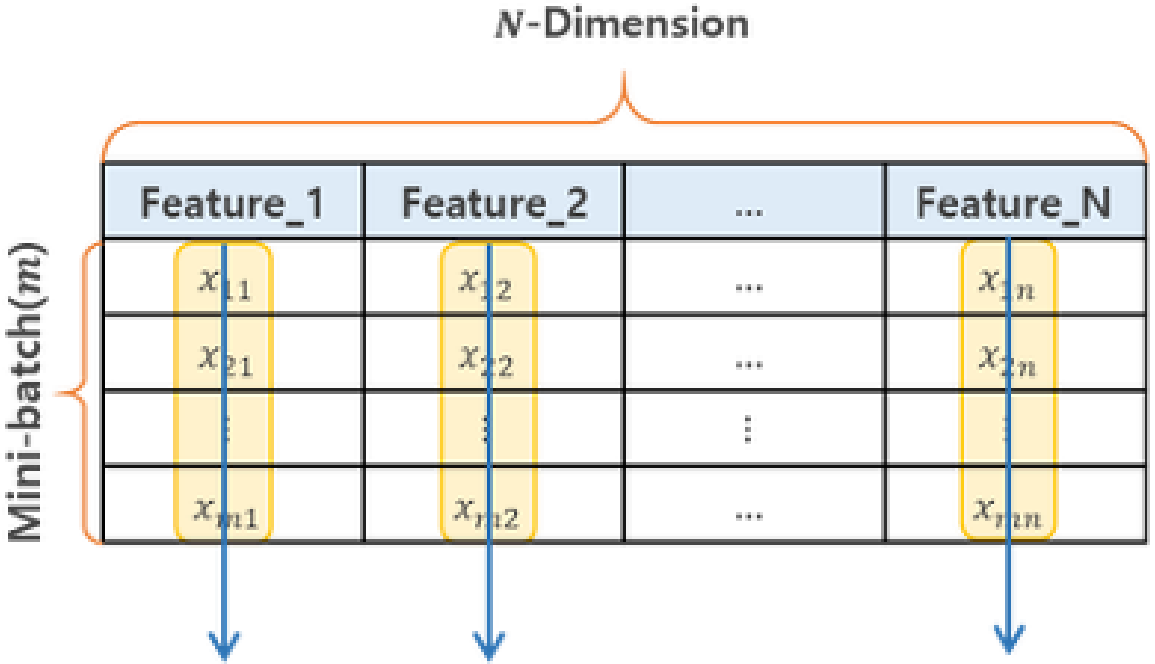
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

// scale and shift

Batch Normalization



Batch Normalization



각 Feature 별 평균(mean)과 분산(variance)을 구함

Normalize

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

Batch Normalization

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Effect of Batch Normalization

