DeepIntoDeep

ML&DL Basics and PyTorch

발표자: 전성후

ML&DL Basics and PyTorch

전성후

Artificial Intelligence in Korea University(AIKU)

Department of Computer Science and Engineering, Korea University

강의자 소개

전성후

- **고려대학교 컴퓨터학과 22학번**

  2023 정보대학 iNThon 데이터톤 트랙 대상

- **고려대학교 CVLAB 학부연구생 (23.12-)**

  Under Prof. Seungryong Kim

  연구 분야 : Video Motion Customization, 3DGS

- **AIKU (22.09-)**

  0기 주니어, 1기 시니어, 2기 기술학술부원,

  3기 학회장, 4기 활동 OB

- **GDSC KU (22.09-)**

  1,2기 | AI Core, 3기 | AI/DevRel Core

Contents

Contents

- Optimization Technique
  - Batch Process and Epoch
  - Stochastic Gradient Descent
  - Advanced Optimizers (SGD, Adam, AdamW)
  - Learning Rate Schedulers
- Introduction to Neural Network
  - Perceptron and Nonlinearity
  - Multi-Layered Perceptron
  - Backpropagation
  - Training Neural Network
- Data Processing
  - Scaling Law
  - Augmentation
  - Data Preprocessing

PyTorch (Hands-on Session)

- Tensor Manipulation
- Dataset & Dataloader
- Backward & Optimization
- Training Pipeline

AI, ML, DL Overview

“What is AI?”

What is AI?

What is AI?

인공지능은 인간의 지능을 모방하여 컴퓨터가 인간처럼 생각하고

학습하며 문제를 해결할 수 있게 만드는 기술입니다.

AI는 기계가 지능적인 작업을 수행하도록 하는 광범위한 분야입니다.

What is AI?

인공지능은 인간의 지능을 모방하여 컴퓨터가 인간처럼 생각하고

학습하며 **문제를 해결할 수 있게** 만드는 기술입니다.

AI는 기계가 **지능적인 작업**을 수행하도록 하는 광범위한 분야입니다.

What is AI?

인공지능은 인간의 지능을 모방하여 컴퓨터가 인간처럼 생각하고

학습하며 문제를 해결할 수 있게 만드는 기술입니다.

AI는 기계가 지능적인 작업을 수행하도록 하는 광범위한 분야입니다.

인간의 지능을 요구하는 영역의 문제들을 처리하기 위한 도구

What is AI?

인간의 지능을 요구하는 영역의 문제들을 처리하기 위한 도구

바둑/체스

이미지 인식 및 생성

챗봇

AI vs. ML vs. DL

셋의 공통점은?

바둑/체스

이미지 인식 및 생성

챗봇

AI vs. ML vs. DL

셋의 공통점은? 딥러닝 기반 모델

AlphaGo

Diffusion Models

GPTs

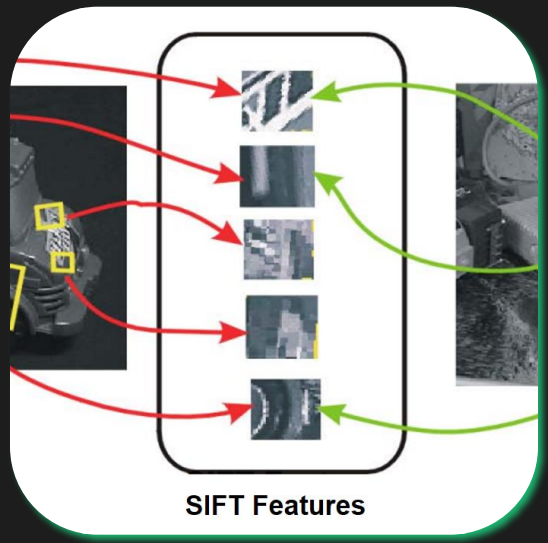
AI vs. ML vs. DL

Deep Learning is Not Everything!

딥러닝은 인간의 지능을 요구하는 영역의 문제들을 처리하기 위한 하나의 알고리즘일 뿐이다.

아주 효율적이고 효과적일 뿐!

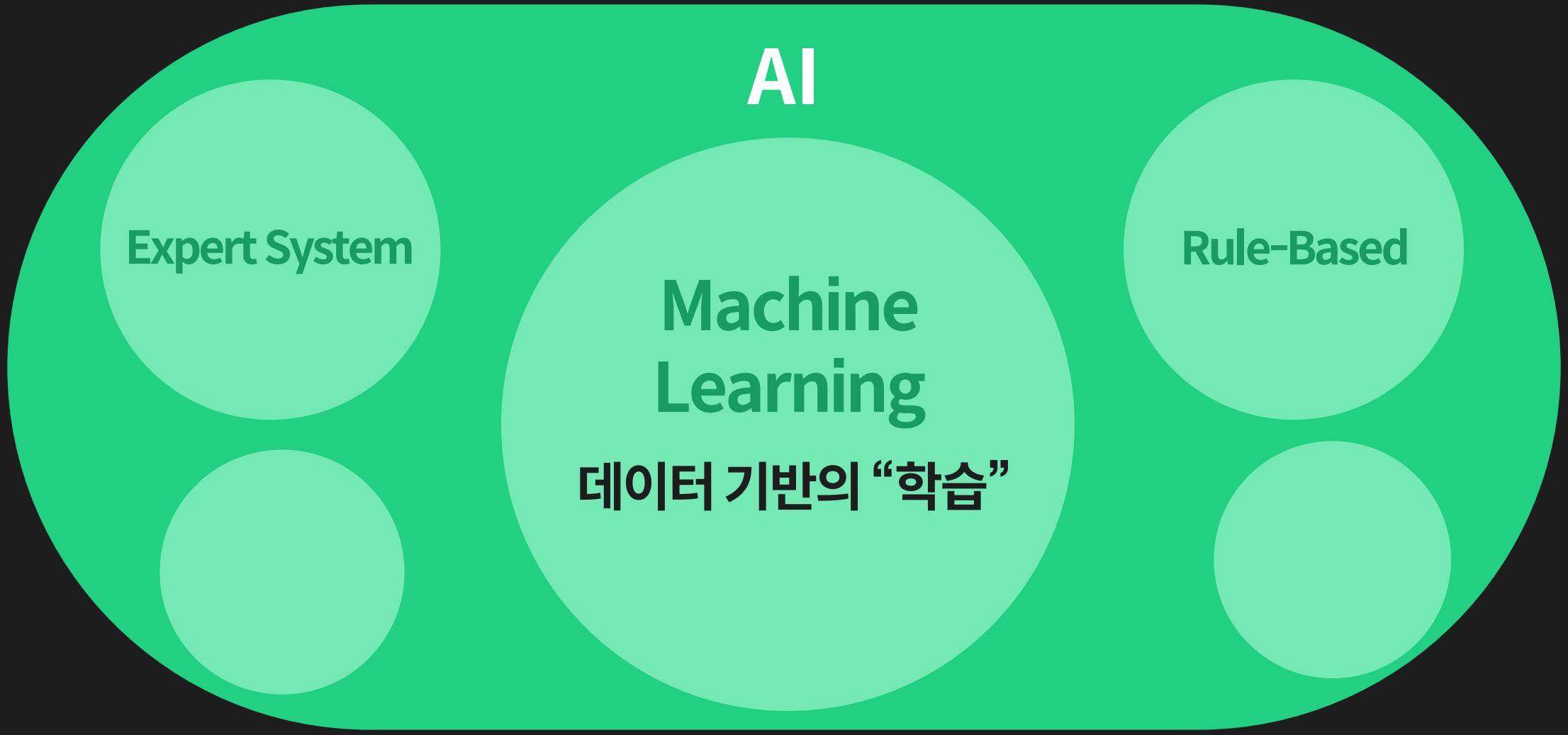
Deep Blue (1996)



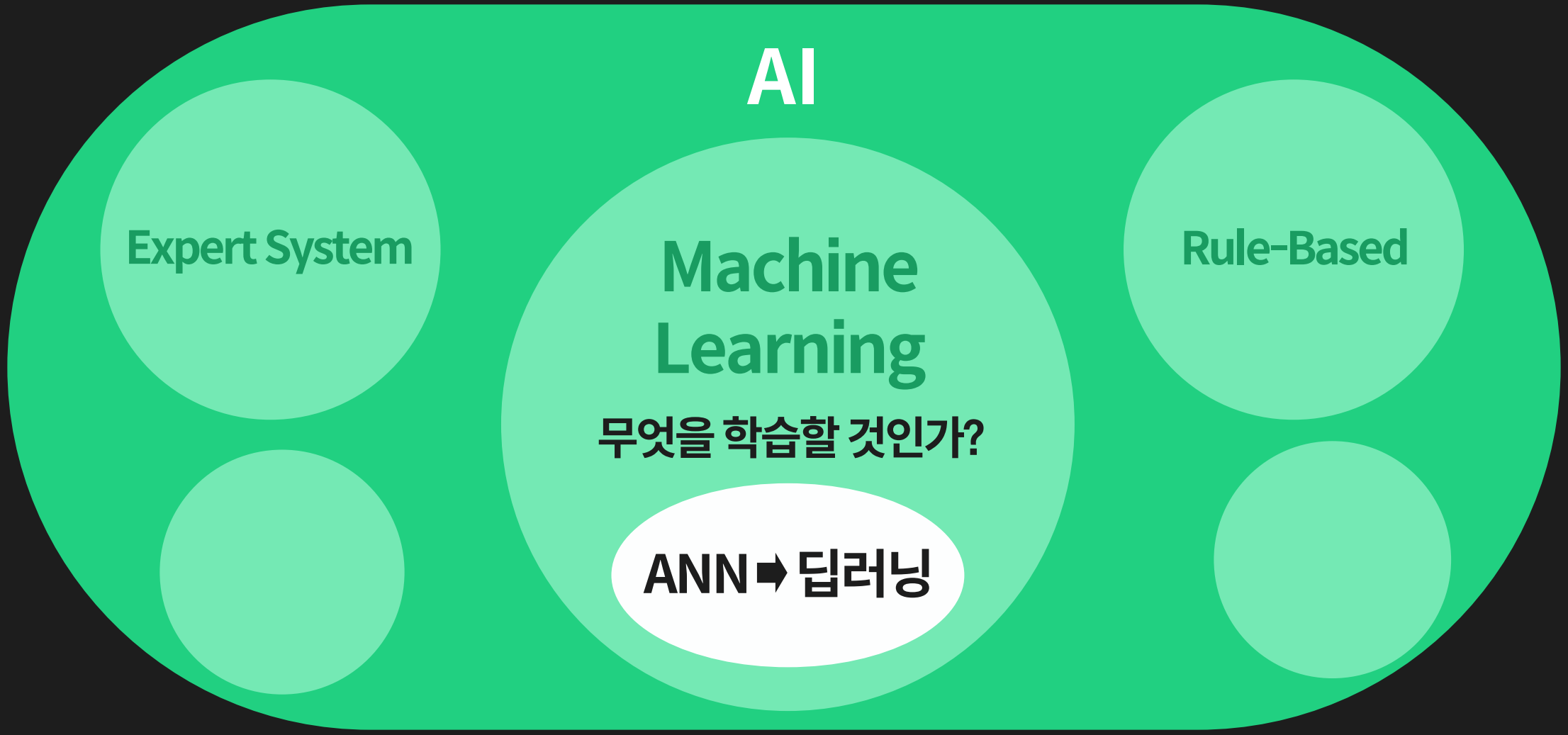
SIFT (2004)

심심이 (2002)

AI vs. ML vs. DL



AI vs. ML vs. DL


AI
Expert System
Machine Learning
무엇을 학습할 것인가?
ANN ➡ 딥러닝
Rule-Based

AI vs. ML vs. DL

AI (Artificial Intelligence)

인간의 지능을 요구하는 영역의 문제들을 처리하기 위한 도구 (= 알고리즘)

ML (Machine Learning)

데이터를 기반으로 모델을 '학습'하여 특정 작업을 수행하는 것

DL (Deep Learning)

ML 모델 중 하나인 '인공신경망'(ANN, Artificial Neural Network)의 방법론

모델을 더 복잡하고, 깊게 만들어 학습시키는 방법론

(Recap?) Definition of Algorithm

입력 : 알고리즘은 0 또는 그 이상의 외부에서 제공된 자료가 존재해야한다.

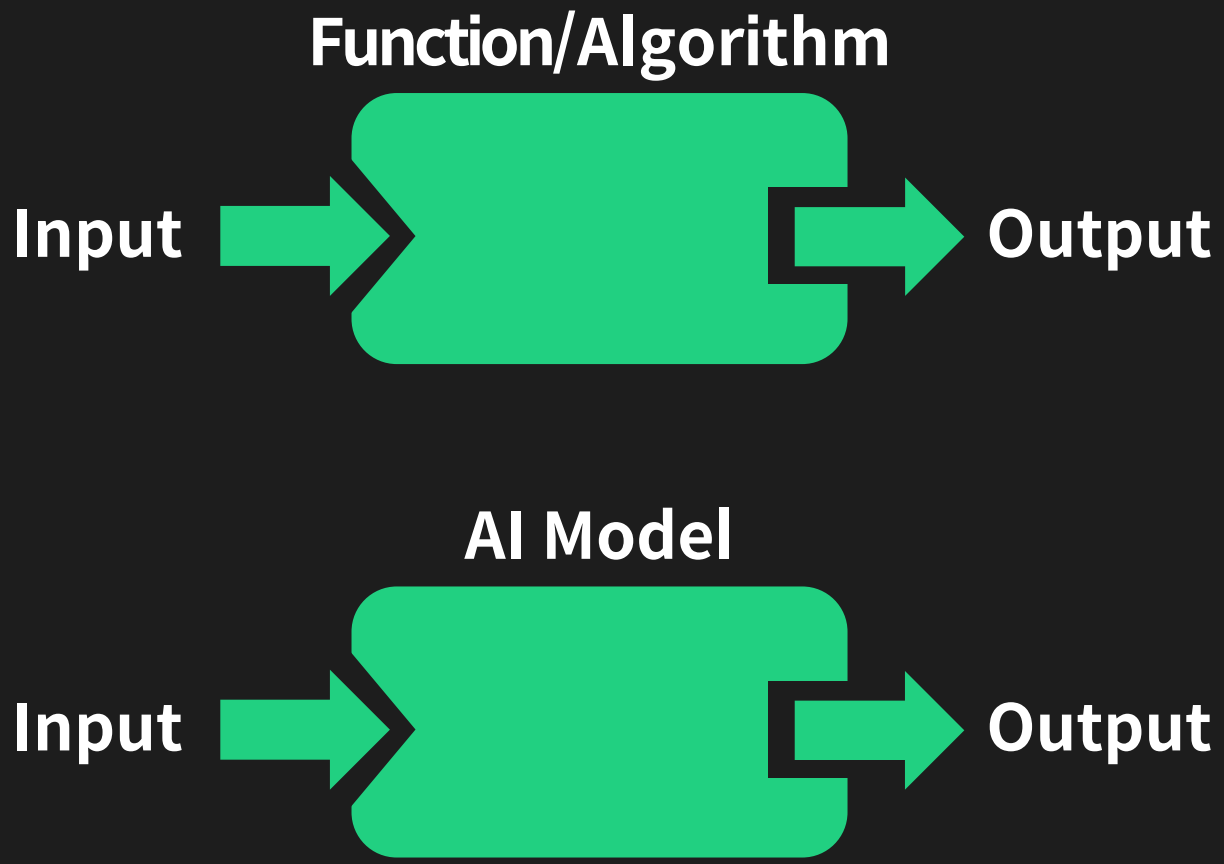
출력 : 알고리즘은 최소 1개 이상의 결과를 가져야한다.

명확성 : 알고리즘의 각 단계는 명확하여 애매함이 없어야 한다.

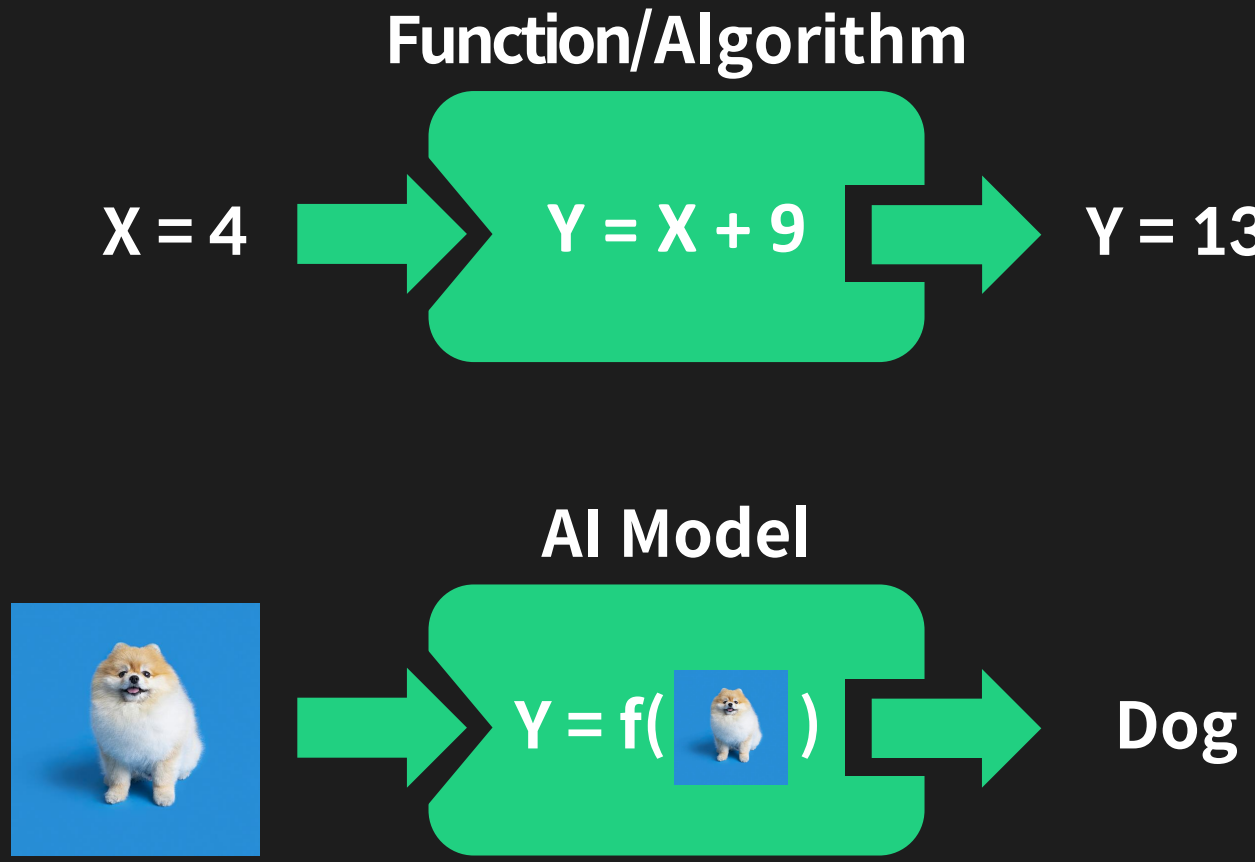
유한성 : 알고리즘은 단계들을 유한한 횟수로 거친 후 문제를 해결하고 종료해야 한다.

효과성 : 알고리즘의 모든 연산들은 사람이 종이와 연필을 이용하여 유한한 시간 안에 정확하게 수행할 수 있을 정도로 충분히 단순해야 한다.

