

Section 2. 딥러닝이란?

목차

- 섹션 0. 강의 소개
- 섹션 1. PyTorch 환경 설정
- **섹션 2. 딥러닝이란?**
- 섹션 3. 손실 함수 (Loss Function)
- 섹션 4. 손실 함수에 대한 심화 이론 (Advanced Topics on Loss Function)
- 섹션 5. 경사 하강 (Gradient Descent)
- 섹션 6. 경사 하강에 대한 심화 이론 (Advanced Topics on Gradient Descent)

Objective

학습 목표

- Deep Learning이란 무엇이고 어디에서 기원하는가?
- 어떤 문제를 풀려고 하는 것인가?
- Neural Network의 기본 구성은 어떻게, 어떻게 학습시키는가?

2-1. 딥러닝이란 뭘까?

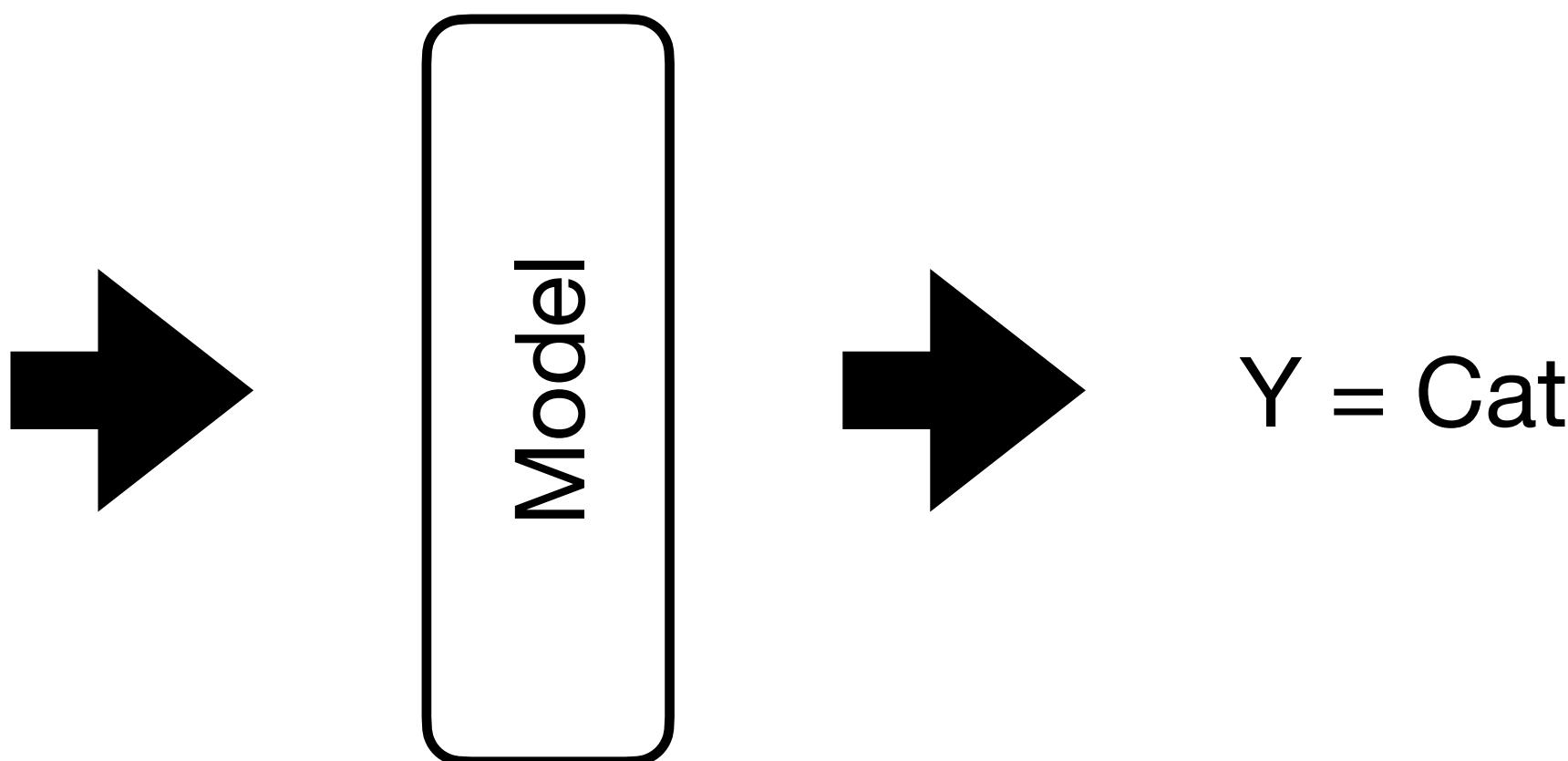
딥러닝이란?

딥러닝으로 어떤 문제를 풀 수 있을까?



Input Data X

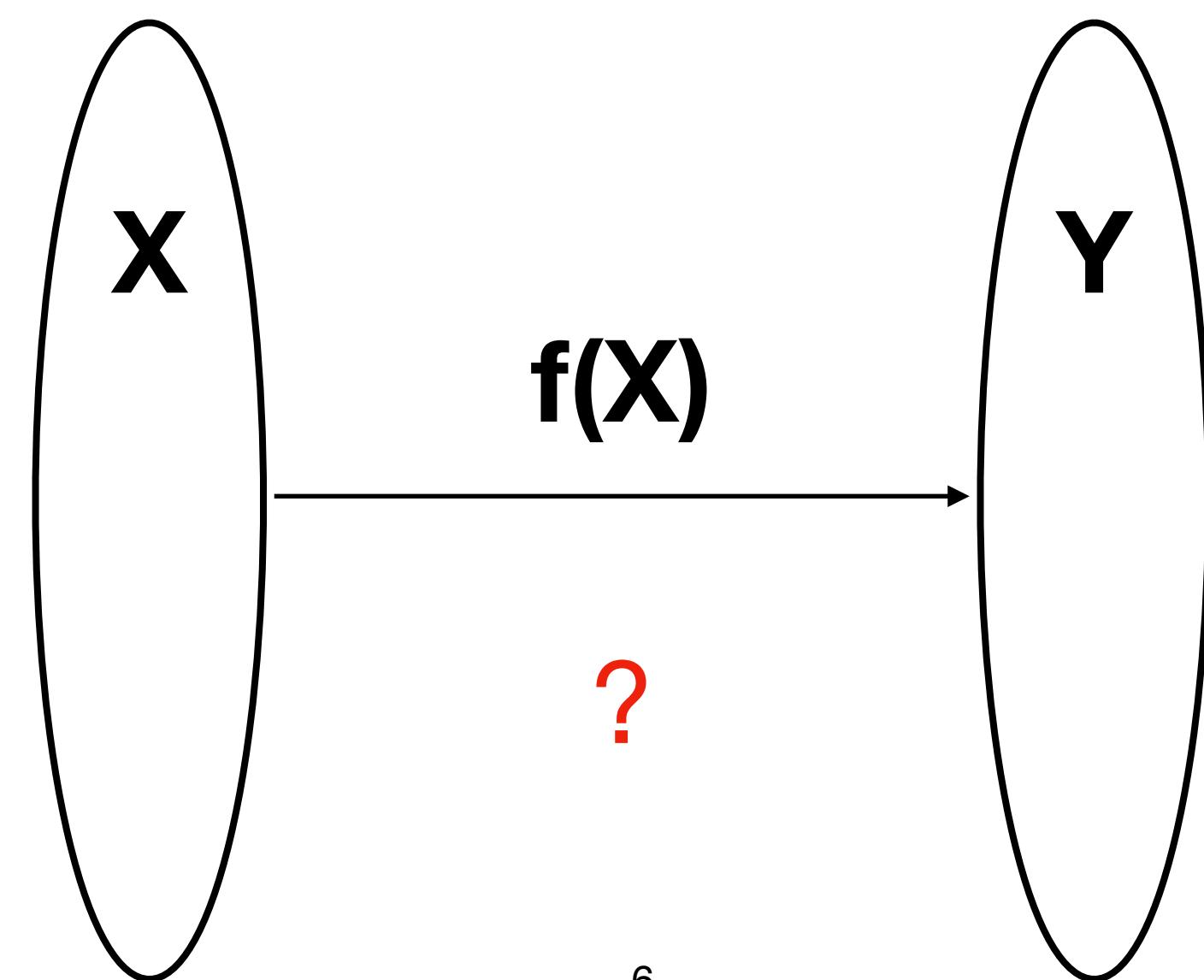
출처: [dreamstime.com](https://www.dreamstime.com)



딥러닝이란?

딥러닝으로 어떤 문제를 풀 수 있을까?

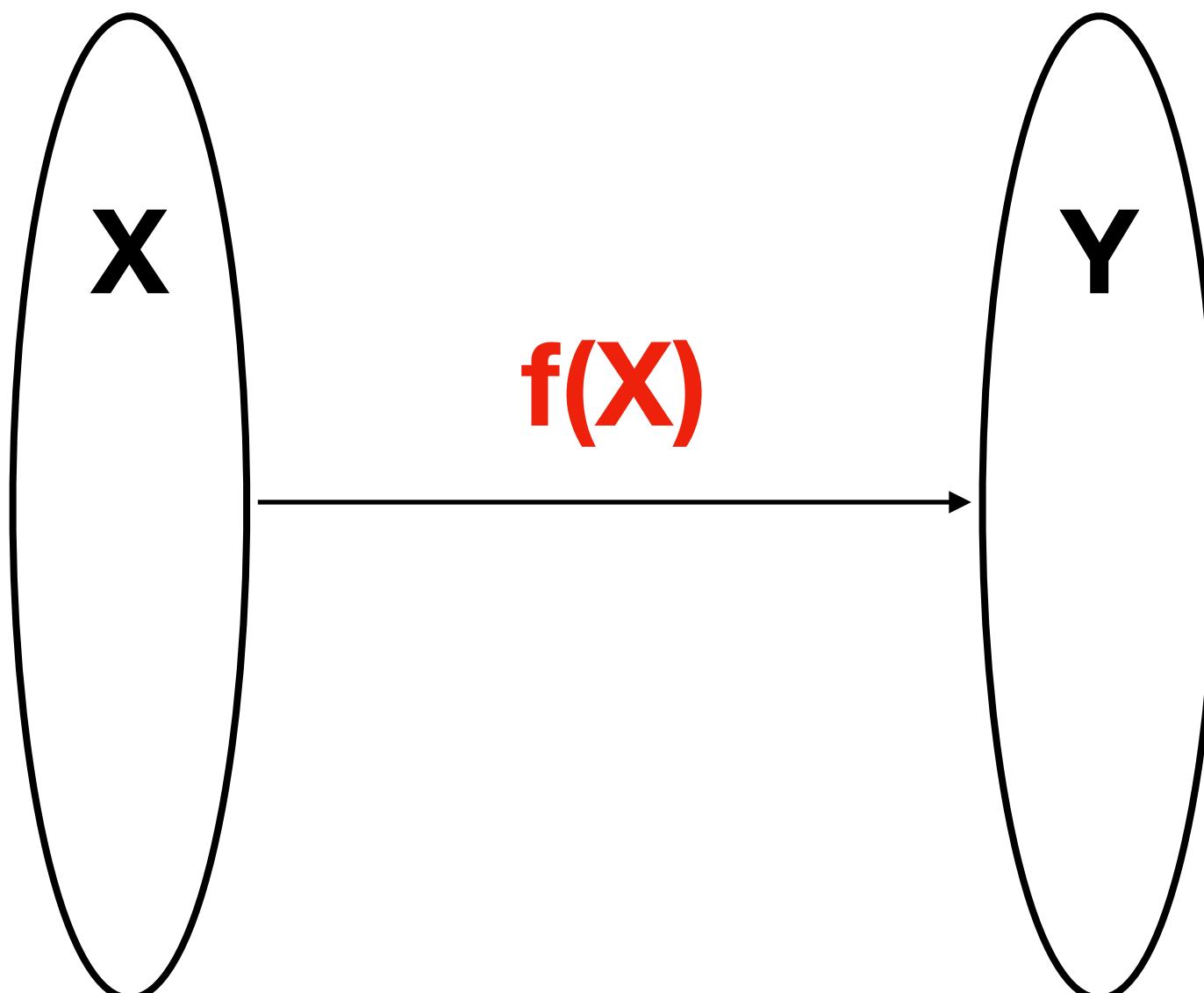
- 입력값 X 가 주어졌을 때, Label Y 의 값은 $Y = f(X)$ 을 따른다.
- $X \rightarrow Y$ 로 mapping 해주는 함수 f 을 어떻게 구할까?



딥러닝이란?

딥러닝으로 어떤 문제를 풀 수 있을까?

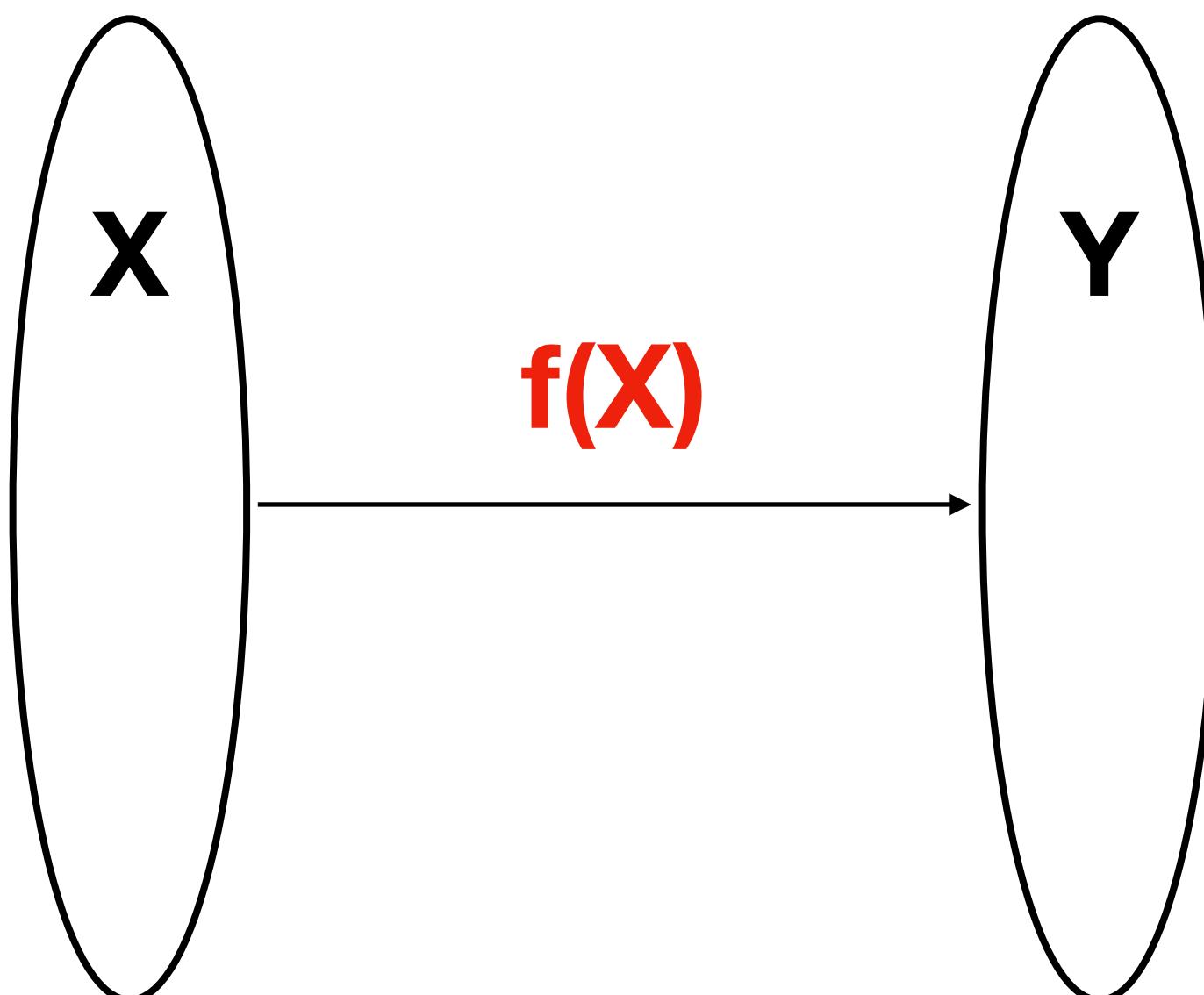
- $X \rightarrow Y$ 로 mapping 해주는 함수 $f(X)$ 을 어떻게 구할까?
- 정확한 $f(X)$ 을 구하기 어려울 수 있다.



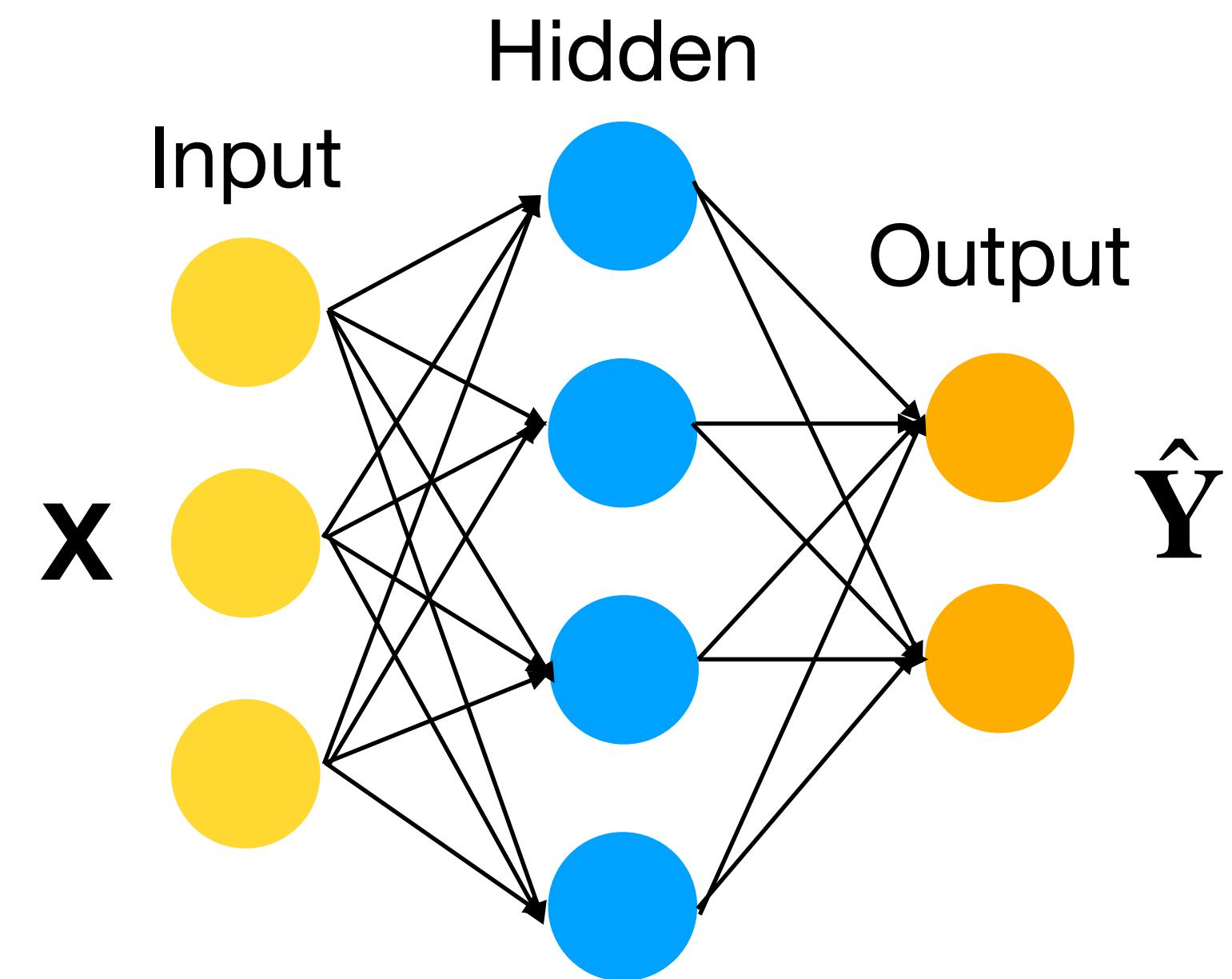
딥러닝이란?

딥러닝으로 어떤 문제를 풀 수 있을까?

- $X \rightarrow Y$ 로 mapping 해주는 함수 $f(X)$ 을 어떻게 구할까?
- 정확한 $f(X)$ 을 구하기 어려울 수 있다.
- $f(X)$ 에 “근사” (approximate) 하는 함수를 Neural Network $\hat{f}(X)$ 으로 모델링 $\hat{Y} = NN(X)$



$$f(X) \approx \hat{f}(X)$$



딥러닝이란?

딥러닝으로 어떤 문제를 풀 수 있을까?

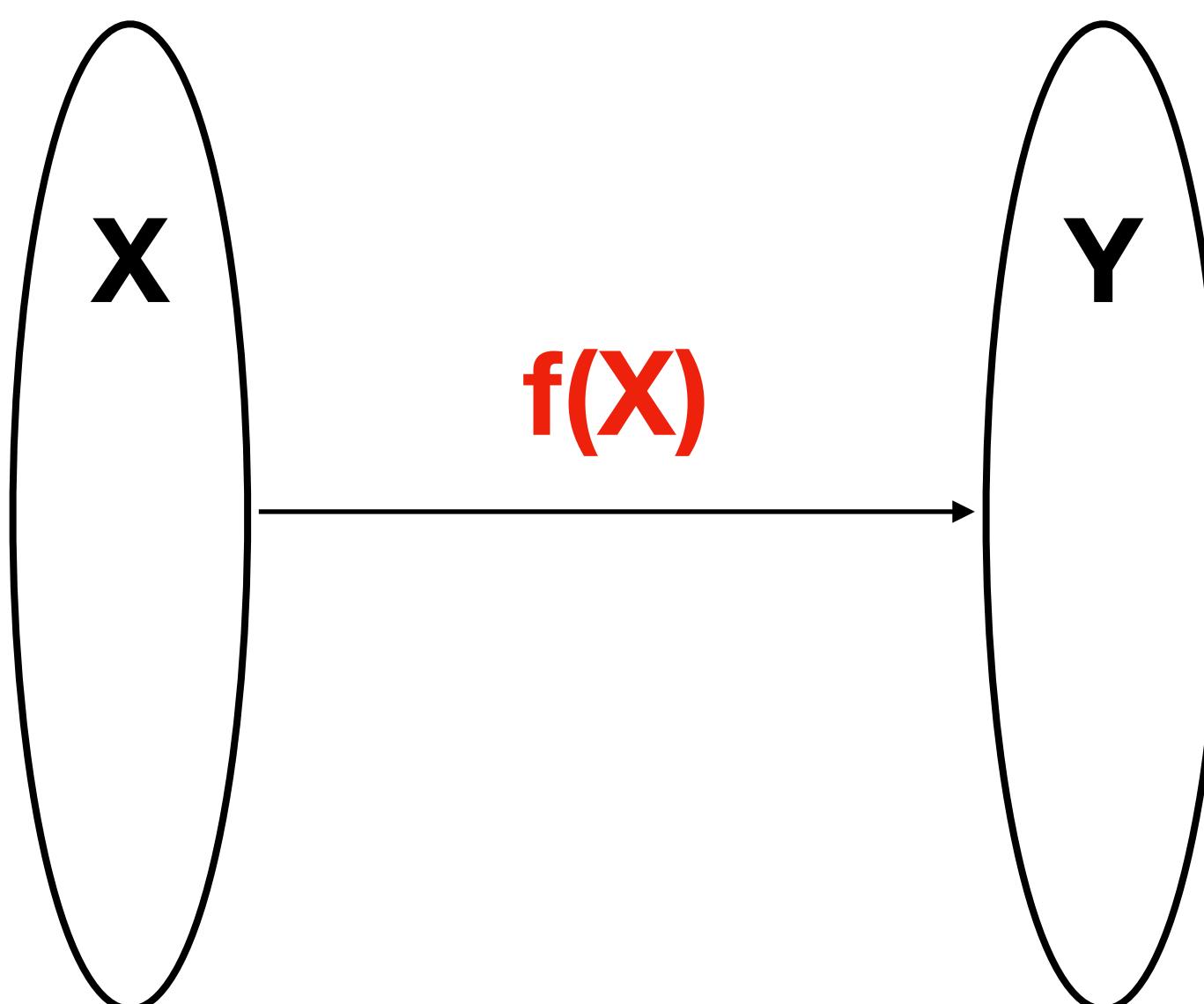
$\hat{Y} = NN(X)$ 가

Neural Network의 출력값

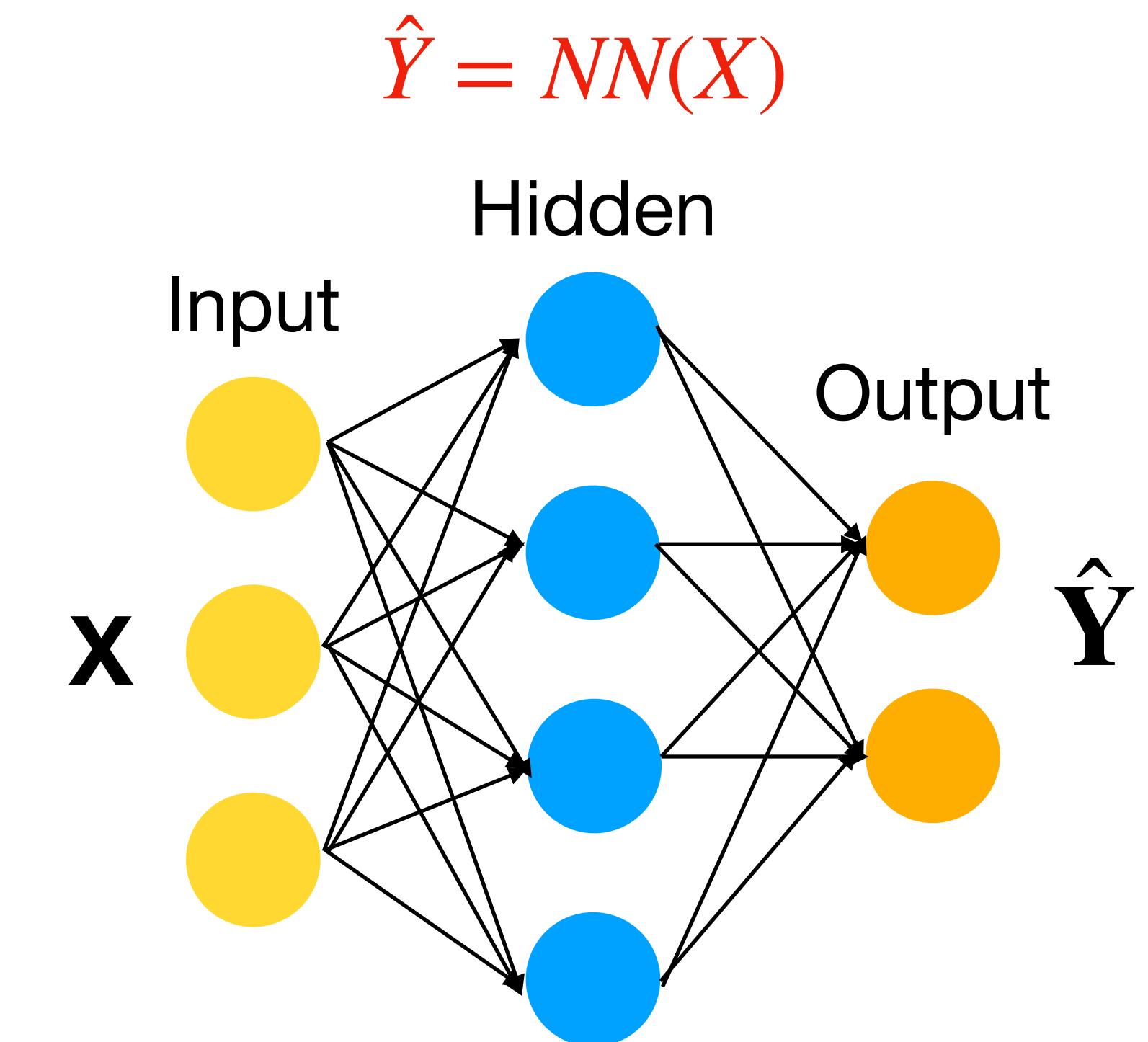
$Y = f(X)$ 에

실제값

최대한 유사하도록 하는 것.



$$f(\mathbf{X}) \approx \hat{f}(\mathbf{X})$$



바로 뉴럴넷을 학습한다는 의미! All rights reserved.

