

BITAMIN 13기 & 14기
24-Summer Session

딥러닝 학습

배치 정규화(오버피팅, 가중치 감소, 드롭아웃), 하이퍼파라미터 최적화

1조 김진호 김나현 김시연 서영우

목차

1. 배치 정규화

1-1

배치 정규화란?

1-2

오버피팅

1-3

가중치 감소

1-4

드롭아웃

2. 하이퍼파라미터

2-1

하이퍼파라미터 종류

2-2

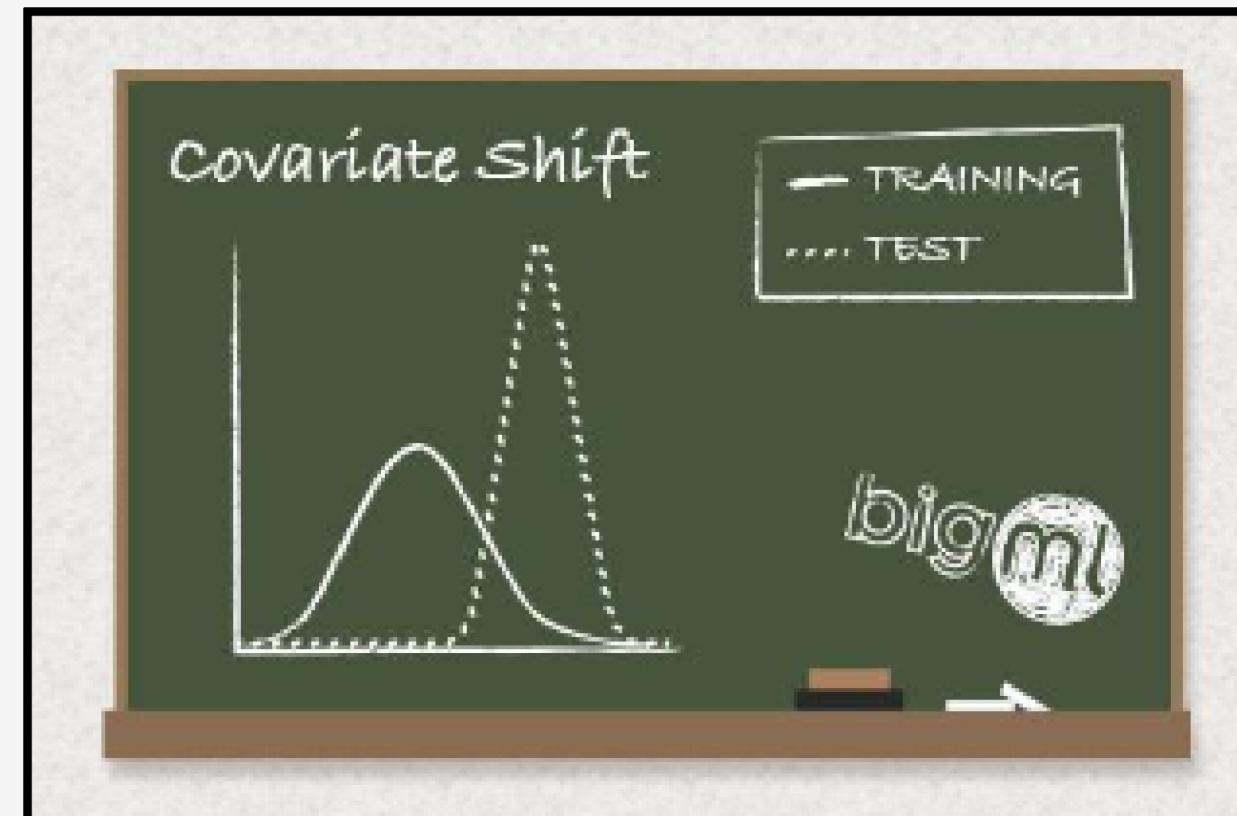
하이퍼파라미터 최적화 방법

공변량 변화

: Covariance Shift

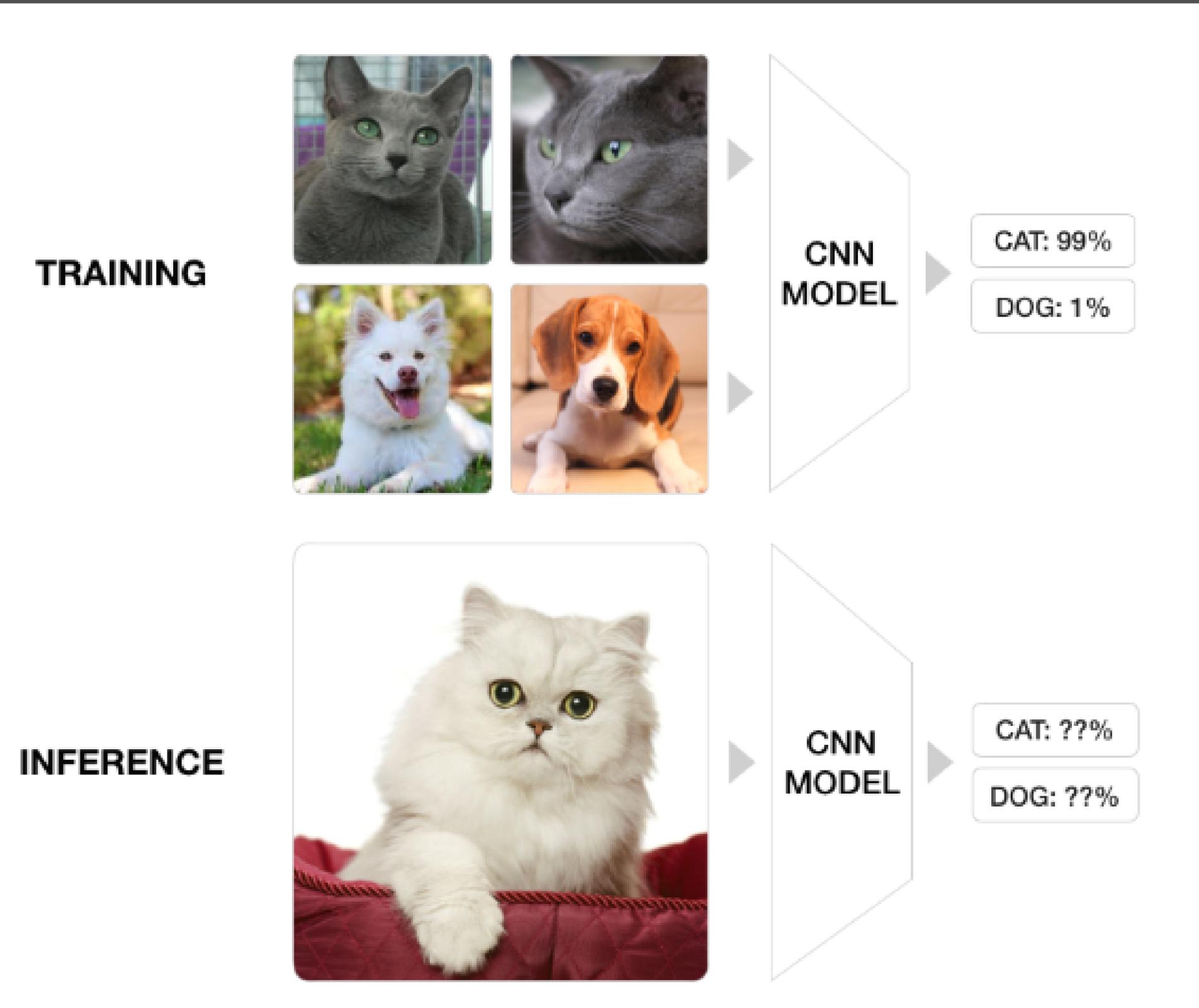
학습에서 불안정화가 일어나는 이유를 '**Internal Covariance Shift**'라고 주장하고 있는데, 이는 네트워크의 각 레이어나 Activation 마다 입력값의 분산이 달라지는 현상을 뜻한다.

Covariate shift는 공변량 변화라고 부르며 입력 데이터의 분포가 학습할 때와 테스트할 때 다르게 나타나는 현상을 말한다.



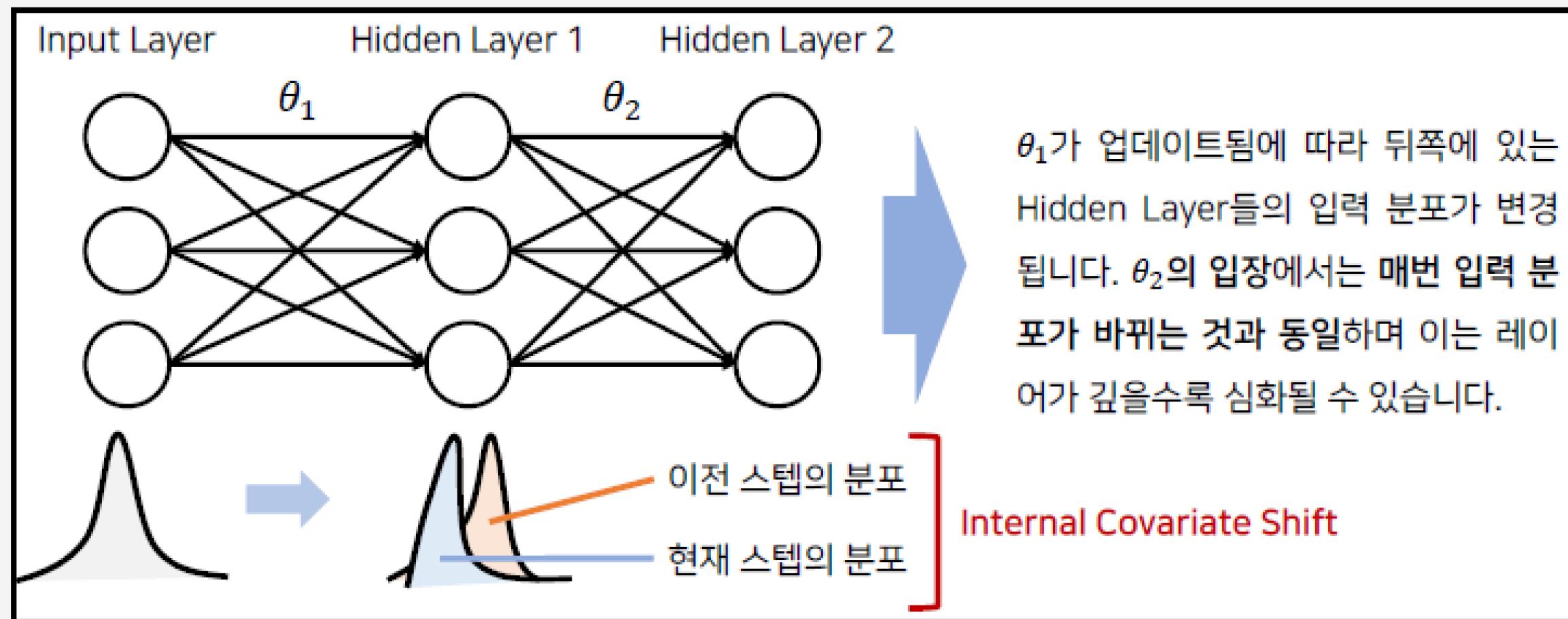
공변량 변화

학습에서 불안정화가 일어나는 이유를 'Inference Error'로 정의한다.
학습 데이터의 분포와 테스트 데이터의 분포가 다른 경우 ->
학습시킨 모델의 분류 성능은 떨어진다. 부
다르게 나타나는 현상을 말한다.
이처럼 학습 데이터와 테스트 데이터의
분포가 다른 것을 covariate shift라
부른다.



공변량 변화 : Covariance Shift

Covariate Shift(공변량 변화)가 뉴럴 네트워크 내부에서 일어나는 현상을 Internal Covariate Shift라고 한다.



정규화와 화이트닝

: Normalization & Whitenning

다른 분포를 가지는 두 데이터를 같은 분포를 가질 수 있게 변환을 해주는 방법으로
Normalization과 Whitenning이 있다.