Tasks of AI/ML/DL

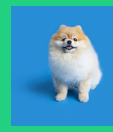


Tasks of AI/ML/DL



Tasks of AI/ML/DL

Discriminitive Model

Y = f()

Dog 80%
Cat 20%

Generative Model (Conditional)

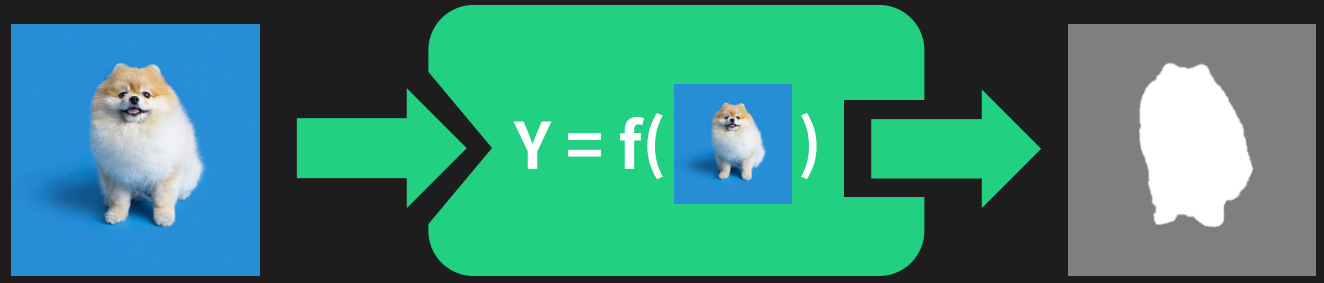
A dog,
blue background

Y = g(A dog, ⋯)

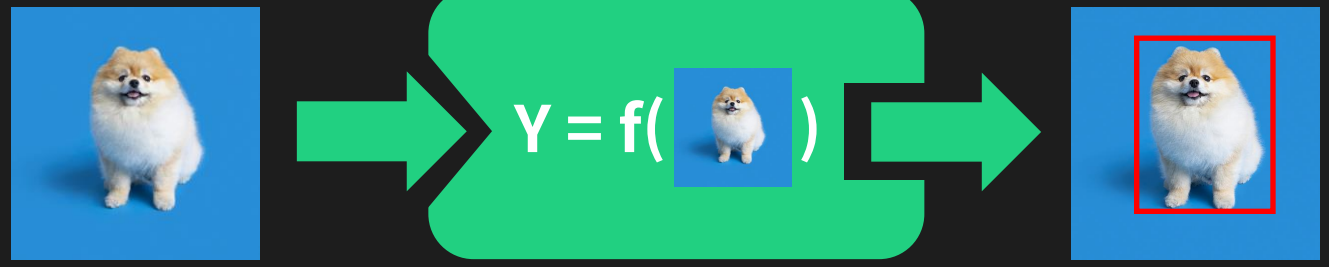
Tasks of AI/ML/DL

Computer Vision (CV)

Segmentation



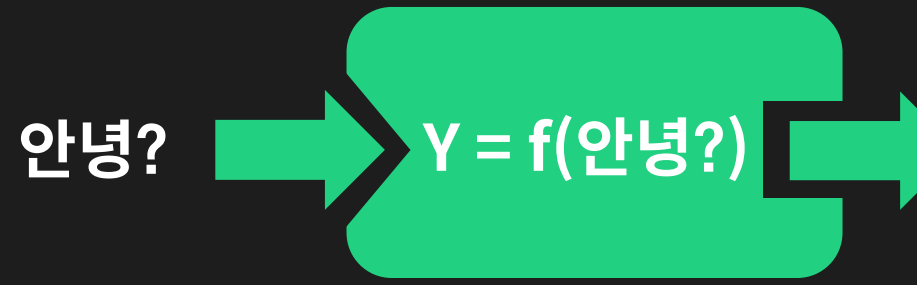
Object Detection



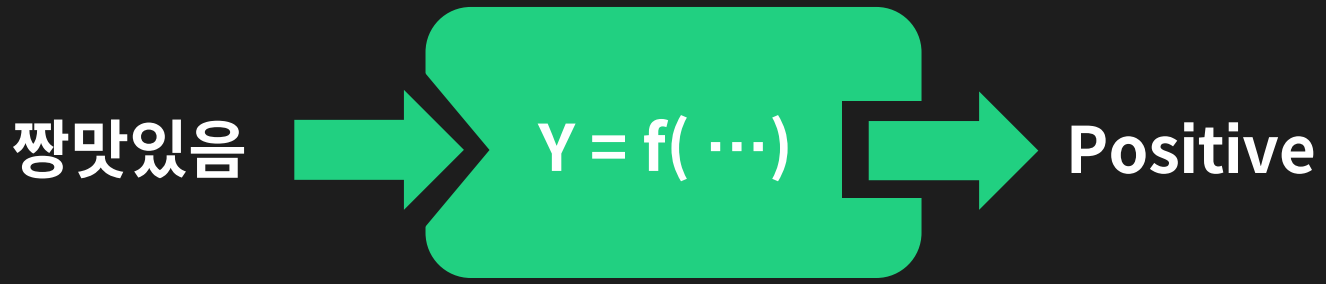
Tasks of AI/ML/DL

Natural Language Processing (NLP)

Machine Translation

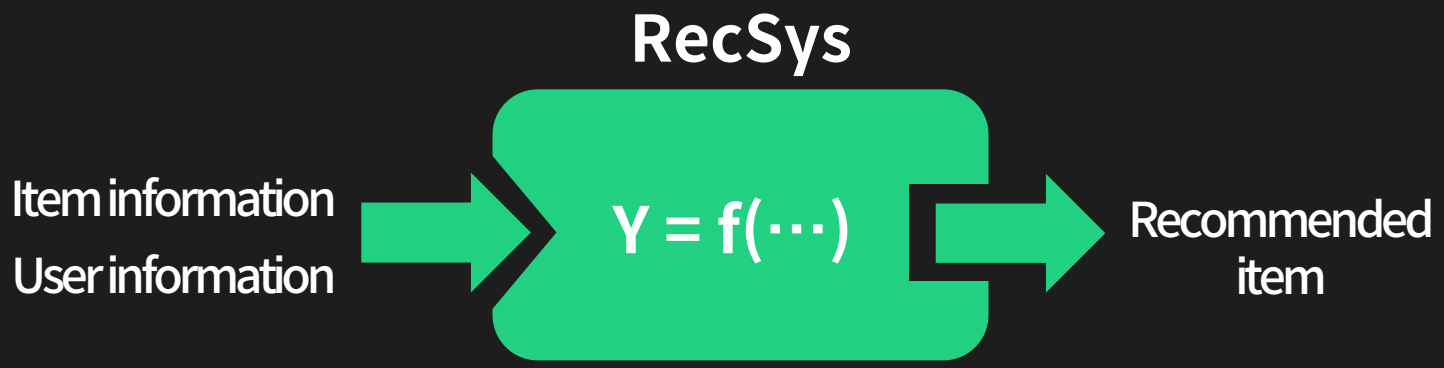


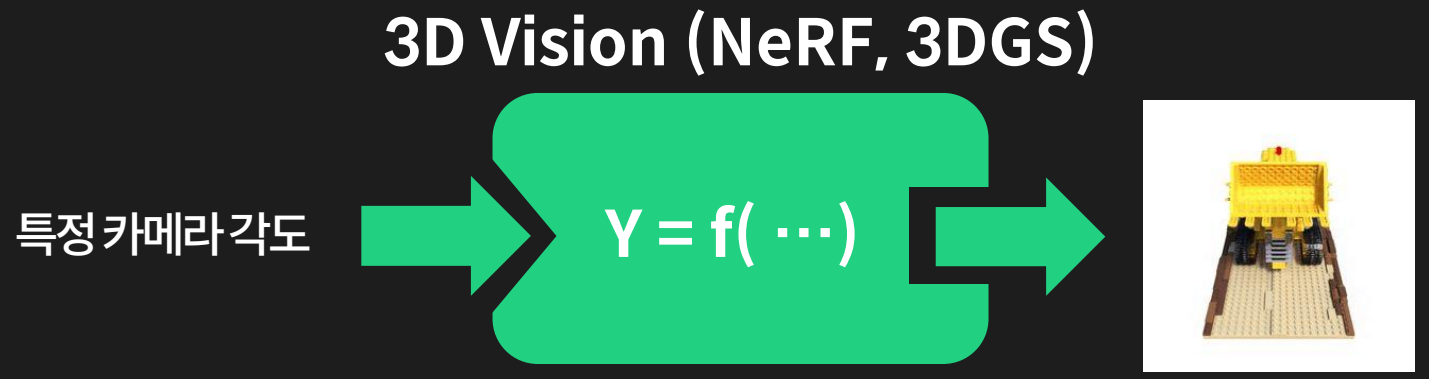
Sentiment Analysis



Tasks of AI/ML/DL

And more..

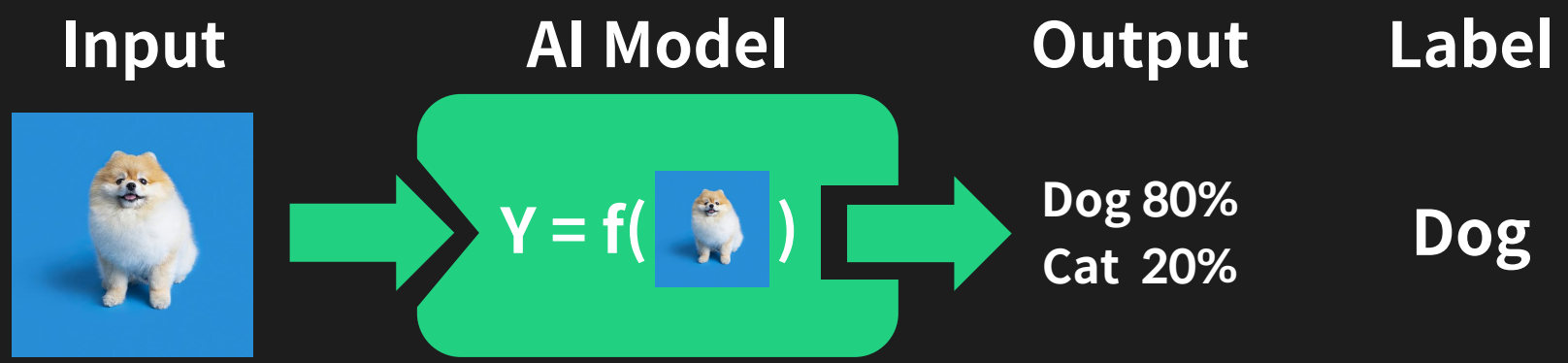




Elements of DL/ML

알고있는 몇 개의 Sample(Input/Label)로,

Input과 Output 사이의 관계를 잘 표현하는 모델를 찾자!



Elements of DL/ML

알고있는 몇 개의 Sample(Input/Label)로,

Input과 Output 사이의 관계를 잘 표현하는 모델를 찾자!

1. Input을 어떻게 AI Model에 넣어줄 것인가?
2. 모델(함수)의 구조를 어떻게 구성할 것인가?
3. 모델이 잘 한다는 것을 어떻게 평가할 수 있는가?
4. 잘 하는 모델을 어떻게 찾을 수 있는가?

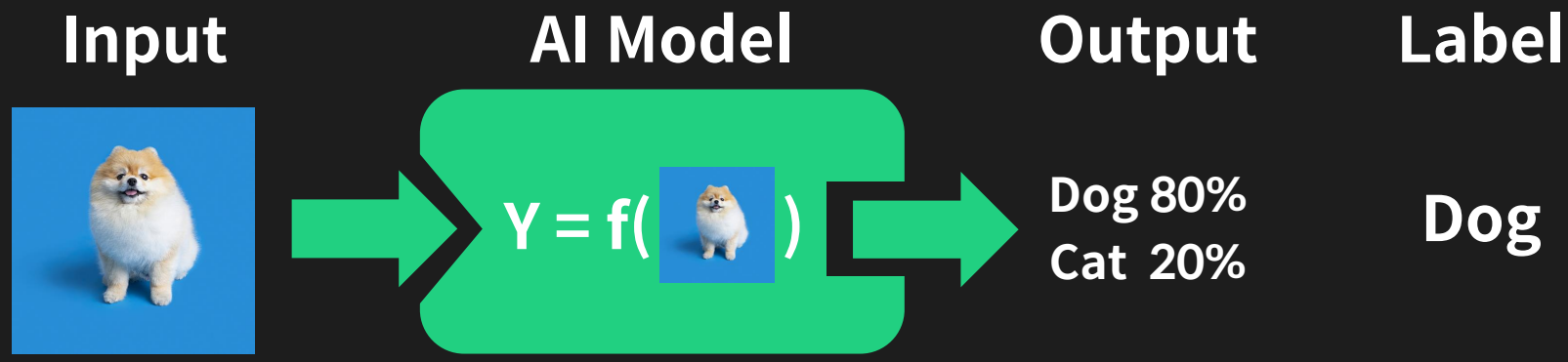
Elements of DL/ML

알고있는 몇 개의 Sample(Input/Label)로,

Input과 Output 사이의 관계를 잘 표현하는 모델를 찾자!

1. Input을 어떻게 AI Model에 넣어줄 것인가? — Dataset (Preprocessing, …)
2. 모델(함수)의 구조를 어떻게 구성할 것인가? — Model Architecture (RNN, CNN, …)
3. 모델이 잘 한다는 것을 어떻게 평가할 수 있는가? — Loss / Evaluation (MSE, MAE, …)
4. 잘 하는 모델을 어떻게 찾을 수 있는가? — Optimization (SGD, Adam, …)

Elements of DL/ML



1. Input을 "어떻게" AI Model에 넣어줄 것인가?

Preprocessing: 모든 데이터는 컴퓨터가 이해할 수 있는 "수"의 형태로 전처리 되어야 한다.

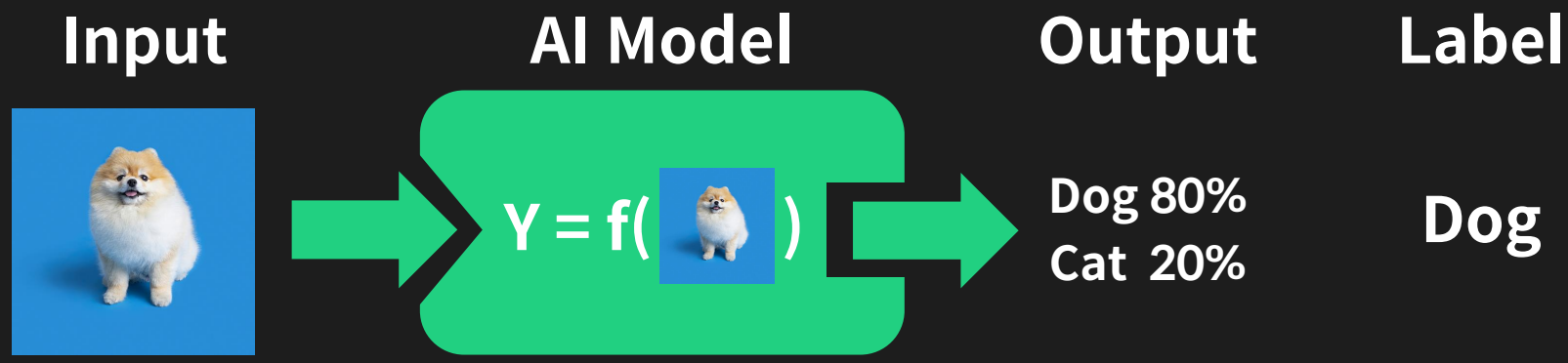
Representation:

Augmentation:

RGB

{{0,0,0.8},
{0,0,0.8}, ...}

Elements of DL/ML

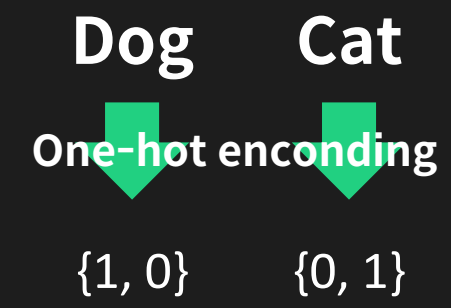


1. Input을 "어떻게" AI Model에 넣어줄 것인가?

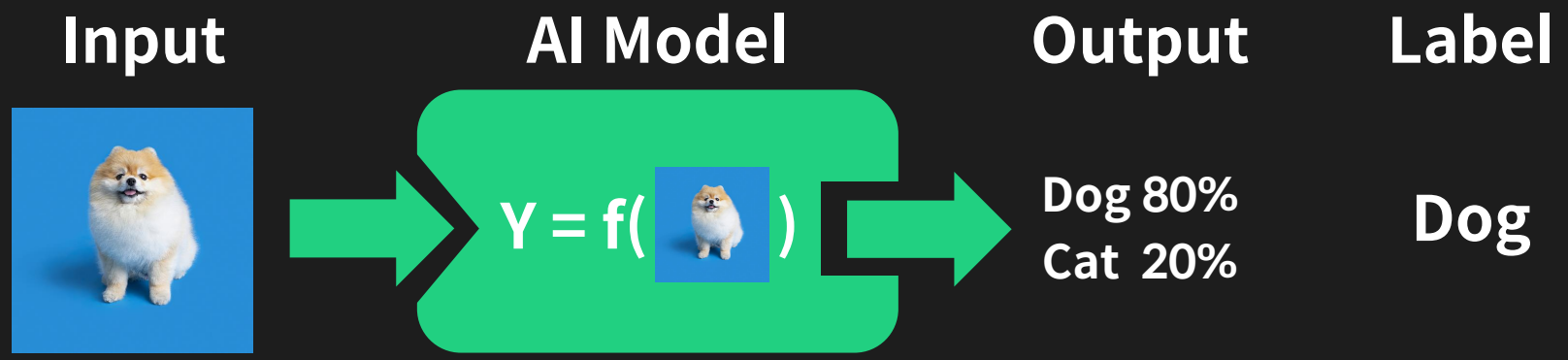
Preprocessing: 모든 데이터는 컴퓨터가 이해할 수 있는 "수"의 형태로 전처리 되어야 한다.

Representation:

Augmentation:



Elements of DL/ML



1. Input을 "어떻게" AI Model에 넣어줄 것인가?

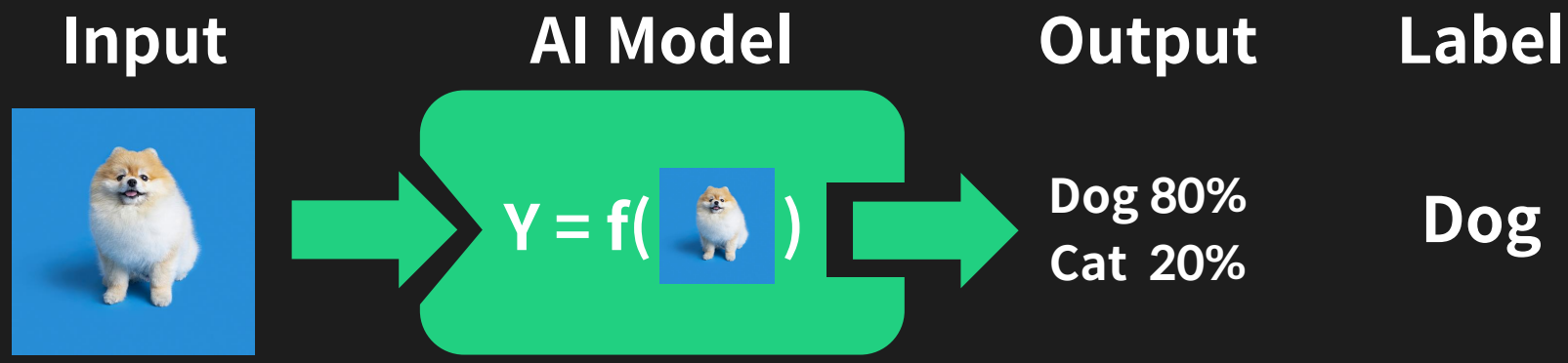
Preprocessing: 모든 이해할데이터는 컴퓨터가 수 있는 "수"의 형태로 전처리 되어야 한다.