딥러닝이란?

딥러닝으로 어떤 문제를 풀 수 있을까?

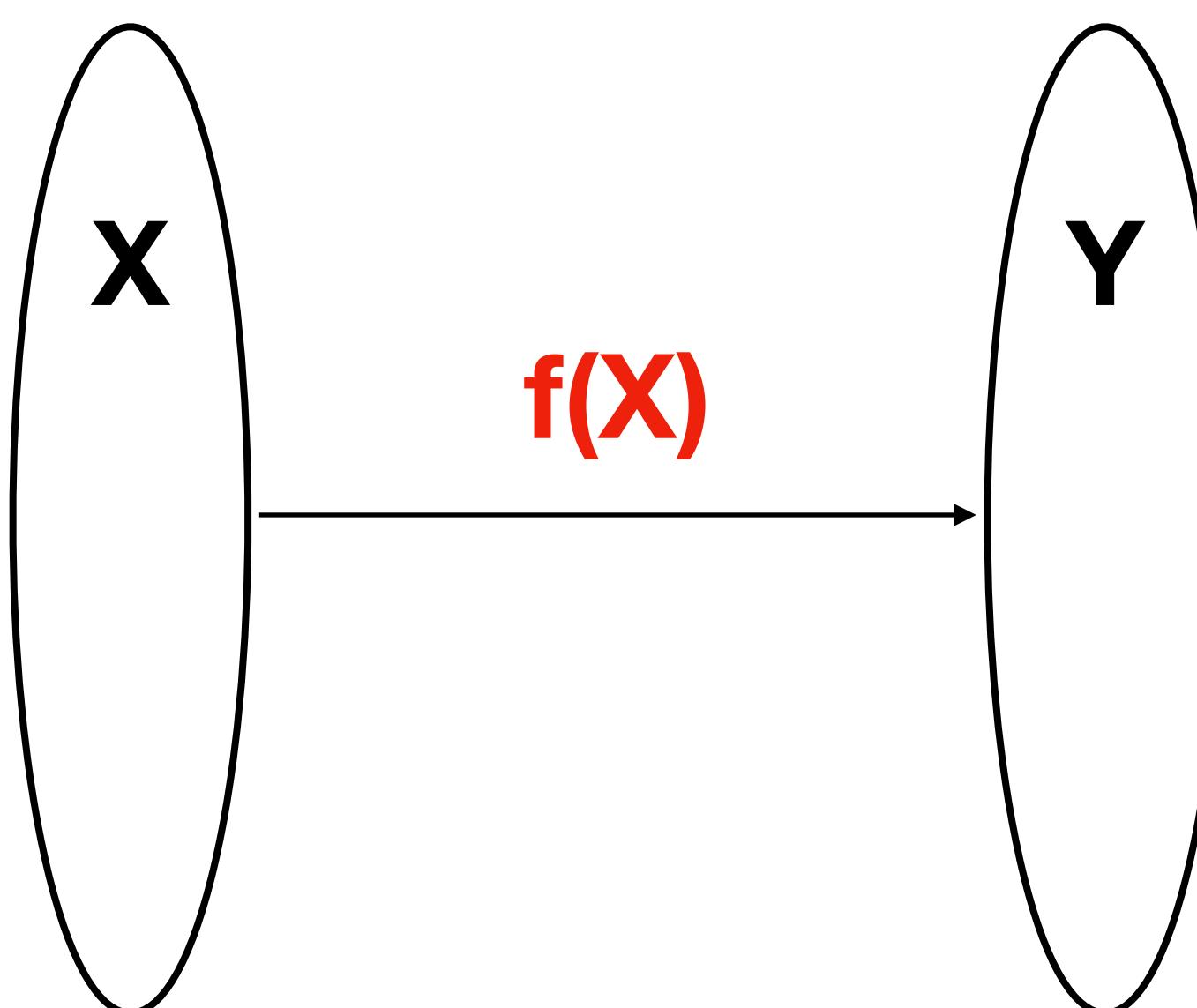
$\hat{Y} = NN(X)$ 가

Neural Network의 출력값

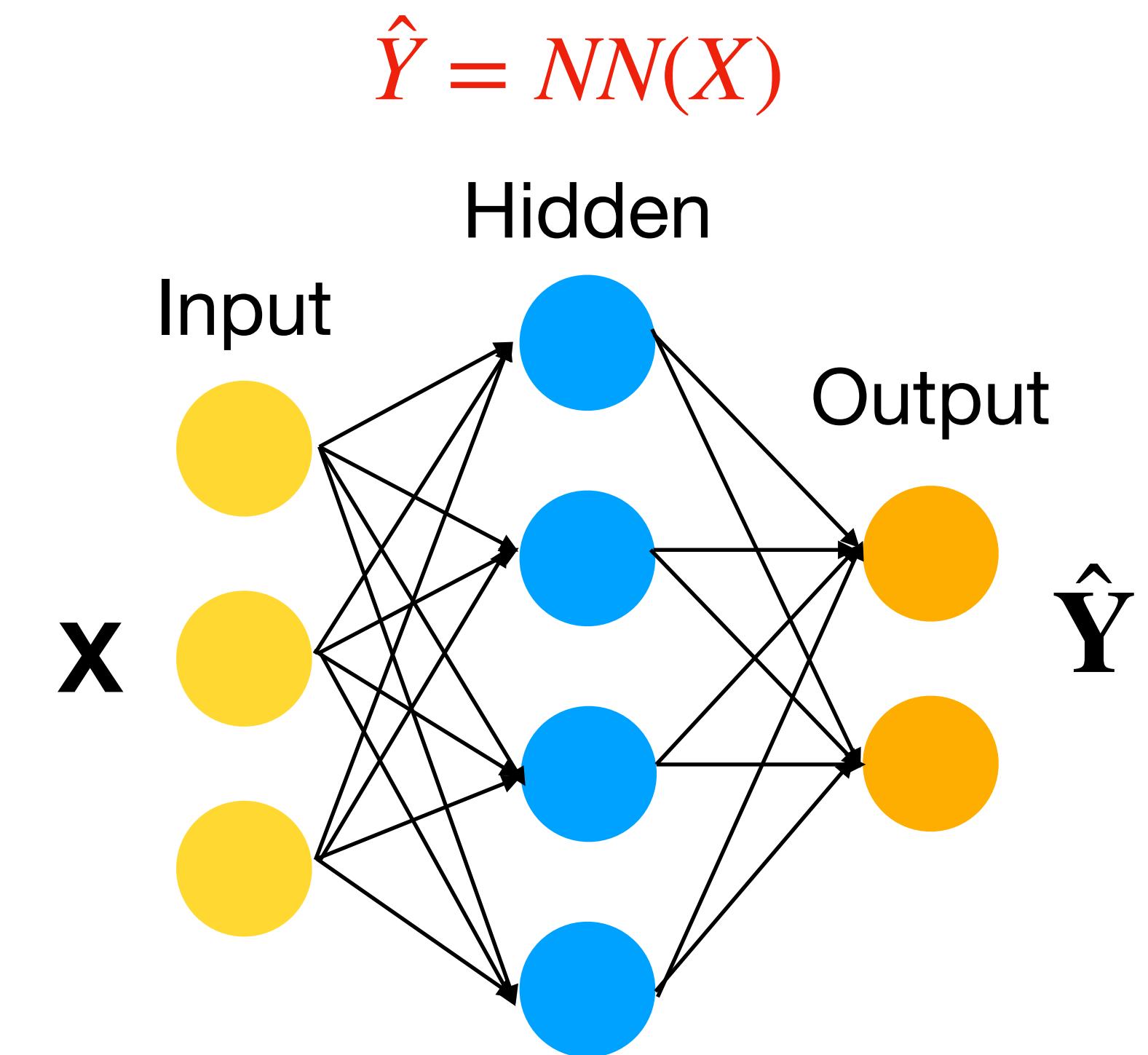
$Y = f(X)$ 에

실제값

최대한 유사하도록 하는 것.



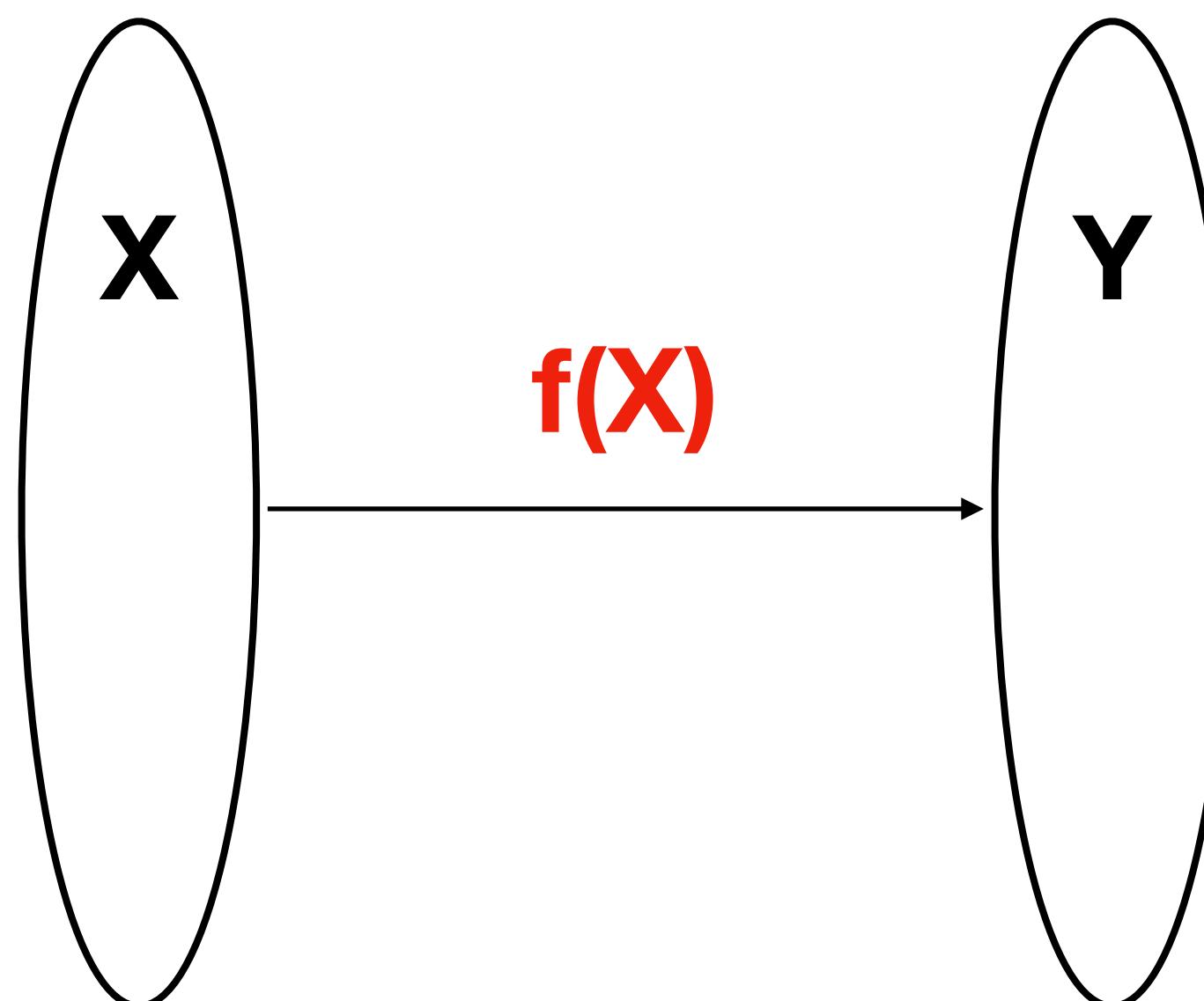
$$f(\mathbf{X}) \approx \hat{f}(\mathbf{X})$$



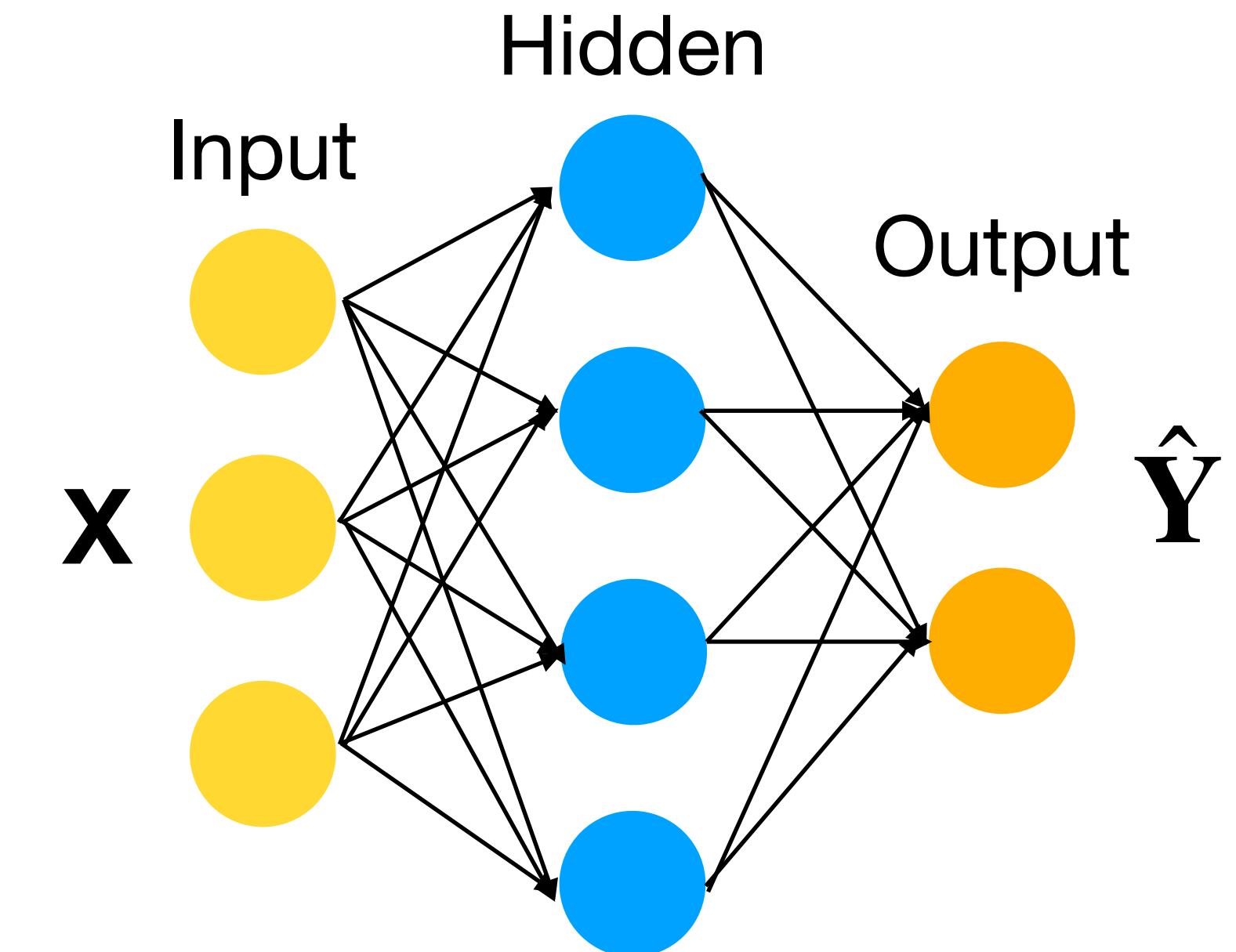
딥러닝이란?

딥러닝으로 어떤 문제를 풀 수 있을까?

- 딥러닝은 Deep Neural Network을 학습시키는 것.
- Deep Neural Network은 하나 이상의 Hidden Layer들로 구성된 Neural Network이다!
- “학습” = 뉴럴넷의 출력값이 $\hat{Y} = NN(X)$ 가 실제값 $Y = f(X)$ 에 더더욱 잘 근사하도록 하는 것이다!



$$f(\mathbf{X}) \approx \hat{f}(\mathbf{X})$$

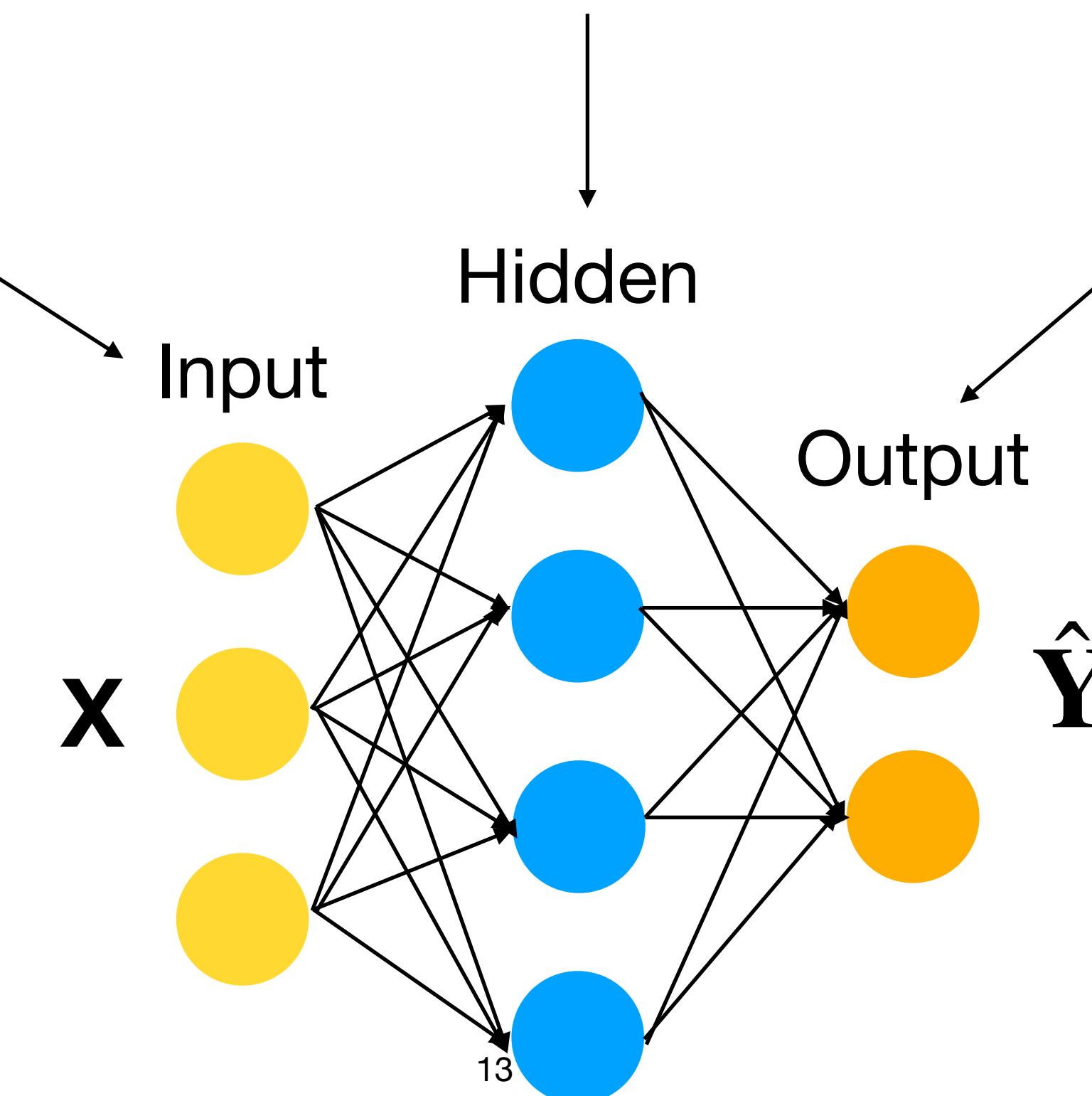


2-2. 뉴럴넷 (Neural Network)은 뭘까?

Neural Network

기본적인 Neural Network의 구성은:

Input Layer + (한 개 이상의) Hidden Layer + Output Layer

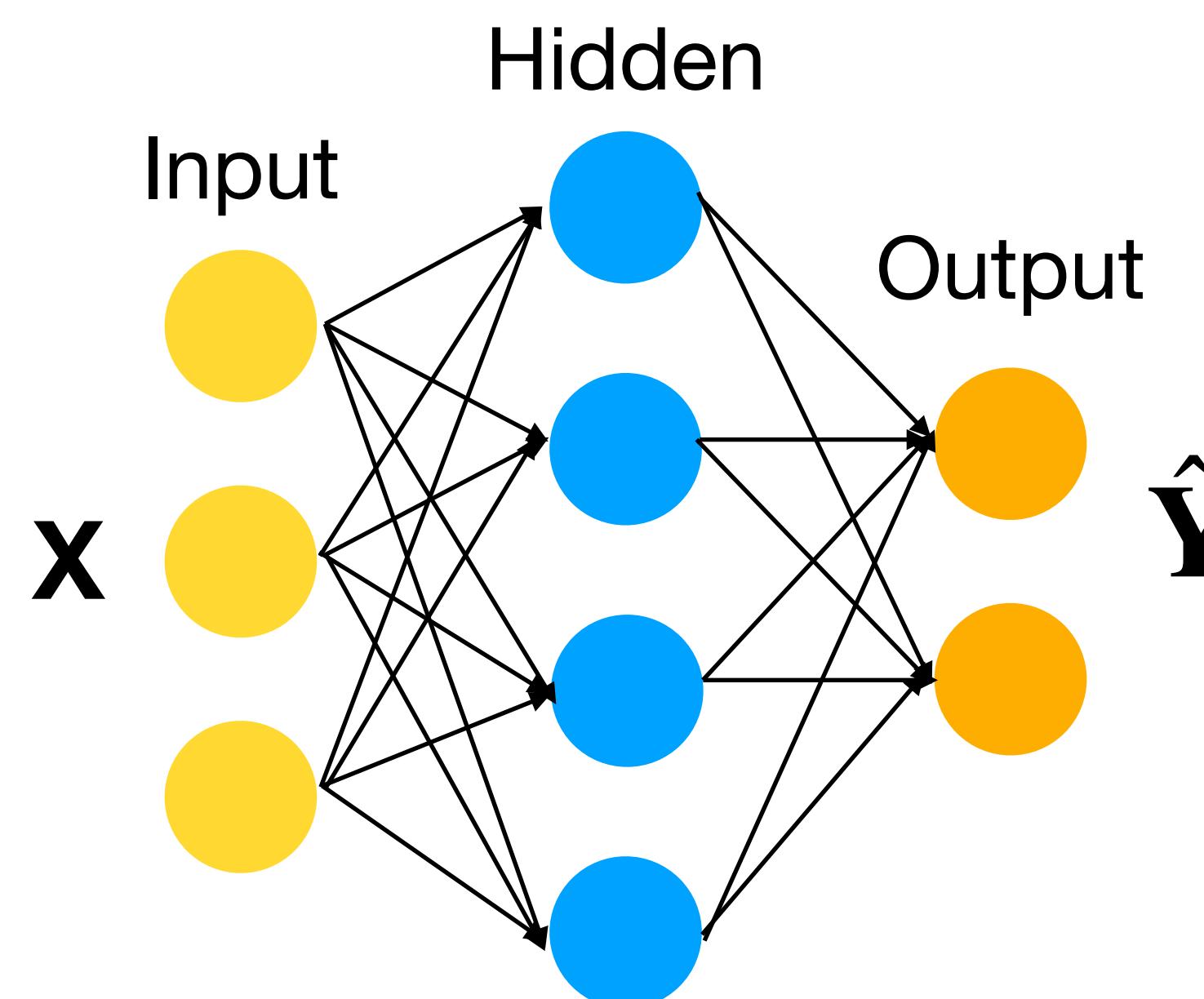


Neural Network

Copyright©2023. Acadential. All rights reserved.

기본적인 Neural Network의 구성은:

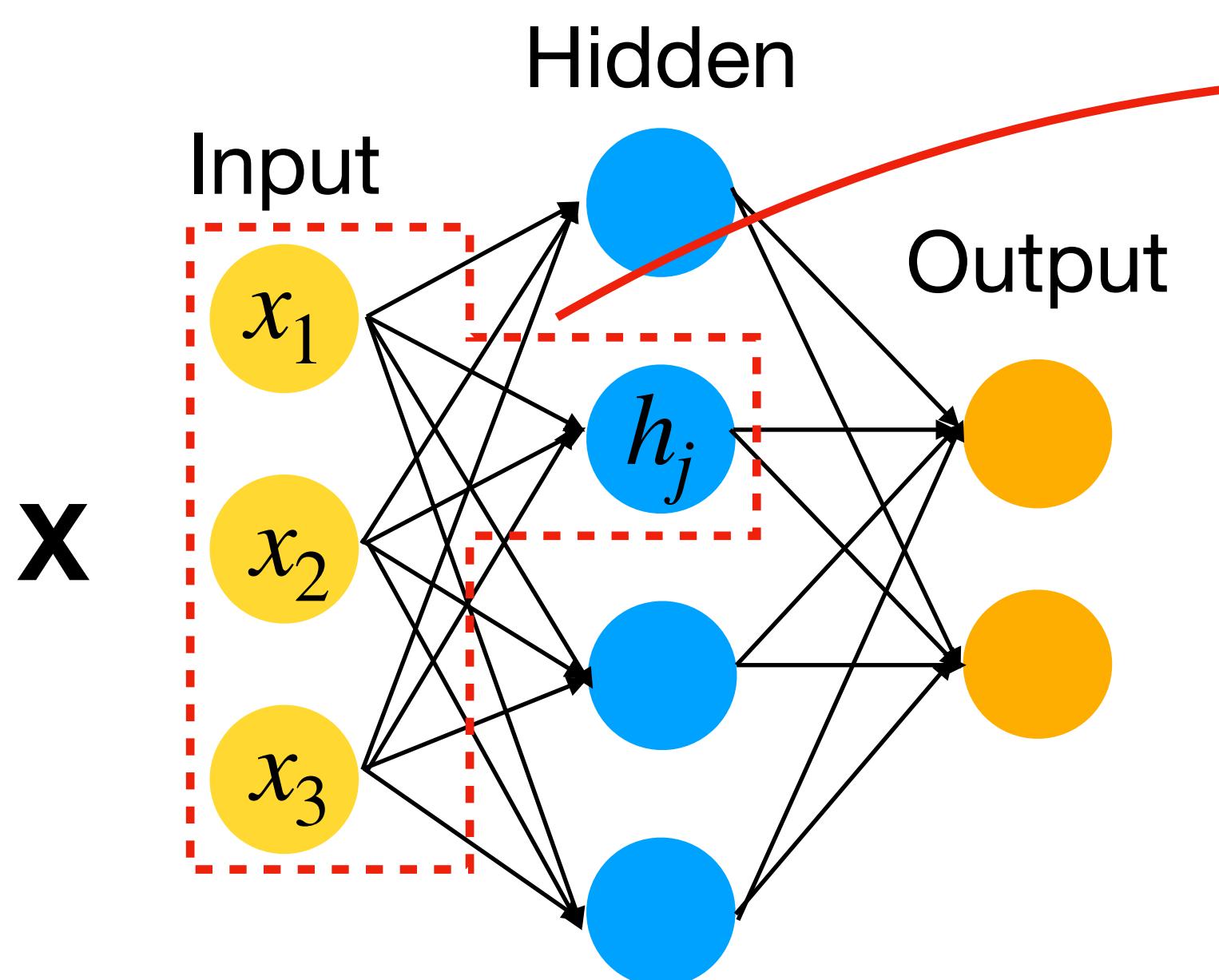
Input Layer + (한 개 이상의) Hidden Layer + Output Layer



- 각 Layer은 뉴런 (Neuron) 들로 구성된다.
- 각 뉴런은 다음 Layer의 뉴런과 연결 (edge) 되어 있다.
- 즉, 뉴론은 이전 Layer의 뉴론들의 출력값을 입력값으로 받는다.
- 각 뉴런은 가중치 weight w 와 활성 함수 activation function으로 구성된다.