정규화(Normalization)는 데이터를 동일한 범위 내의 값을 갖도록 하는 기법

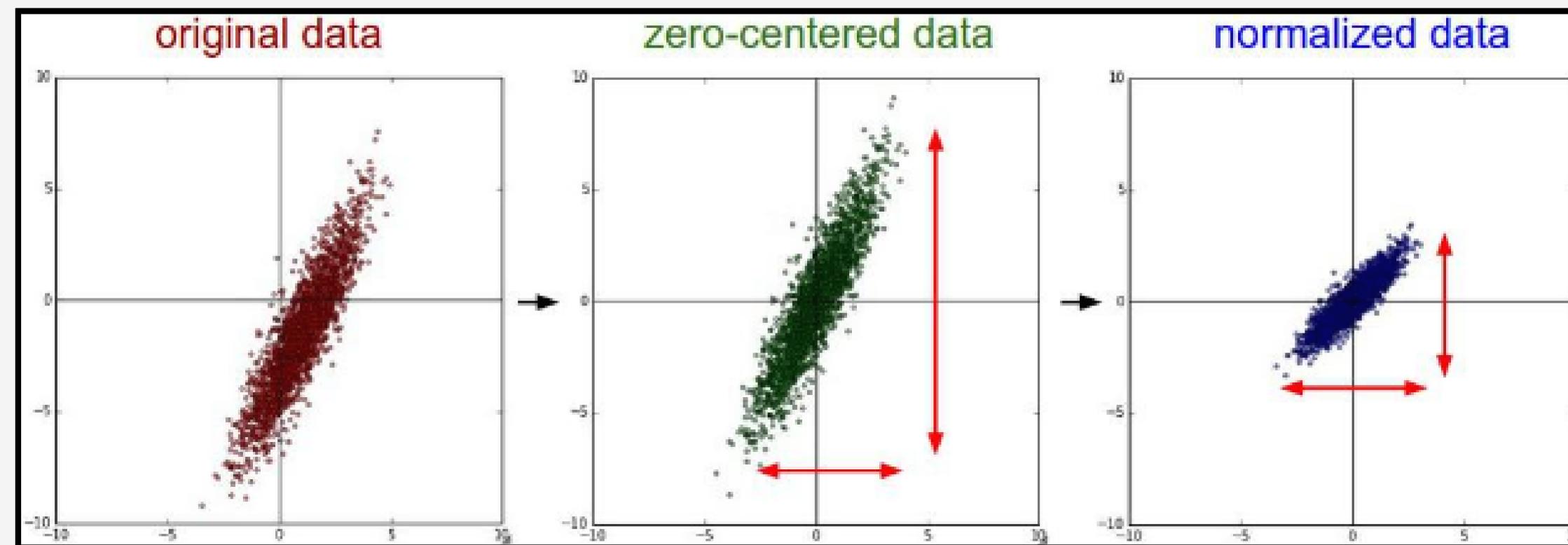
- Min-Max : 최소 0, 최대 1
- Standardization : 평균 0, 표준편차 1



정규화와 화이트닝

: Normalization & Whitenning

Normalization



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

정규화와 화이트닝

: Normalization & Whitenning

다른 분포를 가지는 두 데이터를 같은 분포를 가질 수 있게 변환을 해주는 방법으로
Normalization과 Whitenning이 있다.

화이트닝(Whitening)은 데이터의 평균을 0, 그리고 공분산을 단위행렬로 갖는 정규분포 형태로 변환하는 기법

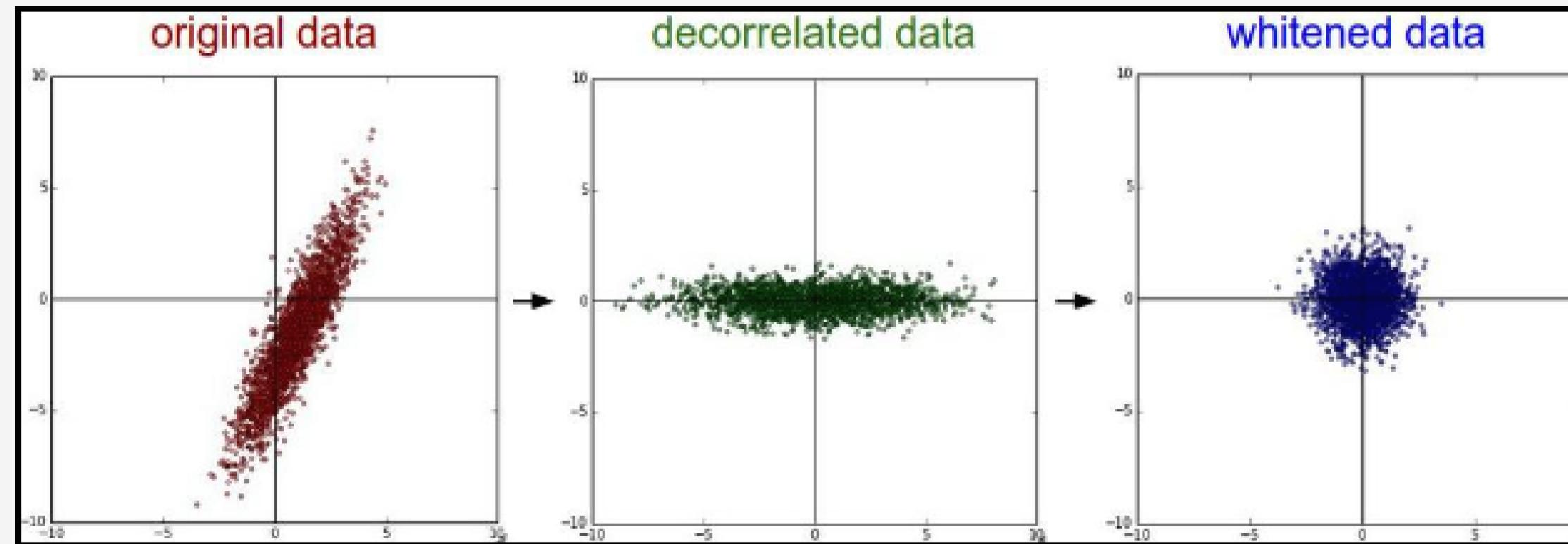
- Decorrelation
- Standardization



정규화와 화이트닝

: Normalization & Whitening

Whitening

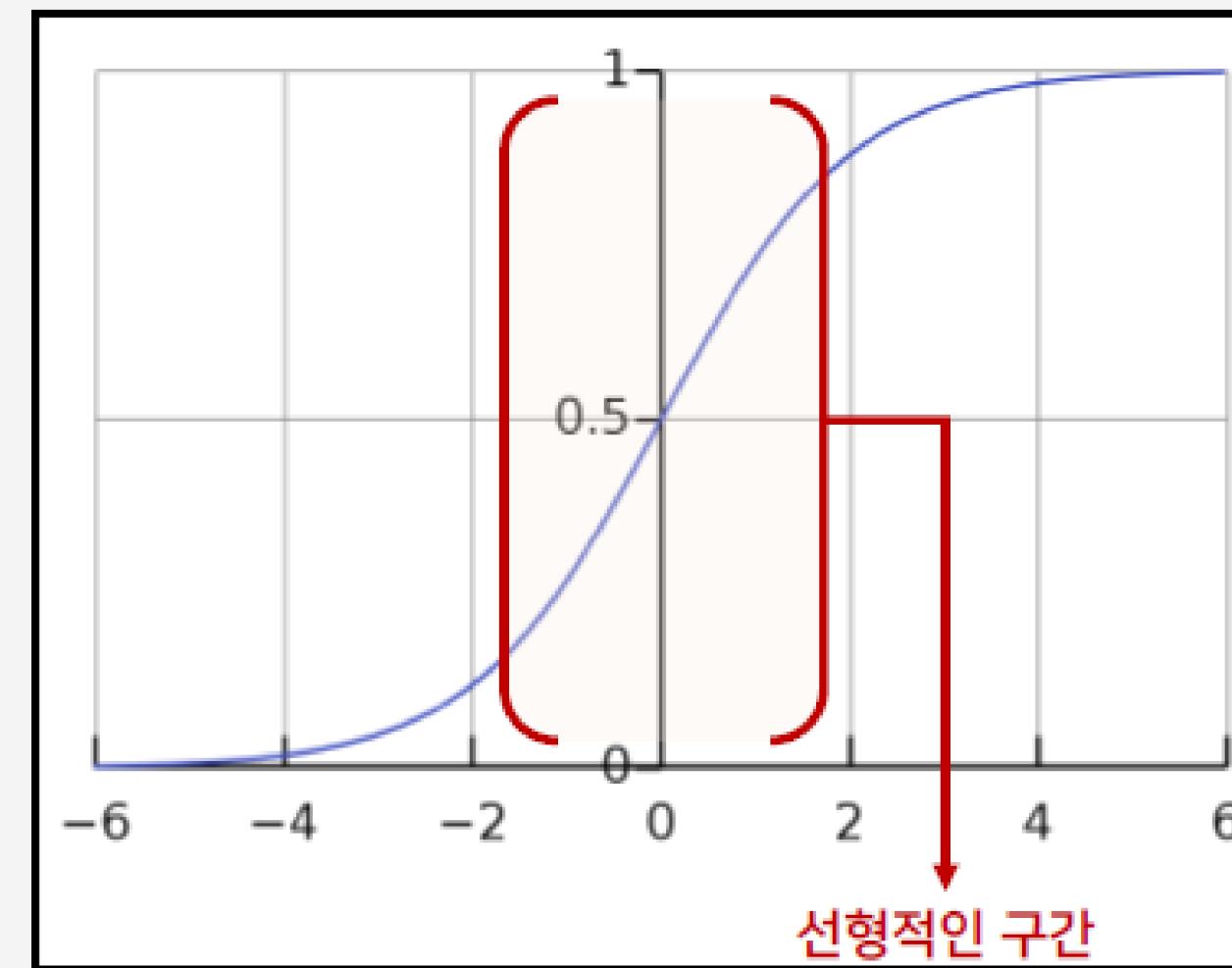


1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

화이트닝의 문제

: Problems with Whitening

평균 0, 표준편차 1로 데이터의 분포를 변환하게 될 경우 활성화 함수로 Sigmoid를 사용하게 되었을 때 비선형성(non-linearity)의 특징을 잃어버리게 되는 문제가 생긴다.

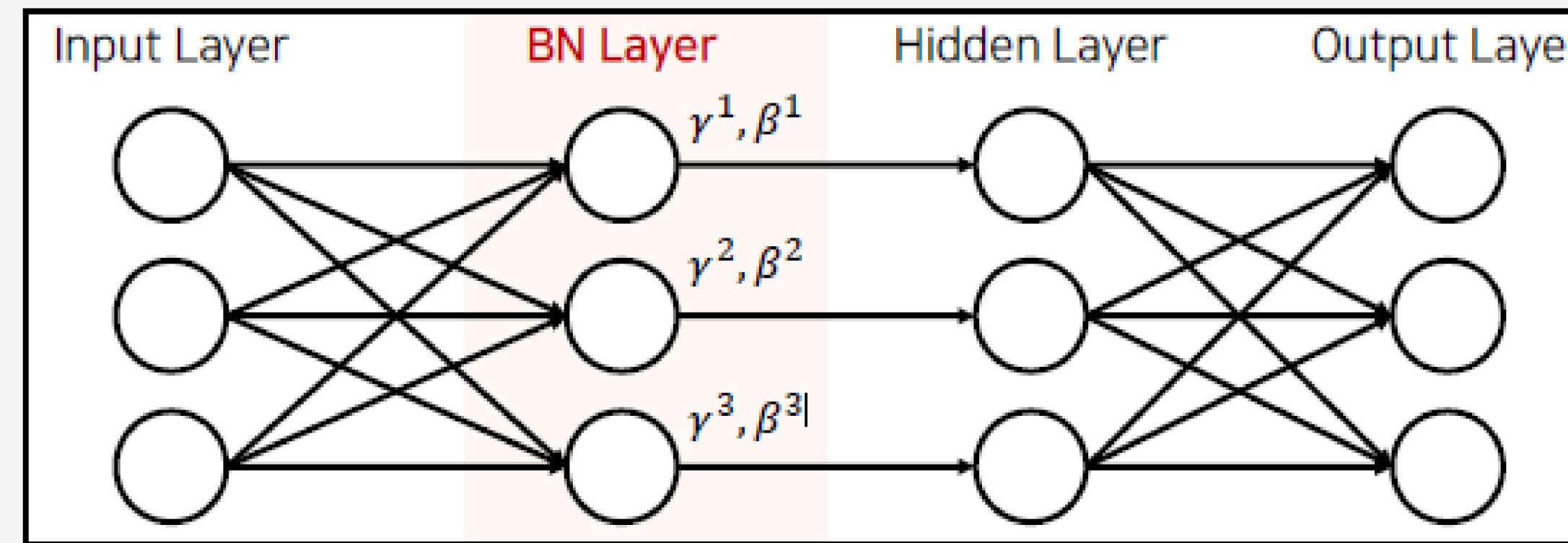


배치 정규화

: Batch Normalization

각 층이 활성화를 적당히 퍼뜨리도록 강제하는 알고리즘

학습 시 Quiz1 을 조정하여 Whitening의 문제점을 해결



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

배치 정규화 알고리즘

: Batch Normalization Alg.