Representation: 데이터를 더 효율적으로 "표현"할 수 있다면, 모델을 더 단순하게 만들 수 있다.

Augmentation:

{귀 : 2개
털: 있음
배경 : 파란색}

Elements of DL/ML



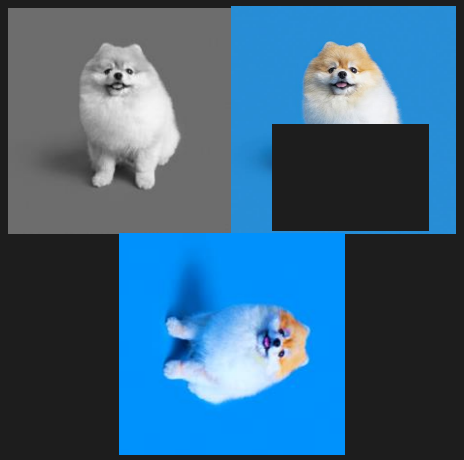
1. Input을 "어떻게" AI Model에 넣어줄 것인가?

Preprocessing: 모든 이해할데이터는 컴퓨터가 수 있는 "수"의 형태로 전처리 되어야 한다.

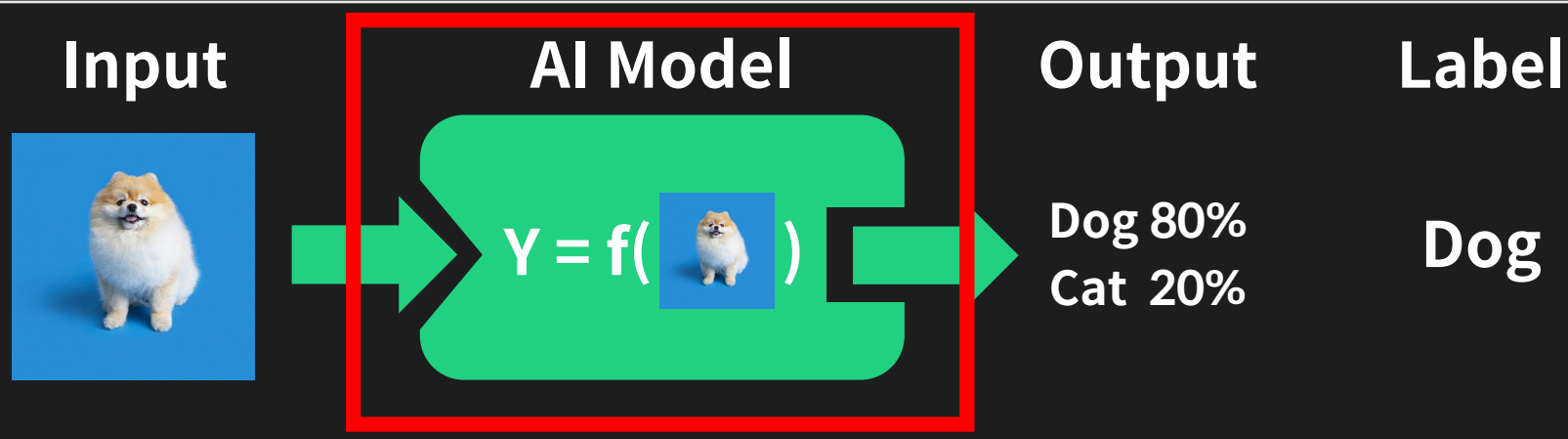
Representation: 데이터를 더 효율적으로 "표현"할 수 있다면, 모델을 더 단순하게 만들 수 있다.

Augmentation: Input 데이터를 변형하여 더 많은 데이터셋을 가지는 효과를 얻을 수 있다.

-> 더 좋은 일반화 성능의 달성



Elements of DL/ML



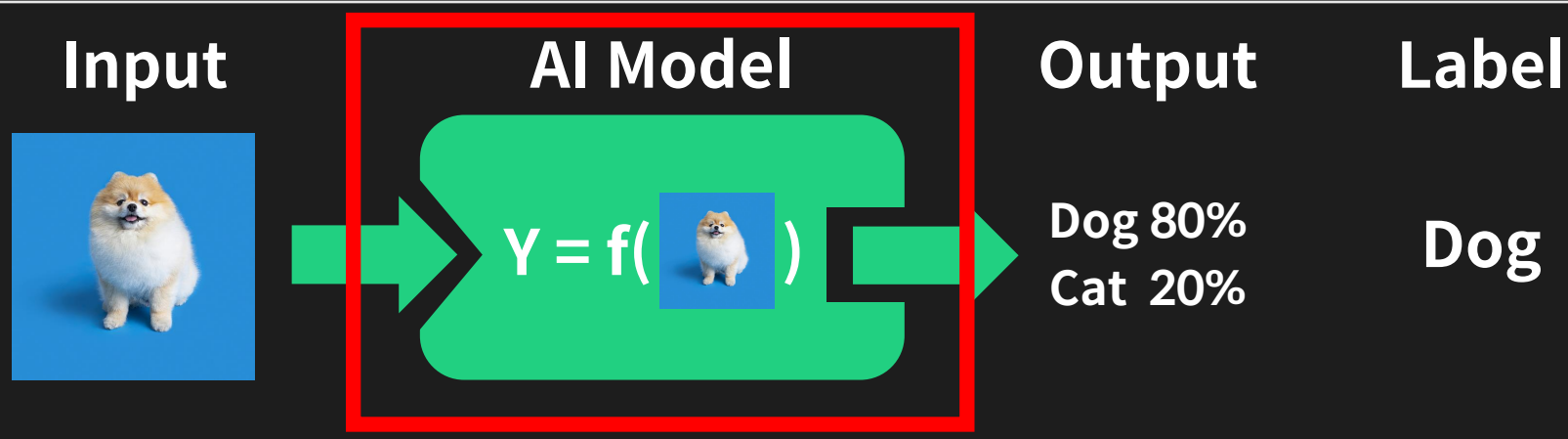
2. 모델(함수)의 구조를 어떻게 구성할 것인가?

알고있는 몇 개의 Sample(Input/Label)로,

Input과 Output 사이의 관계를 잘 표현하는 모델를 찾자!

- 모델(함수)를 찾기 위해서는, 데이터에 대한 최소한의 가정이 필요하다.

Elements of DL/ML

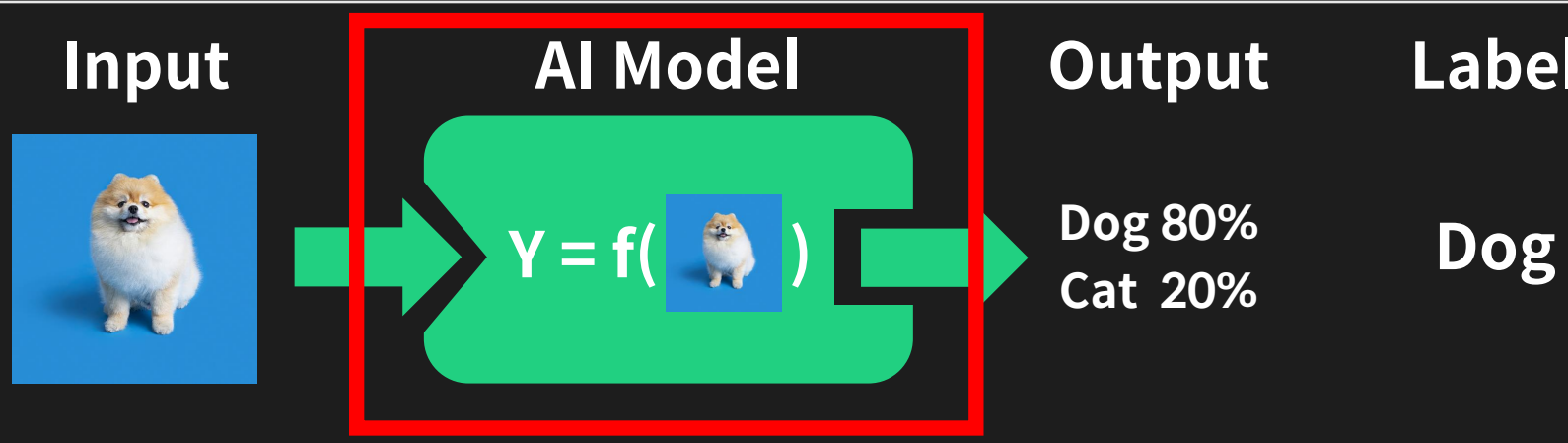


2. 모델(함수)의 구조를 어떻게 구성할 것인가?

- 모델(함수)를 찾기 위해서는 데이터에 대한 최소한의 가정이 필요하다. (Inductive Bias)
- 집의 크기와 집값은 선형적인 관계일 것이라는 가정, 즉,

$$(\text{집값}) = a \times (\text{집 크기}) + b$$

Elements of DL/ML



2. 모델(함수)의 구조를 어떻게 구성할 것인가?

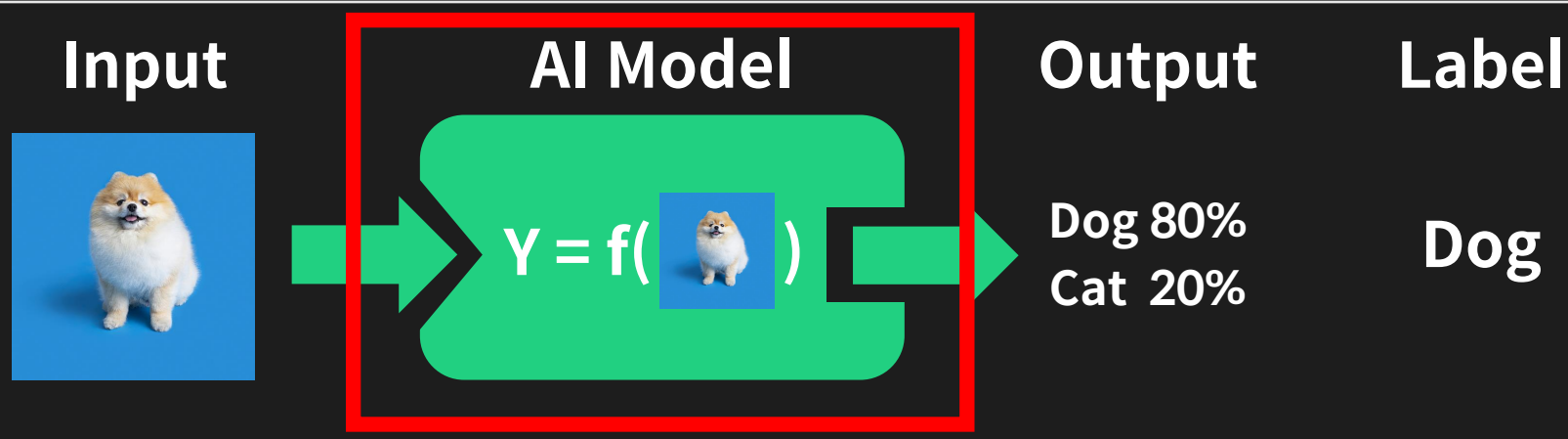
- 모델(함수)를 찾기 위해서는 데이터에 대한 최소한의 가정이 필요하다. (Inductive Bias)
- 집의 크기와 집값은 선형적인 관계일 것이라는 가정, 즉,

$$y = ax + b$$

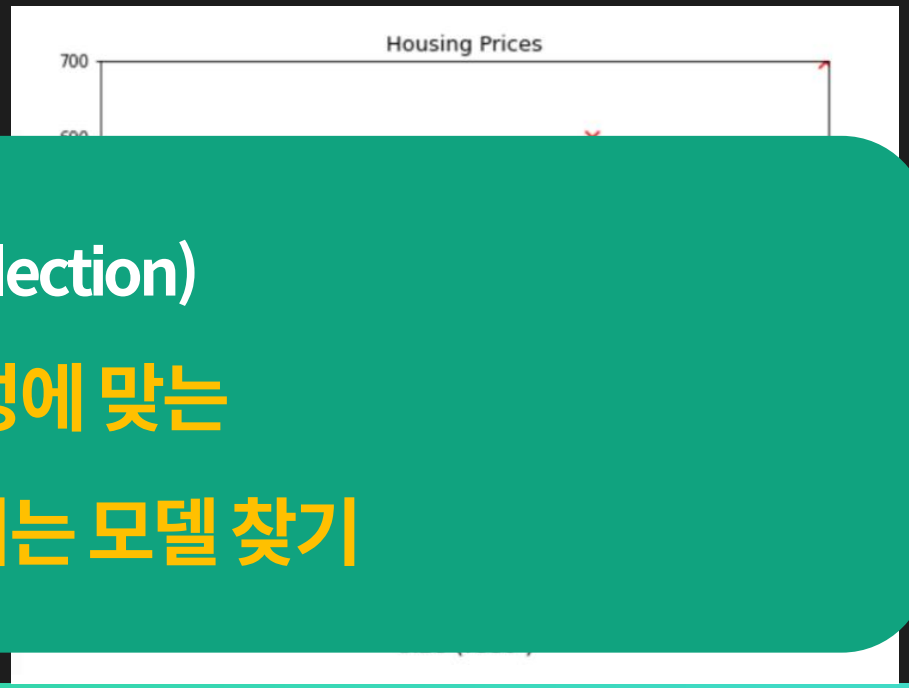
모델

"선형적 관계"라는 가정은 유지하며,
파라미터(a, b)만을 조정하며
최적의 모델을 찾는다.

Elements of DL/ML



2. 모델(함수)의 구조를 어떻게 구성할 것인가?

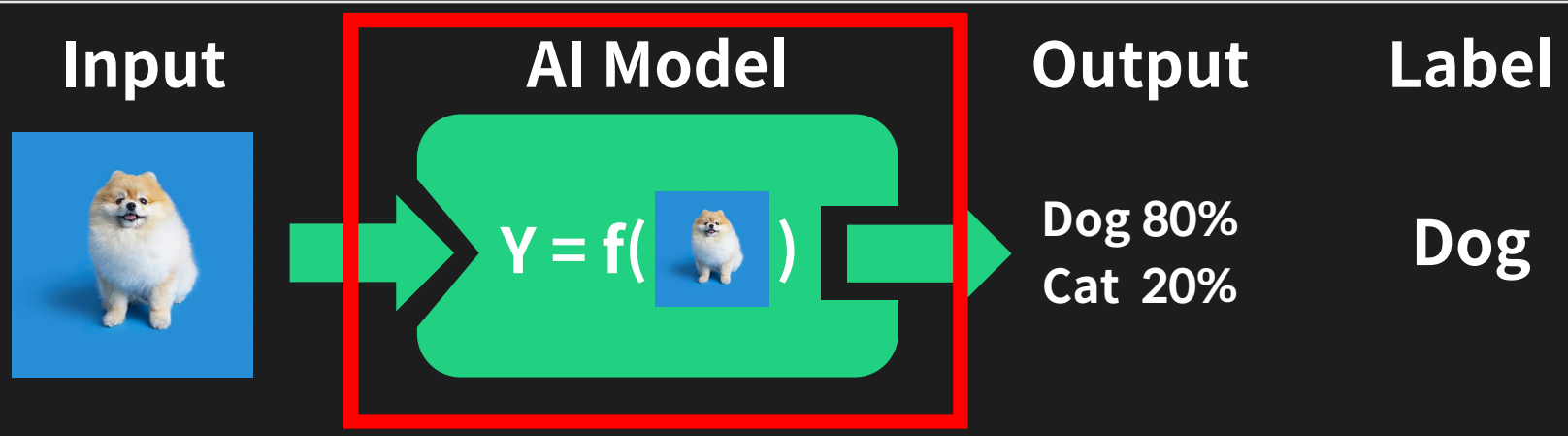


모델의 구조 찾기 (Model Selection)

= Task나 Dataset의 특성에 맞는

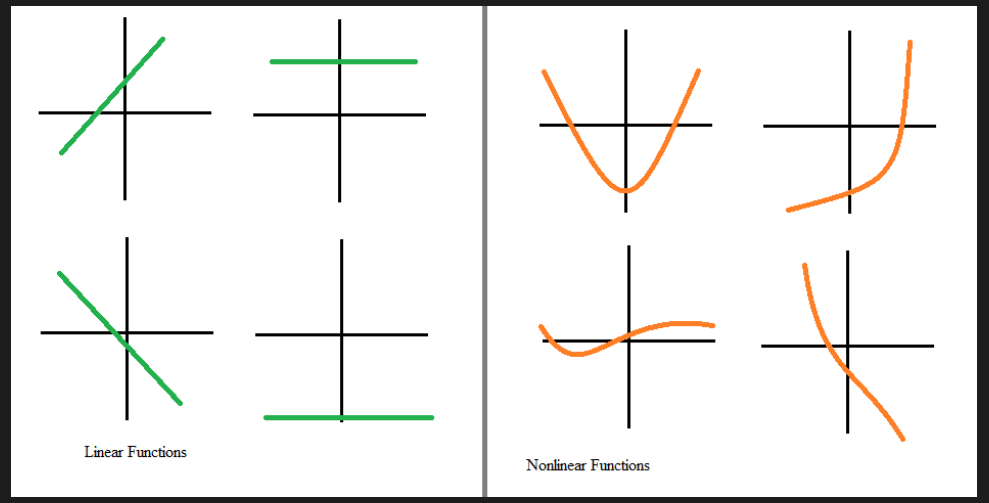
적당한 Inductive Bias를 가지는 모델 찾기

Elements of DL/ML

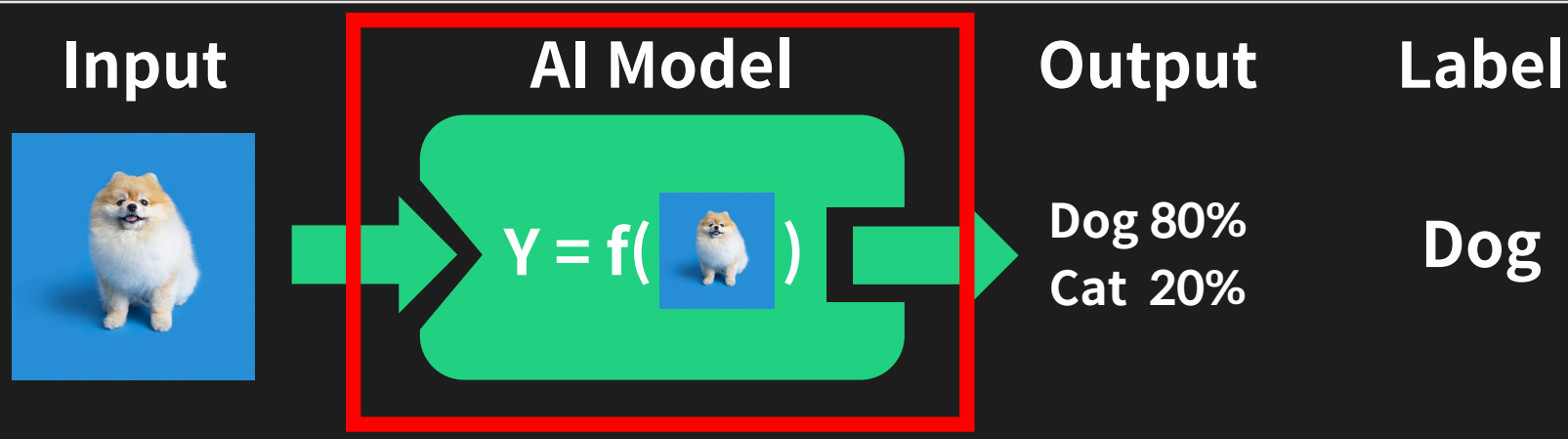


2. 모델(함수)의 구조를 어떻게 구성할 것인가?