Fast Convergence

High Learning rate

빠른 학습 속도 구현 가능,
Optimum에 더 빠른 수렴

Not sensitive to Weight Initialization

Initial value에 덜 민감

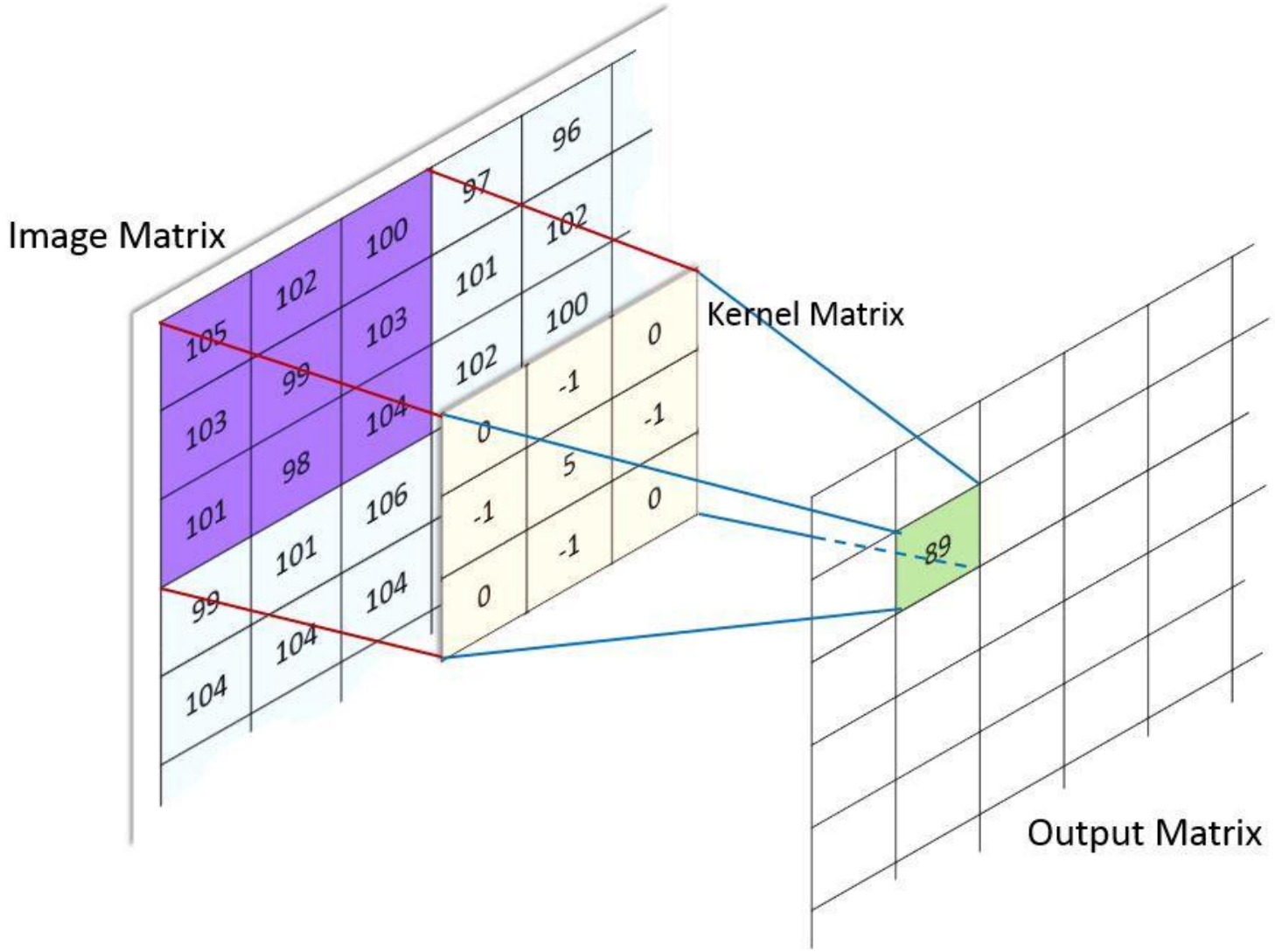
Weight initial Setting 문제를
덜 고민하는 것이 가능

Regularization