학습 단계

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

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

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

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

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



배치 정규화 알고리즘

: Batch Normalization Alg.

학습 단계

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

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

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

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

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



배치 정규화 알고리즘

: Batch Normalization Alg.

학습 단계

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

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

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

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

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



배치 정규화 알고리즘

: Batch Normalization Alg.

학습 단계

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

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

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

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

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



배치 정규화 알고리즘

: Batch Normalization Alg.

학습 단계

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

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

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

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

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



배치 정규화 알고리즘

: Batch Normalization Alg.

추론 단계

```
Train  $N_{BN}^{tr}$  to optimize the parameters  $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$ 
 $N_{BN}^{inf} \leftarrow N_{BN}^{tr}$  // Inference BN network with frozen
// parameters

for  $k = 1 \dots K$  do
    // For clarity,  $x \equiv x^{(k)}$ ,  $\gamma \equiv \gamma^{(k)}$ ,  $\mu_B \equiv \mu_B^{(k)}$ , etc.
    Process multiple training mini-batches  $\mathcal{B}$ , each of
    size  $m$ , and average over them:
         $E[x] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ 
         $Var[x] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$ 

    In  $N_{BN}^{inf}$ , replace the transform  $y = BN_{\gamma, \beta}(x)$  with
     $y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right)$ 
end for
```



배치 정규화 알고리즘

: Batch Normalization Alg.

추론 단계

```
Train  $N_{BN}^{tr}$  to optimize the parameters  $\Theta \cup$ 
 $\{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$ 
 $N_{BN}^{inf} \leftarrow N_{BN}^{tr}$  // Inference BN network with frozen
// parameters
모집단 추정 방식
for  $k = 1 \dots K$  do
    // For clarity,  $x \equiv x^{(k)}$ ,  $\gamma \equiv \gamma^{(k)}$ ,  $\mu_B \equiv \mu_B^{(k)}$ , etc.
    Process multiple training mini-batches  $B$ , each of
         $E[x^{(k)}] = E_B[\mu_B^{(k)}]$ 
         $Var[x^{(k)}] = \frac{m}{m-1} E_B[\sigma_B^{(k)2}]$ 
end for
```



배치 정규화 알고리즘

: Batch Normalization Alg.

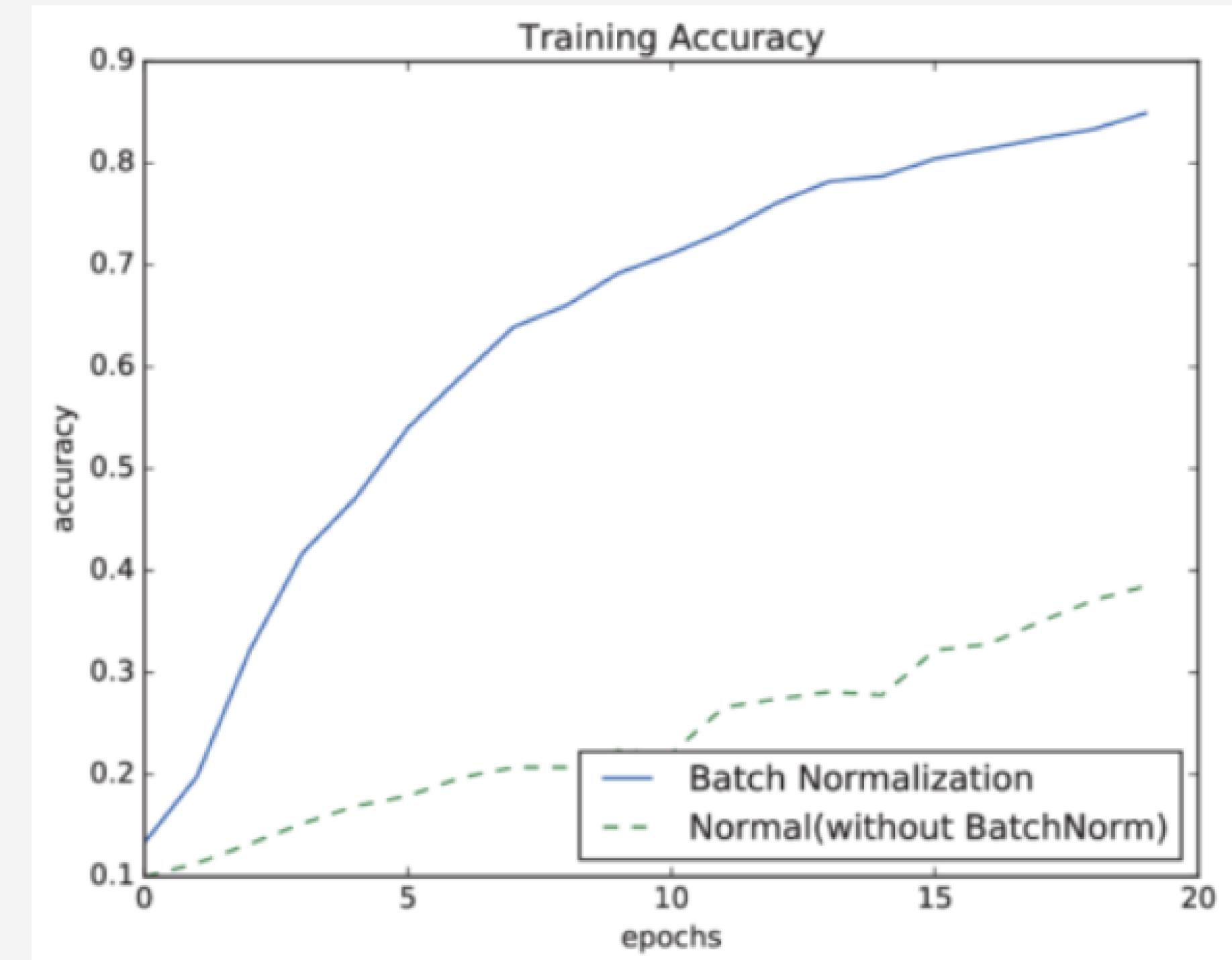
추론 단계

```
Train  $N_{BN}^{tr}$  to optimize the parameters  $\Theta \cup$ 
 $\{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$ 
 $N_{BN}^{inf} \leftarrow N_{BN}^{tr}$  // Inference BN network with frozen
// parameters
// 이동 평균
for  $k = 1 \dots K$  do
    // For clarity,  $x \equiv x^{(k)}$ ,  $\gamma \equiv \gamma^{(k)}$ ,  $\mu_B \equiv \mu_B^{(k)}$ , etc.
    Process multiple training mini-batches  $B_h$  each of
         $\hat{\mu} \leftarrow \alpha \hat{\mu} + (1 - \alpha) \mu_\beta^{(i)}$ 
         $\hat{\sigma} \leftarrow \alpha \hat{\sigma} + (1 - \alpha) \sigma_\beta^{(i)}$ 
    end for
```