- Linear/Nonlinear function
- Perceptron / MLP
- CNN
- RNN/LSTM
- Transformers

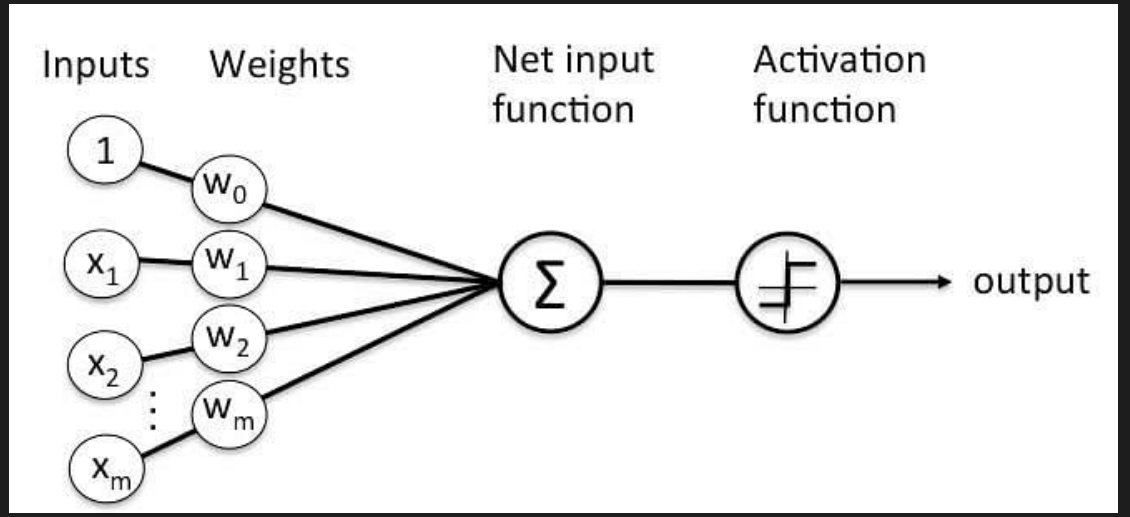


Elements of DL/ML

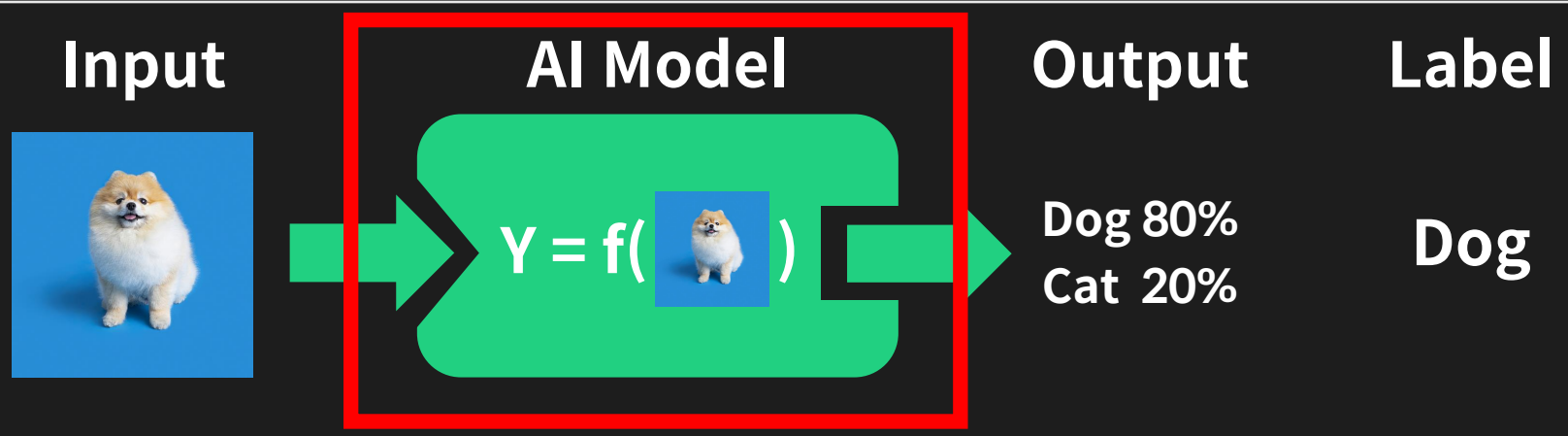


2. 모델(함수)의 구조를 어떻게 구성할 것인가?

- Linear/Nonlinear function
- Perceptron / MLP
- CNN
- RNN/LSTM
- Transformers

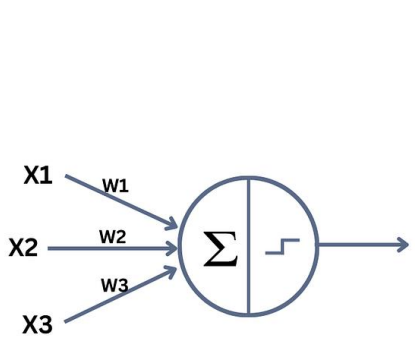


Elements of DL/ML

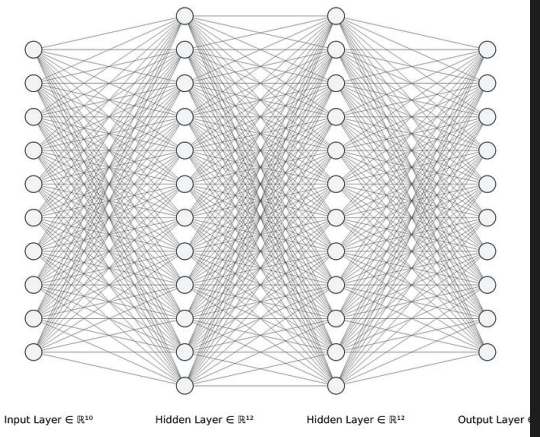


2. 모델(함수)의 구조를 어떻게 구성할 것인가?

- Linear/Nonlinear function
- Perceptron / MLP
- CNN
- RNN/LSTM
- Transformers

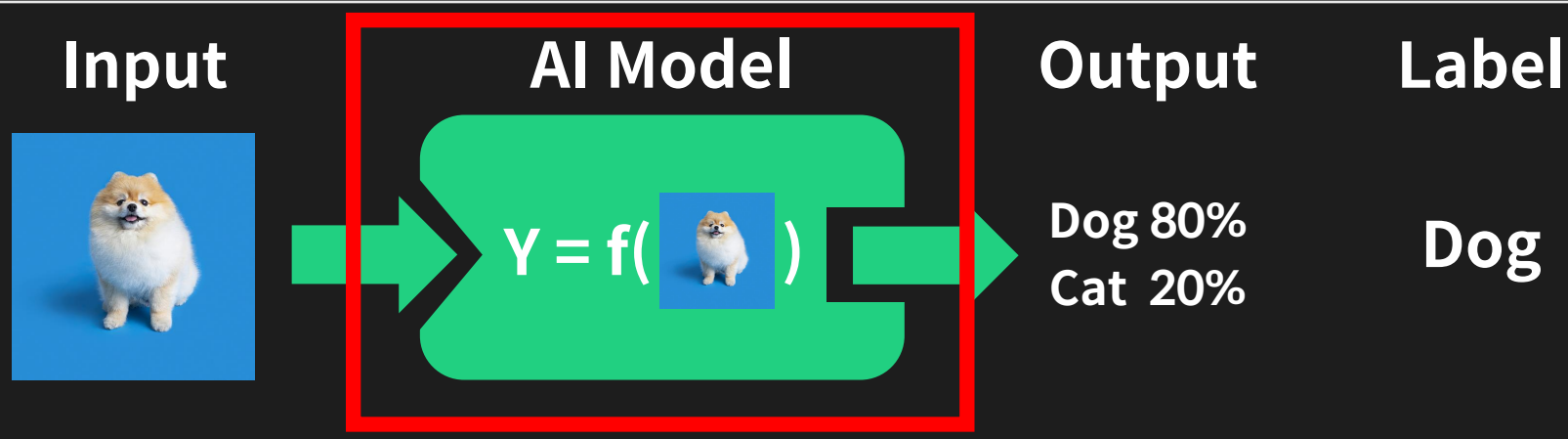


Single-layer perceptron



Multi-layer perceptron

Elements of DL/ML



2. 모델(함수)의 구조를 어떻게 구성할 것인가?

- Linear/Nonlinear function
- Perceptron / MLP
- CNN
- RNN/LSTM
- Transformers

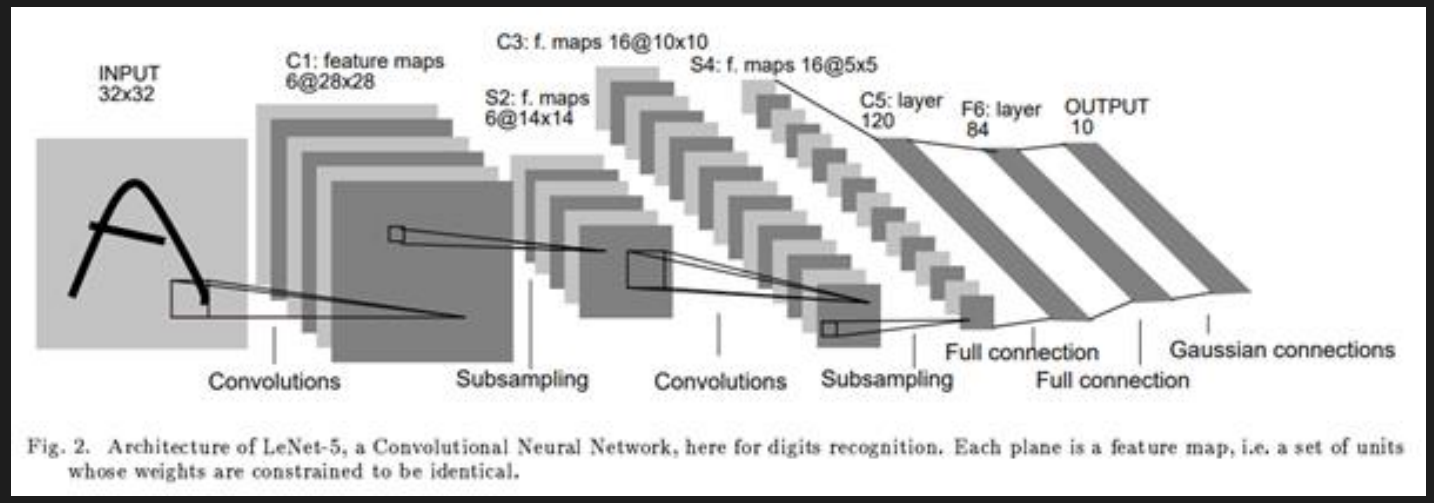
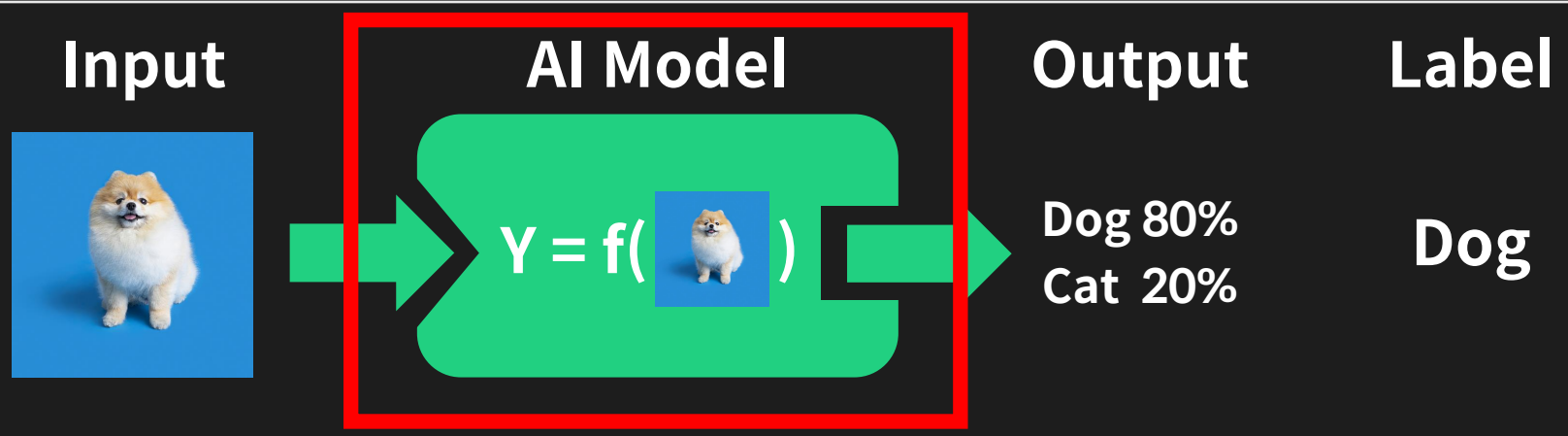


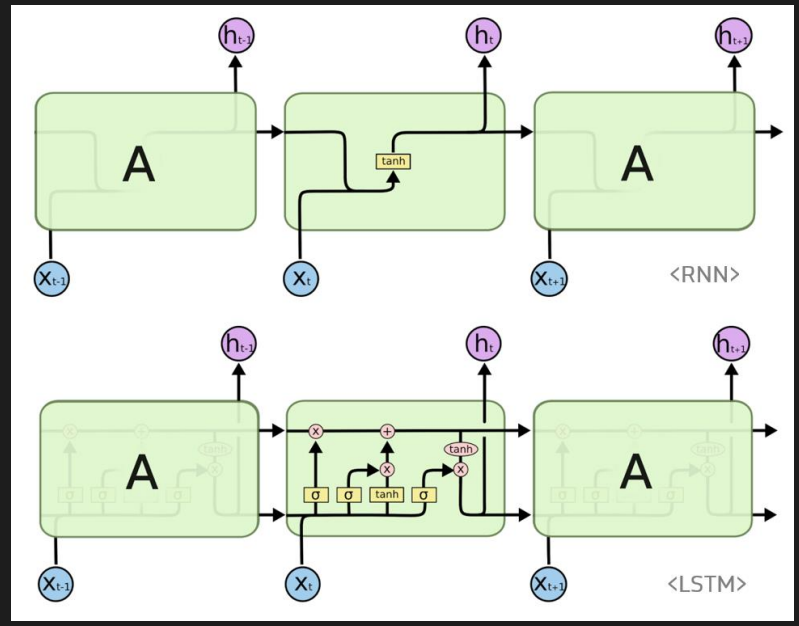
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Elements of DL/ML

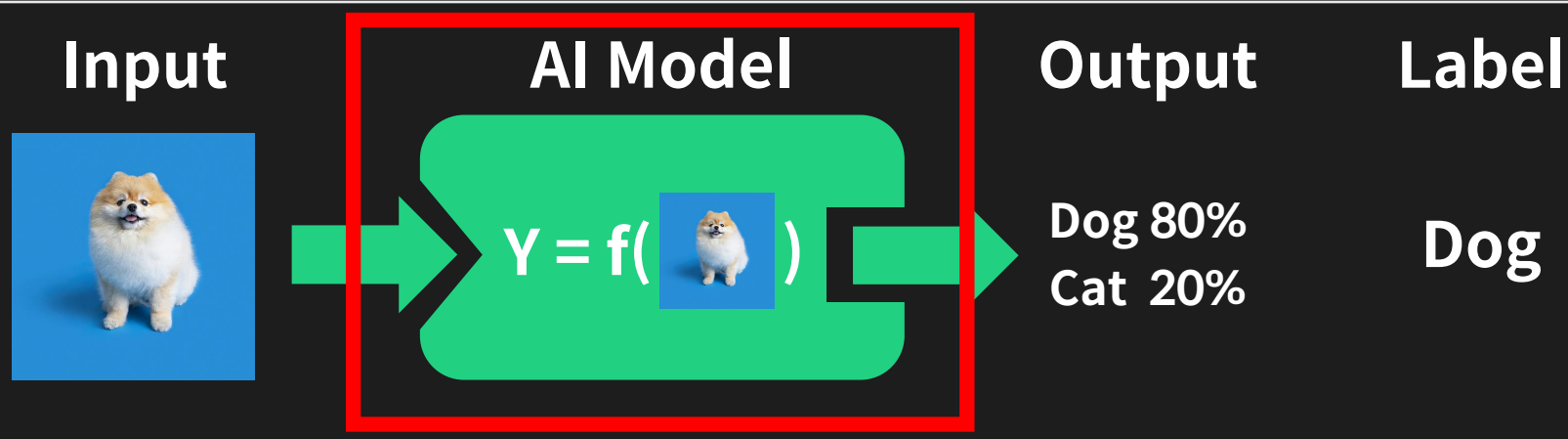


2. 모델(함수)의 구조를 어떻게 구성할 것인가?

- Linear/Nonlinear function
- Perceptron / MLP
- CNN
- RNN/LSTM
- Transformers

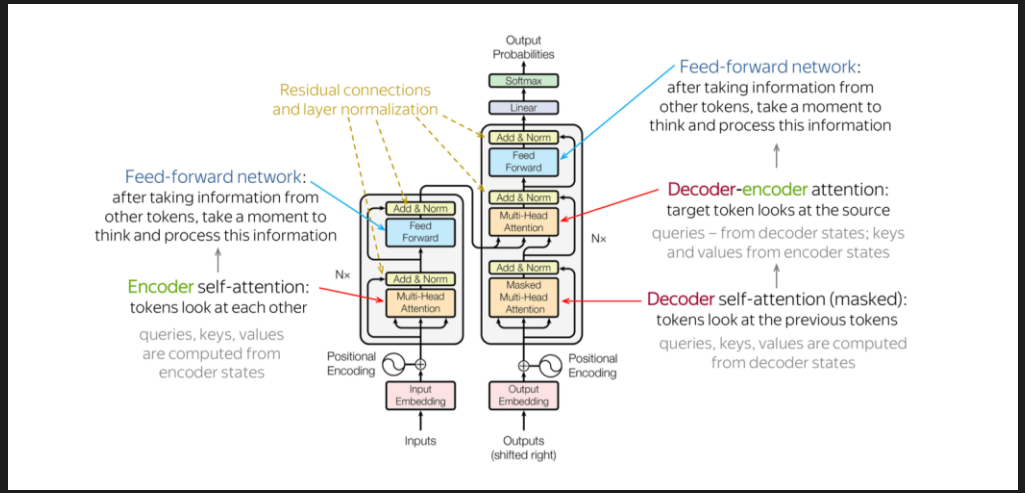


Elements of DL/ML

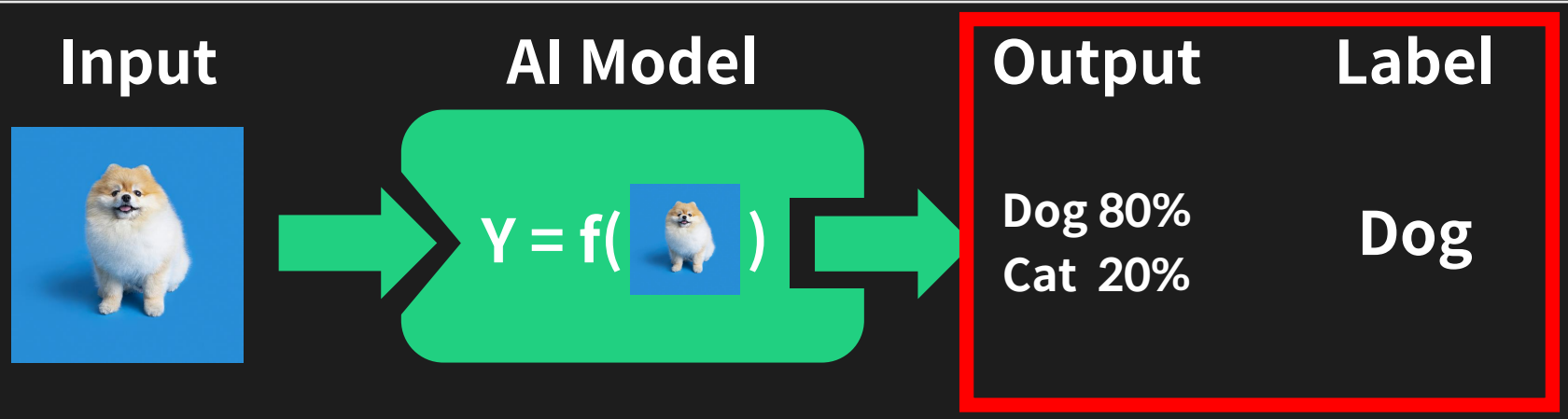


2. 모델(함수)의 구조를 어떻게 구성할 것인가?