딥러닝이란?

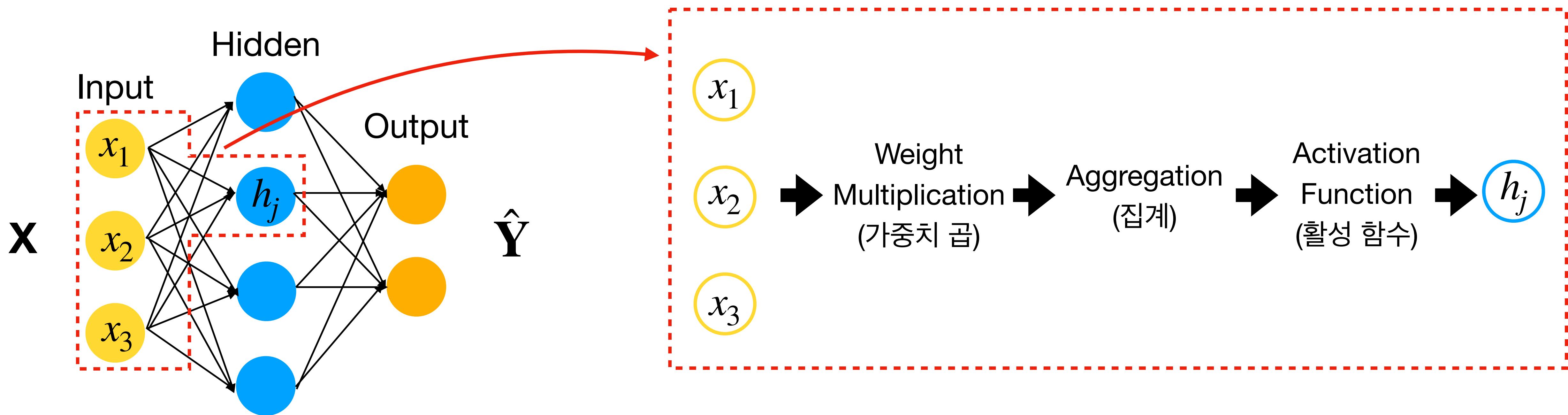
Deep Neural Network란?



- x_1, x_2, x_3
 - 뉴럴넷에 입력되는 값
 - Input Layer의 1, 2, 3번째 뉴론이 출력하는 값
 - 다음 Layer의 Input 값으로 전달
- h_j
 - Hidden Layer의 j번째 뉴론이 출력하는 값
 - 이전 Layer인 Input Layer의 출력값을 입력값으로 사용하여 계산됨
 - 다음 Layer의 Input 값으로 전달

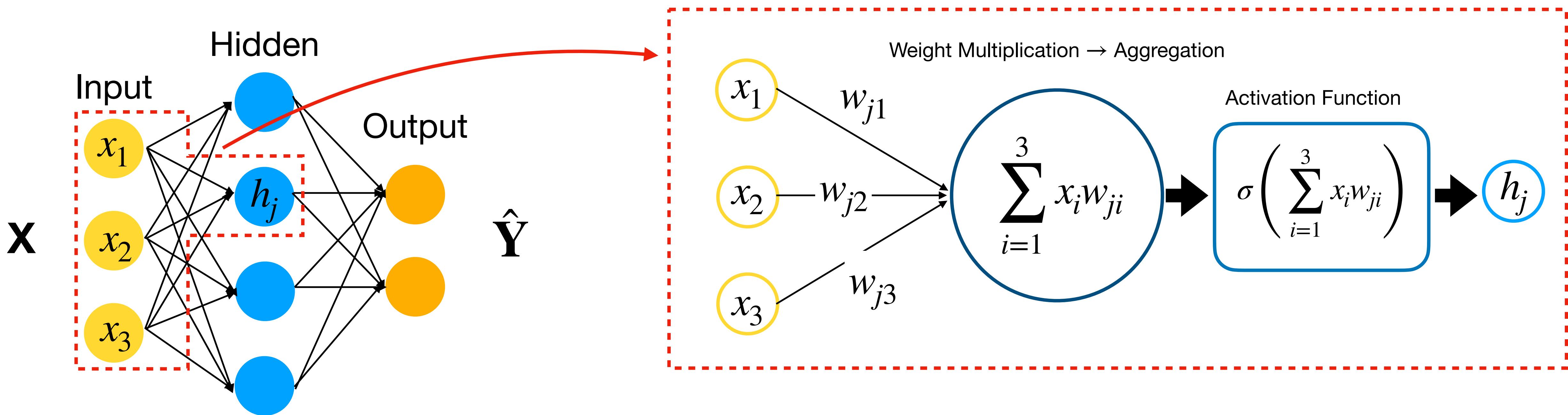
딥러닝이란?

Deep Neural Network란?



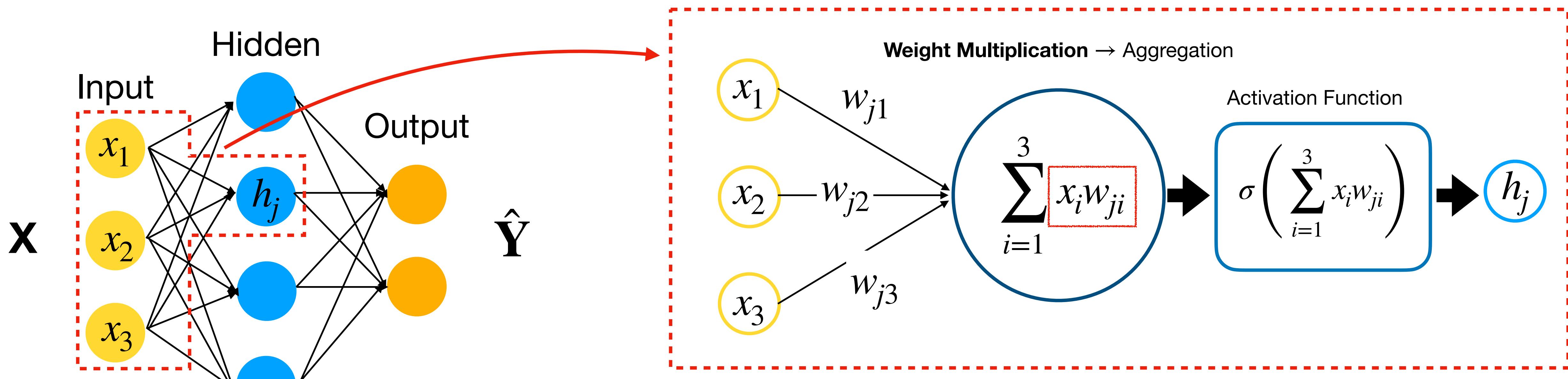
딥러닝이란?

Deep Neural Network란?



딥러닝이란?

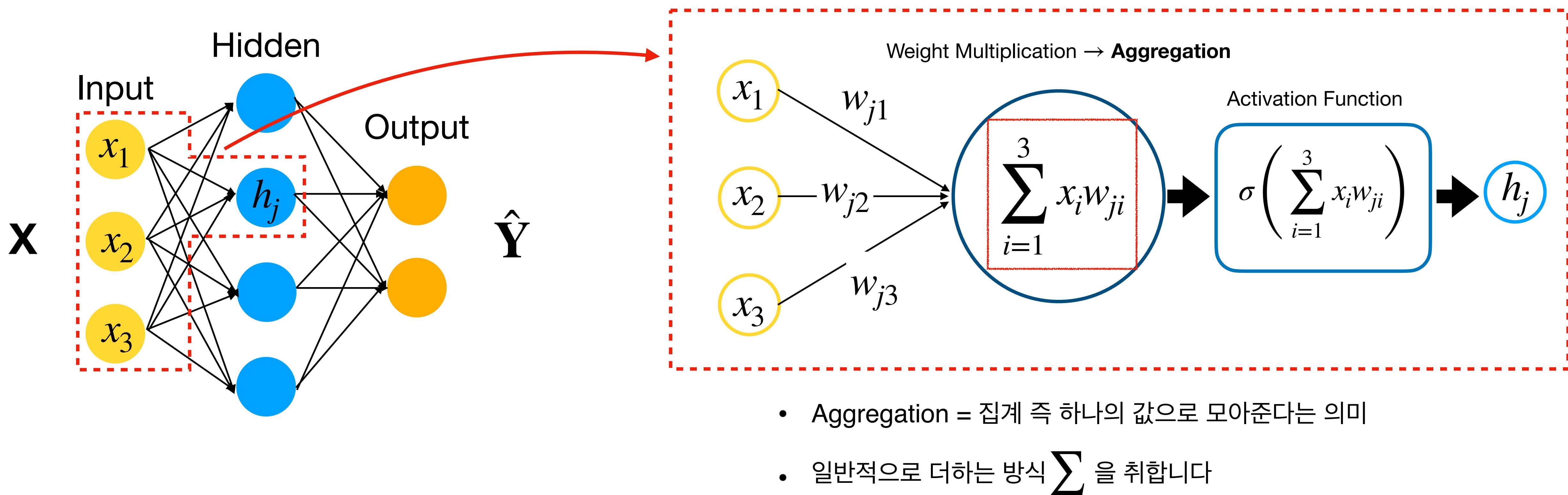
Deep Neural Network란?



- w_{j1} = input layer의 첫 번째 뉴론과 hidden layer의 j 번째 뉴론간의 가중치
- 각 가중치가 연결되어 있는 입력값들과 곱해진다

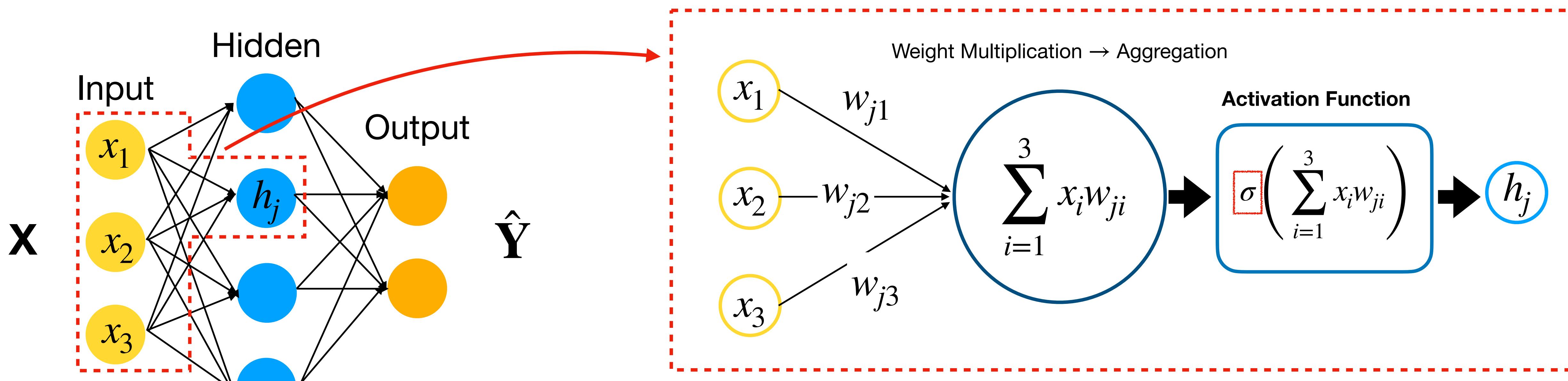
딥러닝이란?

Deep Neural Network란?



딥러닝이란?

Deep Neural Network란?



- Activation Function = 비선형 활성 함수
- e.g. Sigmoid, Tanh, ReLU, SoftMax

딥러닝이란?

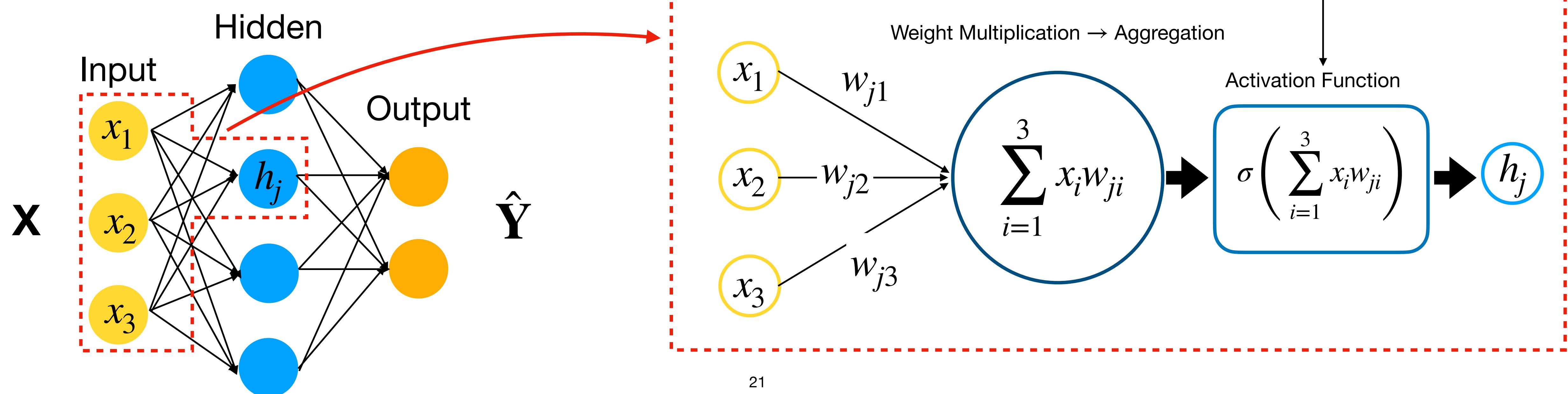
Deep Neural Network란?

- 각 뉴론은 Weight, Activation Function으로 구성되어 있다.

- $\sum_{i=1}^3 x_i w_{ji}$: 이전 Layer의 출력값 x_i 은 가중치 w_{ji} 에 곱해져서 합해진다. (weight multiplication → aggregation)

- $\sigma(\cdot)$: 합해진 값들은 activation function σ 을 통하여 해당 뉴런의 최종 출력값 h_j 을 구한다.

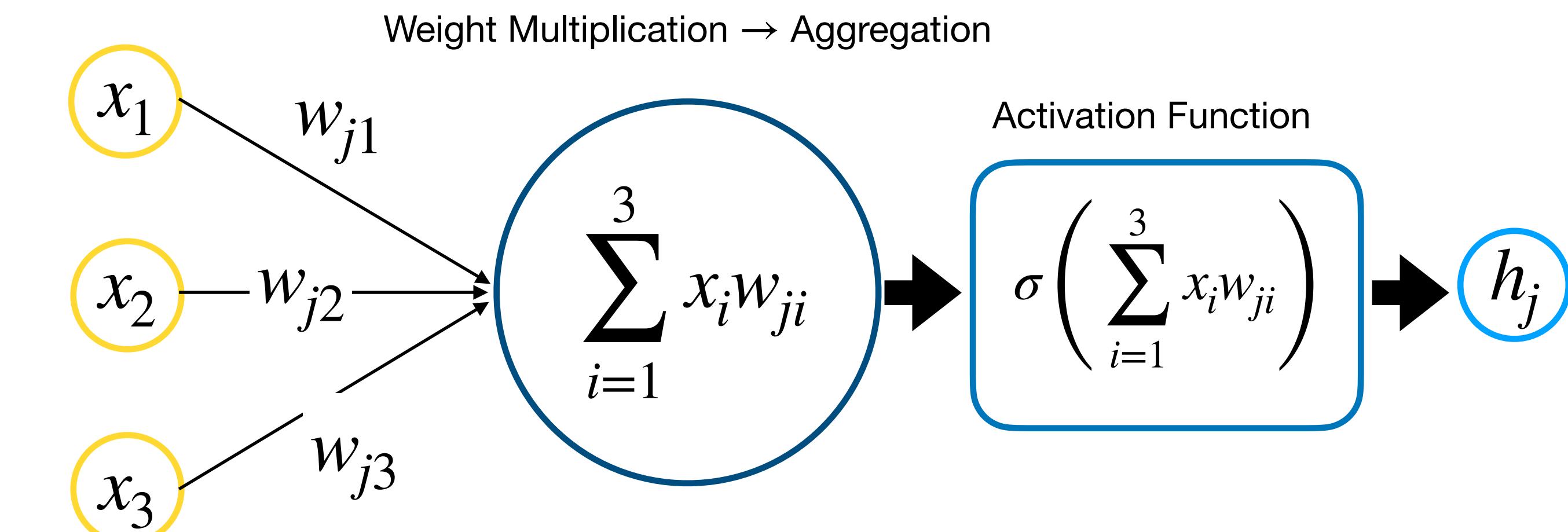
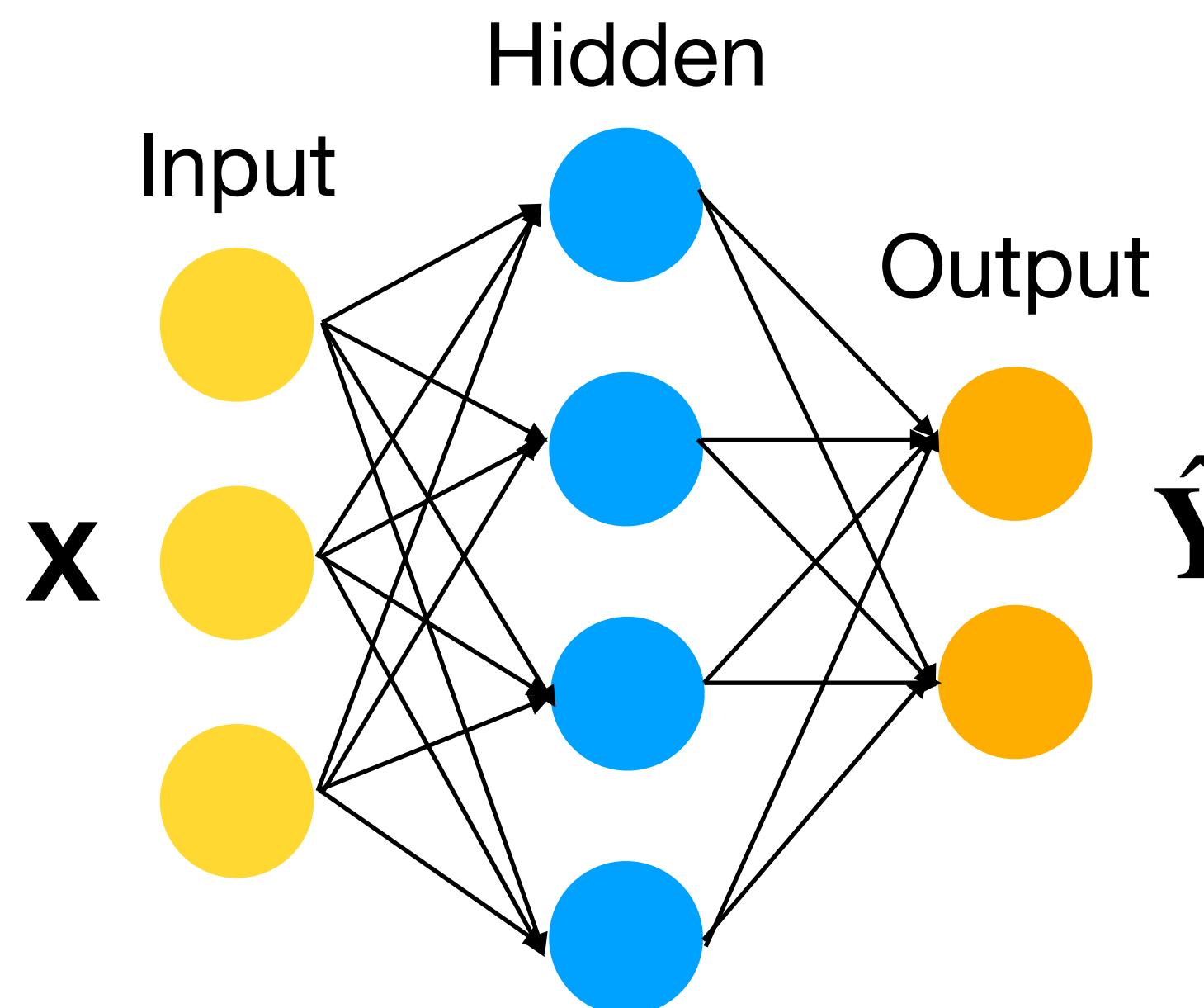
Activation Function에서 자세히 다를 예정!



Neural Network

Copyright©2023. Acadential. All rights reserved.

- 그렇다면 우리는 왜 이것을 Neural Network라고 부르는 것일까?
- Neural Network가 파생된 기원이 있는걸까?

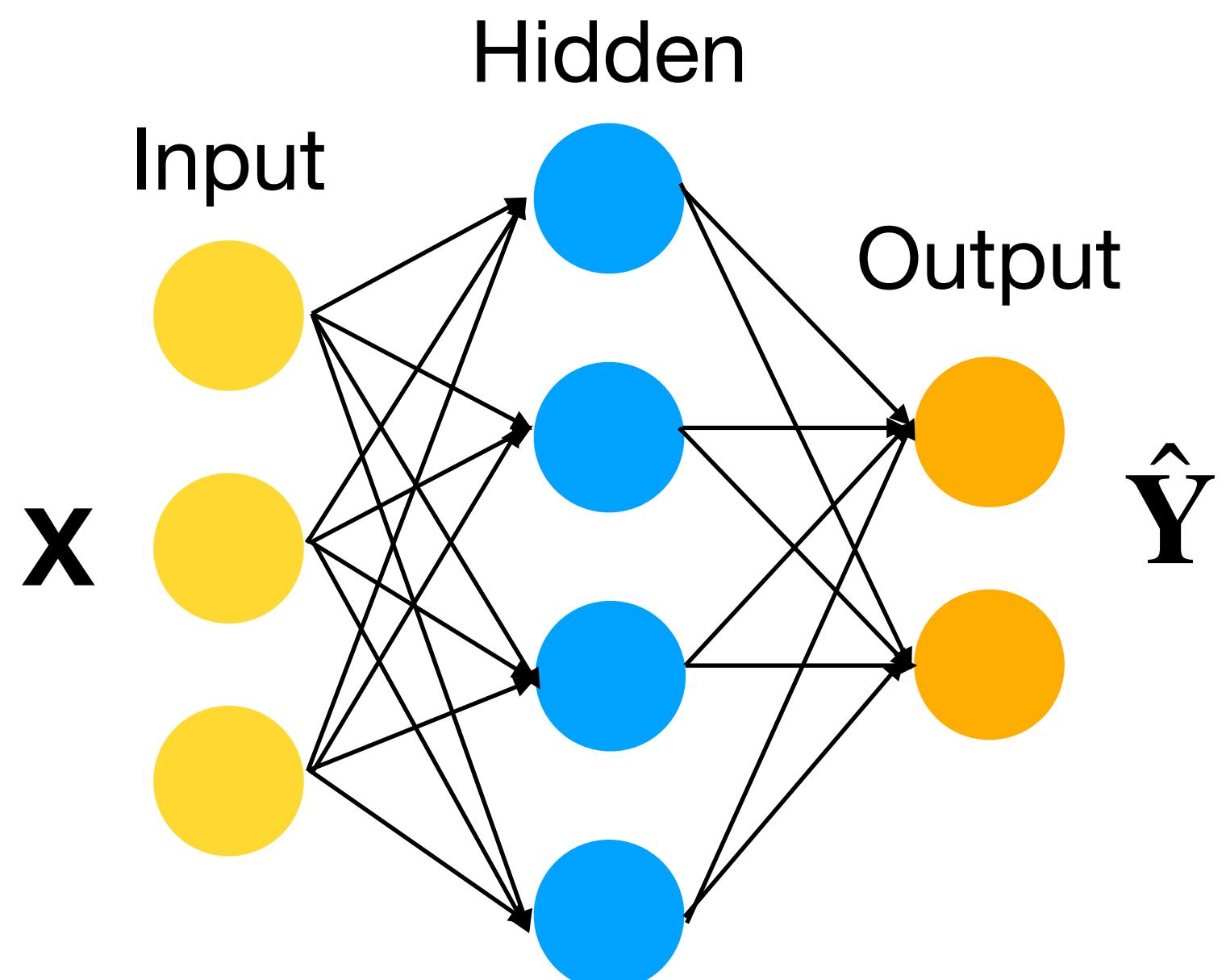


Neural Network의 기원

Neural Network

Neural Network의 기원

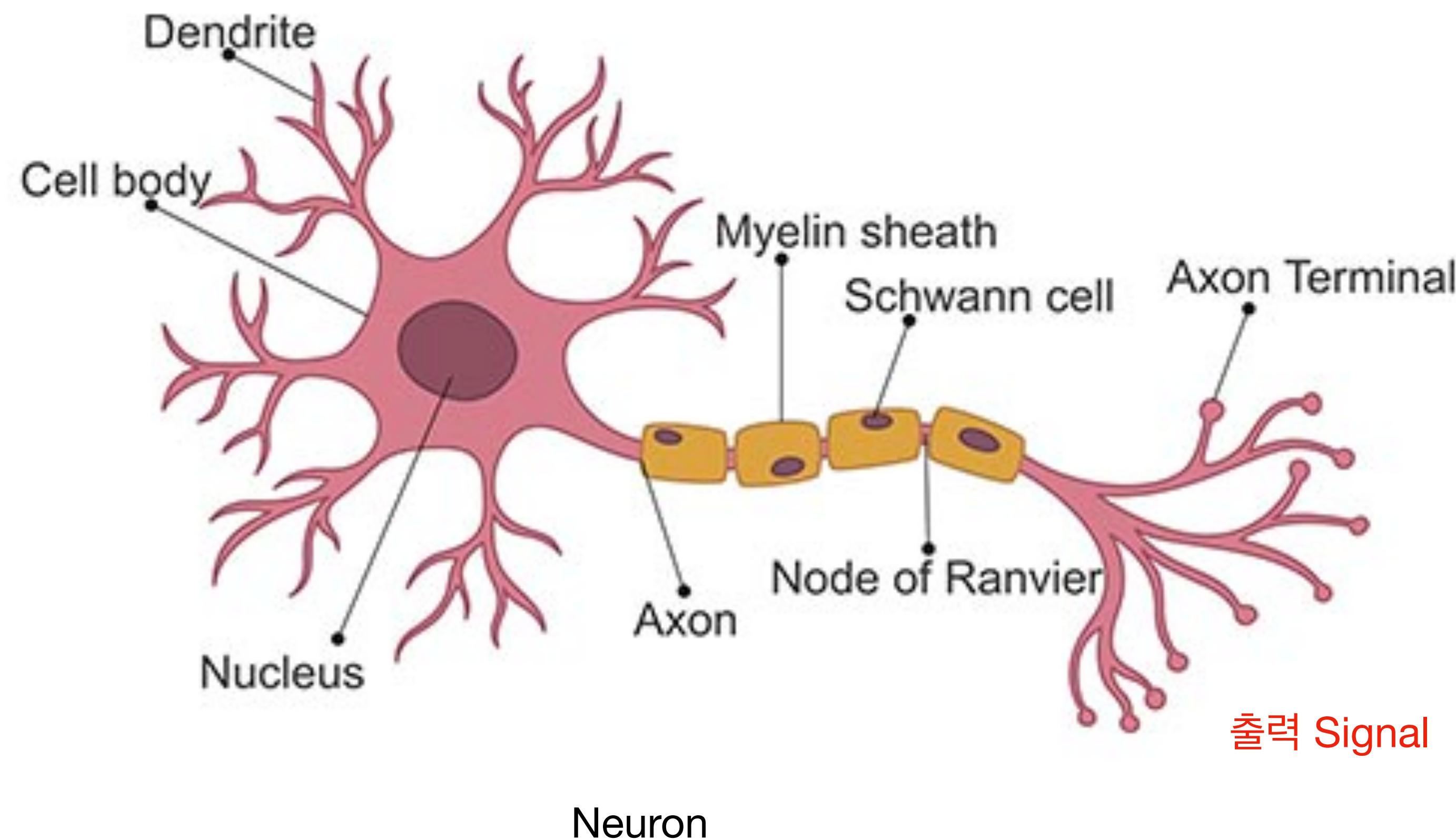
- 왜 Neural Network라고 부르는 것일까?
- 기원: 뇌의 뉴론 (Neuron)