배치 정규화의 효과

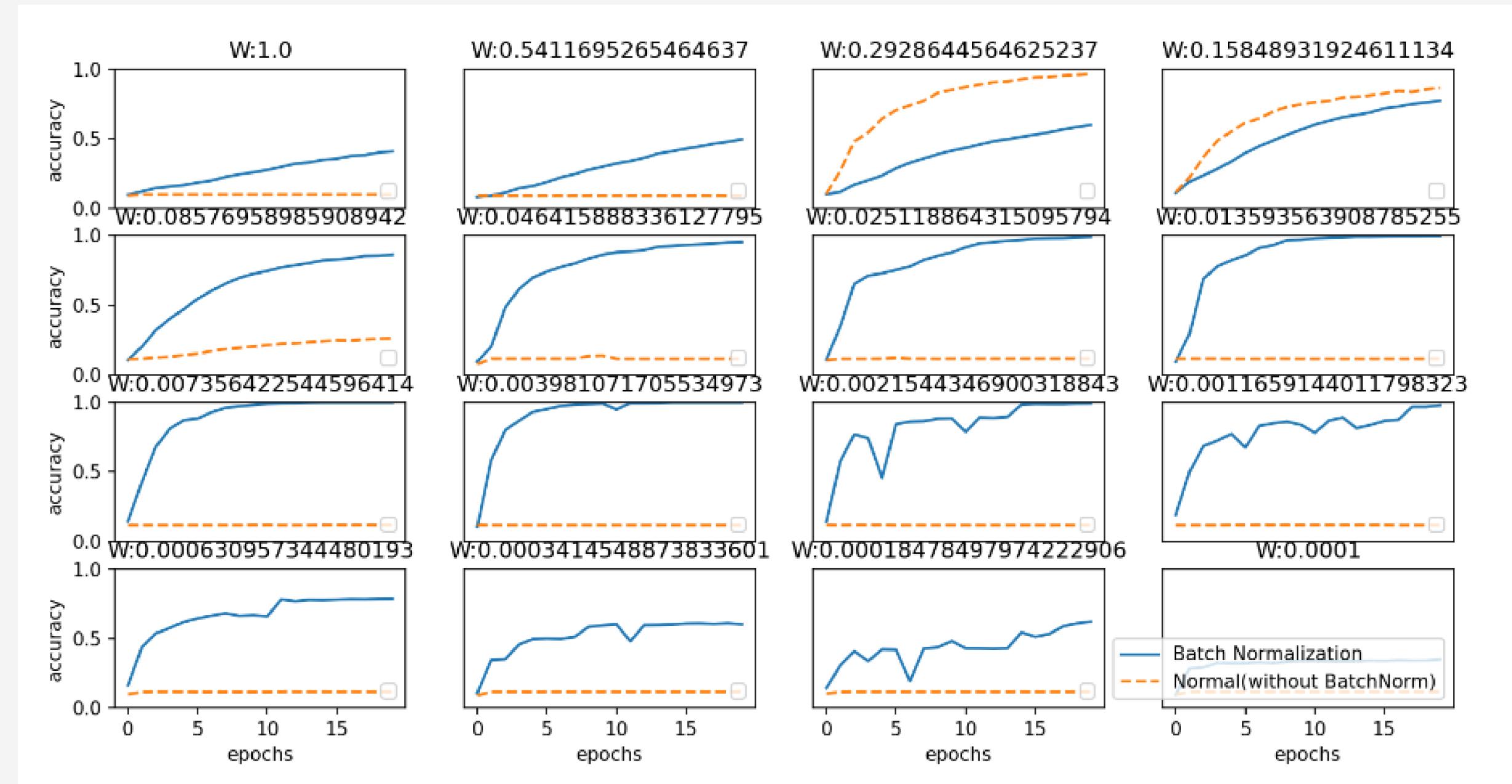
: Batch Normalization Effect



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃 2. 하이퍼파라미터 – 종류 / 최적화 방법

배치 정규화의 효과

: Batch Normalization Effect



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃 2. 하이퍼파라미터 – 종류 / 최적화 방법

오버피팅

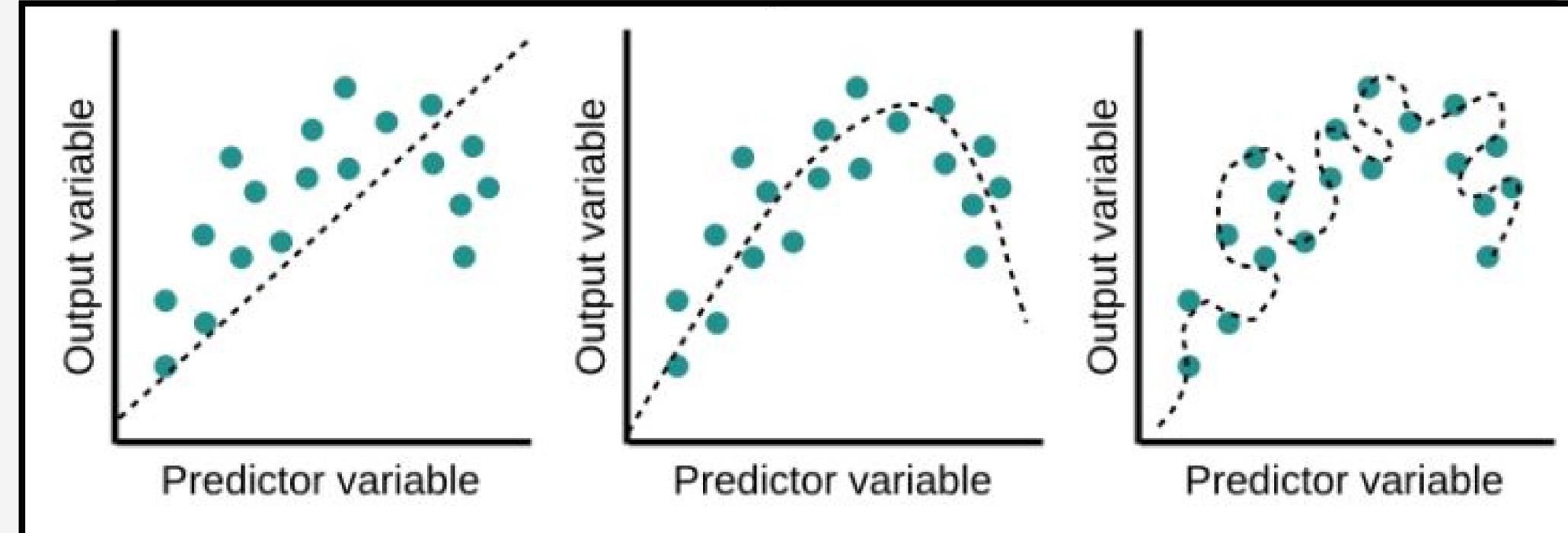
: Overfitting

: 신경망이 훈련 데이터에 과도하게 맞춰져서 훈련 데이터에서는 높은 성능을 보이지만,
새로운 데이터나 테스트 데이터에서는 성능이 저하되는 현상

언제 일어나나?

: 매개변수가 많고 표현력이 높은 모델일 경우 훈련 데이터가 적을 경우

Quiz2



Ex) Mnist 데이터 300개로 오버피팅 재현

```
import os
import sys

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.multi_layer_net import MultiLayerNet
from common.optimizer import SGD

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

# 오버피팅을 재현하기 위해 학습 데이터 수를 줄임
x_train = x_train[:300]
t_train = t_train[:300]

# weight decay(가중치 감쇠) 설정 =====
#weight_decay_lambda = 0 # weight decay를 사용하지 않을 경우
weight_decay_lambda = 0.1
# =====

network = MultiLayerNet(input_size=784,
                         hidden_size_list=[100, 100, 100, 100, 100, 100],
                         output_size=10,
                         weight_decay_lambda=weight_decay_lambda)
optimizer = SGD(lr=0.01) # 학습률이 0.01인 SGD로 매개변수 갱신

max_epochs = 201
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)
epoch_cnt = 0

for i in range(1000000000):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    grads = network.gradient(x_batch, t_batch)
    optimizer.update(network.params, grads)

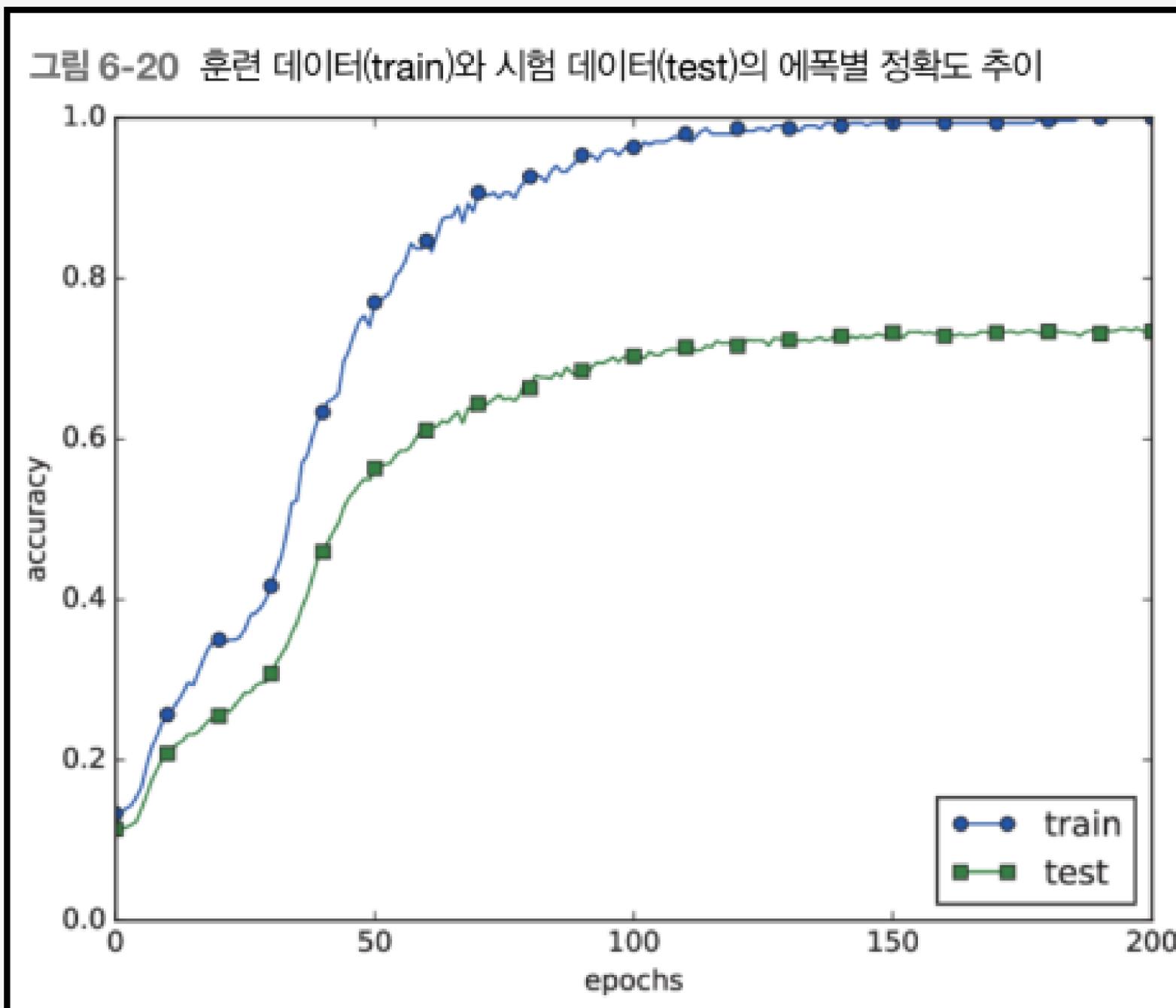
    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)

        print("epoch:" + str(epoch_cnt) + ", train acc:" +
              str(train_acc) + ", test acc:" + str(test_acc))

    epoch_cnt += 1
    if epoch_cnt >= max_epochs:
        break
```



Ex) Mnist 데이터 300개로 오버피팅 재현



- 학습 데이터에서는 뛰어난 정확도를 보이지만, 테스트 데이터에서는 그만큼의 성능을 보여주지 못함 -> 오버피팅이 발생했기 때문



가중치 감소

: Weight Decay

: 학습 과정에서 큰 가중치에 대해서는 큰 페널티를 부과하여 **오버피팅을 억제하는 방법**

L1 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M W_j^2$$

Loss function Regularization Term

L1 규제 (Lasso) :

기존 손실값 + (람다λ값이 곱해진) 가중치의 절대값

L2 규제 (Ridge) :