- Linear/Nonlinear function
- Perceptron / MLP
- CNN
- RNN/LSTM
- Transformers



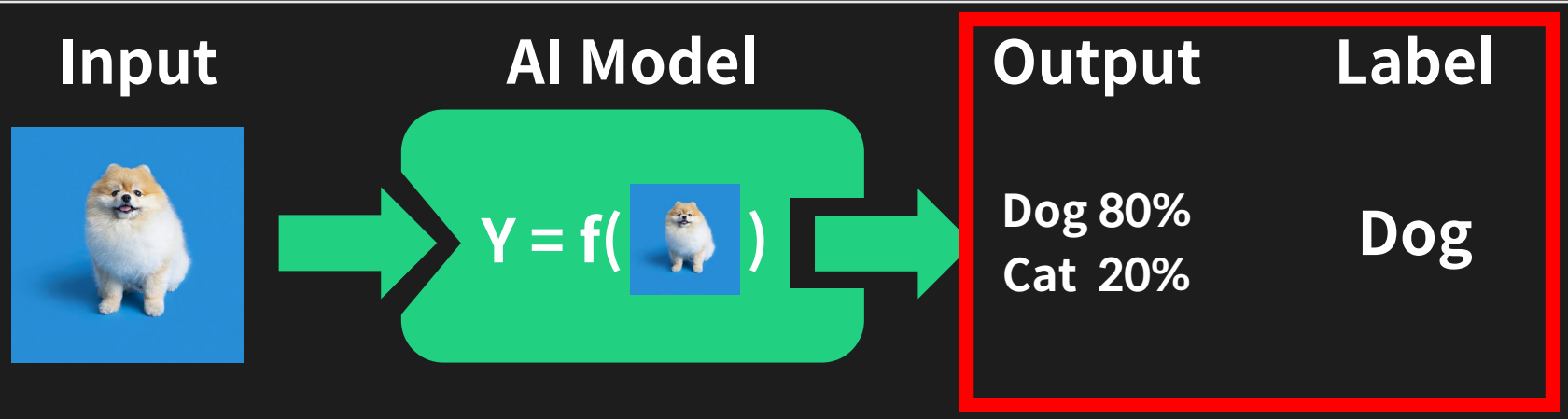
Elements of DL/ML



3. 모델이 잘 한다는 것을 어떻게 평가할 수 있는가?

1. 모델의 Output과 알고있는 정답 (Label)이 가까워지기를 바람 : Training / Optimization
2. 모델이 학습 중 보지 못한 예시에 대해서도 잘 예측하기를 바람 : Generalization

Elements of DL/ML



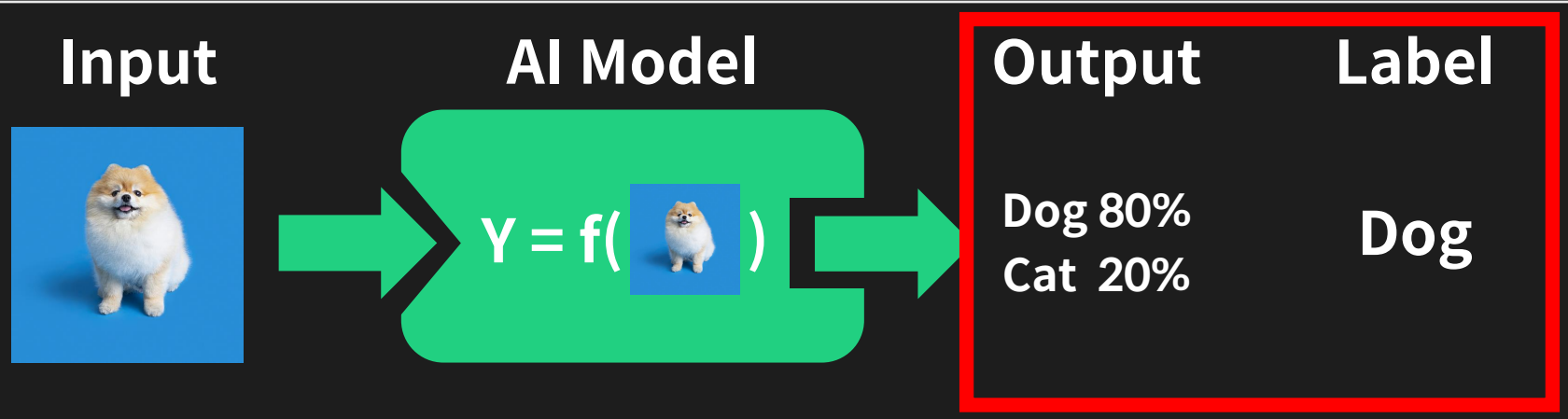
3. 모델이 잘 한다는 것을 어떻게 평가할 수 있는가?

1. 모델의 Output과 알고있는 정답 (Label)이 가까워지기를 바람 : Training / Optimization
   "얼마나 가까운지 " 를 측정하는 기준 -> Loss function

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Elements of DL/ML



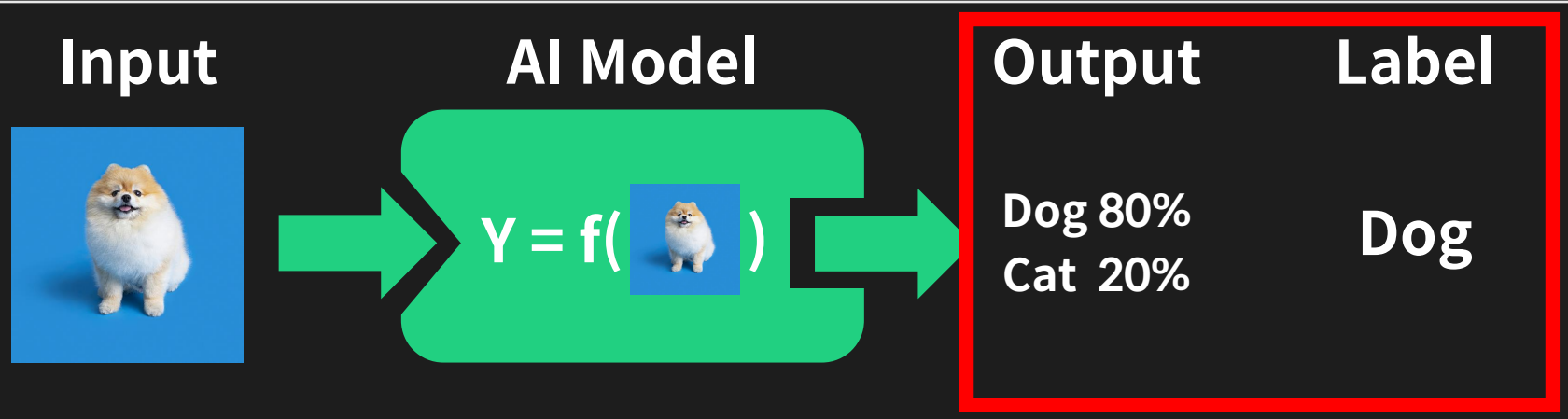
3. 모델이 잘 한다는 것을 어떻게 평가할 수 있는가?

2. 모델이 학습 중 보지 못한 예시에 대해서도 잘 예측하기를 바람 : Generalization (일반화)

데이터셋 중 학습 중 보지 못하게 하고, 이를 통해 일반화 성능을 측정한다.

=> Validation/Test Set Split

Elements of DL/ML



4. 잘 하는 모델을 어떻게 찾을 수 있는가?

"잘 하는 것"의 기준 = Loss function

그렇다면, 주어진 데이터셋에 대해서 Loss(Error)를 최소화하는 모델(함수)를 찾으면 된다!

Elements of DL/ML

4. 잘 하는 모델을 어떻게 찾을 수 있는가?

"잘 하는 것"의 기준 = Loss function

그렇다면, 주어진 데이터셋에 대해서 Loss(Error)를 최소화하는 모델(함수)를 찾으면 된다!

직접 미분하여 최솟값 찾기

모델이 복잡해질 수록,
데이터가 많아질 수록
직접 계산하기 매우 어려워짐.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \quad [1.0]$$

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_0}\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2\right) \quad [1.1]$$

$$= \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_0}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \quad [1.2]$$

$$= \frac{1}{m}\sum_{i=1}^{m}2\left(h_\theta(x^{(i)}) - y^{(i)}\right)\frac{\partial}{\partial\theta_0}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \quad [1.3]$$

$$= \frac{2}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \quad [1.4]$$

Elements of DL/ML

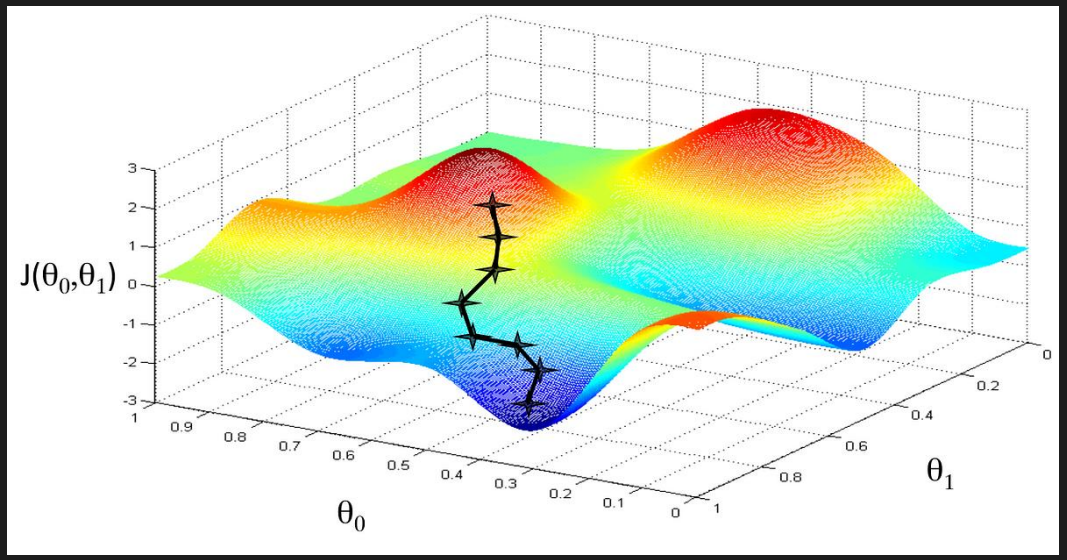
4. 잘 하는 모델을 어떻게 찾을 수 있는가?

"잘 하는 것"의 기준 = Loss function

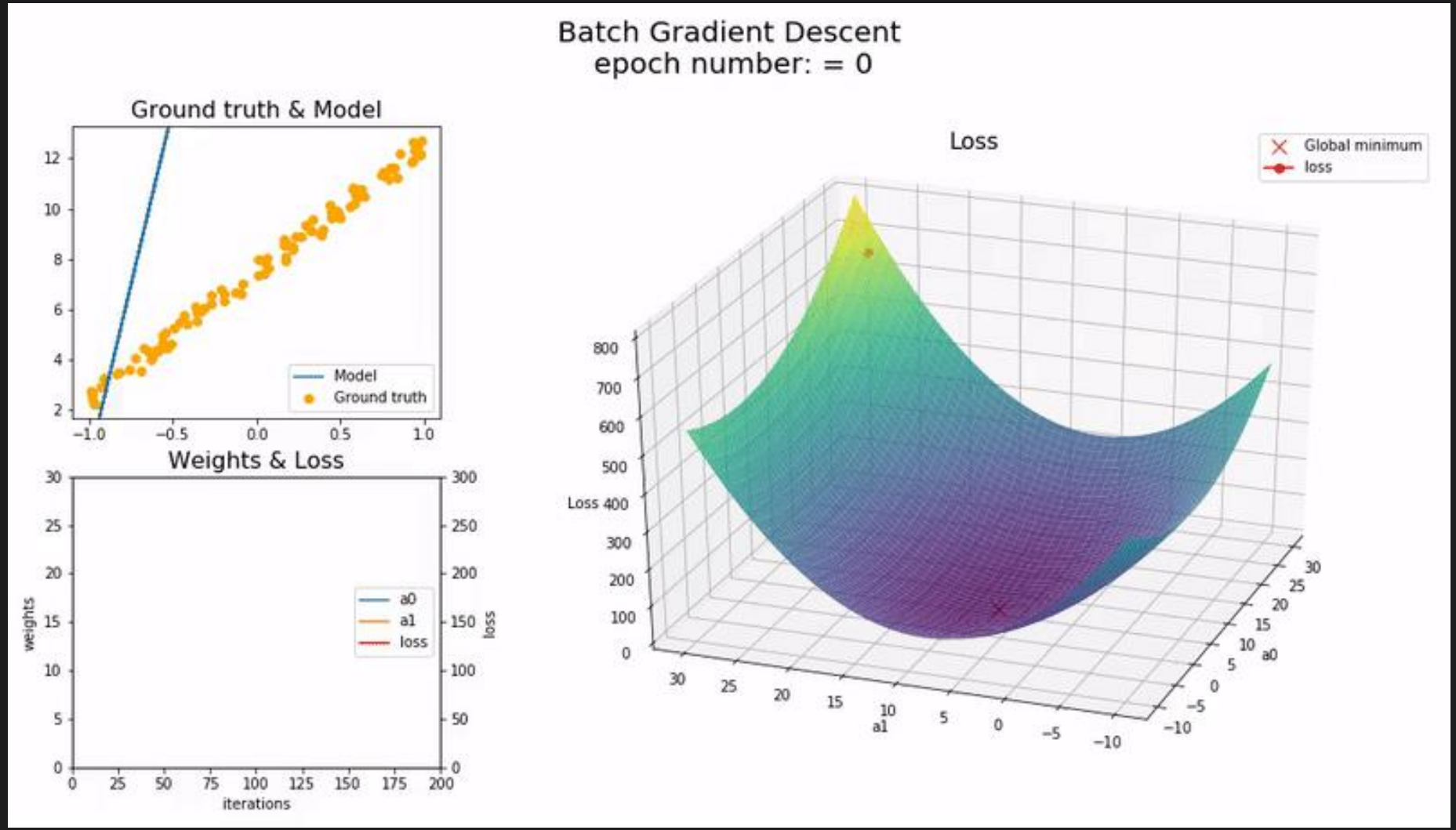
그렇다면, 주어진 데이터셋에 대해서 Loss(Error)를 최소화하는 모델(함수)를 찾으면 된다!

Gradient Descent (경사하강법)

함수의 값이 낮아지는 방향
(미분 계수의 반대 방향)으로
값을 바꾸어가며 최솟값을 찾는 방법
SGD, Adam 등 다양한 변형 존재

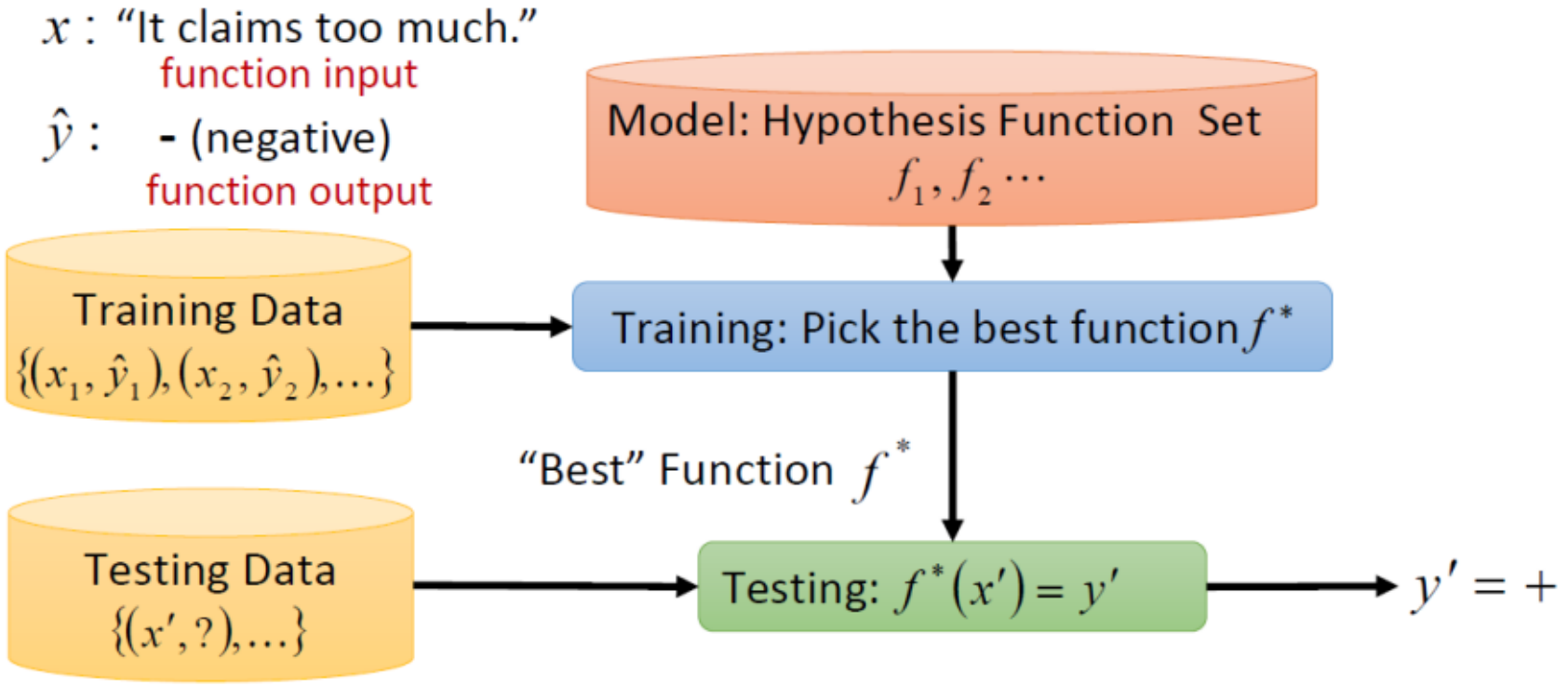


Elements of DL/ML



Overall Pipeline of ML

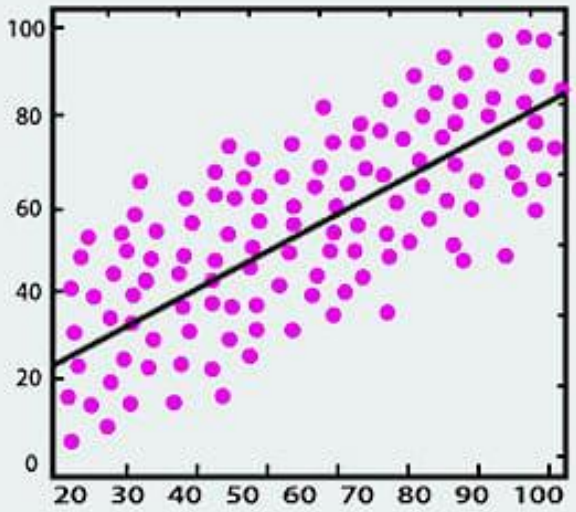
- **Training** is to pick the best function given the observed data.
- **Testing** is to predict the label using the learned function.