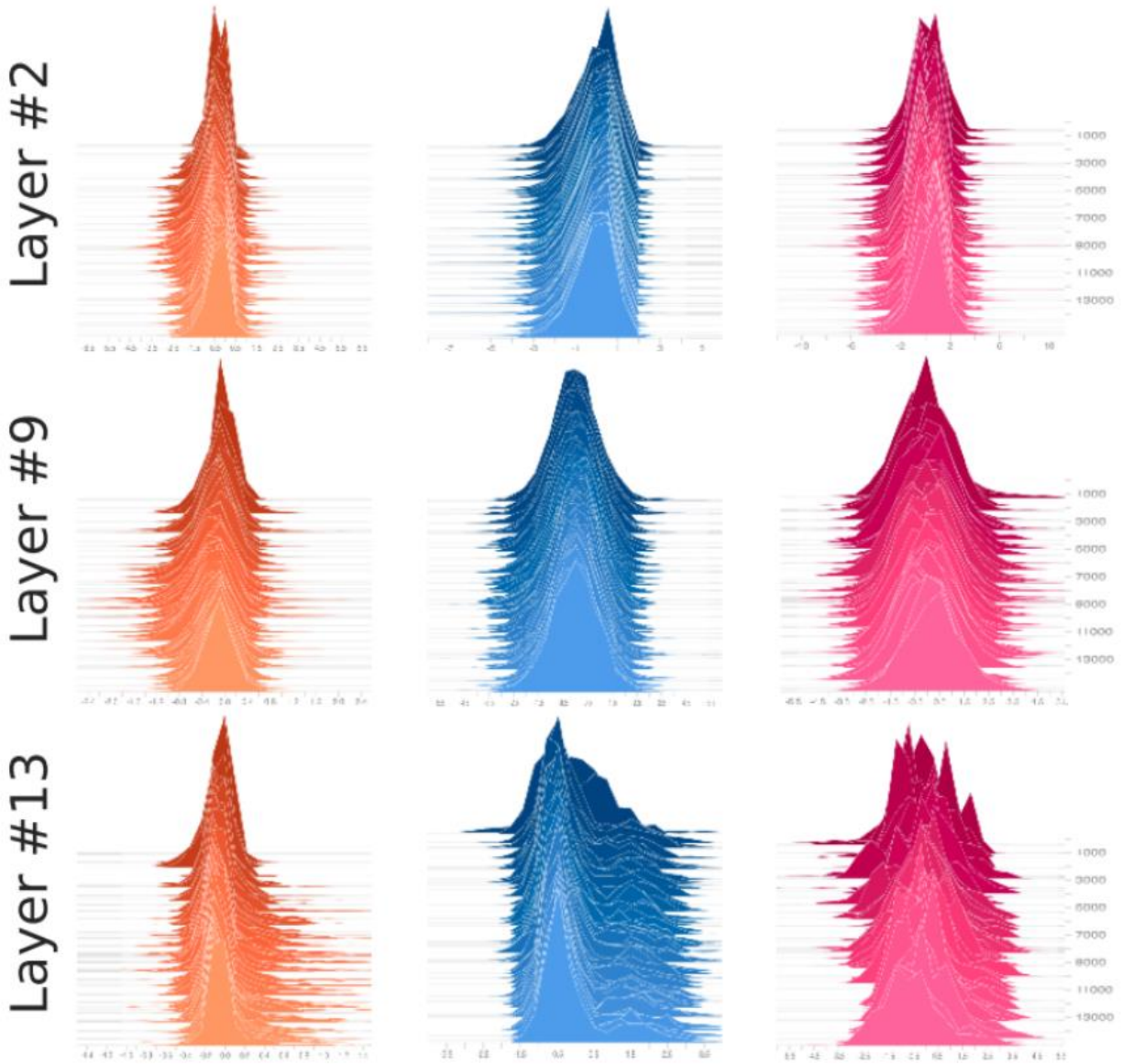
Dropout과 비슷한 효과

Dropout과 Batch Normalization을
동시에 사용하지는 않음!
둘 모두 비슷한 효과!!

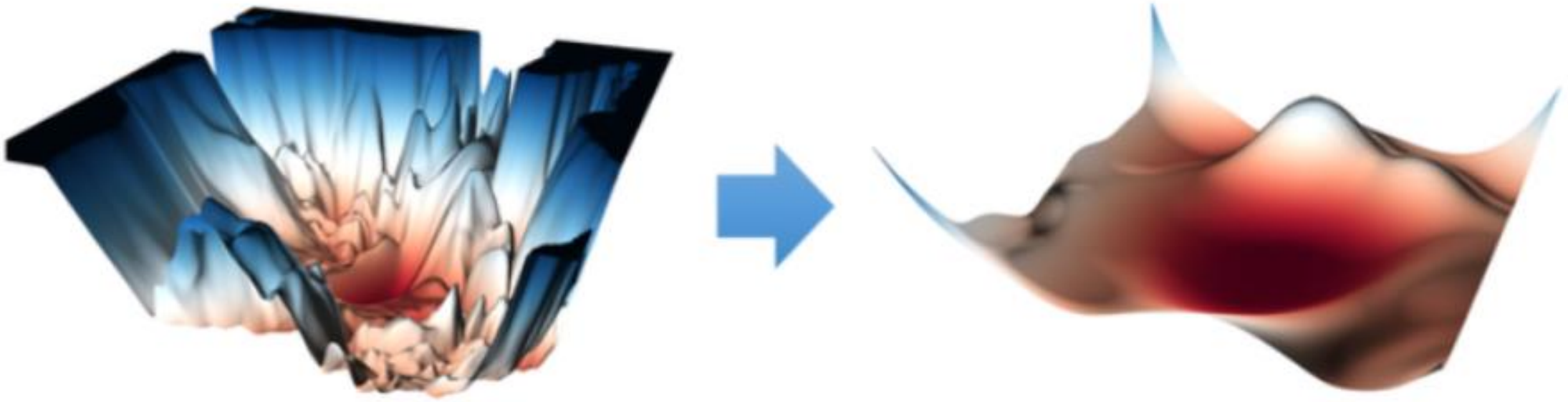
Batch-Normalization in CNN



Does Batch Normalization help to solve Internal Covariance Shifting?