기존 손실값 + 가중치의 제곱값

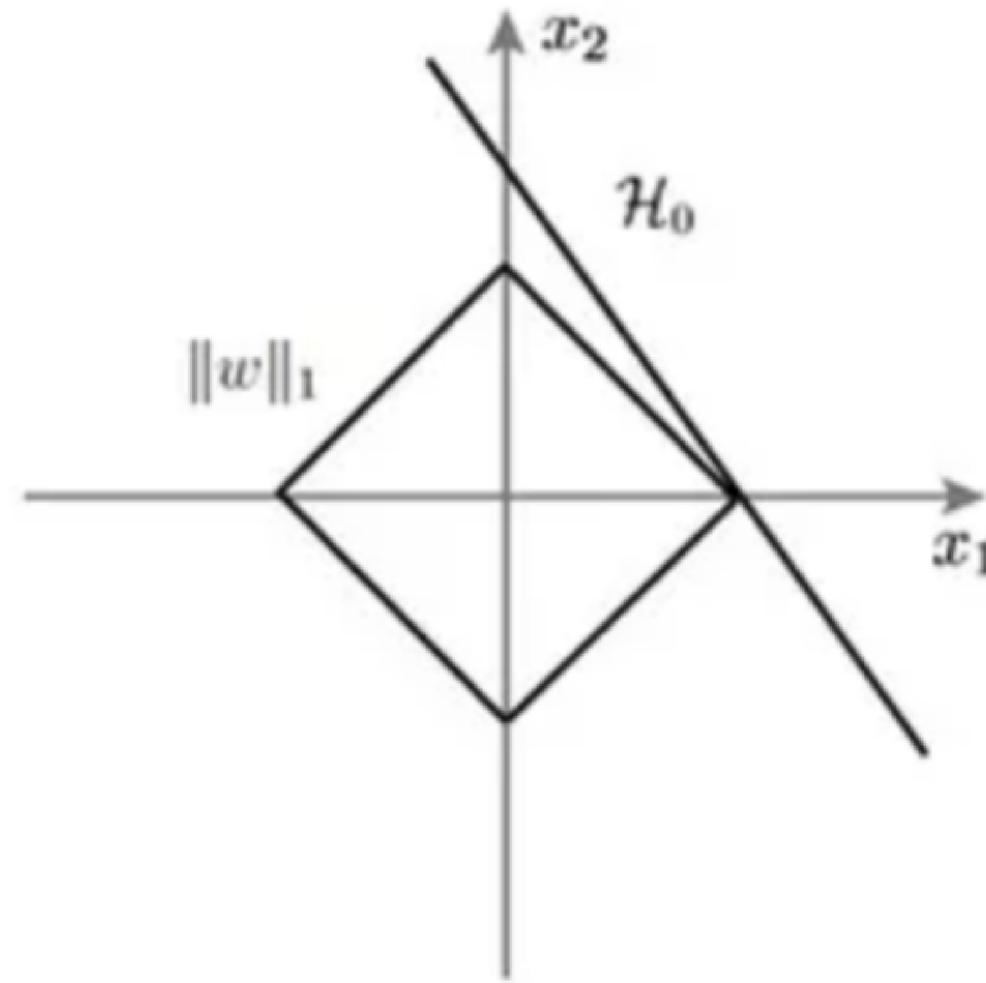
각 가중치(W)를 제곱한 것의 총합(np.sum)을 2로 나눠준 값



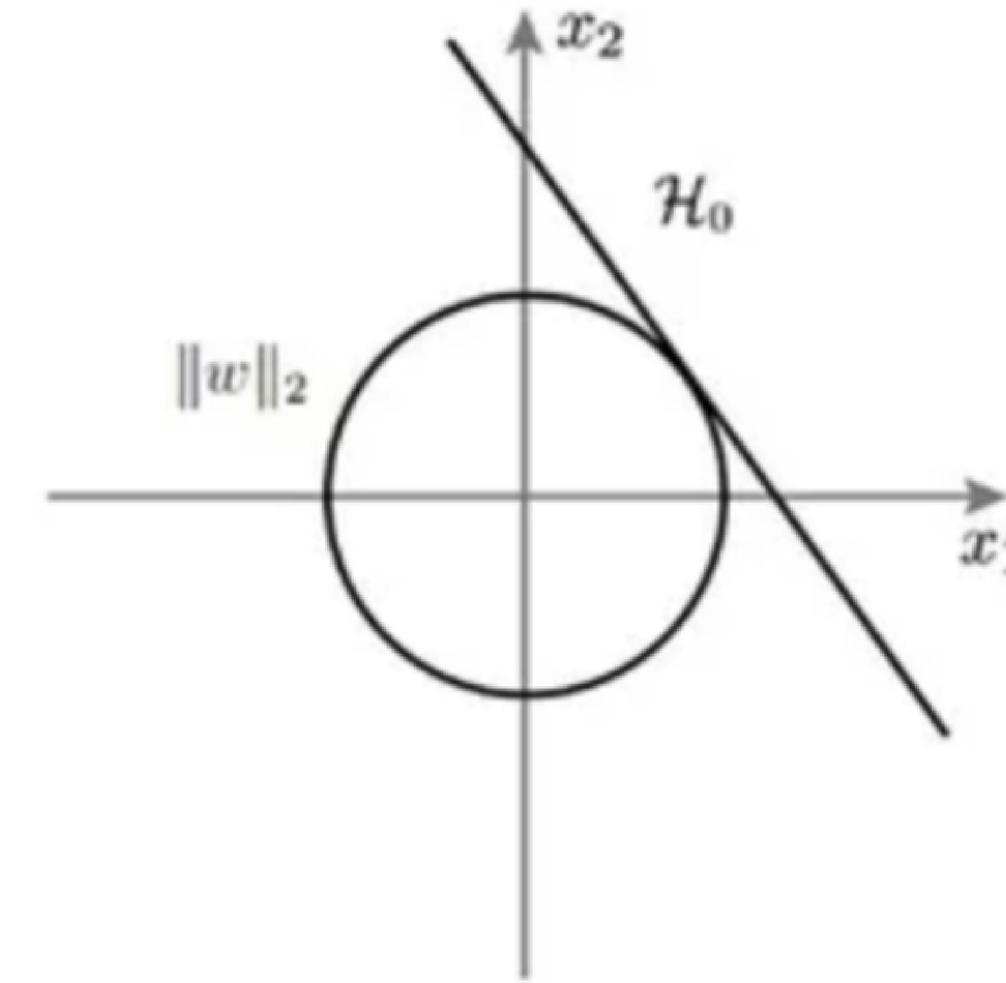
가중치 감소

: Weight Decay

A L1 regularization



B L2 regularization



가중치 감소

: Weight Decay

※ L2 정규화 구현 코드

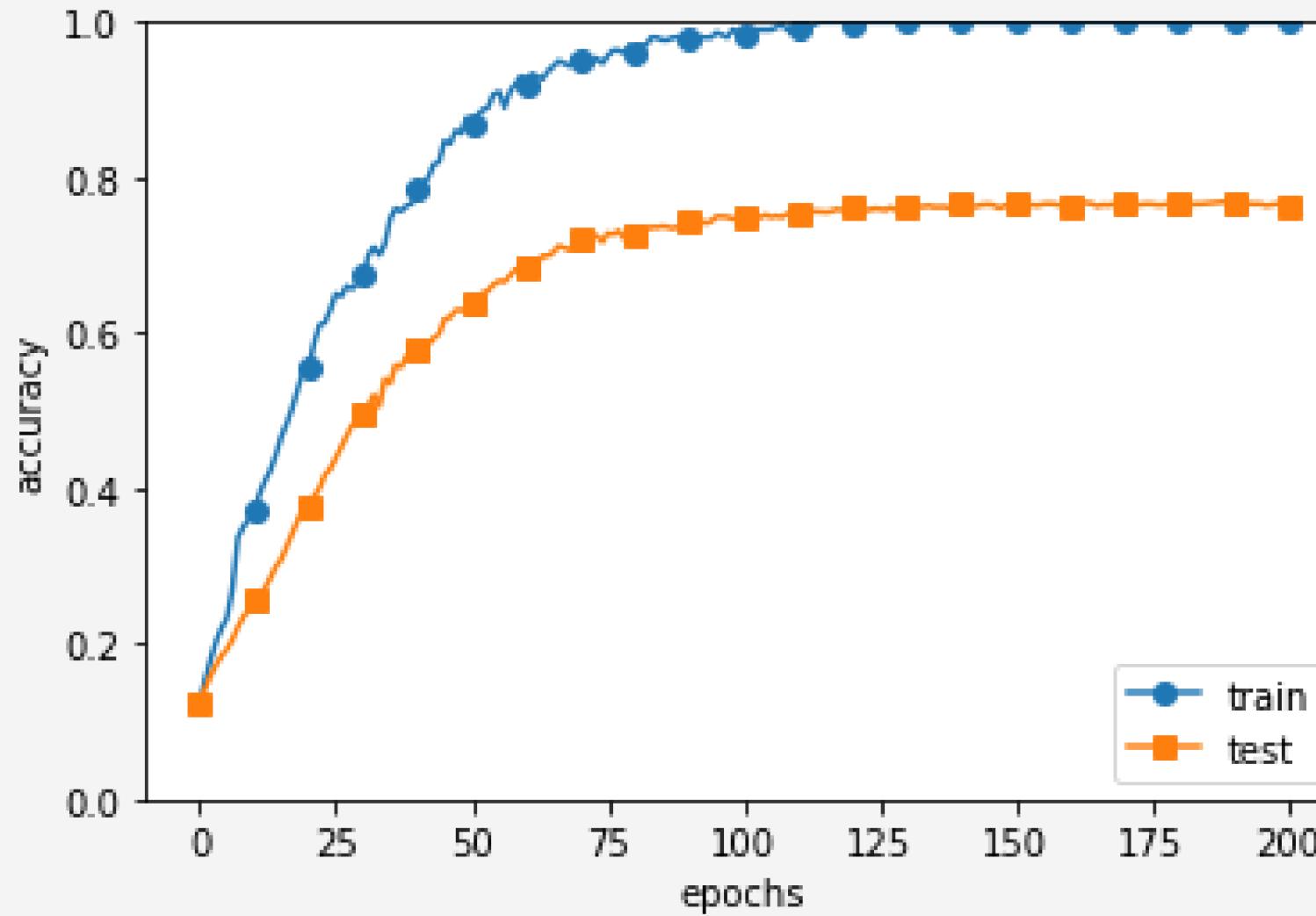
```
class TwoLayerNet:  
    ...  
  
    def loss(self, x, t):  
        y = self.predict(x)  
  
        if self.l2:  
            weight_decay = 0  
            for idx in range(1, 3):  
                W = self.params['W' + str(idx)]  
                weight_decay += 0.5 * weight_decay_lambda * np.sum(W ** 2)  
        return self.lastlayer.forward(y, t) + weight_decay  
  
    y = self.predict(x)  
    return self.lastlayer.forward(y, t)
```