ML&DL Basics

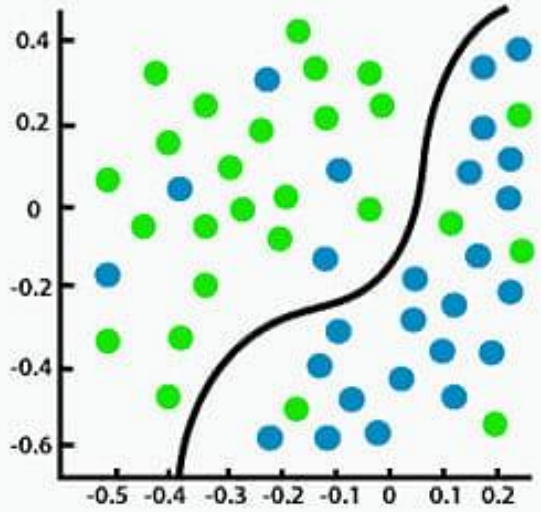
Linear & Logistic Regression

Classification vs. Regression



Regression

versus



Classification

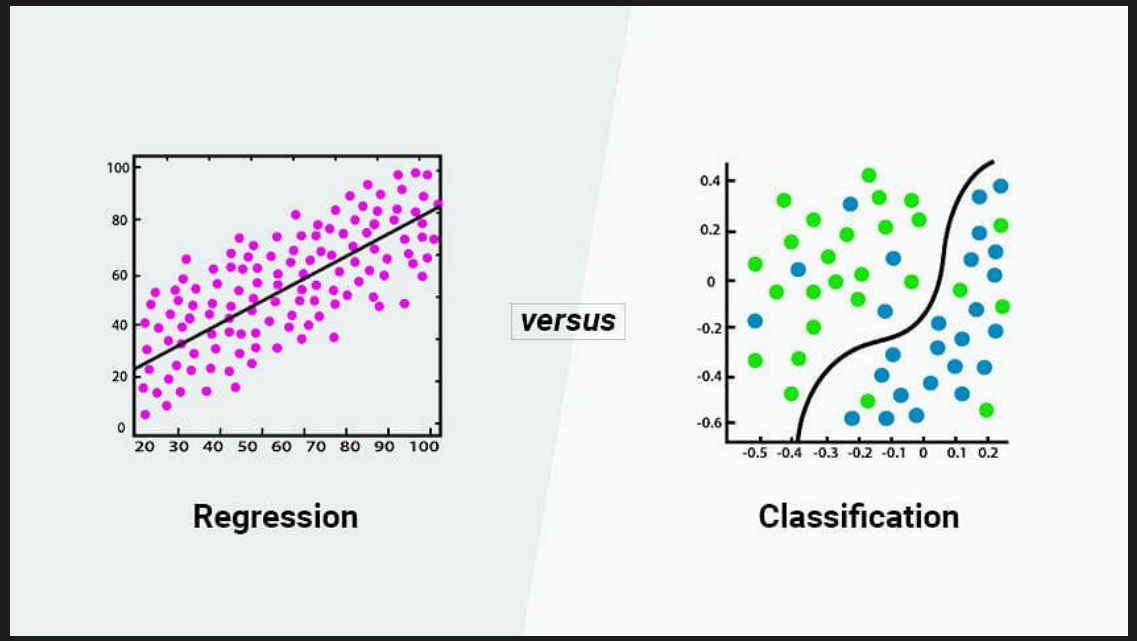
Classification vs. Regression

Classification

- 남성 vs. 여성, 양성 vs. 음성과 같은
  이산적(Discrete)인 값을 예측하는 Task

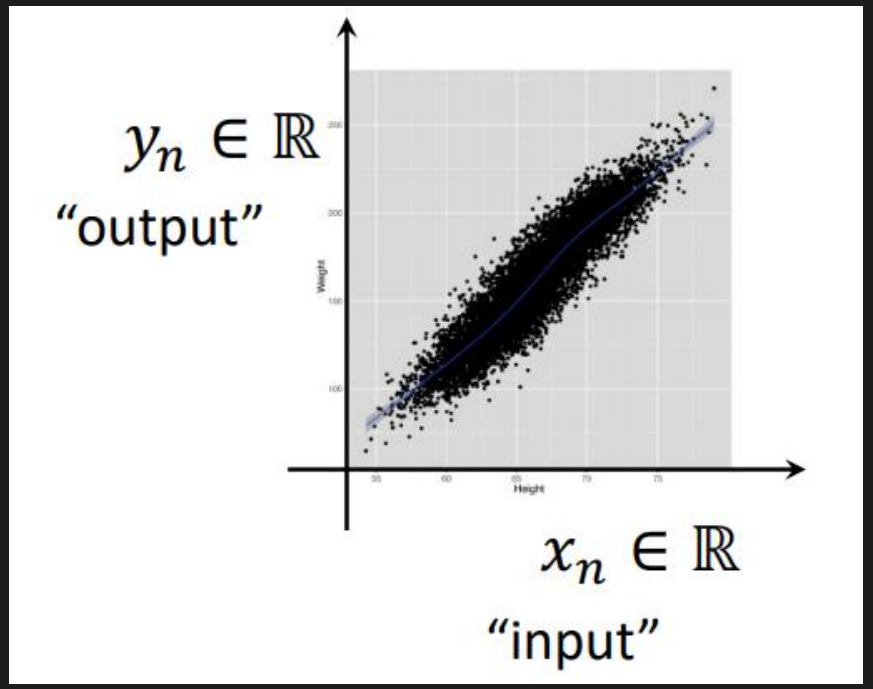
Regression

- 집값, 성적, 주가와 같은
  연속적(Continuous)인 값을 예측하는 Task

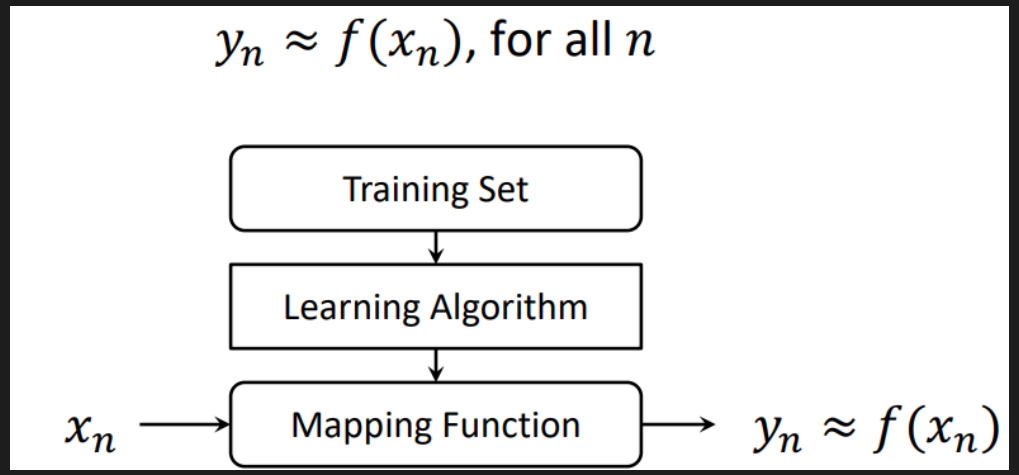


Regression

집값, 성적, 주가와 같은 연속적(Continuous)인 값을 예측하는 Task

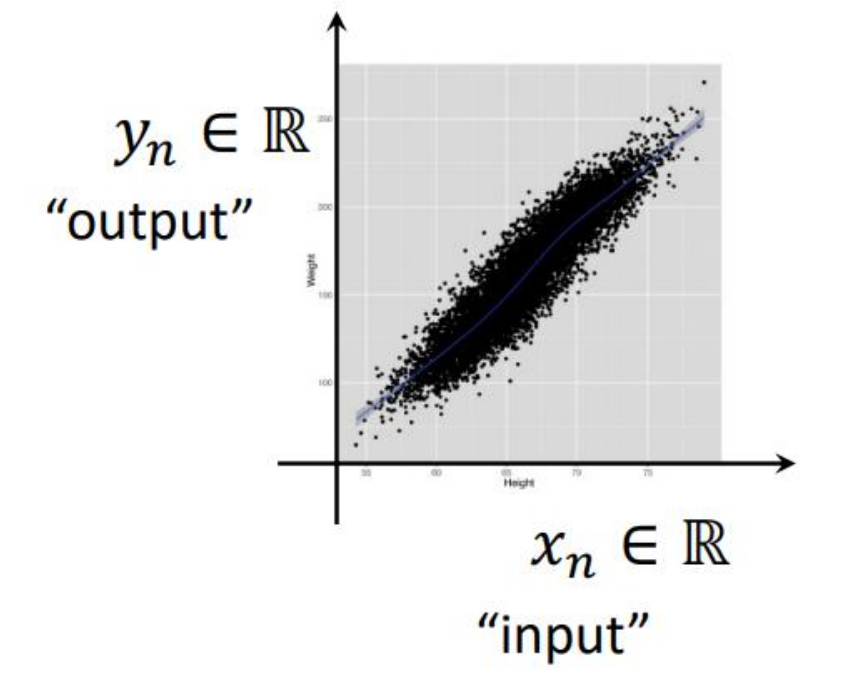


- We need to find a mapping function that approximates the output "well enough" given inputs



Linear Regression

Linear regression is a model that assumes a linear relationship between input variable and output variable.

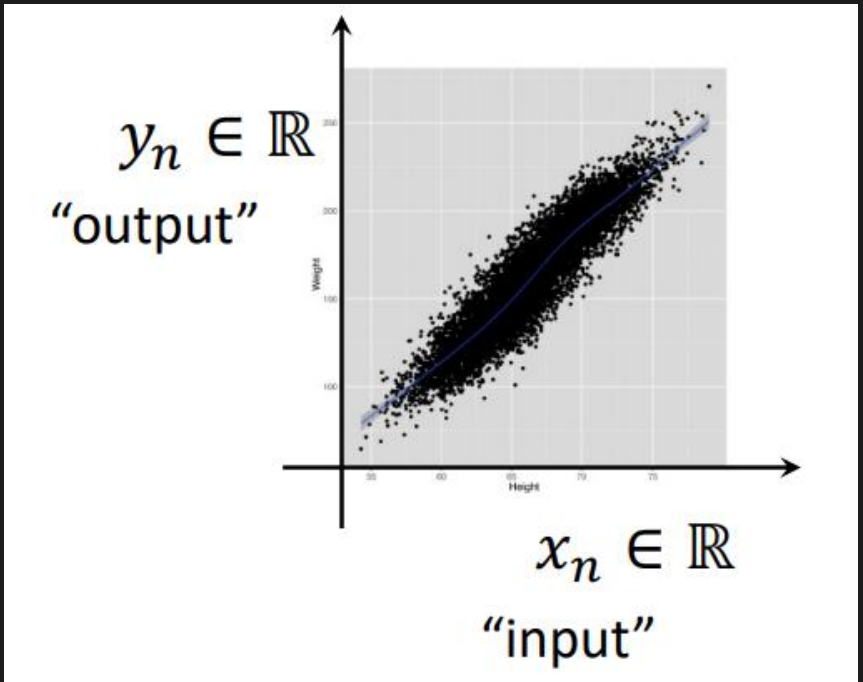


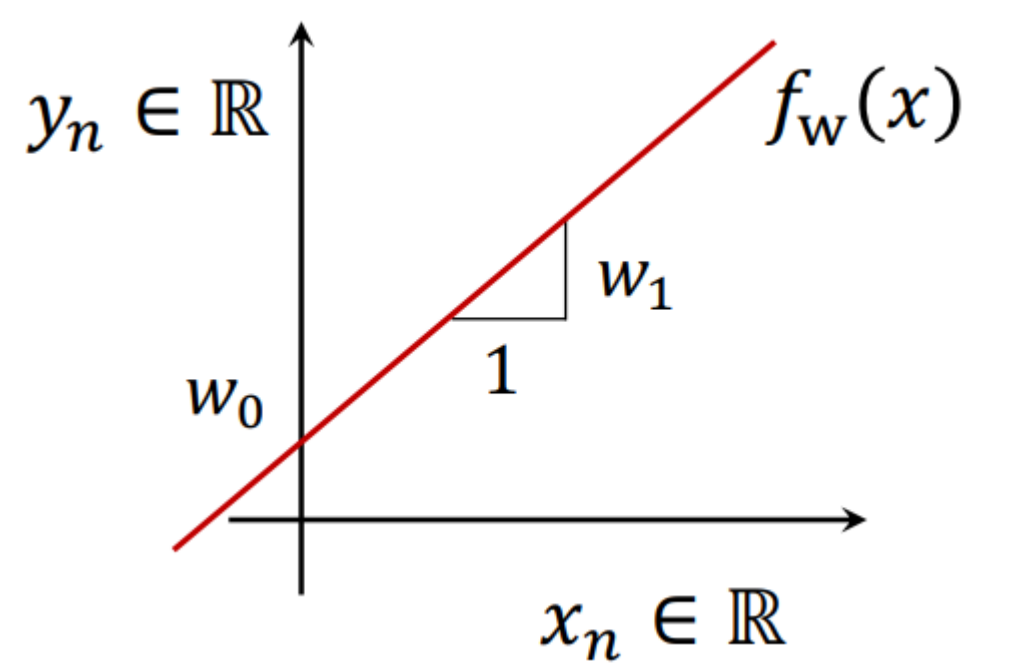
왜 하필 "Linear" Regression인가?

- 간단하고 이해하기 쉬움
- 간단한 변형으로 Linear Model에 Fit할 수 있음 (Feature Engineering)
- ML의 모든 기초 개념을 배울 수 있음

Linear Regression

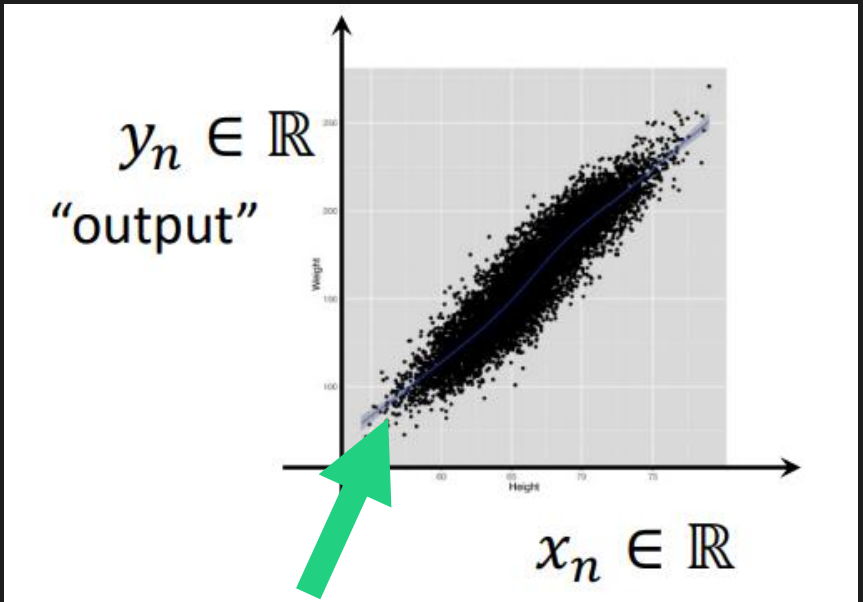
$$y_n \approx f(x_n) = f_w(x_n) = w_0 + w_1 x_n$$

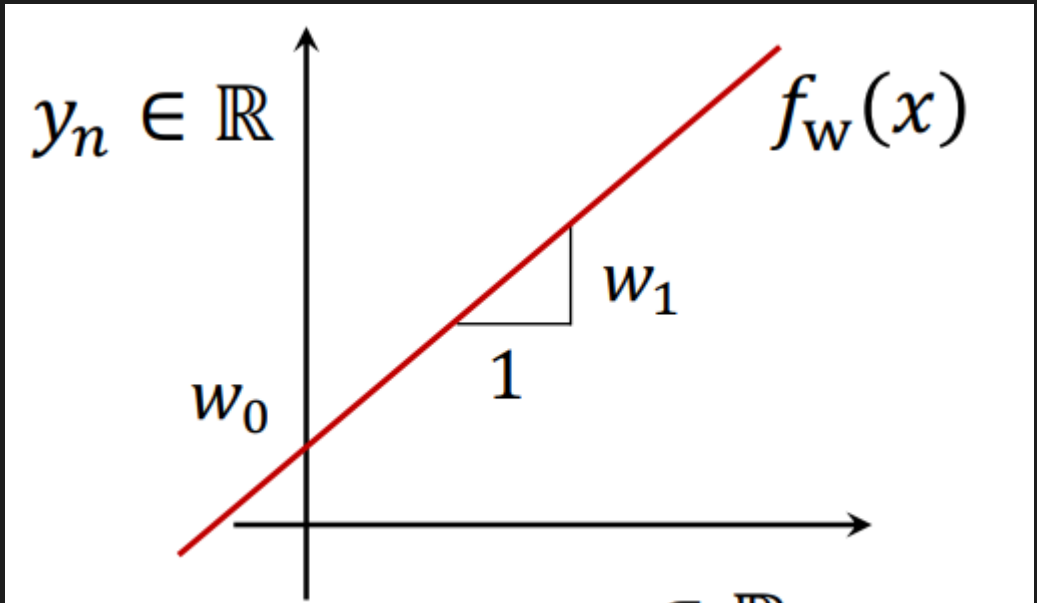




Linear Regression

$$y_n \approx f(x_n) = f_w(x_n) = w_0 + w_1 x_n$$





목적 : 주어진 데이터셋에 대해서 Loss(Error)를 최소화하는 모델 찾기

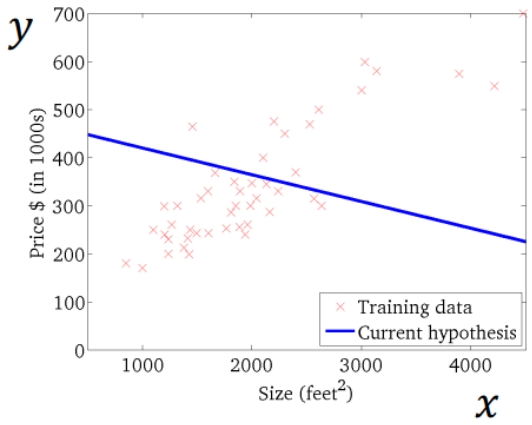
Linear Regression

데이터셋 $\{(x_n, y_n)_{n=1}^N\}$ 에 대해서,

- Model:
$$f_{\mathrm{w}}(x) = w_0 + w_1 x$$
- Parameters:
$$\mathrm{w} = (w_0, w_1)$$
- Cost function:
$$L(\mathrm{w}) = \frac{1}{N}\sum_{n=1}^{N}(f_{\mathrm{w}}(x_n) - y_n)^2$$
- Goal:
$$\mathrm{w}^* = \underset{\mathrm{w}}{\operatorname{argmin}}\, L(\mathrm{w})$$