Does Batch Normalization help to solve Internal Covariance Shifting?



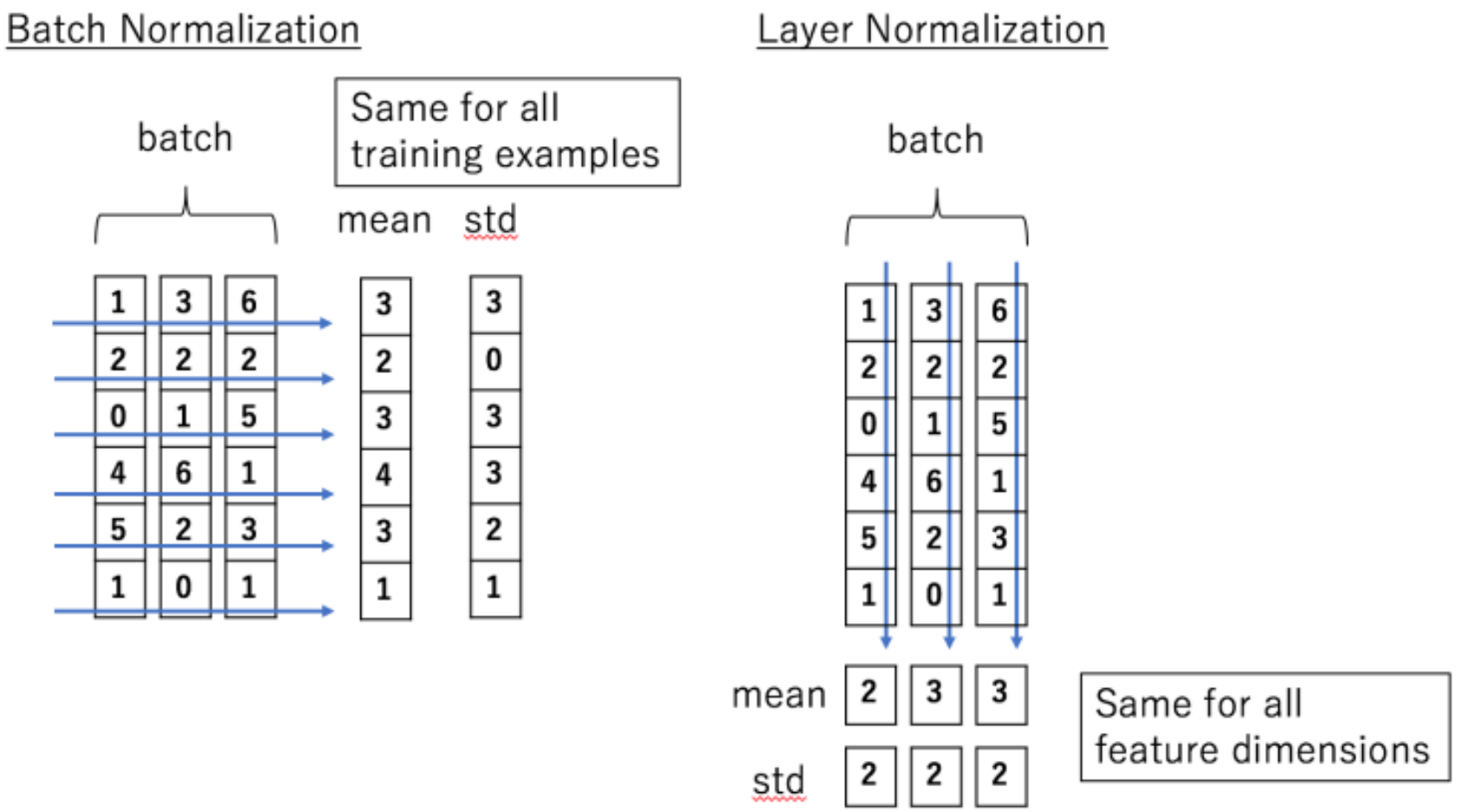
Layer 배치 순서

1. Convolution layer
2. **Batch normalization**
3. Activation
4. **Dropout**
5. Maxpooling

1. Dense layer
2. **Batch normalization**
3. Activation(**+Layer normlization**)
4. **Dropout**

물론, 저자들마다 하는 얘기가 다름!!

Layer Normalization / Batch-Renormalization



감사합니다