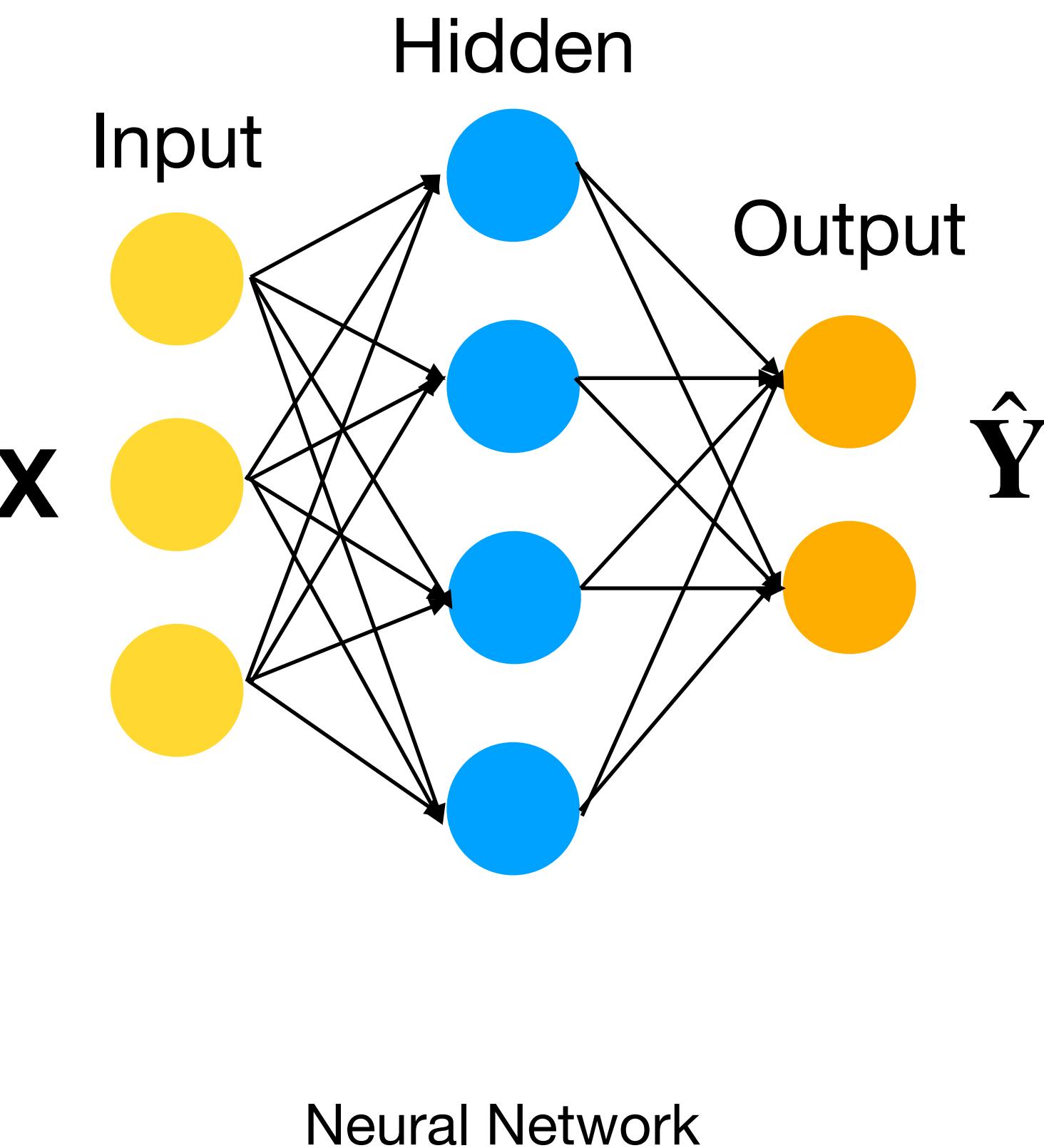
Neural Network

Neural Network의 기원: 노의 Neuron

입력 Signal



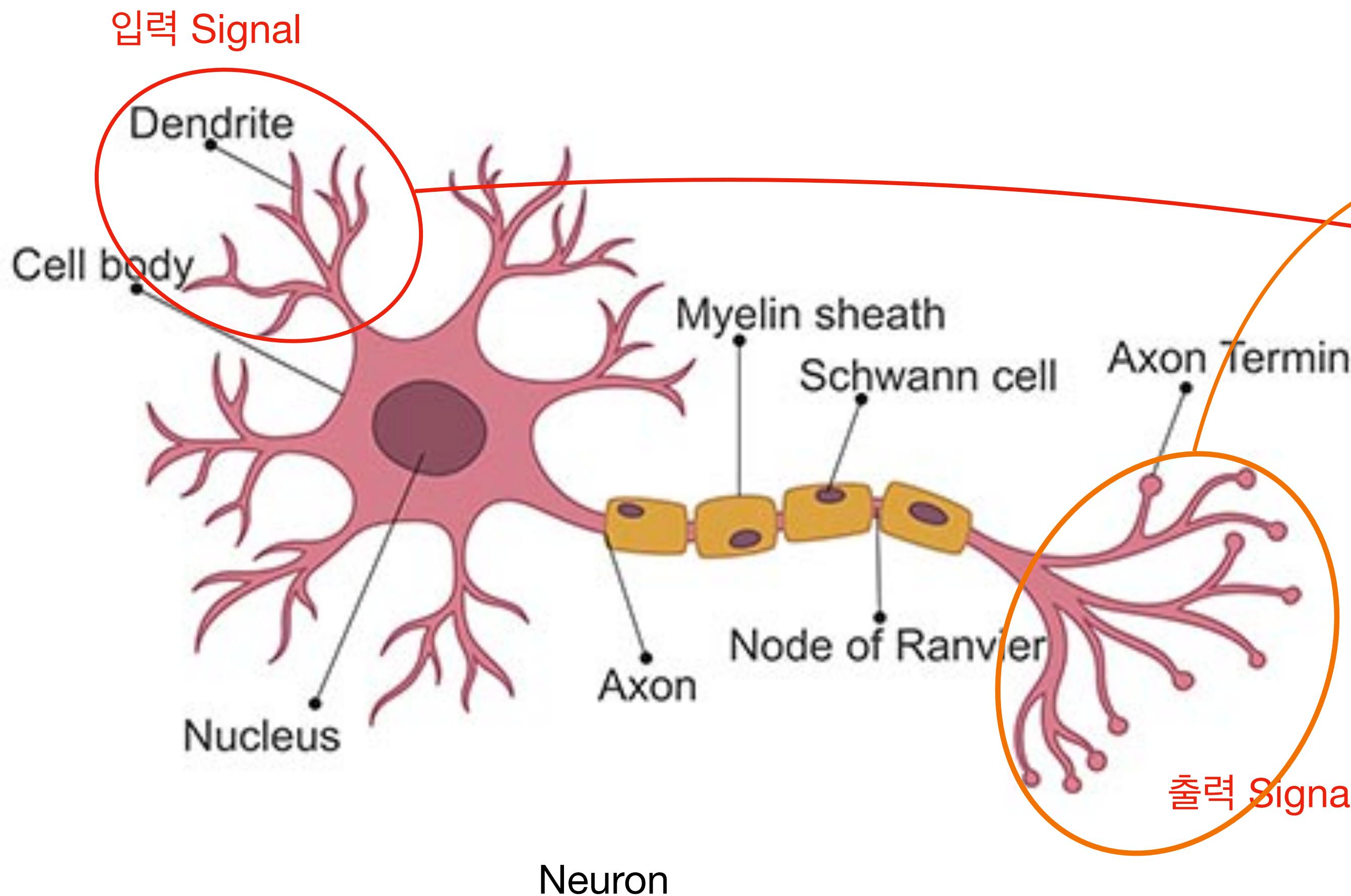
출력 Signal



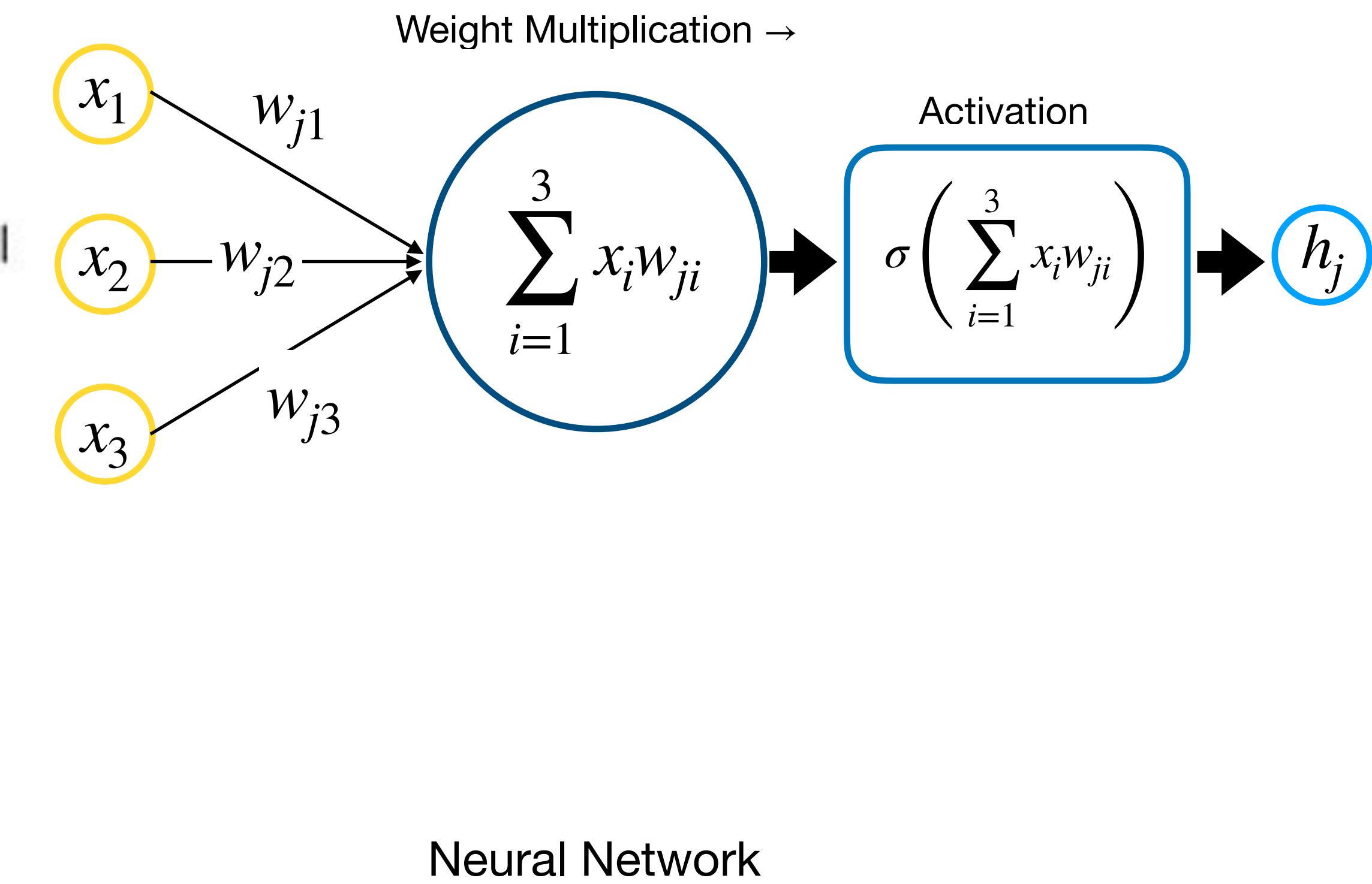
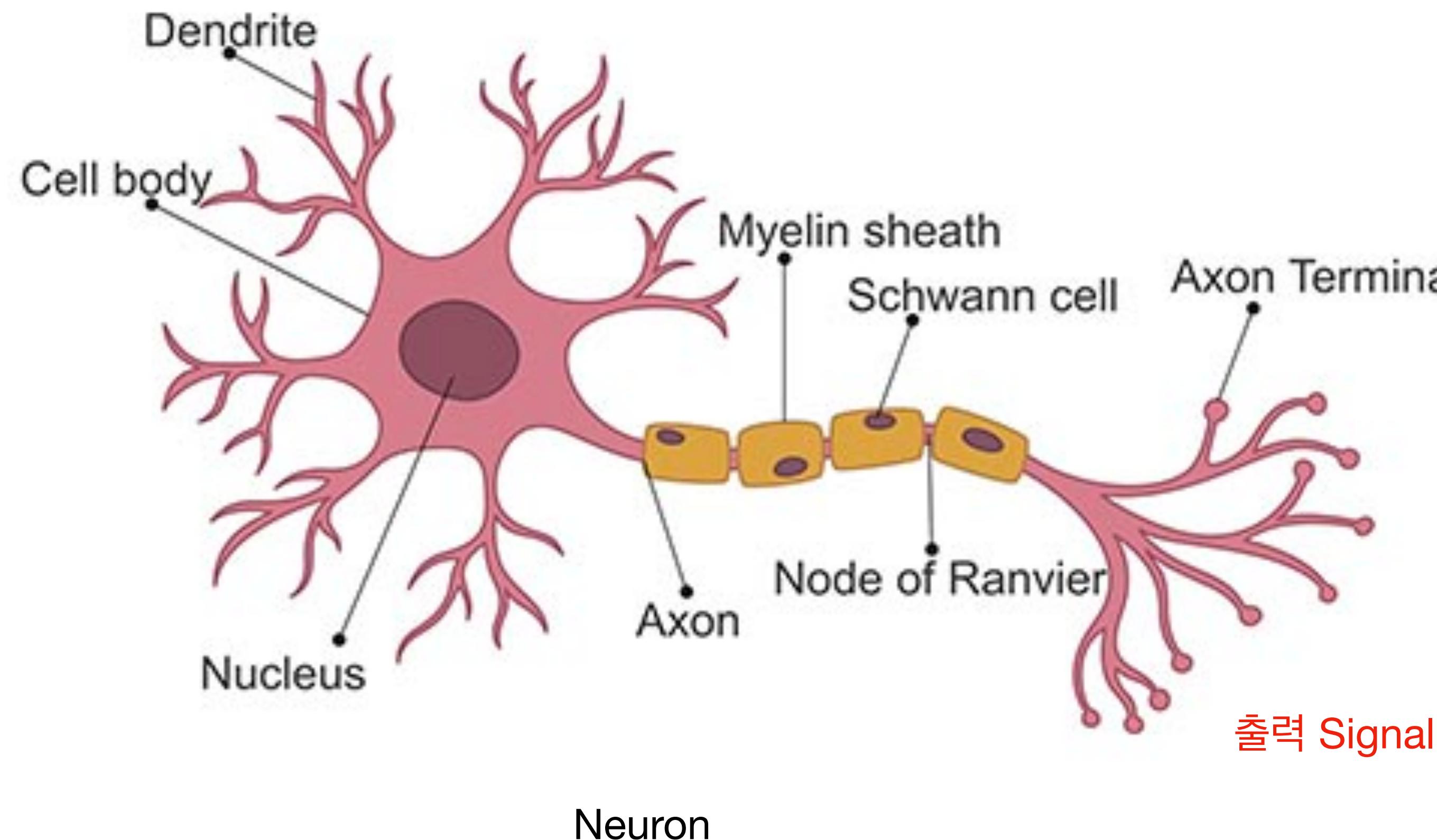
Neural Network

Neural Network

Neural Network의 기원: 노의 Neuron



Neural Network의 기원: 노의 Neuron

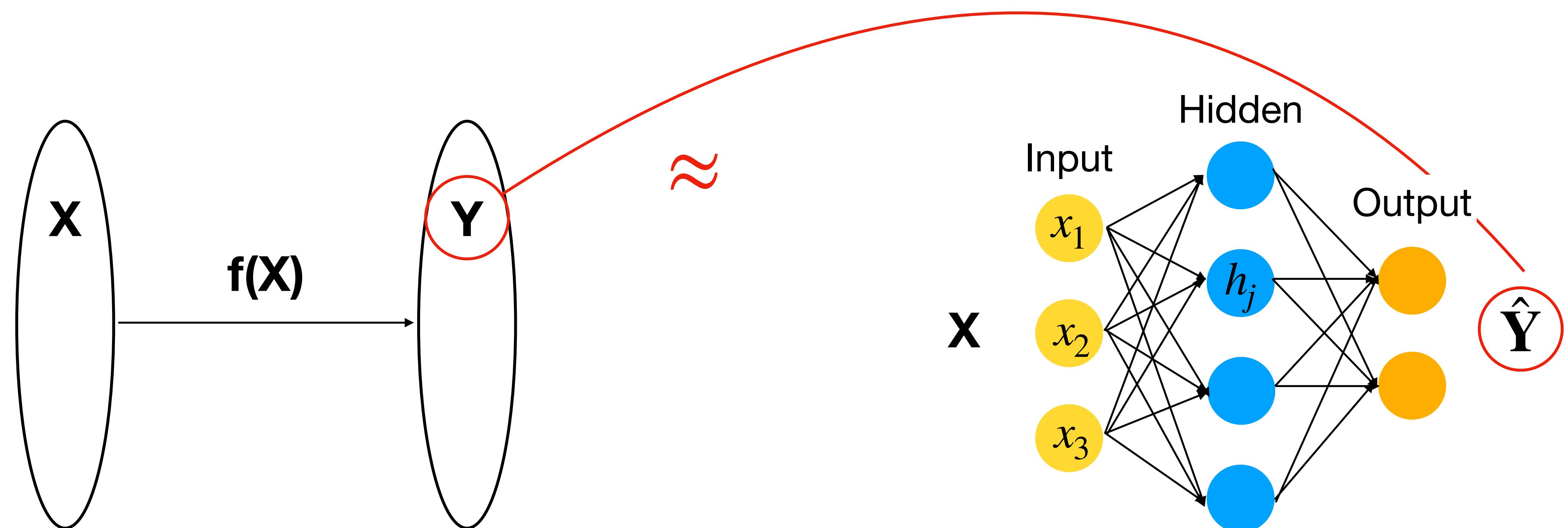


2-3 Neural Network의 동작 원리

Neural Network의 동작 원리

Question:

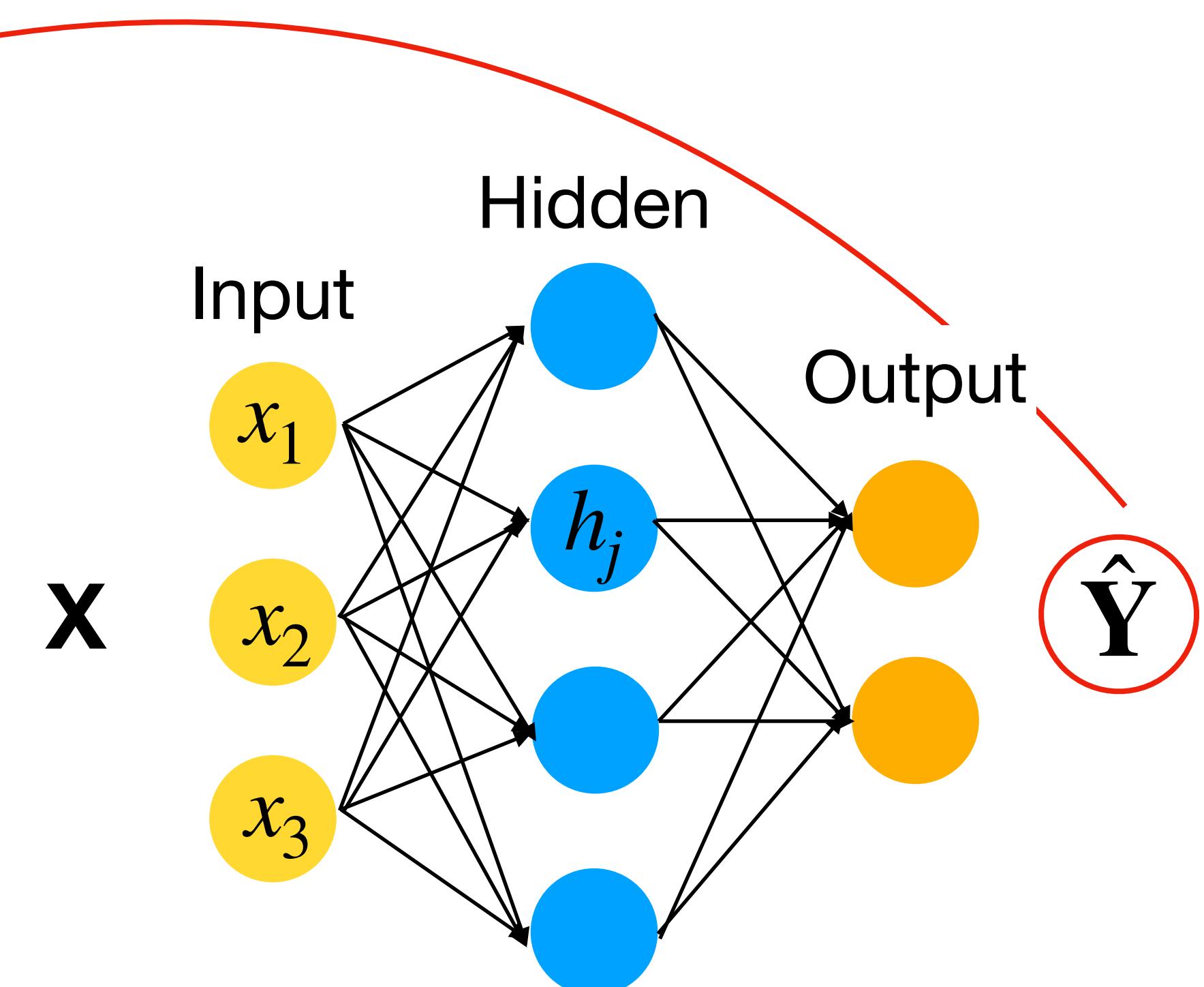
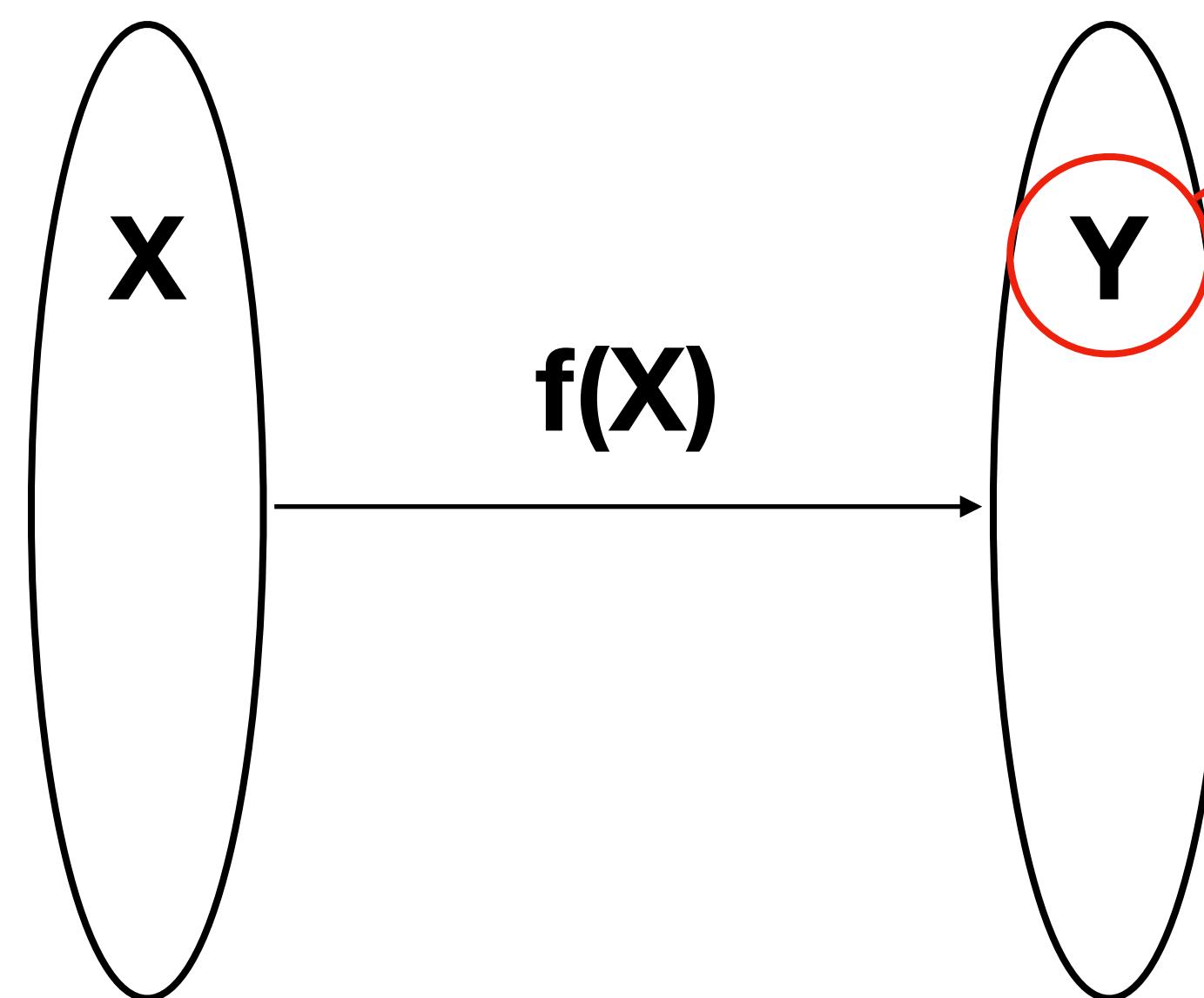
- 어떻게 해야 Neural Network의 출력값인 \hat{Y} 가 실제값인 Y 에 최대한 근사 $Y \approx \hat{Y}$ 할까?