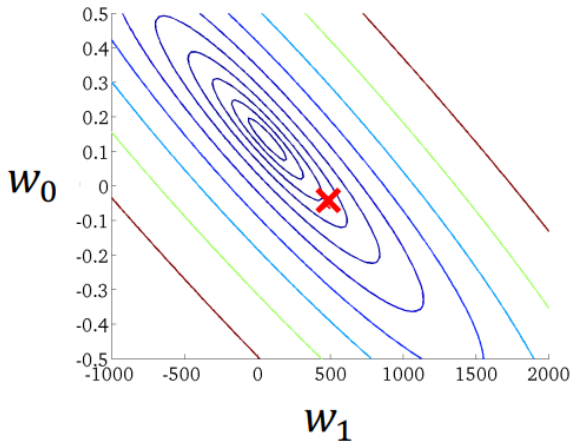
$f_{\mathrm{w}}(x)$

(for fixed w, this is a function of x)



$L(\mathrm{w})$

(function of the parameter w)



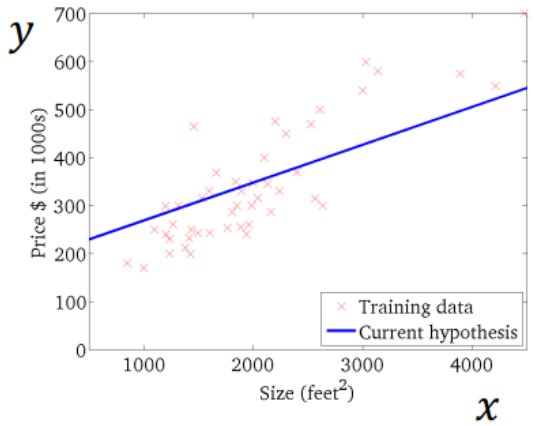
Linear Regression

데이터셋 $\{(x_n, y_n)_{n=1}^N\}$ 에 대해서,

- Model:
$$f_{\mathrm{w}}(x) = w_0 + w_1 x$$
- Parameters:
$$\mathrm{w} = (w_0, w_1)$$
- Cost function:
$$L(\mathrm{w}) = \frac{1}{N}\sum_{n=1}^{N}(f_{\mathrm{w}}(x_n) - y_n)^2$$
- Goal:
$$\mathrm{w}^* = \underset{\mathrm{w}}{\operatorname{argmin}}\, L(\mathrm{w})$$

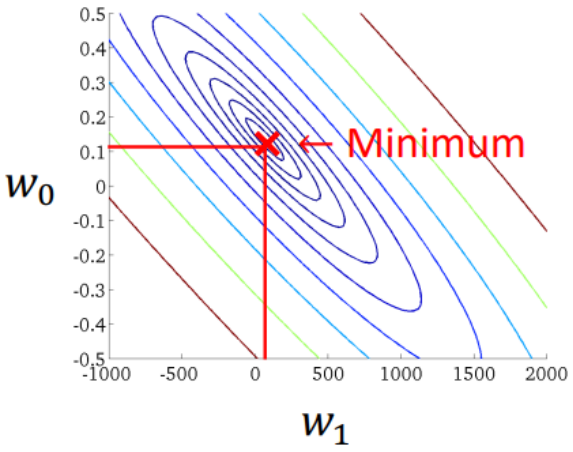
$f_{\mathrm{w}}(x)$

(for fixed w, this is a function of x)



$L(\mathrm{w})$

(function of the parameter w)



Loss Function (= Cost Function, Objective)

- Mean Squared Error (MSE) vs. Mean Absolute Error (MAE)

$$MSE = \frac{1}{N}\sum_{i}^{N}(pred_i - target_i)^2$$

$$MAE = \frac{1}{N}\sum_{i}^{N}|(pred_i - target_i)|$$

- MAE는 L1 Loss, MSE는 L2 Loss 로 많이 불림.

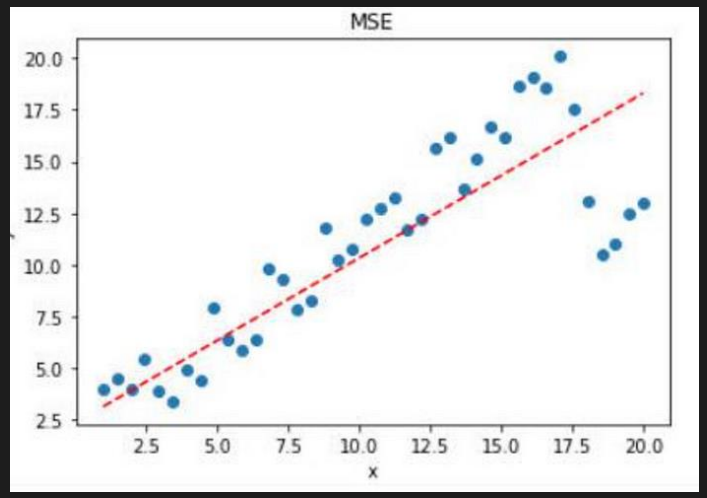
Loss Function (= Cost Function, Objective)

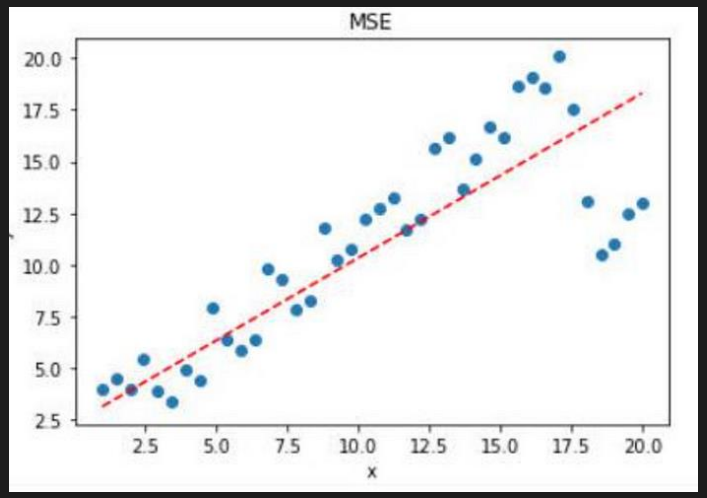
- Mean Squared Error (MSE) vs. Mean Absolute Error (MAE)

$$MSE = \frac{1}{N}\sum_{i}^{N}(pred_i - target_i)^2$$

$$MAE = \frac{1}{N}\sum_{i}^{N}|(pred_i - target_i)|$$

- MAE는 L1 Loss, MSE는 L2 Loss 로 많이 불림.
- MSE는 이상치 (Outlier)에 취약하다. (Why?)





Loss Function (= Cost Function, Objective)

- Mean Squared Error (MSE) vs. Mean Absolute Error (MAE)

$$MSE = \frac{1}{N}\sum_{i}^{N}(pred_i - target_i)^2$$

$$MAE = \frac{1}{N}\sum_{i}^{N}|(pred_i - target_i)|$$

- MAE는 L1 Loss, MSE는 L2 Loss 로 많이 불림.
- MSE는 이상치 (Outlier)에 취약하다. (Why?)
- 하지만, 어떤 Loss가 더 좋은 결과를 낼 지는 해봐야 알 수 있다.

Optimization

- Model:

$$f_{\mathrm{w}}(x) = w_0 + w_1 x$$

- Parameters:

$$\mathrm{w} = (w_0, w_1)$$

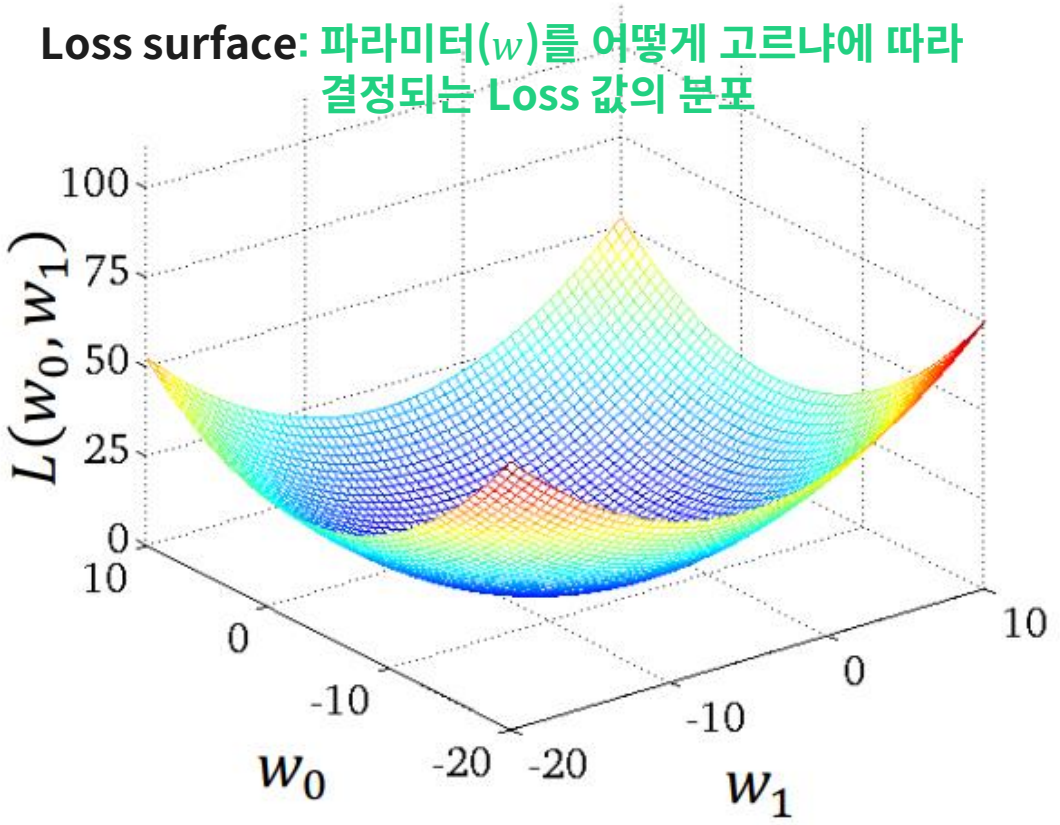
- Cost function:

$$L(\mathrm{w}) = \frac{1}{N}\sum_{n=1}^{N}(f_{\mathrm{w}}(x_n) - y_n)^2$$

- Goal:

$$\mathrm{w}^* = \underset{\mathrm{w}}{\operatorname{argmin}}\, L(\mathrm{w})$$

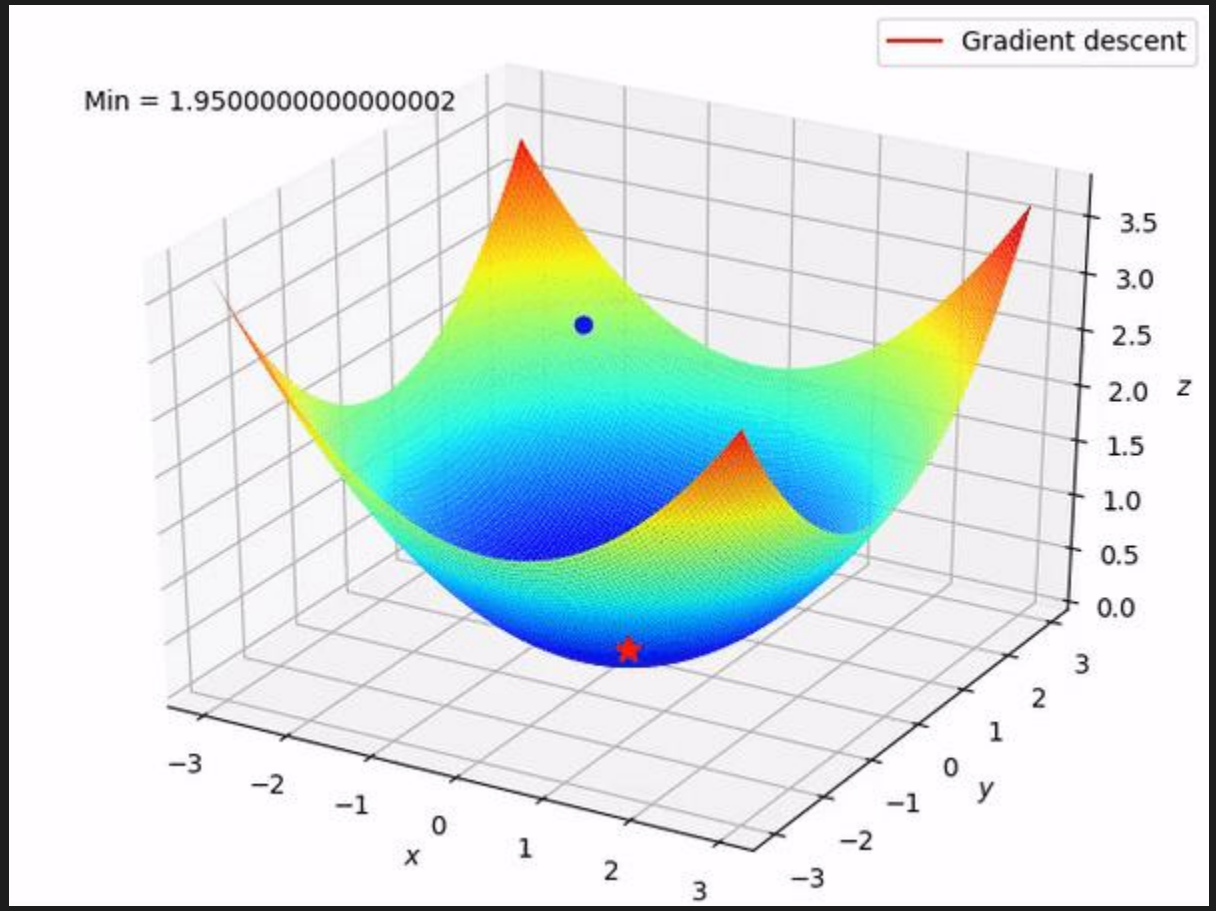
Loss surface: 파라미터(w)를 어떻게 고르냐에 따라 결정되는 Loss 값의 분포



어떻게 찾을 것인가? : Gradient Descent

Gradient Descent

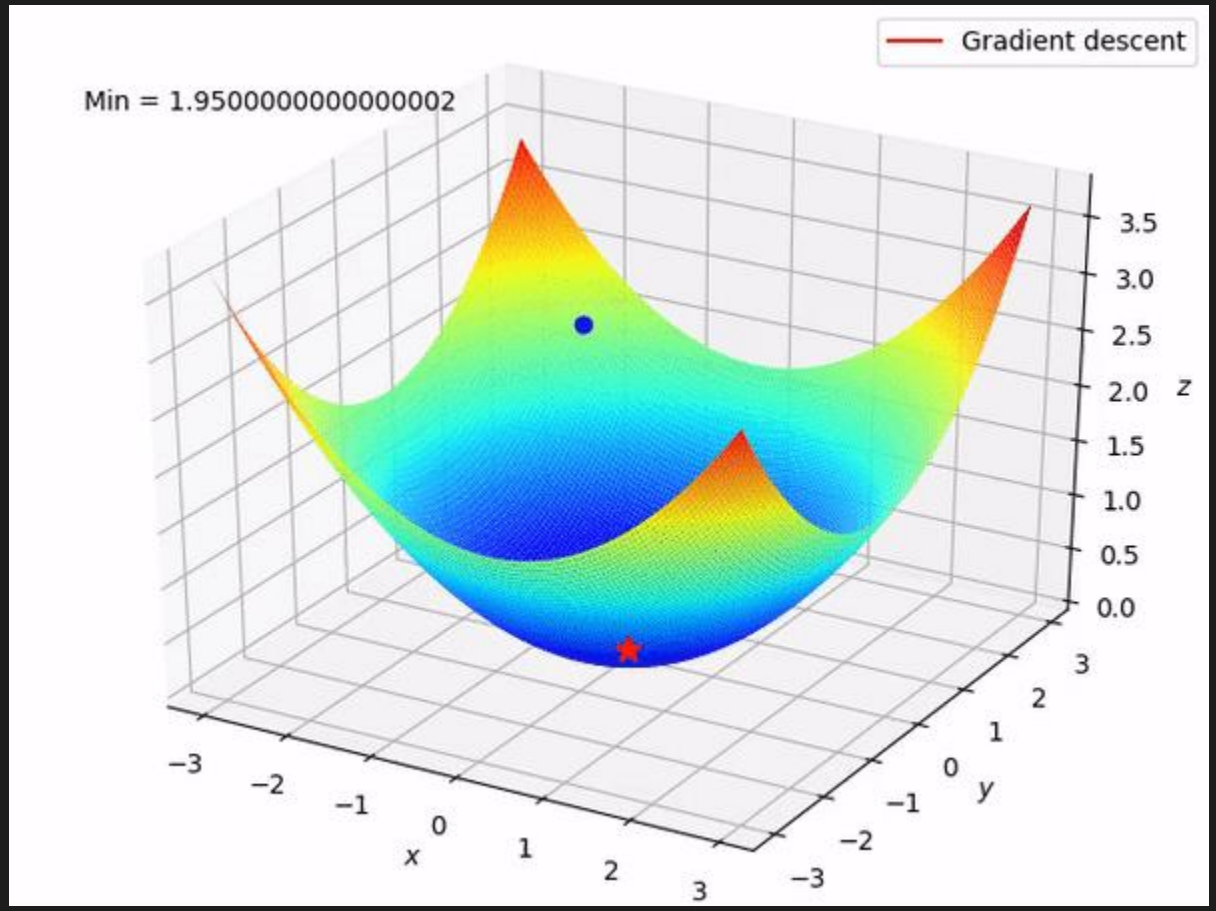
- 한 점에서의 Gradient는 그 점에서의 함수의 접선의 기울기와 같다.
- Gradient는 가장 큰 "증가 방향"을 나타낸다.
- 즉, Gradient의 반대 방향을 따라가다 보면, 함수의 극소값을 찾을 수 있다.



Gradient Descent

- 한 점에서의 Gradient는 그 점에서의 함수의 접선의 기울기와 같다.
- Gradient는 가장 큰 "증가 방향"을 나타낸다.
- 즉, Gradient의 반대 방향을 따라가다 보면, 함수의 극소값을 찾을 수 있다.

Q: Where start? → Initialization!
Q: Which direction? → Gradient!
Q: How much move? → Learning rate!