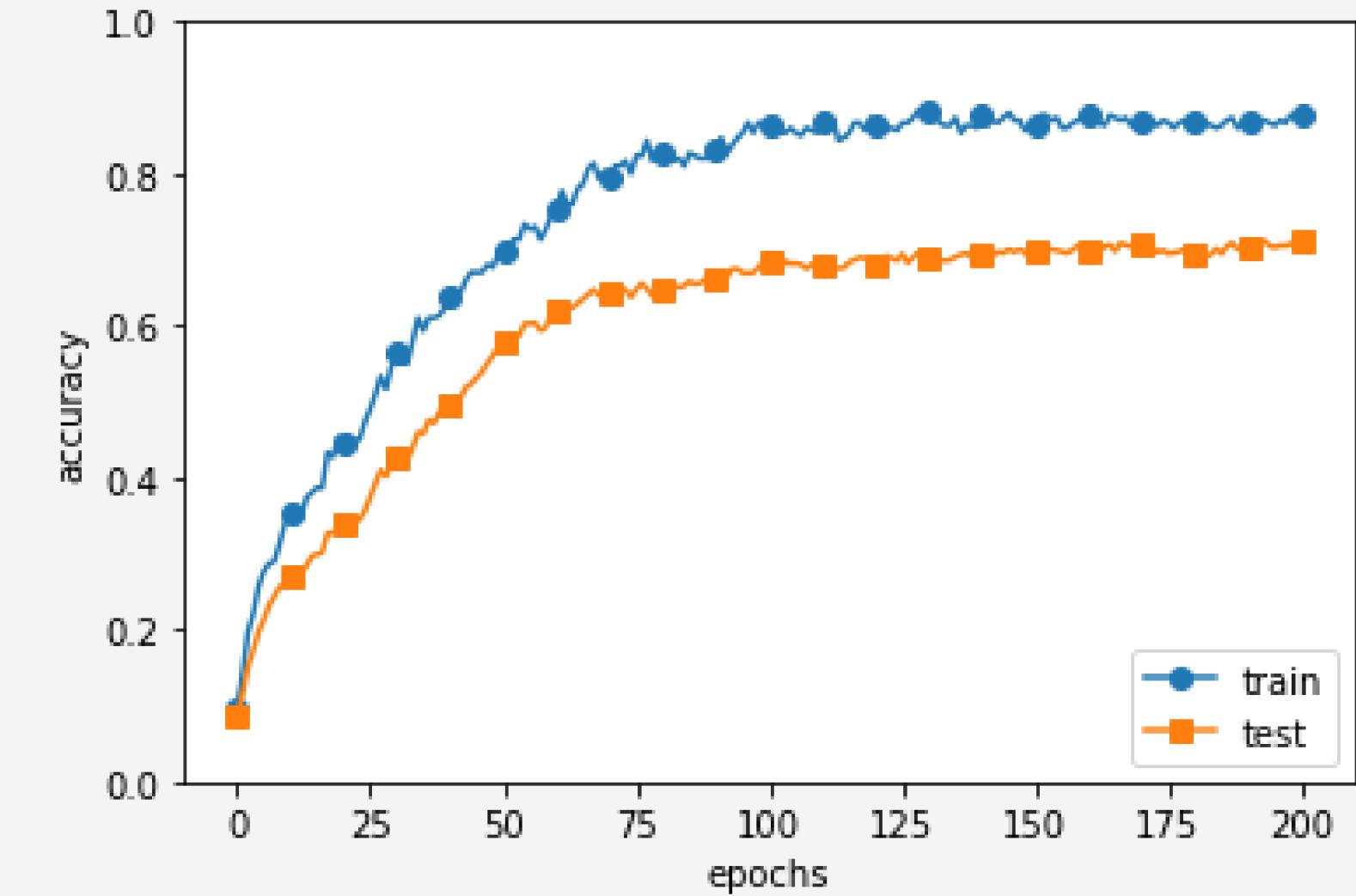
가중치 감소

: Weight Decay

가중치 감소 이전



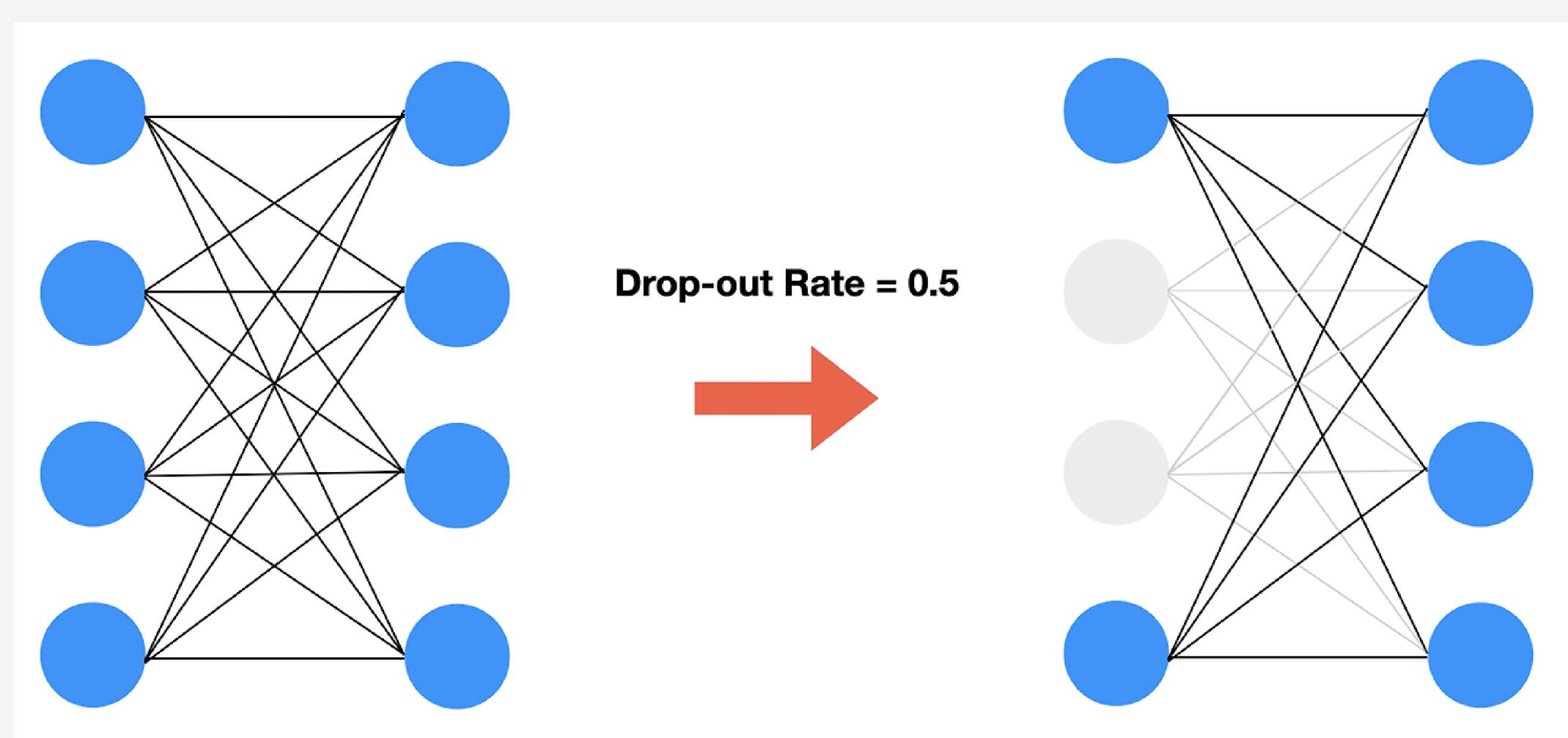
가중치 감소 이후



드롭아웃

: Dropout

뉴런을 임의로 삭제해서 학습하는 방법
훈련 시 은닉층의 뉴런을 무작위로 골라 삭제함

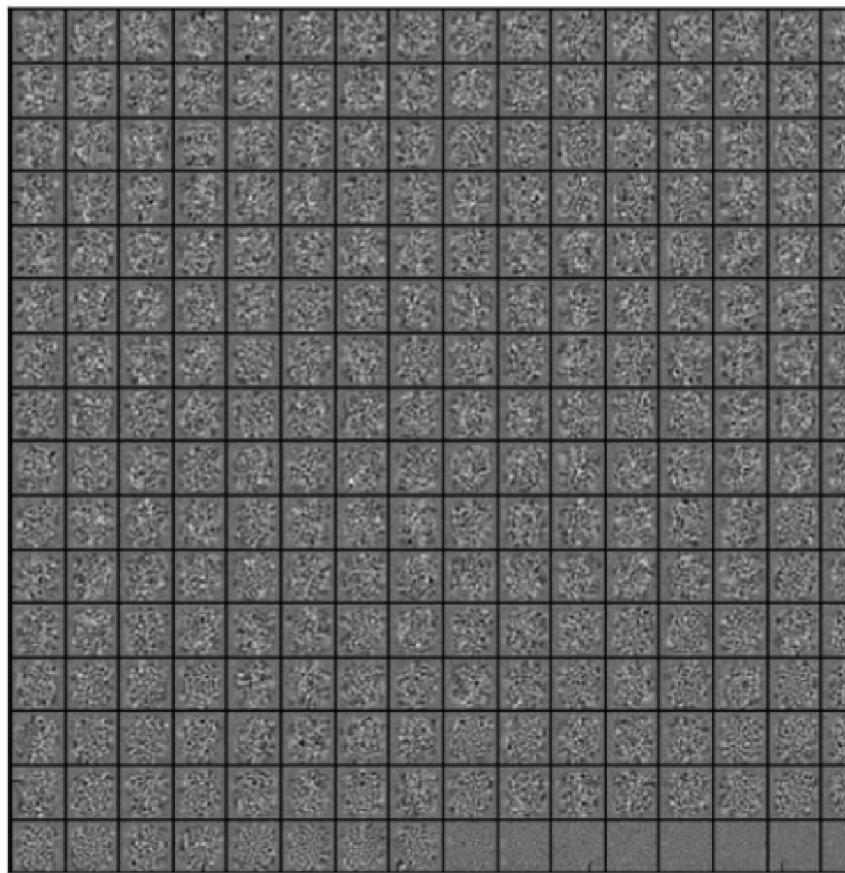


1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

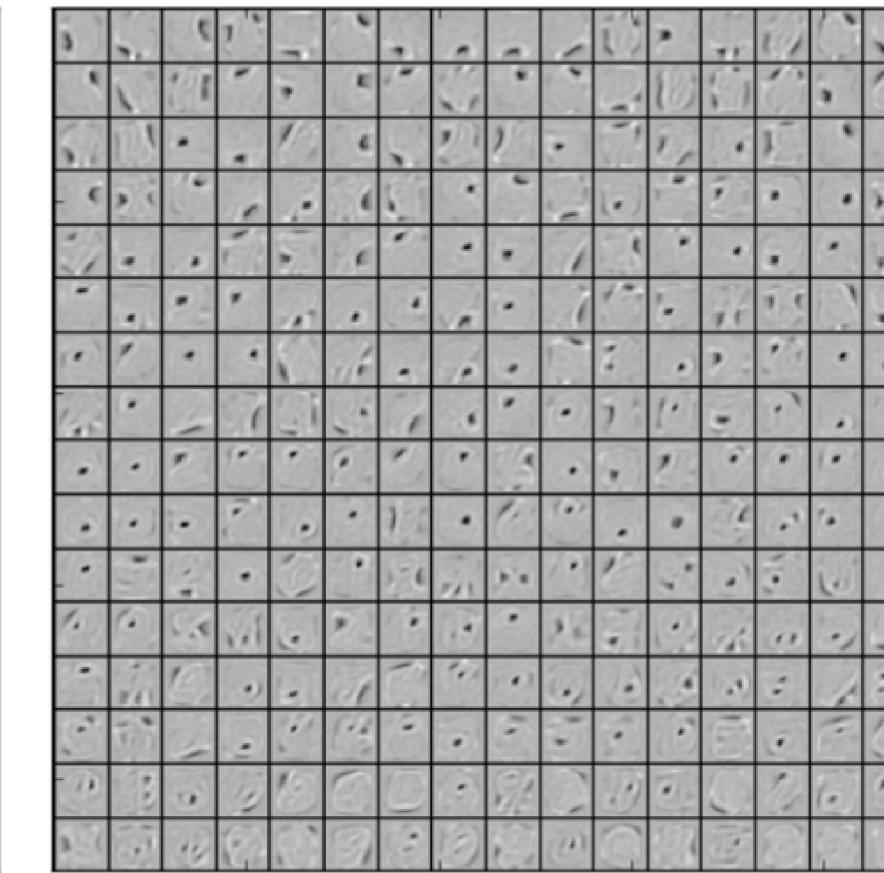
드롭아웃

: Dropout

- p 값: 각 훈련 반복 중 무작위로 탈락되는 뉴런의 비율.
- 0.2 - 0.5가 적절. 너무 낮으면 과적합을 효과적으로 방지할 수 없고, 탈락률이 너무 높으면 학습을 방해함.



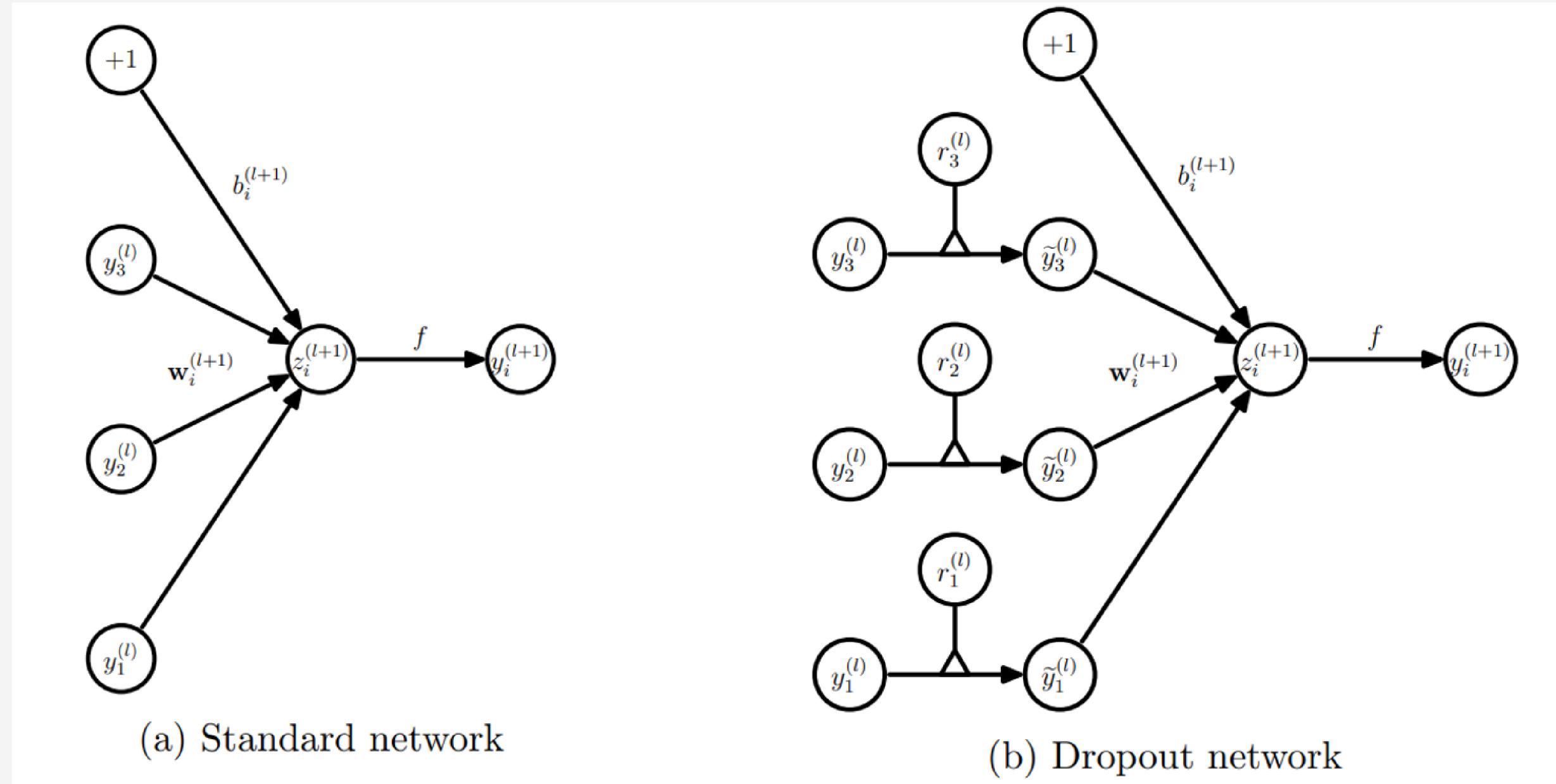
(a) Without dropout



(b) Dropout with $p = 0.5$.



드롭아웃 : Dropout



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

드롭아웃

: Dropout

- **순전파**

- self.mask에 삭제할 뉴런을 False로 표시
- self.mask는 x와 형상이 같은 배열을 무작위로 생성
- 그 값이 dropout_ratio보다 큰 원소만 True로 설정

- **역전파**

- 순전파때 신호를 통과시키는 뉴런은 역전파때도 통과
- 아닌 뉴런은 역전파 때도 차단

- **평가**

- 모든 뉴런에 신호를 전달.
- 단, 각 뉴런의 출력에 훈련 때 삭제한 비율을 곱하여 출력
- → 앙상블에서 여러 모델의 평균을 내는 효과



드롭아웃

: Dropout

※(드롭아웃 예시 코드)

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



하이퍼파라미터

: Hyperparameter

Hyperparameter는 주로 알고리즘 사용자가 경험에 의해 직접 세팅하는 값

- 정해진 최적의 값이 없음
- 외적인 요소, 데이터 분석 등을 통해 얻어지는 것이 아님
- 모델의 parameter값을 측정하기 위해 알고리즘 구현 과정에서 사용됨

Hyperparameter Tuning(최적화) :

- 모델이나 데이터에 맞춰 여러 시도 후 모델이 맞는 하이퍼파라미터를 찾아 나가는 과정
- 튜닝 방법 : Manual Search, Grid Search, Random Search, Bayesian Optimization etc..