Neural Network의 동작 원리

Question:

- 어떻게 해야 Neural Network의 출력값인 \hat{Y} 가 실제값인 Y 에 최대한 근사 $Y \approx \hat{Y}$ 할까?
- 각 Layer의 weight w 을 최적화해야 한다.



Neural Network의 동작 원리

Copyright©2023. Acadential. All rights reserved.

Question:

- 어떻게 해야 Neural Network의 출력값인 \hat{Y} 가 실제값인 Y 에 최대한 근사 $Y \approx \hat{Y}$ 할까?
- 각 Layer의 weight w 을 최적화해야 한다.

- Weight w_{ji} 이 바뀌면 주어진 Input x_i 값에 대한 Output y 값도 바뀐다
- 주어진 Input x_i 들에 대해서 최대한 실제값 y 과 유사하게 Output \hat{y} 을 출력해주는 가중치 w_{ji} 들의 조합을 찾고 싶은 것
- 각 Layer의 weight w_{ji} 을 적절하게 조정해서 주어진 Input에 대해서 출력되는 값 \hat{y} 이 실제값 y 에 최대한 잘 근사하도록 최적화하는 것

Neural Network의 동작 원리

Question:

- 어떻게 해야 Neural Network의 출력값인 \hat{Y} 가 실제값인 Y 에 최대한 근사 $Y \approx \hat{Y}$ 할까?
- 각 Layer의 weight w 을 최적화해야 한다.
- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?
- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Neural Network의 동작 원리

Weight 값의 최적화

Question:

- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?
- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Neural Network의 동작 원리

Weight 값의 최적화

Question:

- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?

모름! 처음에는 weight 값을 랜덤하게 정의함!

- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Neural Network의 동작 원리

Weight 값의 최적화

Question:

- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?

모름! 처음에는 weight 값을 랜덤하게 정의함!

- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Gradient Descent (경사 하강)을 통한 Loss function (손실 함수) 값을 최소화하도록

weight 값을 최적화하여 점진적으로 모델의 예측 정확도를 높인다.

Neural Network의 동작 원리

Weight 값의 최적화

Question:

- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?

섹션 11. 초기화 (Initialization)에서 자세히 다룰 예정

모름! 처음에는 weight 값을 랜덤하게 정의함!

- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Gradient Descent (경사 하강)을 통한 Loss function (손실 함수) 값을 최소화하도록

weight 값을 최적화하여 점진적으로 모델의 예측 정확도를 높인다.

Neural Network의 동작 원리

Weight 값의 최적화

Question:

- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?

섹션 11. 초기화 (Initialization)에서 자세히 다룰 예정

모름! 처음에는 weight 값을 랜덤하게 정의함!

- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Gradient Descent (경사 하강)을 통한 Loss function (손실 함수) 값을 최소화하도록

weight 값을 최적화하여 점진적으로 모델의 예측 정확도를 높인다.

섹션 4. 경사 하강법 (Gradient Descent)
에서 자세히 다룰 예정

Neural Network의 동작 원리

Weight 값의 최적화

Question:

- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?

섹션 11. 초기화 (Initialization)에서 자세히 다룰 예정

모름! 처음에는 weight 값을 랜덤하게 정의함!

- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Gradient Descent (경사 하강)을 통한 Loss function (손실 함수) 값을 최소화하도록

weight 값을 최적화하여 점진적으로 모델의 예측 정확도를 높인다.

섹션 4. 경사 하강법 (Gradient Descent)
에서 자세히 다룰 예정

섹션 3. 손실 함수 (Loss Function)
에서 자세히 다룰 예정

Neural Network의 동작 원리

Weight 값의 최적화

Question:

- Weight 값을 어떻게 정의해야 예측값이 최대한 정확할까?

모름! 처음에는 weight 값을 랜덤하게 정의함!

- Weight 값을 어떻게 최적화해야 모델의 예측값이 더 정확해질 수 있을까?

Gradient Descent (경사 하강)을 통한 **Loss function** (손실 함수) 값을 최소화하도록

weight 값을 최적화하여 점진적으로 모델의 예측 정확도를 높인다.

즉, 처음에는 랜덤한 weight 값에 따라 모델의 예측값도 random 하지만 weight 값을 최적화하여 점차 모델의 정확도를 높임!

See you in the practicals!

2-4. Deep Learning 실무 기초 개념

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

목차:

- Training, Validation, Test dataset
- Overfitting
- K-fold Cross Validation
- Hyperparameter Tuning
- Loss와 Evaluation Metric

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

- 1. Training (학습 / 훈련) 데이터셋**
- 2. Validation (검증) 데이터셋**
- 3. Test (시험) 데이터셋**

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

1. Training (학습 / 훈련) 데이터셋

- 모델을 학습시키는 용도
- (딥러닝 모델의 경우) 학습 데이터셋에 대해서 **Gradient Descent**하여 **Loss**가 최소화되도록 모델의 **weight** 최적화

2. Validation (검증) 데이터셋

3. Test (시험) 데이터셋

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

1. Training (학습 / 훈련) 데이터셋

2. Validation (검증) 데이터셋

- 모델의 성능 평가와 **Hyperparameter Tuning**에 사용
- 학습 데이터셋에 대한 “**Overfitting**” 될 수 있기에 Validation 데이터셋으로 성능 평가

3. Test (시험) 데이터셋

Hyperparameter란

- 모델 구조와 학습 방식에 영향을 주는 Parameter
- 모델의 최종 성능에도 영향을 준다.

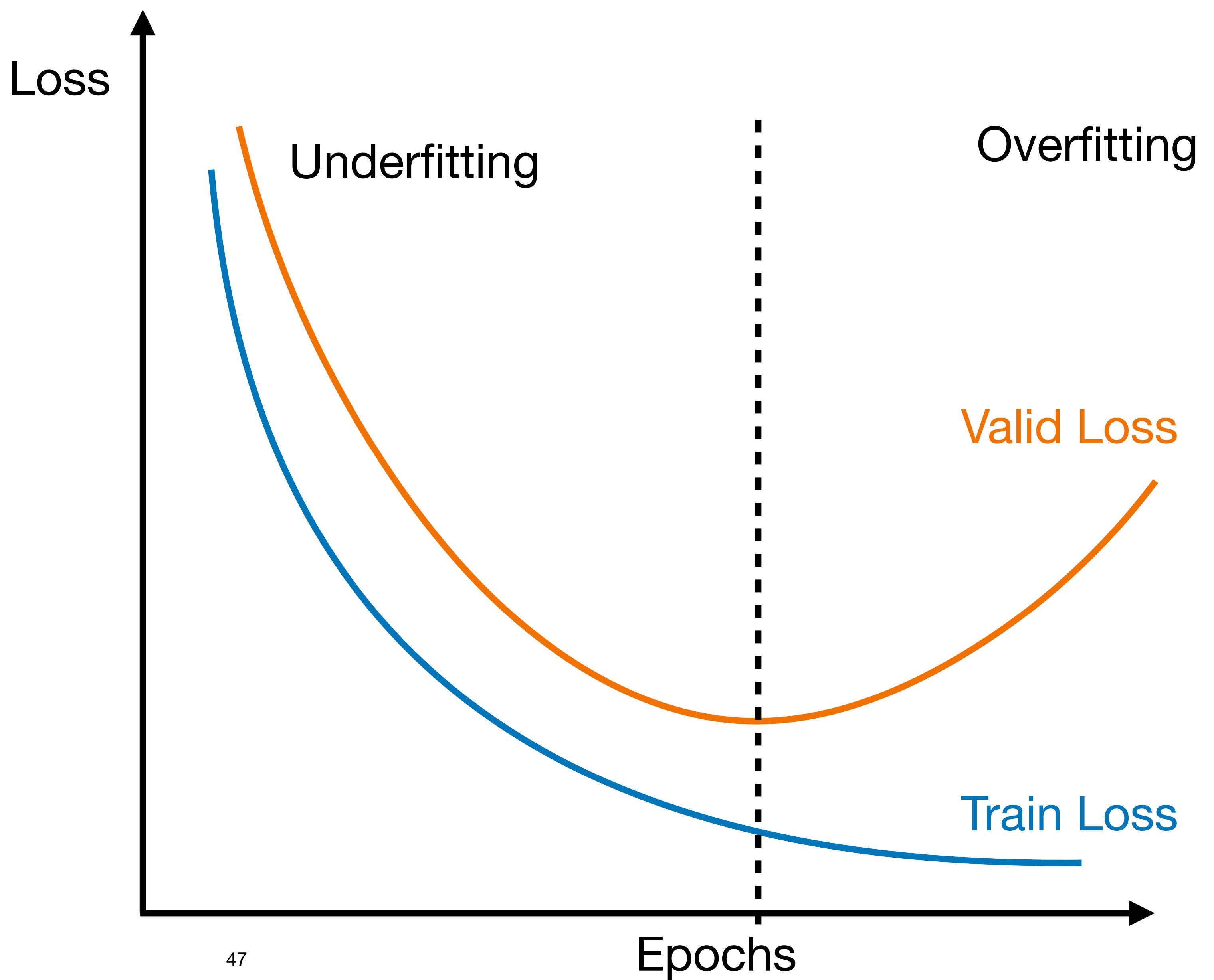
Overfitting

Copyright©2023. Acadential. All rights reserved.

Overfitting (과적합)이란

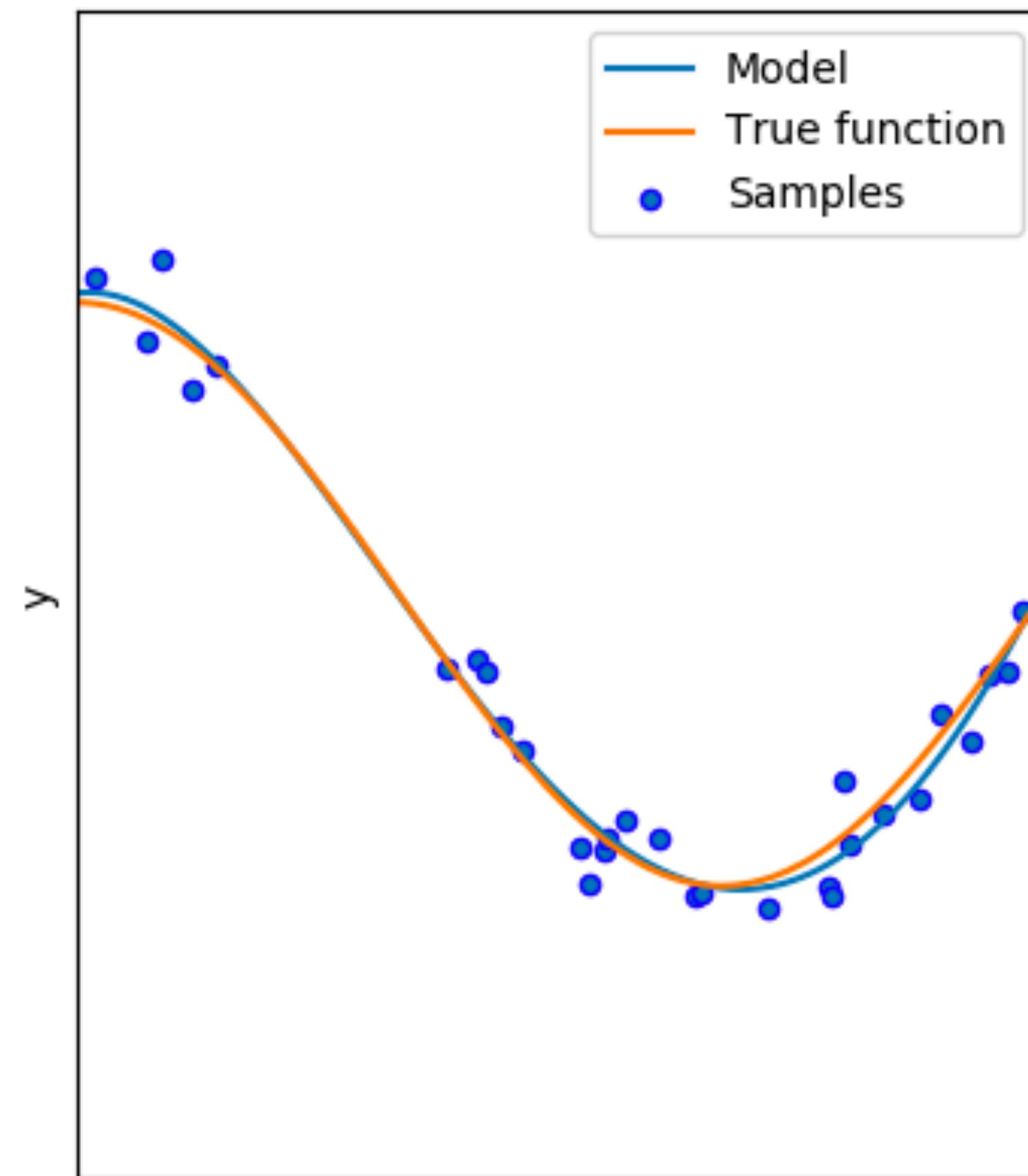
- **Unseen data**에 대해서 모델의 예측값이 일반화되지 않는 경우.
- 뉴럴넷 모델이 학습 데이터에 있는 **noise (노이즈)**에 대해서도 학습하여 일반화 성능 (**generalizability**)가 저하되는 현상.

Overfitting

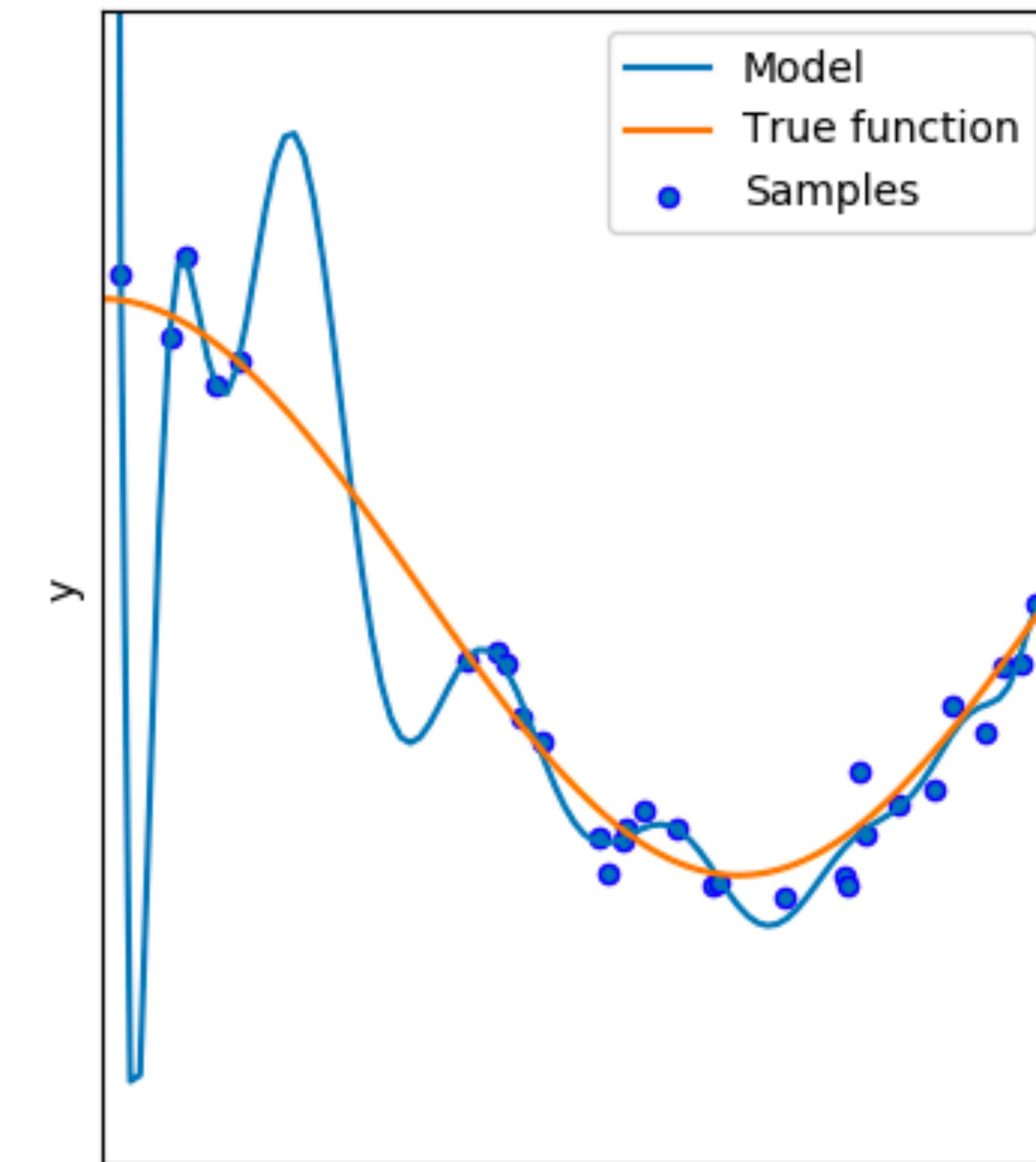


Overfitting

Degree 4



Degree 15



Good Fit

High Variance

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

1. Training (학습 / 훈련) 데이터셋

2. Validation (검증) 데이터셋

- 모델의 성능 평가와 **Hyperparameter Tuning**에 사용
- 학습 데이터셋에 대한 “**Overfitting**” 될 수 있기에 Validation 데이터셋으로 성능 평가

3. Test (시험) 데이터셋

Overfitting란

- 학습 데이터의 Noise마저 학습하여 일반화 성능이 감소하는 것.

Hyperparameter Tuning이란

- 가장 최적의 Hyperparameter 조합을 찾는 것.
- Validation 성능이 가장 높은 조합 찾기

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

1. Training (학습 / 훈련) 데이터셋

2. Validation (검증) 데이터셋

3. Test (시험) 데이터셋

- 검증 단계에서 선택한 최적의 모델의 최종 성능을 평가하는데 사용
- Hyperparameter Tuning을 과도하게 적용하는 경우 Validation dataset에 대해서 unintentional overfitting 발생할 수 있다.

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할

- 원본 데이터셋을 K 개의 서로 겹치지 않는 부분 집합으로 나눈다. (일반적으로 K=5 혹은 10 사용)

2. 반복 학습 및 검증

원본 데이터셋

3. 성능 측정



4. 최종 성능 평가 혹은 Hyperparameter Tuning

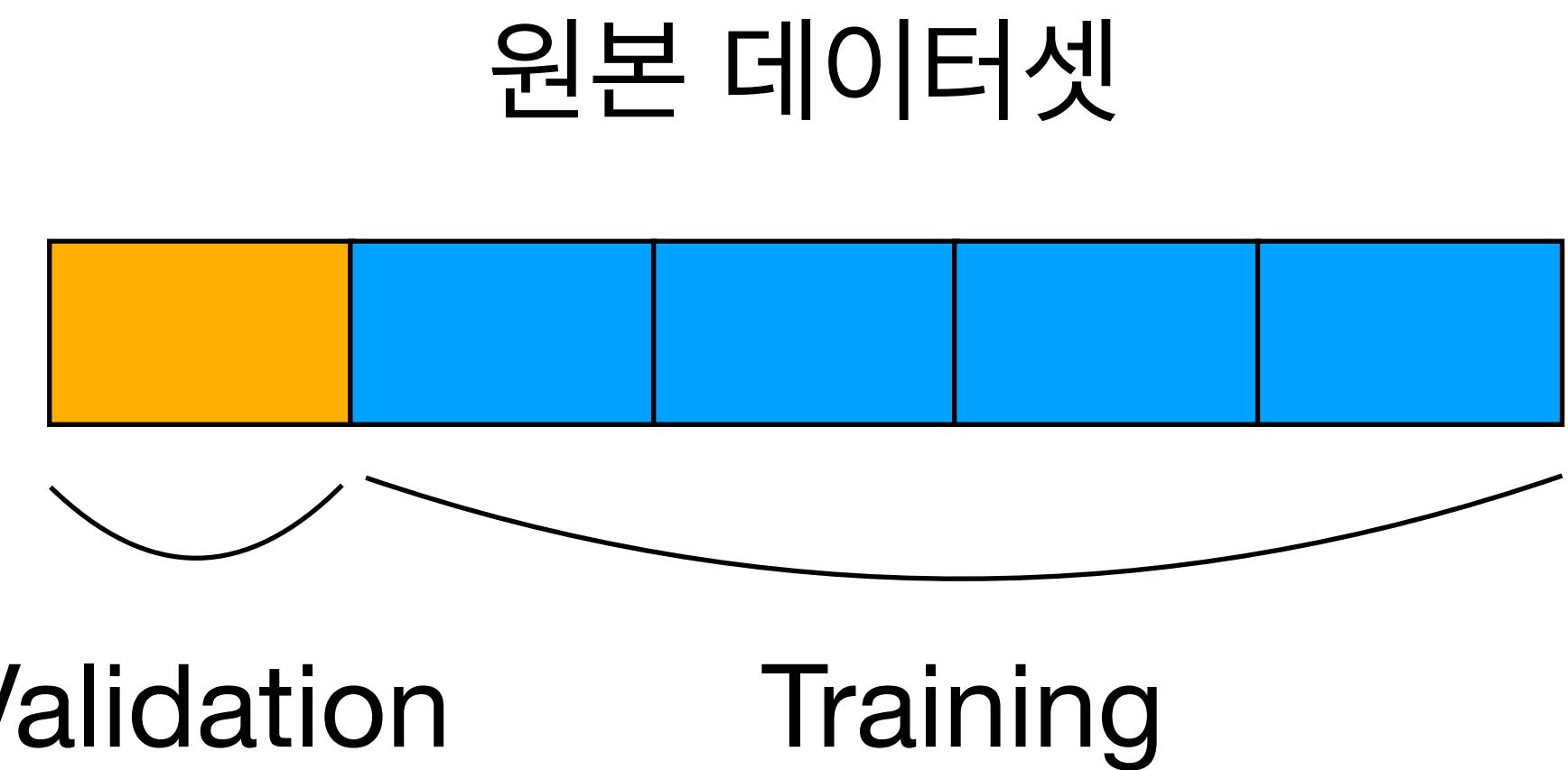
K 등분

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할
2. 반복 학습 및 검증
 - 하나의 폴드를 검증 데이터로 사용
 - K-1개의 폴드를 훈련 데이터로 사용하여 모델 학습
3. 성능 측정
4. 최종 성능 평가 혹은 Hyperparameter Tuning

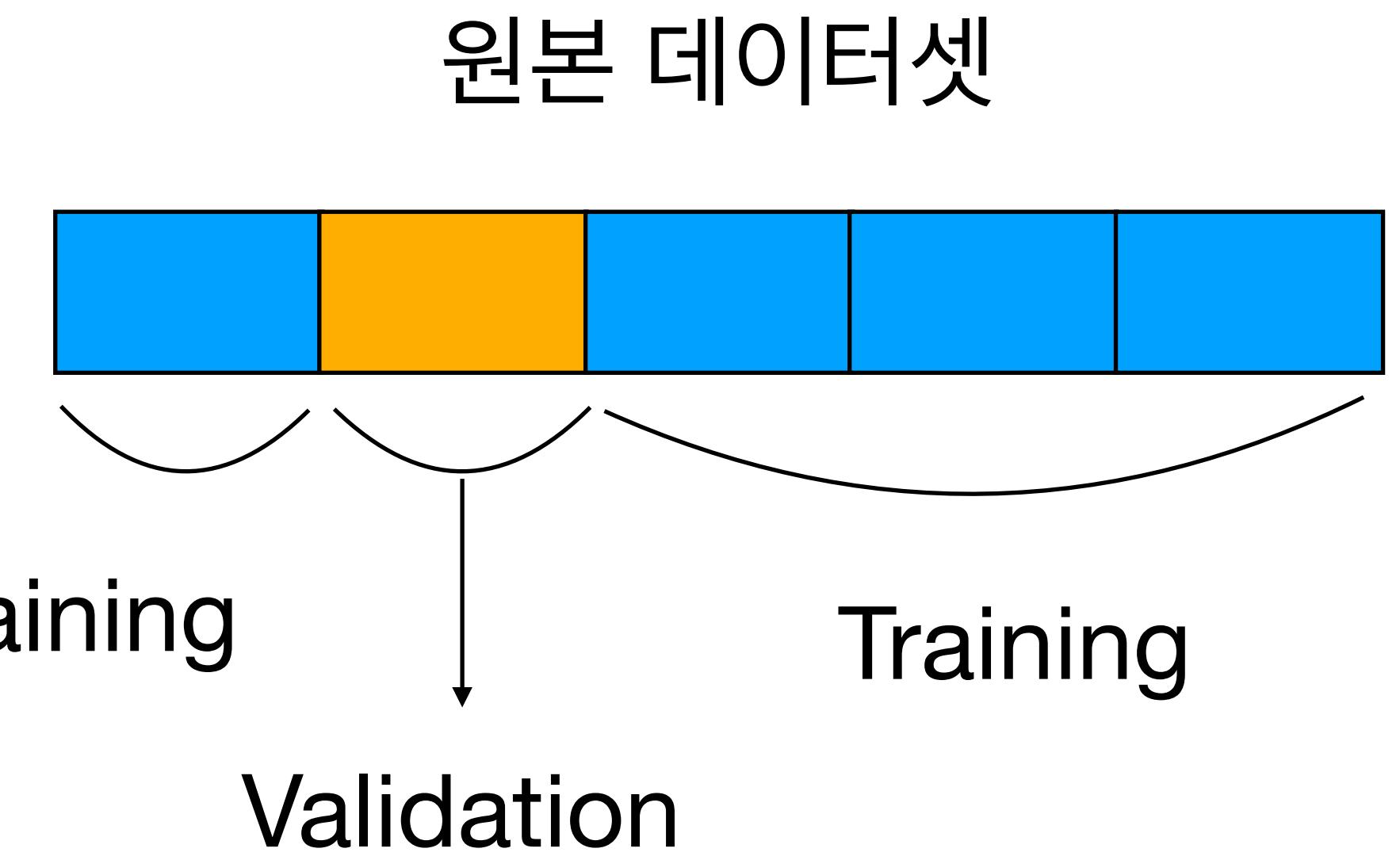


Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할
2. 반복 학습 및 검증
 - 하나의 폴드를 검증 데이터로 사용
 - $K-1$ 개의 폴드를 훈련 데이터로 사용하여 모델 학습
3. 성능 측정
4. 최종 성능 평가 혹은 Hyperparameter Tuning

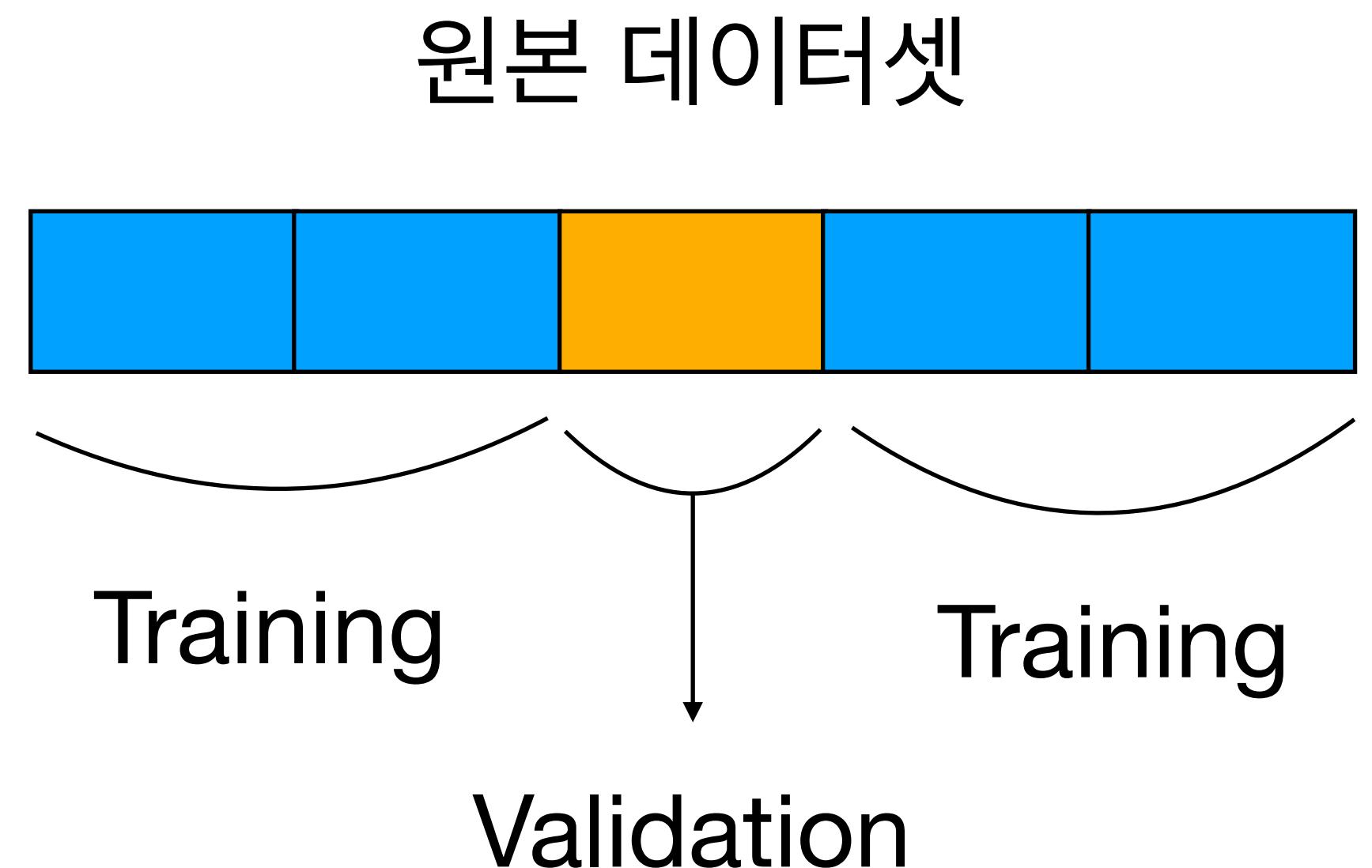


Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할
2. 반복 학습 및 검증
 - 하나의 폴드를 검증 데이터로 사용
 - $K-1$ 개의 폴드를 훈련 데이터로 사용하여 모델 학습
3. 성능 측정
4. 최종 성능 평가 혹은 Hyperparameter Tuning

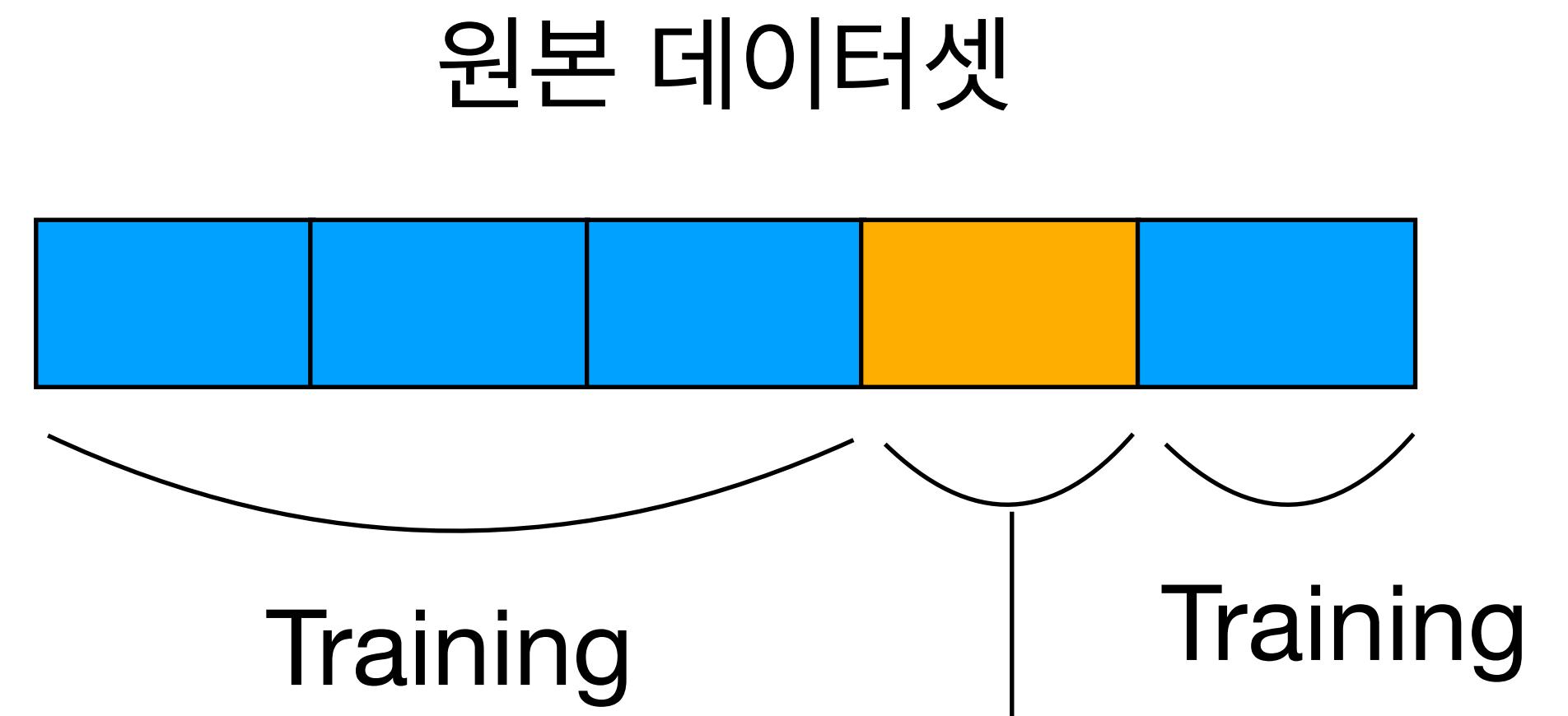


Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할
2. 반복 학습 및 검증
 - 하나의 폴드를 검증 데이터로 사용
 - K-1개의 폴드를 훈련 데이터로 사용하여 모델 학습
3. 성능 측정
4. 최종 성능 평가 혹은 Hyperparameter Tuning

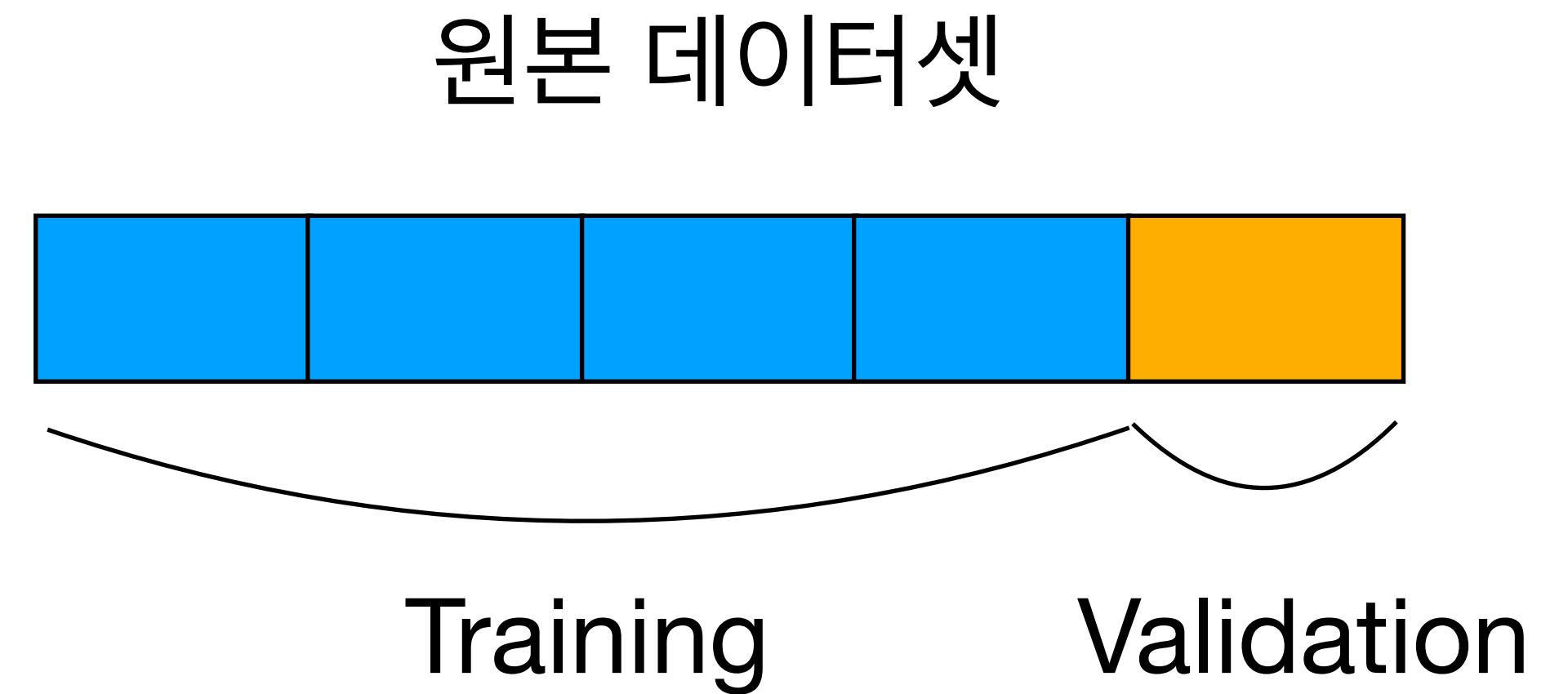


Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할
2. 반복 학습 및 검증
 - 하나의 폴드를 검증 데이터로 사용
 - K-1개의 폴드를 훈련 데이터로 사용하여 모델 학습
3. 성능 측정
4. 최종 성능 평가 혹은 Hyperparameter Tuning



Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할
2. 반복 학습 및 검증
3. 성능 측정
 - 각 반복마다 모델은 검증용 폴드에 대한 성능 평가
 - 모든 K 번의 평가를 완료한 후에 평균 성능 계산
4. 최종 성능 평가 혹은 Hyperparameter Tuning

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

1. 데이터 분할
2. 반복 학습 및 검증
3. 성능 측정
4. 최종 성능 평가 혹은 Hyperparameter Tuning
 - Fold들에 대한 평균 성능을 최종 성능 지표로 사용
 - 혹은 Hyperparameter Tuning에 사용

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

K-fold Cross Validation

- 장점:
 - 데이터를 효과적으로 활용하여 모델의 성능을 평가
 - 과적합 문제를 예방하고 모델의 일반화 성능을 더 신뢰성 있게 추정
 - 검증 데이터의 선택에 따라 모델의 평가 결과가 크게 변하지 않는다

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

Loss와 Evaluation Metric의 차이

- **Loss Function**
 - 예측한 값과 실제 타깃 값 사이의 차이
 - 학습 단계에서 Loss을 최소화하는 방향으로 모델의 Weight을 조정
 - 미분 가능해야한다!
 - Cross Entropy Loss, Mean Squared Loss, 등등
- **Evaluation Metric**

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

Loss와 Evaluation Metric의 차이

- Loss Function
- **Evaluation Metric**
 - 학습된 모델의 성능을 평가하는데 사용되는 지표
 - 손실함수와 달리 평가 지표는 더 직관적이다.
 - 정확도 (Accuracy), 정밀도 (Precision), 재현율 (Recall), F1 Score 등등

Deep Learning 실무 기초 개념

Copyright©2023. Acadential. All rights reserved.

Loss와 Evaluation Metric의 차이

- Loss function:
 - 모델 학습 단계에서 모델의 가중치를 조정하기 위한 목적으로 예측 오차를 측정
- Evaluation Metric:
 - 학습된 모델의 성능을 평가하고 보고하기 위해 사용

2-5. [실습] PyTorch 기초 - Tensor

PyTorch의 Tensor

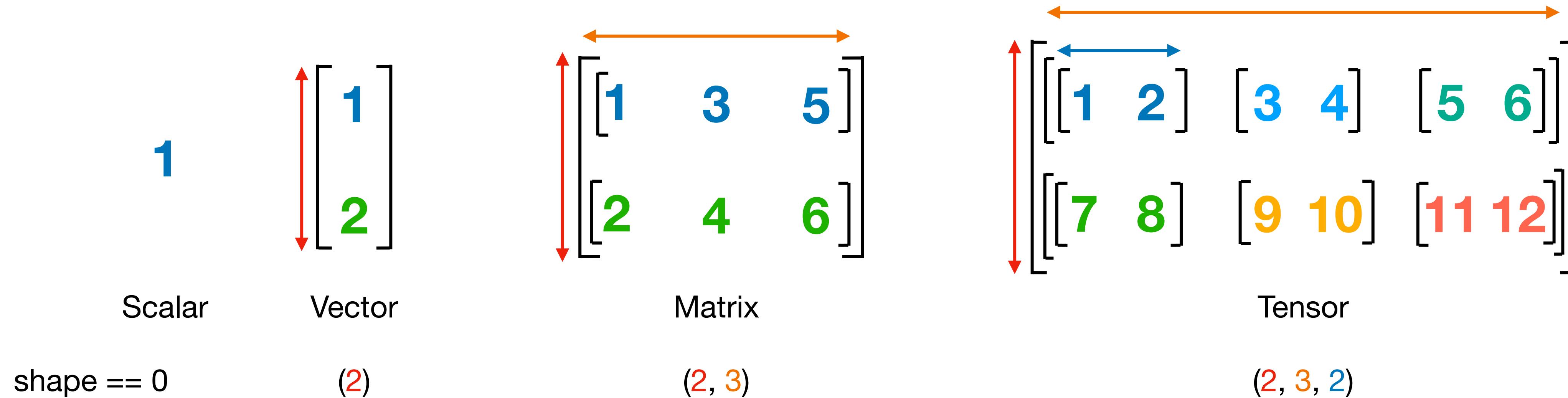
Tensor (torch.Tensor)

- Numpy의 배열 (array)와 행렬 (matrix)과 매우 유사한 자료구조
- PyTorch에서 scalar, vector, matrix, tensor등등을 표현하는데 사용

PyTorch의 Tensor

Tensor (torch.Tensor)

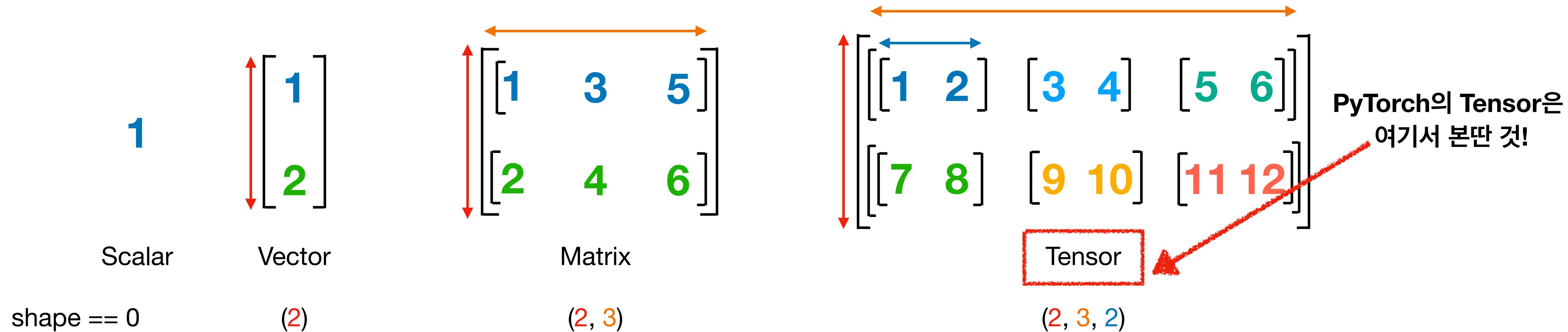
- Numpy의 배열 (array)와 행렬 (matrix)과 매우 유사한 자료구조이다.
- PyTorch에서 scalar, vector, matrix, tensor등등을 표현하는데 사용



PyTorch의 Tensor

Tensor (torch.Tensor)

- Numpy의 배열 (array)와 행렬 (matrix)과 매우 유사한 자료구조이다.
- PyTorch에서 scalar, vector, matrix, tensor등등을 표현하는데 사용



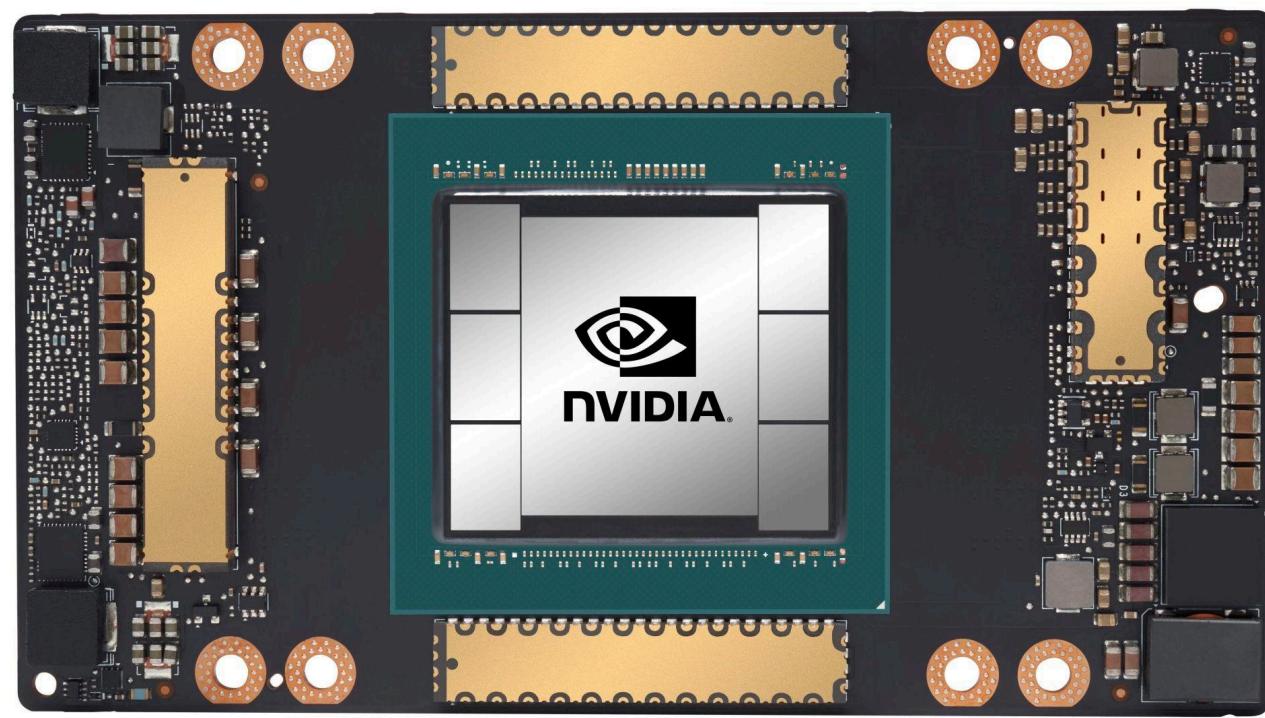
PyTorch의 Tensor

Tensor (torch.Tensor)

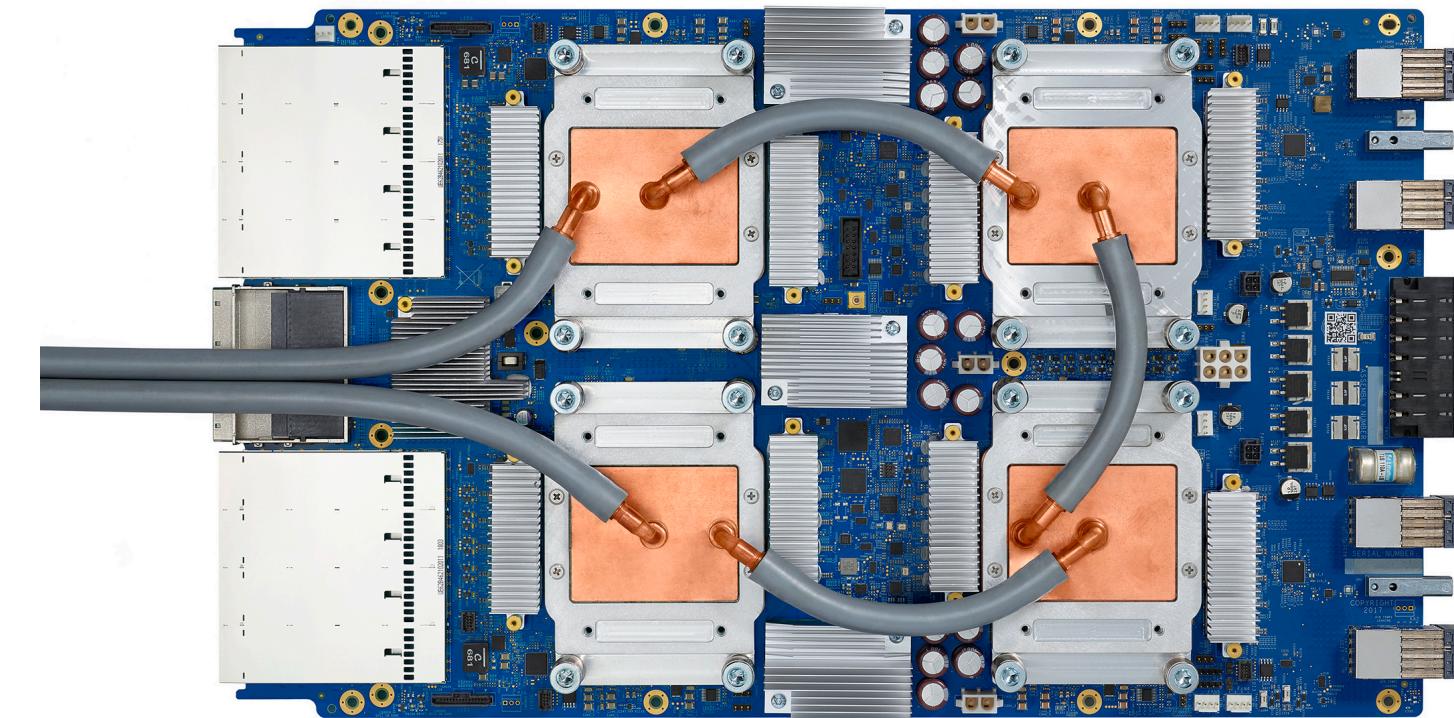
- GPU나 TPU와 같은 연산 가속을 위한 특수한 하드웨어에서 실행할 수 있다.



CPU



GPU



TPU

Hardware accelerator

(CPU, GPU, TPU의 차이에 대해서는 다음 영상 참고! https://www.youtube.com/watch?v=6ZDoFomU10A&ab_channel=DIDYOUKNOW)

PyTorch의 Tensor

Tensor (torch.Tensor)

- Backward pass에서 계산된 Gradient (.grad)을 저장한다.
- 기본적으로 torch.Tensor에 어떤 operation (더하기, 곱셈 등등)을 취하면 해당 operation이 Computational Graph에 기록

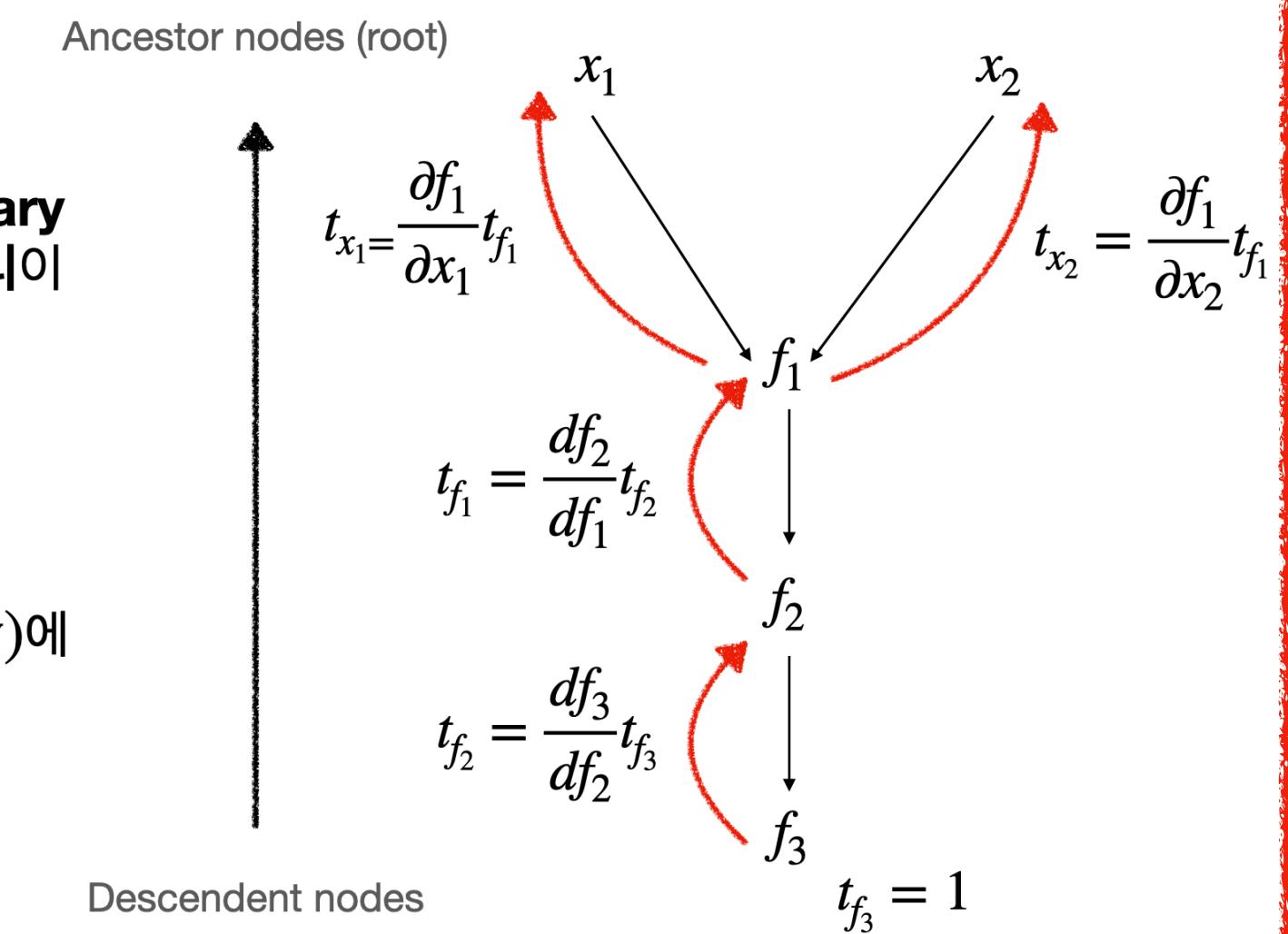
이론편의 6-4. Auto Differentiation 참고!

Auto Differentiation Auto Differentiation의 Summary

이것이 바로 PyTorch, Tensorflow 등의 딥러닝 Library들의 Backward Propagation의 핵심개념이자 작동원리이다!

모두 Reverse Differentiation을 활용해서 Gradient Descent에 필요한 Gradient를 계산하는 것이다!