하이퍼파라미터의 종류

〈네트워크 계층과 관련된 하이퍼파라미터〉

- 은닉층의 뉴런 개수 (Hidden Unit)
 - 훈련 데이터에 대한 학습 최적화 결정 변수
 - 첫 Hidden Layer의 뉴런 수가 Input Layer 보다 큰 것이 효과적
- Dropout에서의 p값
 - 과적합을 피하기 위한 정규화 기법
 - 학습데이터에 과적합하게 학습된 model을 실제로 사용하기 위해 범용성을 위한 기법으로 Hidden Layer의 Neuron들을 일정 비율로 배제하고 학습
- 가중치 초기화 (Weight Initialization)
 - 학습 성능에 대한 결정 변수



하이퍼파라미터의 종류

〈학습에 사용되는 하이퍼파라미터〉

- **학습률 (learning rate)**

- gradient의 방향으로 얼마나 빠르게 이동할 것인지 결정하는 변수
- 너무 작으면 학습의 속도가 늦고, 너무 크면 학습이 불가

- **모멘텀 (momentum)**

- 학습 방향을 유지하려는 성질
- 모멘텀을 사용하면 학습 방향이 바로 바뀌지 않고, 일정한 방향을 유지하며 움직임
- 같은 방향의 학습이 진행된다면 가속을 가지며 더 빠른 학습을 기대할 수 있음

- **에포크 (epoch, training epochs)**

- 전체 트레이닝 셋이 신경망을 통과한 횟수를 의미
- ex. 1-epoch는 전체 트레이닝 셋이 하나의 신경망에 적용되어 순전파와 역전파를 통해 신경망을 한 번 통과했다는 것을 의미



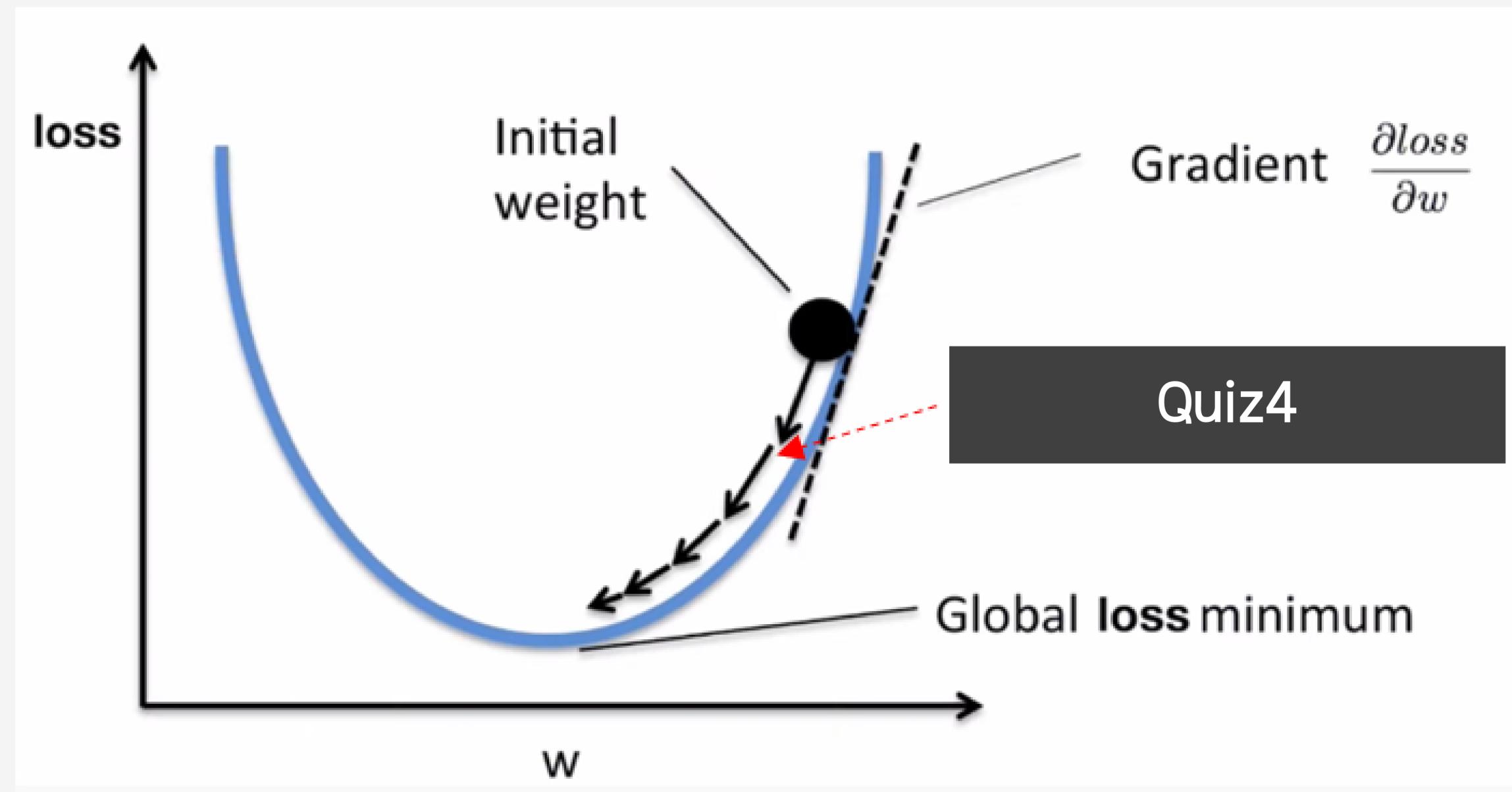
하이퍼파라미터의 종류

〈학습에 사용되는 하이퍼파라미터〉

- 배치 사이즈 (batch size)
 - 미니 배치 크기(Mini-batch Size)
 - 배치셋 수행을 위해 전체 학습 데이터를 등분하는 크기
- 반복 (iteration)
 - Quiz3 를 마치는데 필요한 미니배치 갯수를 의미
- 손실함수 (Cost Function)
 - 입력에 따른 기대 값과 실제 값의 차이를 계산하는 함수
 - 평균 제곱 오차, 교차 엔트로피 오차 ...
- 정규화 파라미터 (Regularization parameter)
 - L1 또는 L2 정규화 방법 사용
 - 사용하는 일반화 변수도 하이퍼 파라미터로 분류된다.



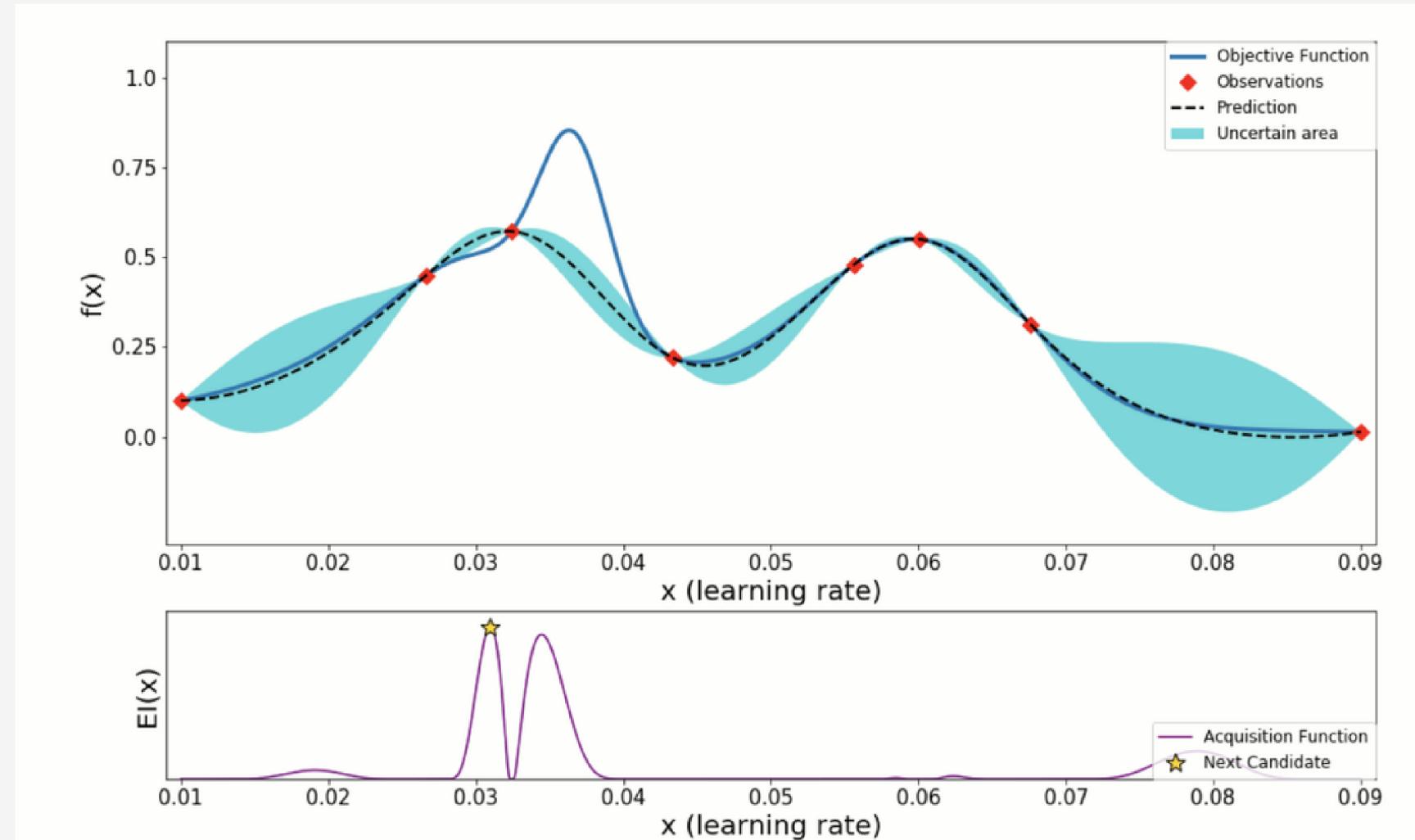
하이퍼파라미터의 종류



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

하이퍼파라미터 최적화 절차

대략적인 범위를 설정하고
그 범위에서 무작위로 하이퍼파라미터 값을 골라낸 후
정확도를 평가하며 최적화



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

하이퍼파라미터 최적화 기법

1. **Manual Search** : 말그대로 A-Z까지 하나하나 넣어보면서 최적의 결과를 찾는 방법
2. **Grid Search** : 하이퍼파라미터를 튜닝해야하는 값들을 하나하나 넣어보면서 최적화된 결과를 찾는 방법
3. **Bayesian Optimization** : 베이지안 기반으로 가장 성능이 높을만한 영역을 중심으로 검색 영역을 줄여나가는 방법
4. **Quiz5** : 데이터 증강 방법론 중 강화학습을 도입하여 어떤 데이터 증강을 사용할지, 얼마만큼의 확률 (probability)로 사용할지, 얼마만큼 변화 (magnitude) 시킬지를 학습