즉, $\mathbf{w} = (w_0, w_1, \dots, w_N)$ 인 어떤 임의의 합성함수 $L(\mathbf{w})$ 에 대해서 우리는 $\frac{dL}{dw_i}$ 을 구할 수 있는 것이다!



Let's look at Jupyter Notebook!

2-6. [실습] PyTorch 기초 Datasets와 DataLoader

Dataset과 Data Loader

Copyright©2023. Acadential. All rights reserved.

- Dataset과 Data Loader은 PyTorch에서 제공하는 추상 클래스 (abstract class) 이다.
- **Dataset** (`torch.utils.data.Dataset`)
 - Mini-batch를 구성할 각 data sample을 하나씩 불러오는 기능을 수행한다.
- **Dataloader** (`torch.utils.data.DataLoader`)
 - Dataset에서 불러온 각 data sample들을 모아서 mini-batch로 구성하는 기능을 수행 한다.

그리고 Data Loader를 통해서 Data sample들을 병렬적으로 불러오고, 데이터 셔플 (`shuffle`) 등등의 작업을 간단하게 수행할 수 있다!

Dataset과 Data Loader

Copyright©2023. Acadential. All rights reserved.

Dataset의 기본 뼈대:

1. `__init__` 함수
2. `__len__` 함수
3. `__getitem__` 함수

```
from torch.utils.data import Dataset

class CustomDataset(Dataset):
    def __init__(self):
        #데이터셋의 전처리를 수행하는 부분
    def __len__(self):
        # 데이터셋의 길이 (즉, 총 샘플의 개수를 명시하는 부분)
    def __getitem__(self, item):
        # 데이터셋에서 특정 1개의 샘플을 가져오는 부분
        # 참고로 item은 index
```

위 함수들을 명시해서 Custom Dataset을 만들 수 있다!

어떻게 customize하는지는 조금 후에 살펴볼 예정!

Dataset과 Data Loader

Data Loader의 input:

- dataset
- batch_size: mini-batch의 크기
- shuffle (binary): 무작위 순서로 데이터를 샘플링 할 것인지
- num_workers: 데이터 로딩에 사용할 subprocess 개수 (병렬 처리)
- pin_memory (binary): GPU memory에 pin 할 것인지
- drop_last (binary): 마지막 mini-batch을 drop할 것인지

Dataset과 Data Loader

Copyright©2023. Acadential. All rights reserved.

참고로

- num_workers 수가 많을수록 데이터 로딩이 더 빠르지만 그만큼 CPU core 개수도 충분해야한다. (CPU core 개수보다 num_workers가 많으면 오히려 느려지는 경우 발생).
- pin_memory=True 로 했을시 GPU (cuda) memory을 미리 할당, 확보시켜줘서 조금 더 빠르게 데이터를 GPU에 올릴 수 있게 해준다.

Dataset과 Data Loader

다음 내용을 Jupyter Notebook에서 살펴보자:

- CIFAR10 (torchvision.datasets.CIFAR10)을 활용
- torch.utils.Dataset으로 Custom Dataset 구현
- torch.utils.DataLoader 활용

Let's look at Jupyter Notebook!

2-7. [실습] PyTorch 기초 Transforms

Transforms

Copyright©2023. Acadential. All rights reserved.

Torchvision에서는 Computer Vision에서 사용되는 다양한 모델, 데이터셋들을 제공

Transforms

Copyright©2023. Acadential. All rights reserved.

Torchvision에서는 Computer Vision에서 사용되는 다양한 모델, 데이터셋들을 제공

여기서 torchvision의 **transforms** 모듈은 이미지 전처리에 관련된 유용한 기능들을 제공

예를 들어

- ToTensor: image을 tensor로 변환하고 0~1의 값으로 normalize
- Data augmentation (데이터 증강):
 - Gaussian Blur
 - Random Affine, 등등

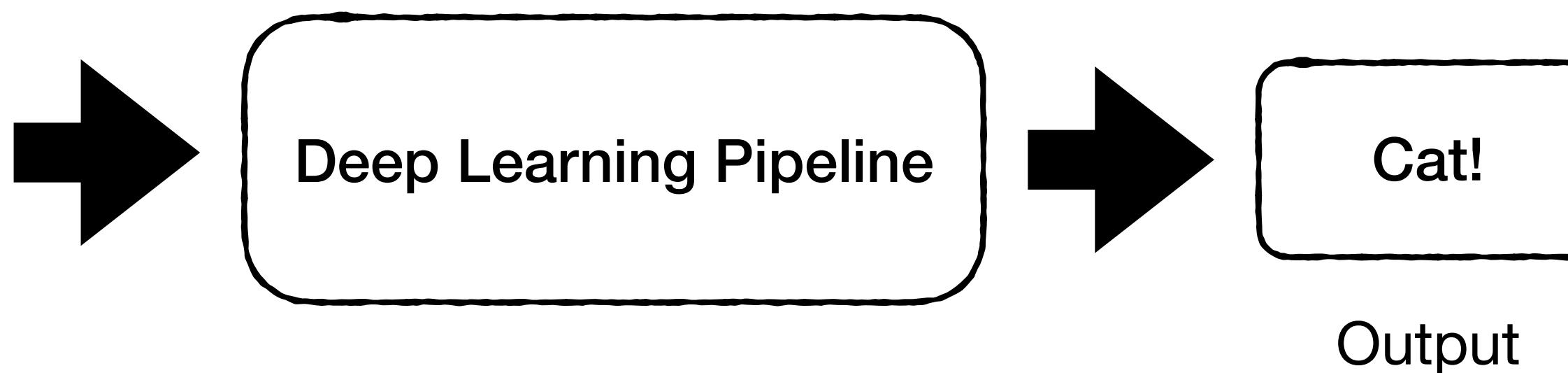
전처리 (Preprocessing)???

Deep Learning Project의 Pipeline을 살펴보자

Transforms Deep Learning Pipeline



Raw Input



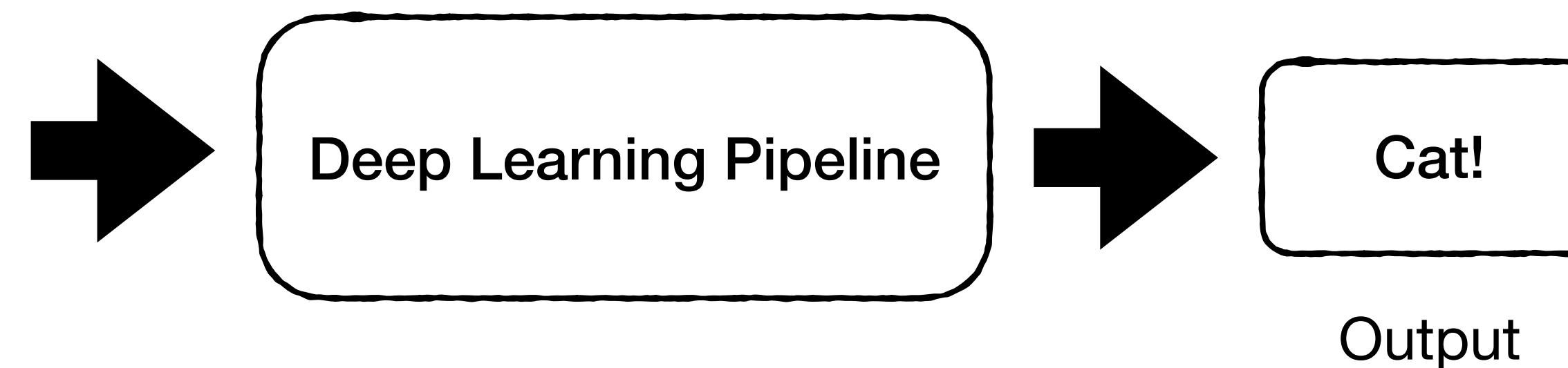
Transforms

Deep Learning Pipeline

참고로 여기서의 pipeline은 MLOps에서의 Machine Learning Pipeline은 아니다.
여기서의 pipeline은 raw input에 대해서 output으로 mapping하는 일련의 과정을 의미.

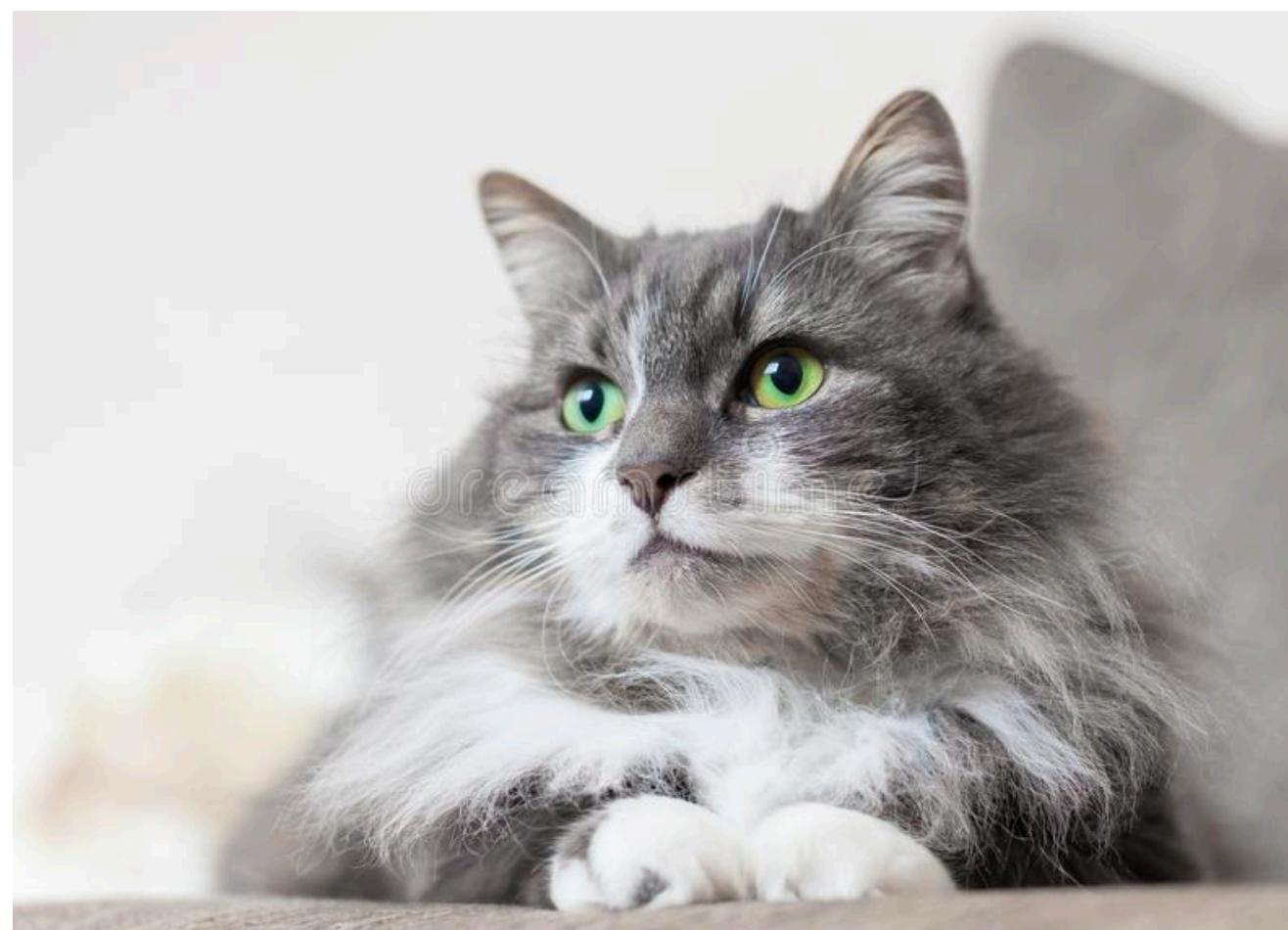


Raw Input

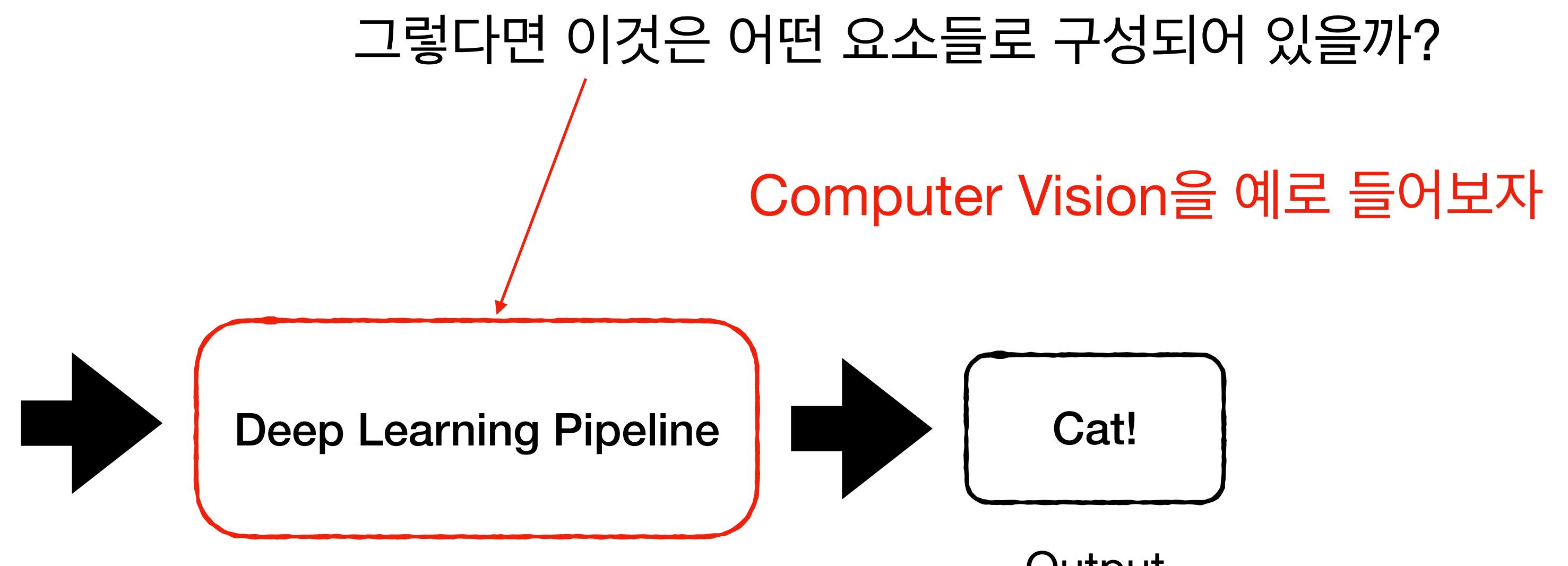


Output

Transforms Deep Learning Pipeline

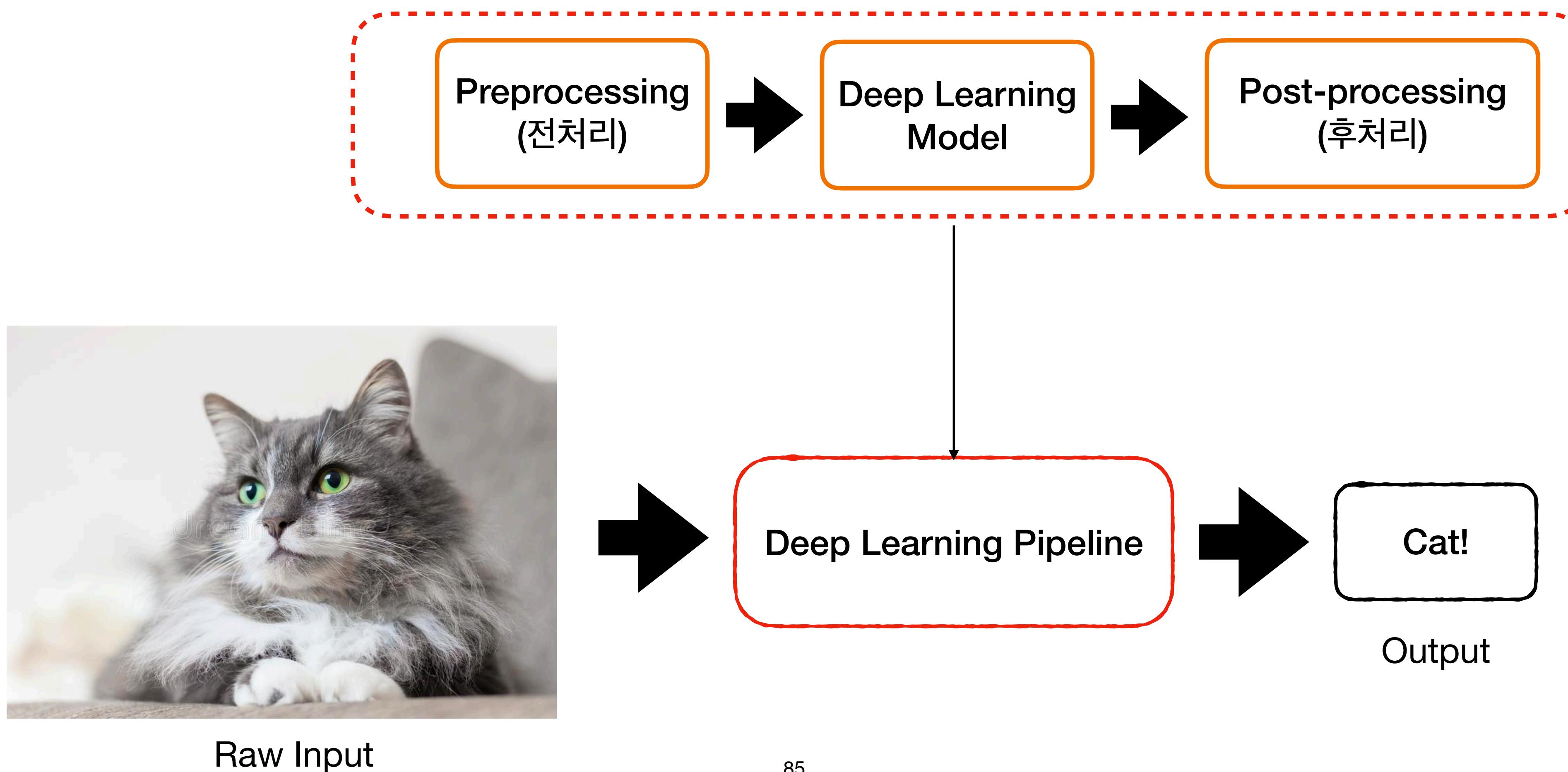


Raw Input



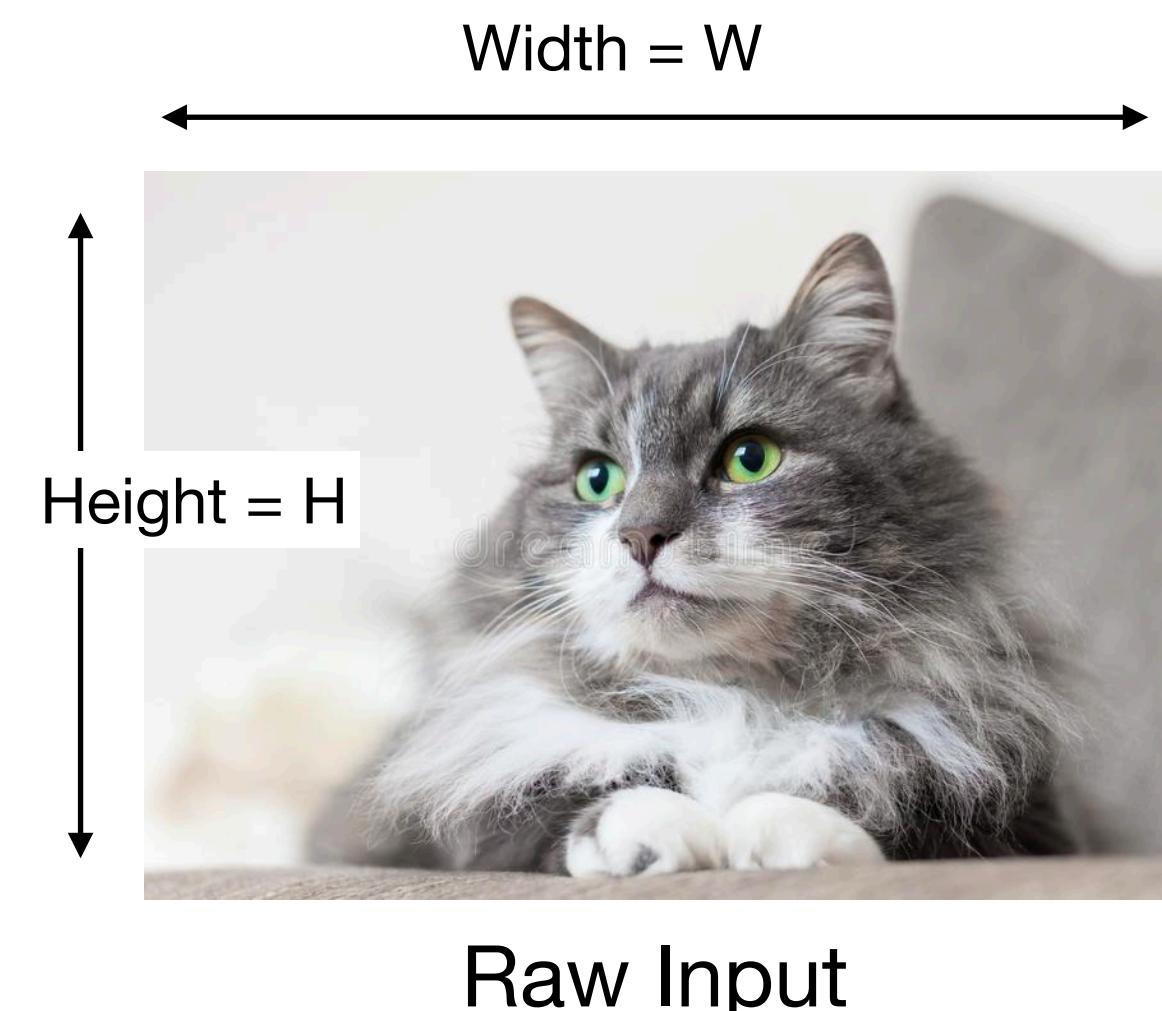
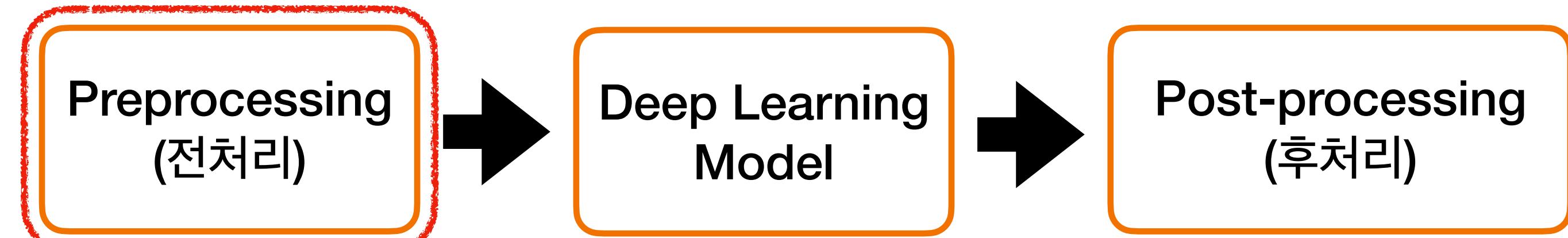
Transforms

Deep Learning Pipeline



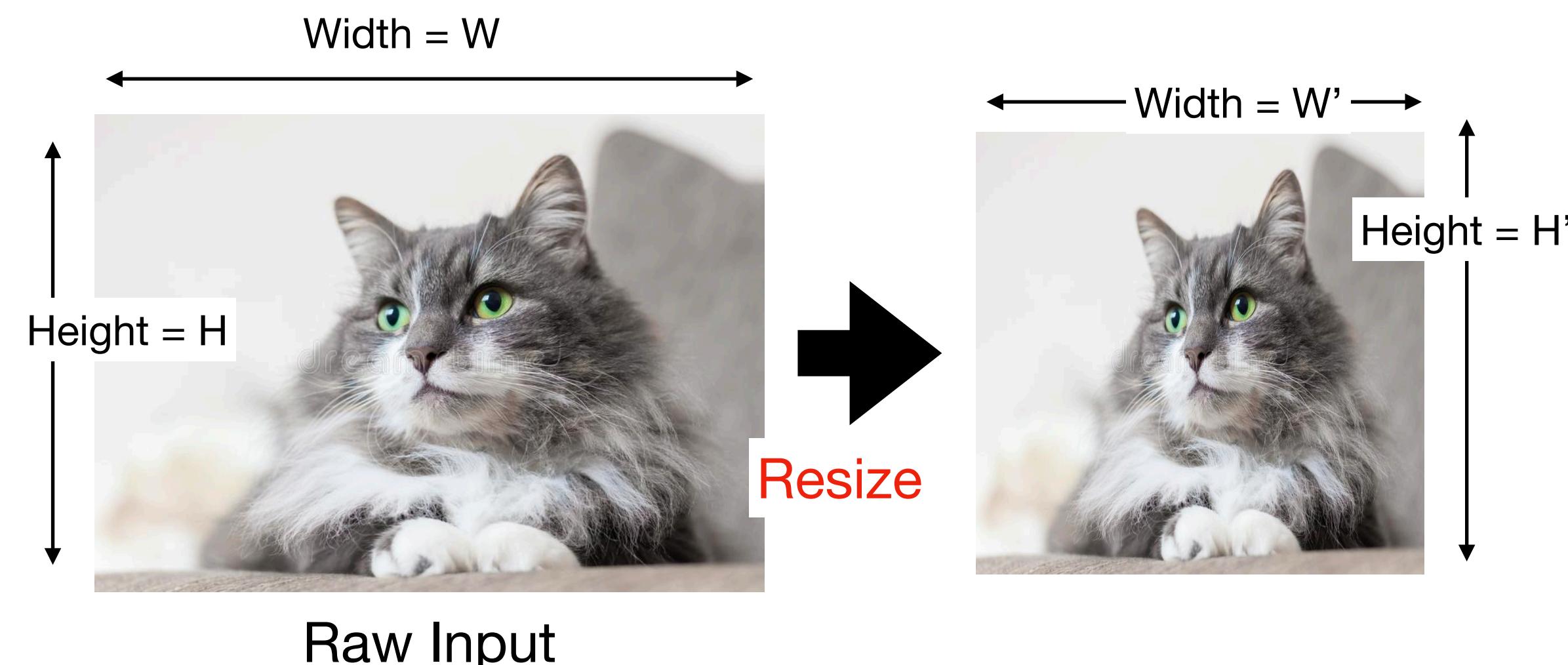
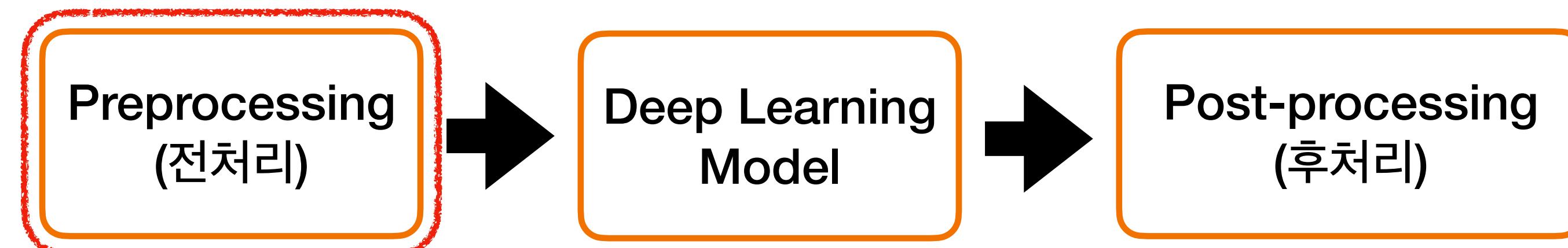
Transforms