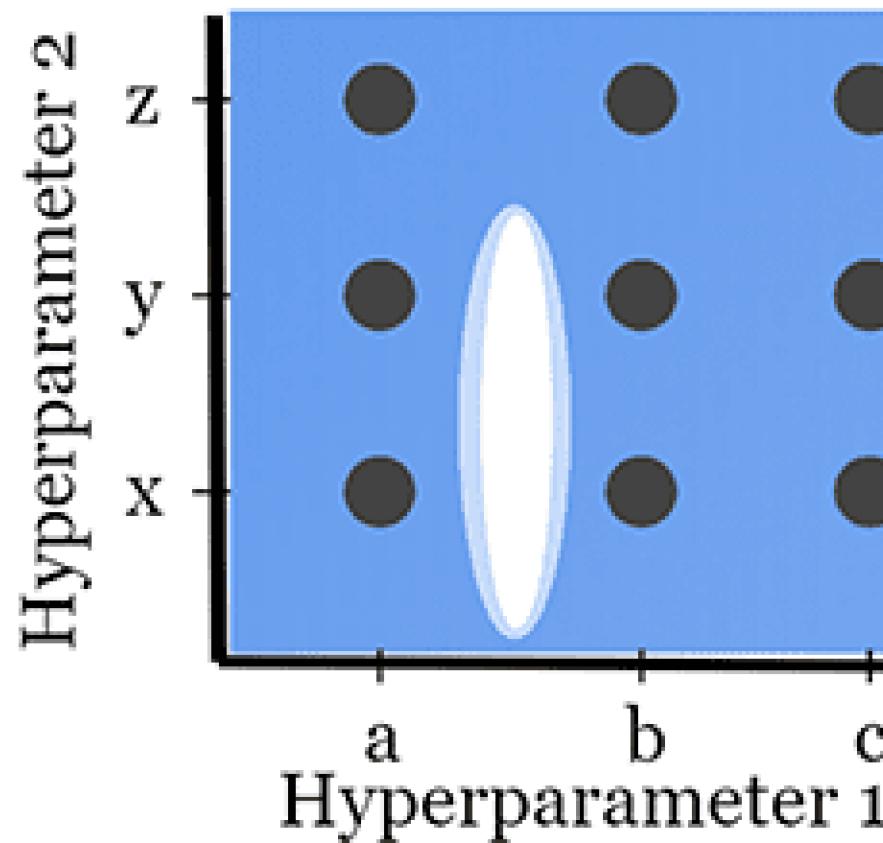


하이퍼파라미터 최적화 기법

Grid Search

Pseudocode

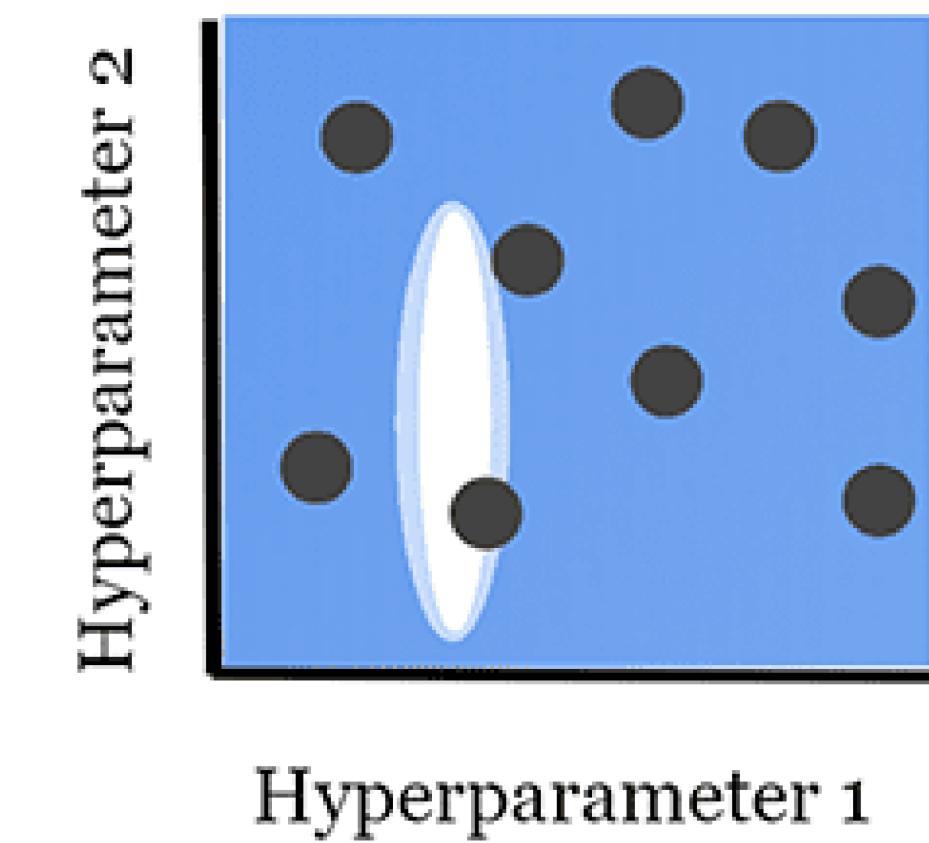
```
Hyperparameter_One = [a, b, c]  
Hyperparameter_Two = [x, y, z]
```



Random Search

Pseudocode

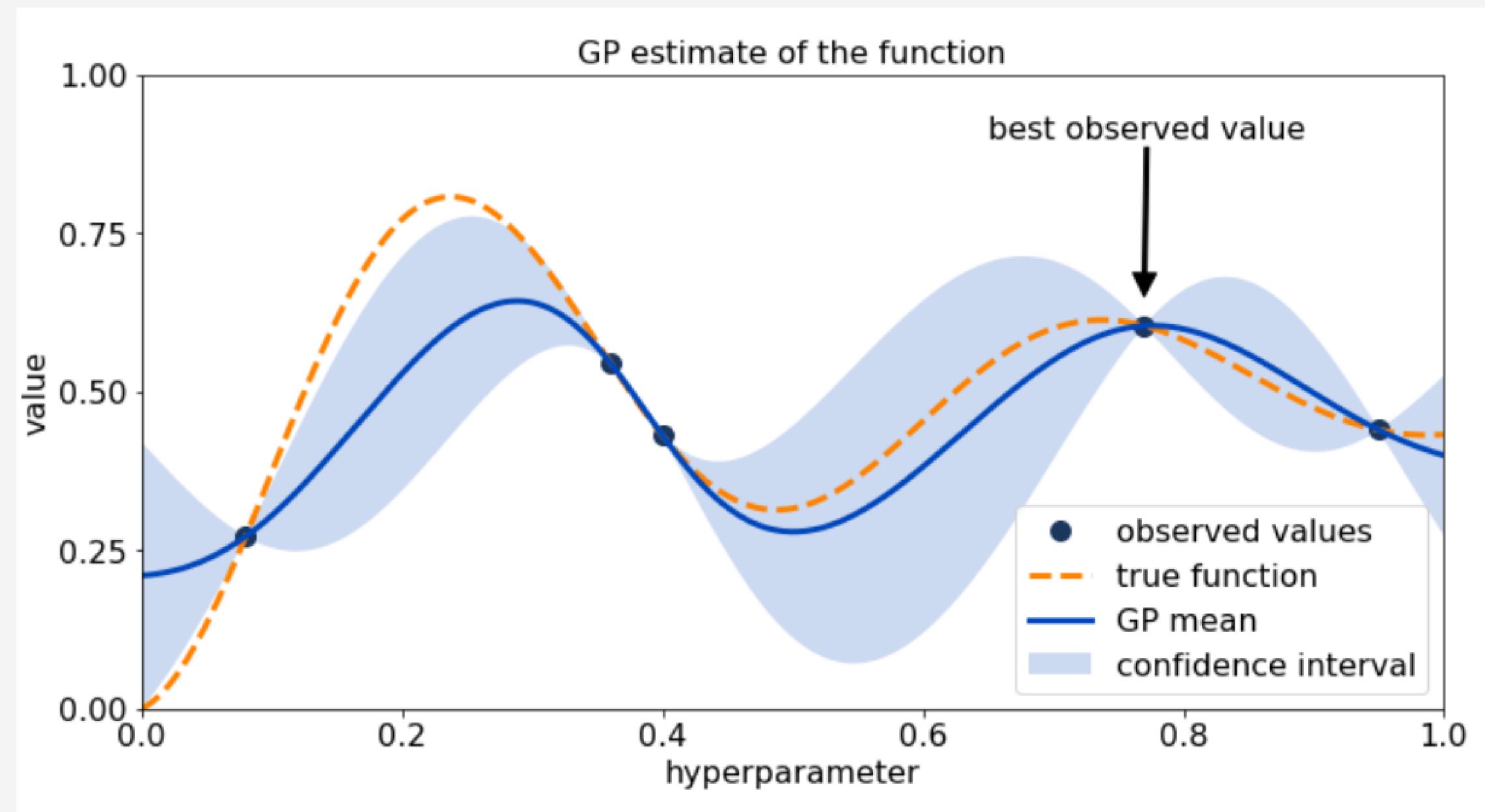
```
Hyperparameter_One = random.num(range)  
Hyperparameter_Two = random.num(range)
```



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

하이퍼파라미터 최적화 기법

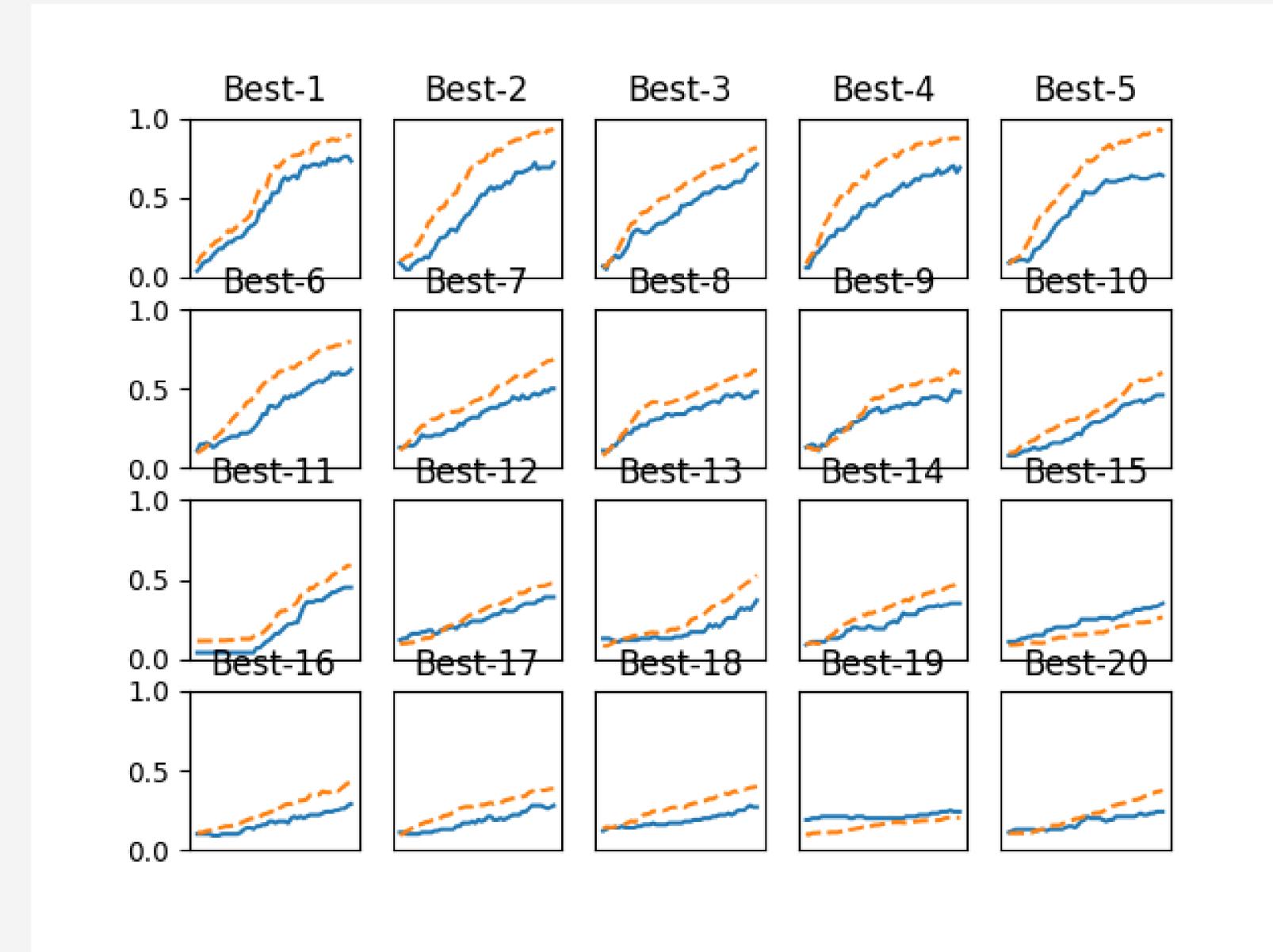
Bayesian Search



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

하이퍼파라미터 최적화 시각화

〈 MNIST 데이터 셋을 사용해서 하이퍼 파라미터를 최적화 〉



1. 배치 정규화 – 오버피팅 / 가중치 감소 / 드롭아웃
2. 하이퍼파라미터 – 종류 / 최적화 방법

참고 자료

- <밑바닥부터 시작하는 딥러닝> 6장 – 사이토 고키 저 / 개앞맵시 역 | 한빛미디어
- <https://eehoeskrap.tistory.com/430>
- <https://velog.io/@js03210/Deep-Learning-Batch-Normalization-%EB%B0%B0%EC%B9%98-%EC%A0%95%EA%B7%9C%ED%99%94>