Deep Learning Pipeline



Transforms

Deep Learning Pipeline: Preprocessing



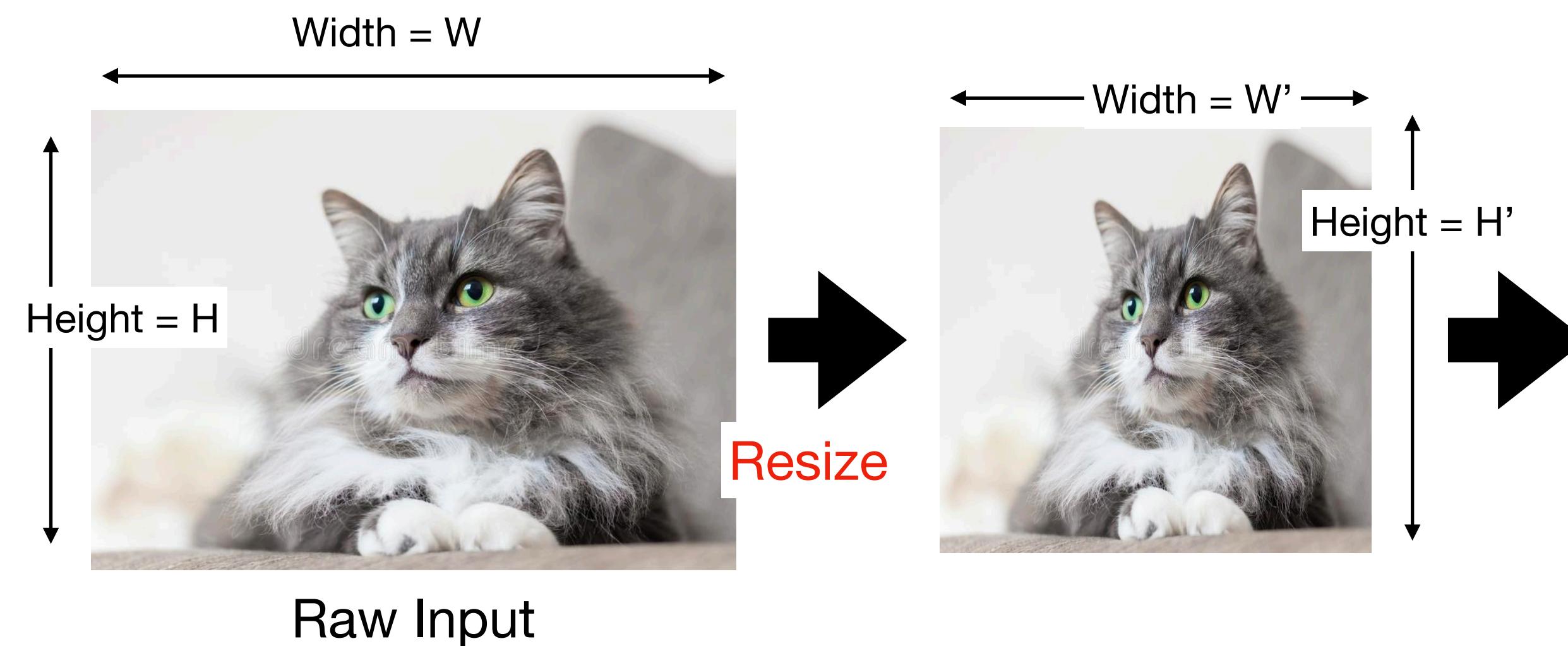
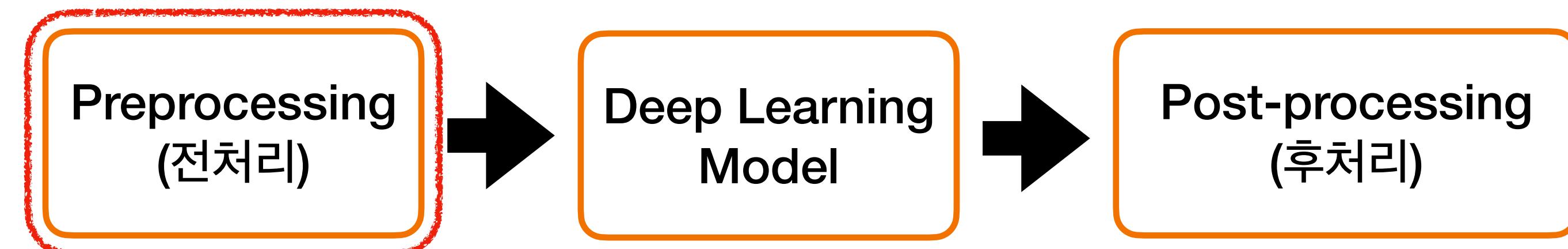
이미지를 정해진 (H' , W')로 resize한다!

왜냐하면 NN의 input으로 mini-batch을 구성하는 데이터들은 동일한 dimension (shape)을 가져야 한다!

그래야 mini-batch을 Matrix 혹은 Tensor로 표현할 수 있다.

Transforms

Deep Learning Pipeline: Preprocessing



```
np.array(img)
```

```
array([[255, 255, 255, 0],  
       [255, 255, 255, 0],  
       [255, 255, 255, 0],  
       ...,  
       [255, 255, 255, 0],  
       [255, 255, 255, 0],  
       [255, 255, 255, 0]],  
  
      [[255, 255, 255, 0],  
       [255, 255, 255, 0],  
       [255, 255, 255, 0],  
       ...,  
       [255, 255, 255, 0],  
       [255, 255, 255, 0],  
       [255, 255, 255, 0]],  
  
      [[255, 255, 255, 0],  
       [255, 255, 255, 0],  
       [255, 255, 255, 0],  
       ...,  
       [255, 255, 255, 0],  
       [255, 255, 255, 0],  
       [255, 255, 255, 0]]]
```

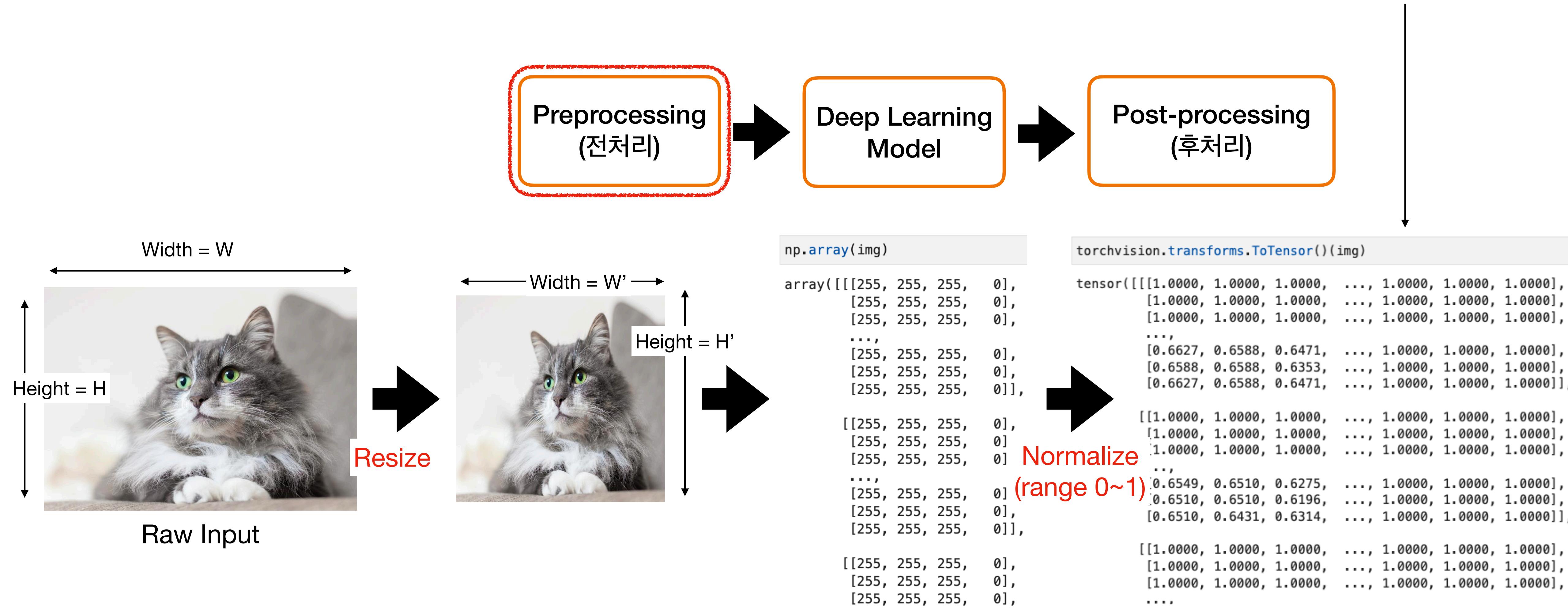
PIL Image을 Numpy array로 변환

하지만 각 r, g, b 채널은 0~255 사이의 integer이다.

Transforms

Deep Learning Pipeline: Preprocessing

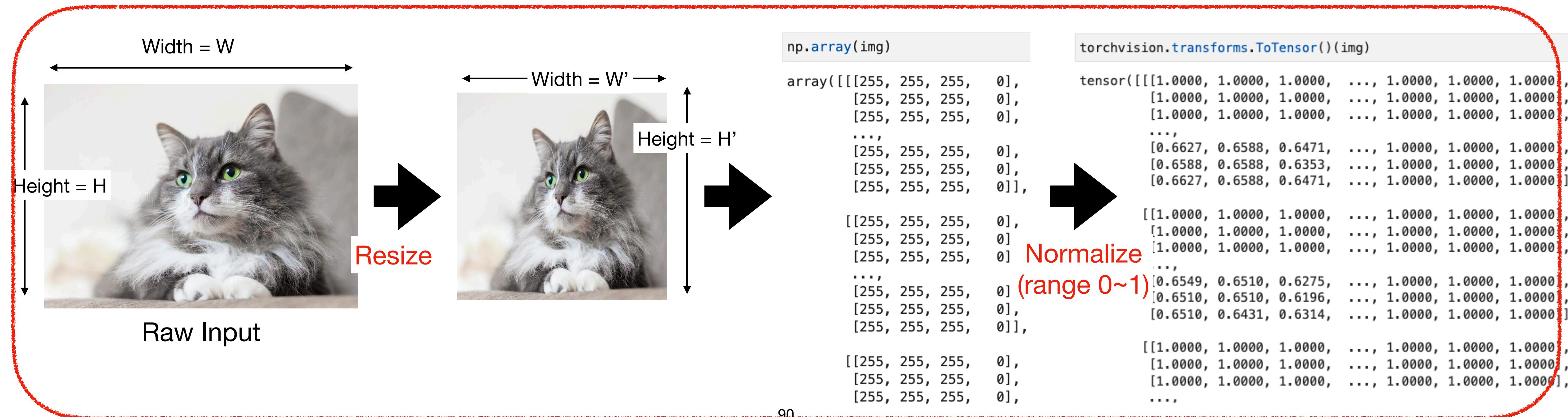
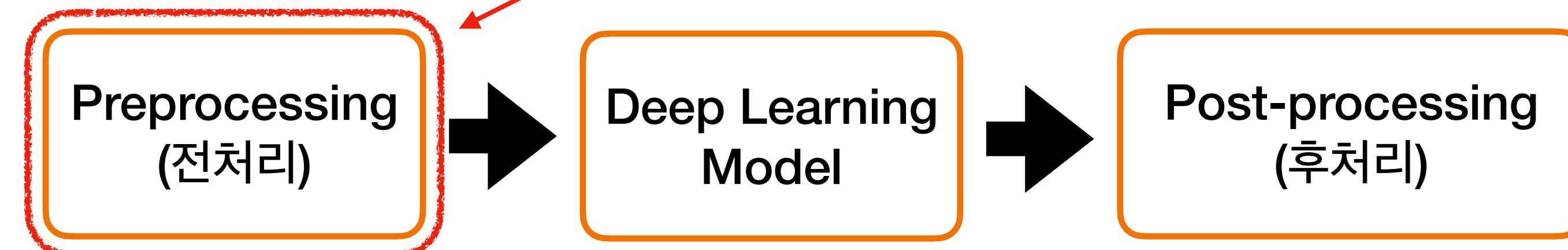
0~1 사이의 값으로 normalize!



Transforms

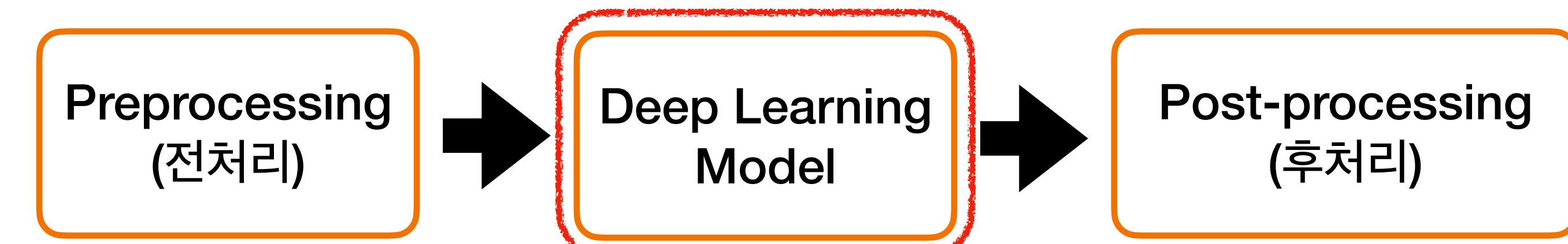
Deep Learning Pipeline: Preprocessing

Torchvision의 Transforms은
CV와 관련된 전처리 함수, 기능
들을 포함한다!



Transforms

Deep Learning Pipeline



Transforms

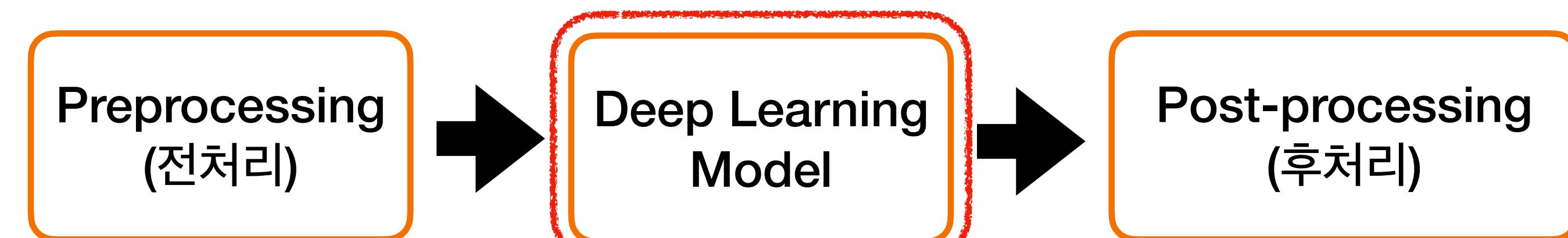
Deep Learning Pipeline: Model

Scalar은 0차원 $\in \mathbb{R}$

Vector은 1차원 $\in \mathbb{R}^D$

행렬은 2차원 $\in \mathbb{R}^{D \times M}$

Tensor은 3차원 $\in \mathbb{R}^{D \times M \times \dots}$



Tensor

```

torchvision.transforms.ToTensor()(img)

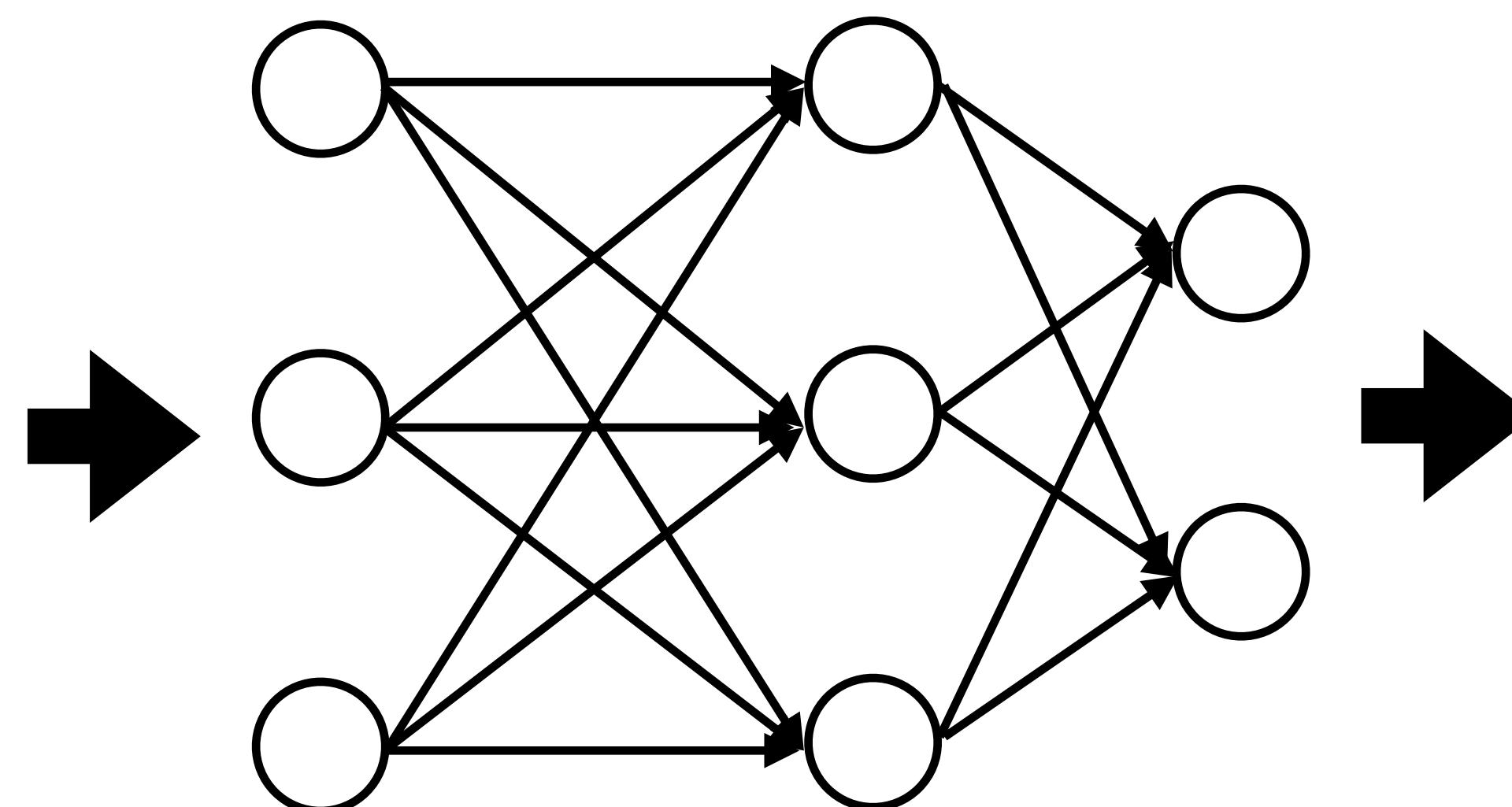
tensor([[1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       [1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       [1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       ...,
       [0.6627, 0.6588, 0.6471, ..., 1.0000, 1.0000, 1.0000],
       [0.6588, 0.6588, 0.6353, ..., 1.0000, 1.0000, 1.0000],
       [0.6627, 0.6588, 0.6471, ..., 1.0000, 1.0000, 1.0000]],

      [[1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       [1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       [1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       ...,
       [0.6549, 0.6510, 0.6275, ..., 1.0000, 1.0000, 1.0000],
       [0.6510, 0.6510, 0.6196, ..., 1.0000, 1.0000, 1.0000],
       [0.6510, 0.6431, 0.6314, ..., 1.0000, 1.0000, 1.0000]],

      [[1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       [1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       [1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000],
       ...,
       [0.6549, 0.6510, 0.6275, ..., 1.0000, 1.0000, 1.0000],
       [0.6510, 0.6510, 0.6196, ..., 1.0000, 1.0000, 1.0000],
       [0.6510, 0.6431, 0.6314, ..., 1.0000, 1.0000, 1.0000]],

      ...
    ]
  
```

Neural Network (Deep Learning Model)



Output

```

model(img)

tensor([0.9900, 0.0100])
  
```

Transforms

Deep Learning Pipeline: Post-processing

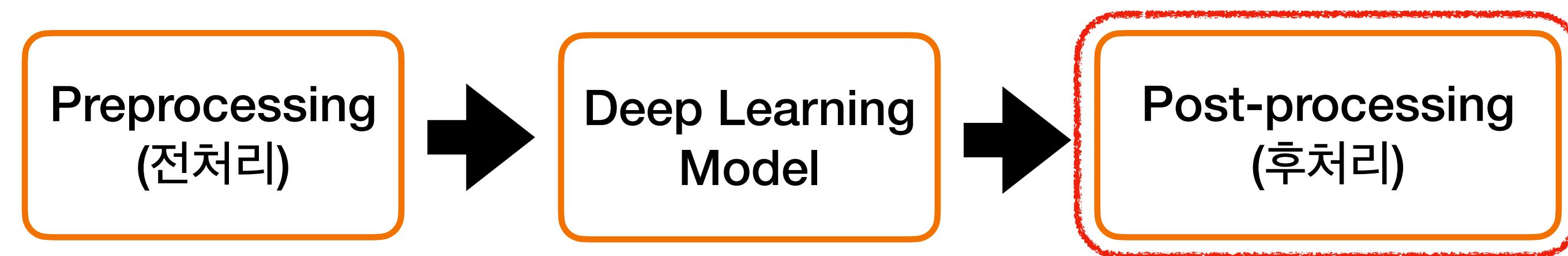


Model Prediction

```
model(img)  
tensor([0.9900, 0.0100])
```

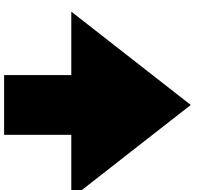
Transforms

Deep Learning Pipeline: Post-processing



Model Prediction

```
model(img)  
tensor([0.9900, 0.0100])
```



Cat!

비록 해당 예제에서는 간단하지만
더 복잡한 Postprocessing의
task도 많이 있다.

e.g. Object Detection,
Segmentation, NLP...

Transforms

다음 내용을 Jupyter Notebook에서 살펴보자:

- torchvision.transforms:
 - Compose
 - GaussianBlur
 - RandomAffine
- Lambda transforms

2-8. [실습] PyTorch로 구현해보는 Neural Network

PyTorch로 구현하는 Neural Network

Copyright©2023. Acadential. All rights reserved.

PyTorch에서 제공하는 `torch.nn`을 활용해서 Neural Network을 구현해보자!

먼저 모델의 “기본 뼈대”는 어떻게 되는지 살펴보자

PyTorch로 구현하는 Neural Network

PyTorch 모델의 기본 뼈대:

- `__init__` 함수:
 - NN을 구성하는 layer들을 명시하고 initialize한다.
- `forward` 함수
 - 입력에 대한 forward pass을 정의 한다.

```
class NeuralNetwork(nn.Module):  
    def __init__(self):  
        super(NeuralNetwork, self).__init__()  
        # Neural Network을 구성하는 layer들을  
        # initialize하는 부분  
  
    def forward(self, x):  
        # Neural Network의 forward pass을 정의하는 부분  
        # x은 input tensor
```

PyTorch로 구현하는 Neural Network

다음 내용을 Jupyter Notebook에서 살펴보자:

- Custom Neural Network 모델의 구현
 - `__init__` 함수
 - `forward` 함수
- `torchsummary`을 사용해서 모델의 summary 뽑기

Let's look at Jupyter Notebook!

2-9. Section 2 요약

Objective

학습 목표

- Deep Learning이란 무엇이고 어디에서 기원하는가?
- 어떤 문제를 풀려고 하는 것인가?
- Neural Network의 기본 구성은 어떻게, 어떻게 학습시키는가?

Section Summary

What is Deep Learning?

- Deep Learning

(Neural Network의 출력값) $\hat{Y} = NN(X)$ 가 (실제값) $Y = f(X)$ 에

최대한 잘 근사 ($\hat{Y} \approx Y$)하도록 학습하는 것.

- 기본적인 Neural Network의 구성

Input Layer + (한 개 이상의) Hidden Layer + Output Layer

- 각 Layer의 Neuron의 구성

가중치 weight w 와 활성화 함수 activation function σ

섹션 6. 활성 함수 (Activation Function)

Section Summary

What is Deep Learning?

- Neural Network가 학습되는 과정 = weight값이 최적화되는 과정

Gradient Descent (경사 하강)을 통한 **Loss function (손실 함수)** 값을 최소화하도록

weight 값을 최적화하여 점진적으로 모델의 예측 정확도를 높인다.

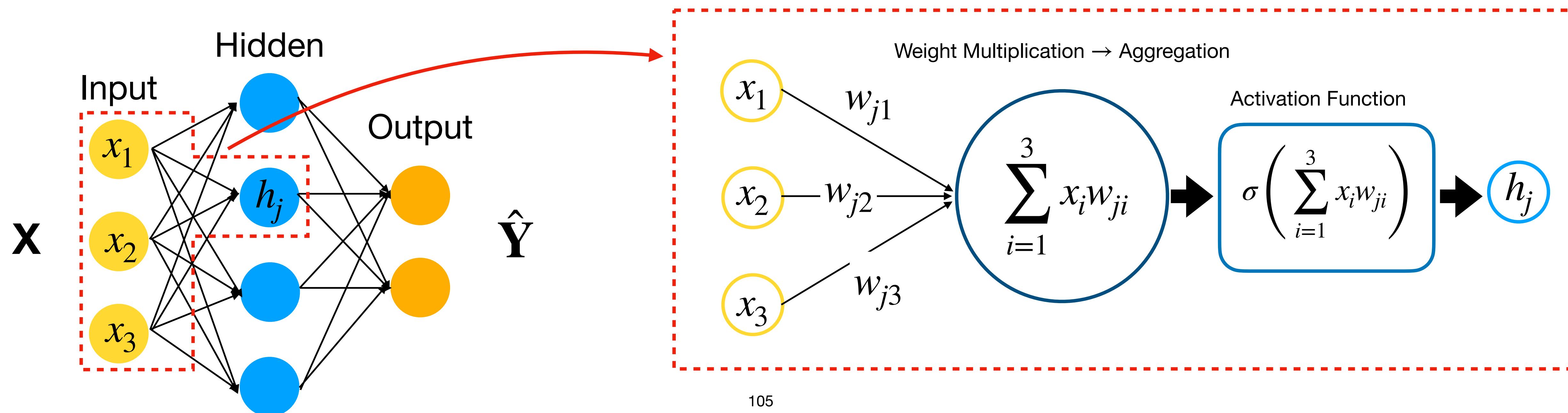
섹션 4. 경사 하강법 (Gradient Descent)

섹션 3. 손실 함수 (Loss Function)

Section Summary

What is Deep Learning?

- 각 뉴론은 Weight, Activation Function으로 구성되어 있다.
- $\sum_{i=1}^3 x_i w_{ji}$: 이전 Layer의 출력값 x_i 은 가중치 w_{ji} 에 곱해져서 합해진다. (weight multiplication → aggregation)
- $\sigma(\cdot)$: 합해진 값들은 activation function σ 을 통하여 해당 뉴런의 최종 출력값 h_j 를 구한다.



Next Up!

Next Up!

- 그렇다면 Loss function (손실함수) 가 뭔지 살펴보자!