Gradient Descent

- Repeat until convergence
  {

$$w_0 \leftarrow w_0 - \alpha \frac{2}{N} \sum_{n=1}^{N} (f_w(x_n) - y_n)$$

$$w_1 \leftarrow w_1 - \alpha \frac{2}{N} \sum_{n=1}^{N} (f_w(x_n) - y_n) x_n$$

  }

Binary Classification

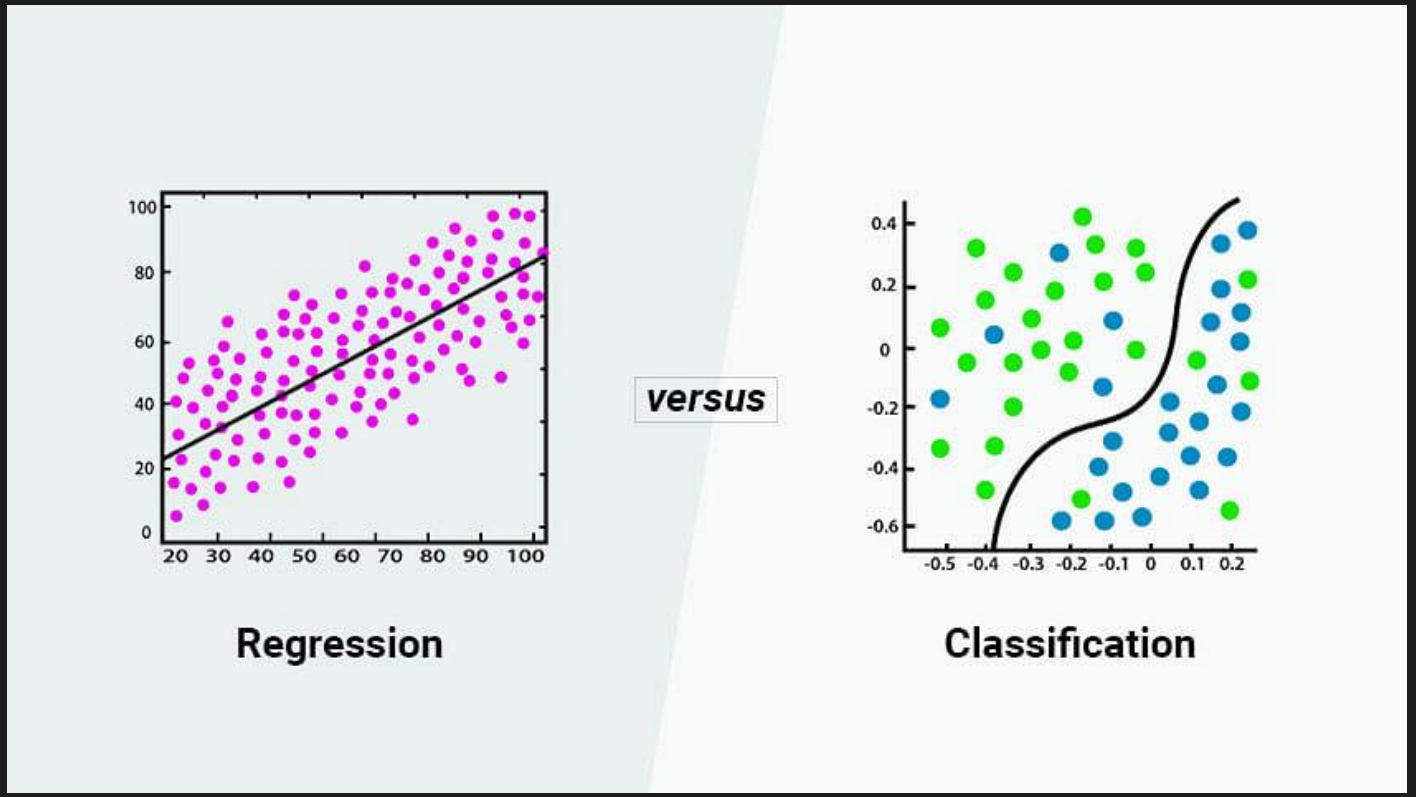
데이터셋 $\{(x_n, y_n)_{n=1}^N\}$ 에 대해서, **샘플** x_n이 **특정 한 클래스** y에 속하는 지를 예측하는 Task.

One vs. All, **즉, 두 개의 클래스 (Positive / Negative)만 있다고 가정함.**

X	Y
종양 크기 : 10mm	암 여부 : 0
키 : 180cm 몸무게 : 70kg	남자 여부 : 1

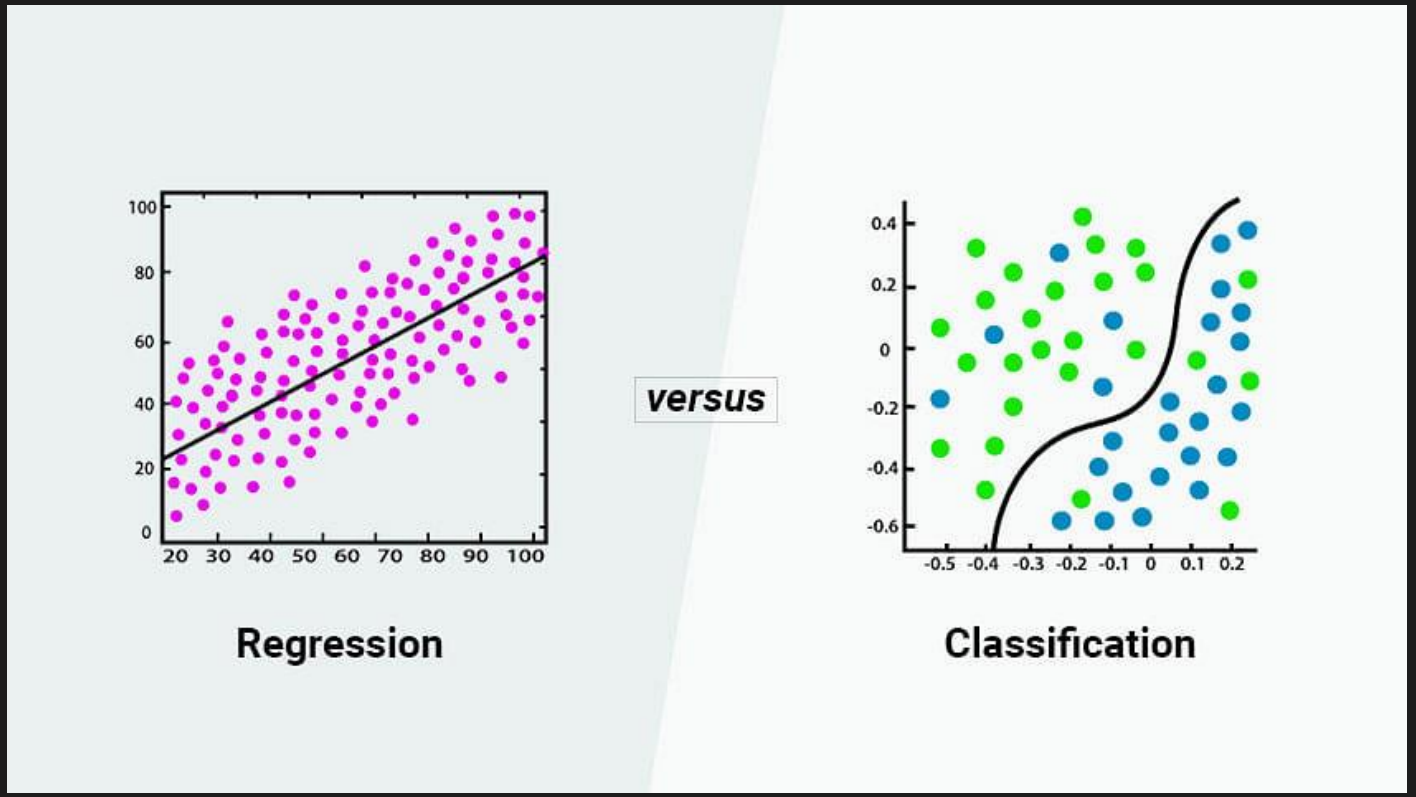
Logistic Regression

Linear Regression으로 분류 문제를 해결할 수 있을까?



Logistic Regression

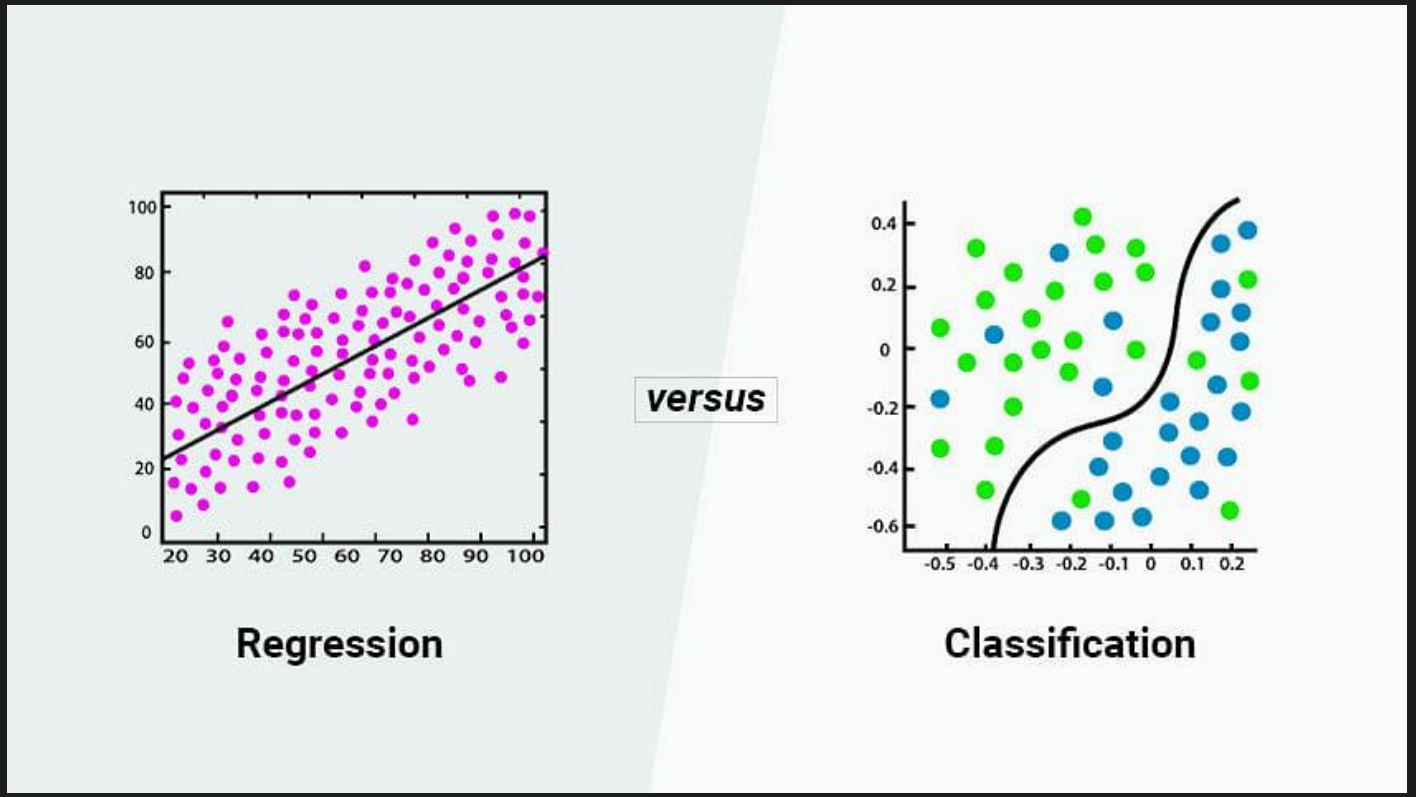
Linear Regression으로 분류 문제를 해결할 수 있을까?



Logistic Regression

Linear Regression으로 분류 문제를 해결할 수 있을까?

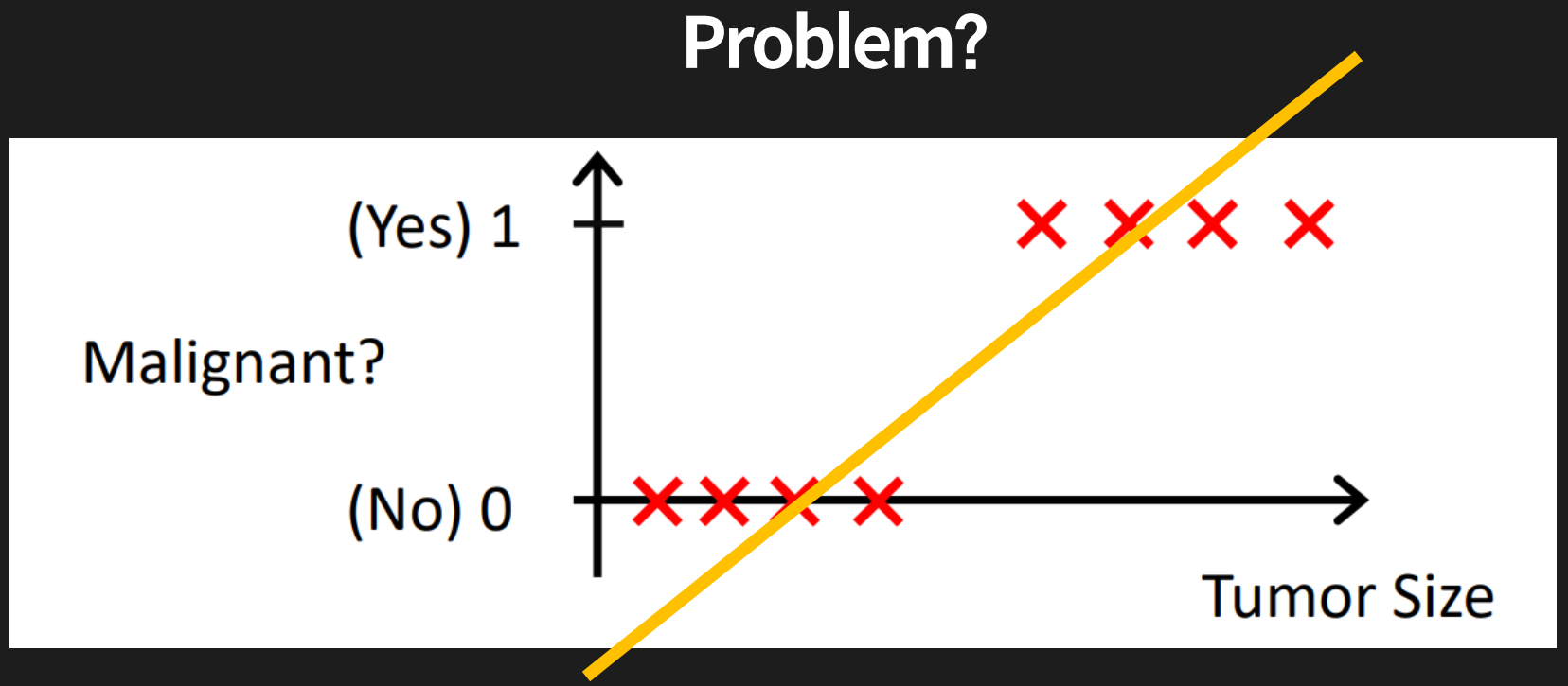
-> “특정 분류에 속할 확률”을 Regression하면 된다.



Logistic Regression

Linear Regression으로 분류 문제를 해결할 수 있을까?

-> “특정 분류에 속할 확률”을 Regression하면 된다.



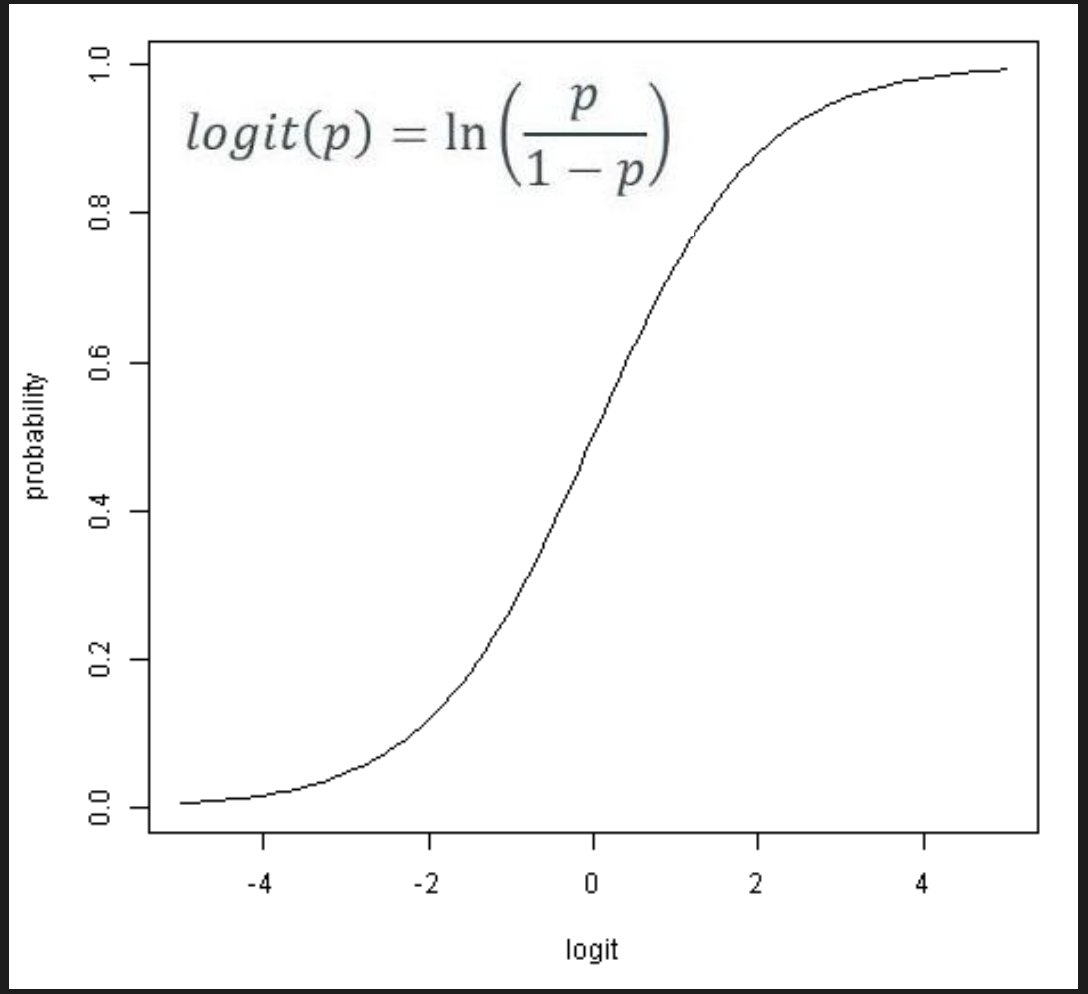
Logit

Logit : odds(승산)에 Log을 취한 것 (Log odds)

- Logit이 0보다 크다 == p가 0.5보다 크다.
- 확률의 범위는 0-1이나, Logit은 [-∞, ∞]이다.

확률 대신, Logit을 Regress해보자!

(단, 여전히 우리의 목표는 확률 구하기)



Logistic Regression

확률 대신, Logit을 Regress해보자! (단, 여전히 우리의 목표는 확률 구하기)

$$\text{logit}(p) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Logistic Regression

확률 대신, Logit을 Regress해보자! (단, 여전히 우리의 목표는 확률 구하기)

$$\text{logit}(p) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

해당 모델로부터 데이터 x가 어떤 클래스에 속할 확률(p)을 역으로 계산할 수 있다.

Logistic Regression Model

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

Sigmoid Function

Logistic Regression

확률 대신, Logit을 Regress해보자! (단, 여전히 우리의 목표는 확률 구하기)

$$\text{logit}(p) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

반대로, Sigmoid 함수의 역할이 Regression된 값을 0과 1의 확률값 범위로 mapping하는 것이라고도 생각할 수 있음.

해당 모델로부터 데이터 x가 어떤 클래스에 속할 확률(p)을 역으로 계산할 수 있다.

Logistic Regression Model

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

Sigmoid Function

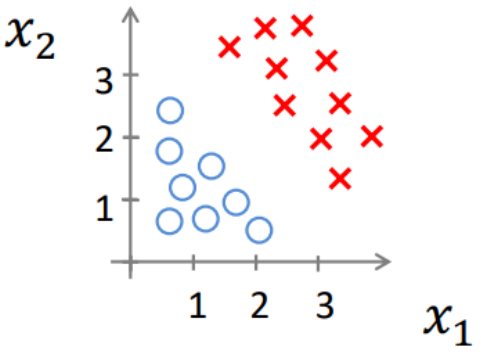
Logistic Regression

$$f_{\mathrm{w}}(\mathrm{x}) = g(\mathrm{w}^T\mathrm{x}) \text{ where } g(z) = 1/(1+\exp(-z))$$

$$f_{\mathrm{w}}(\mathrm{x}) = g(\mathrm{w}^T\mathrm{x}) = g(w_0 + w_1x_1 + w_2x_2)$$

$$f_{\mathrm{w}}(\mathrm{x}) = \frac{1}{1+\exp(-\mathrm{w}^T\mathrm{x})}$$

$$= \frac{1}{1+\exp(-w_0 - w_1x_1 - w_2x_2)}$$



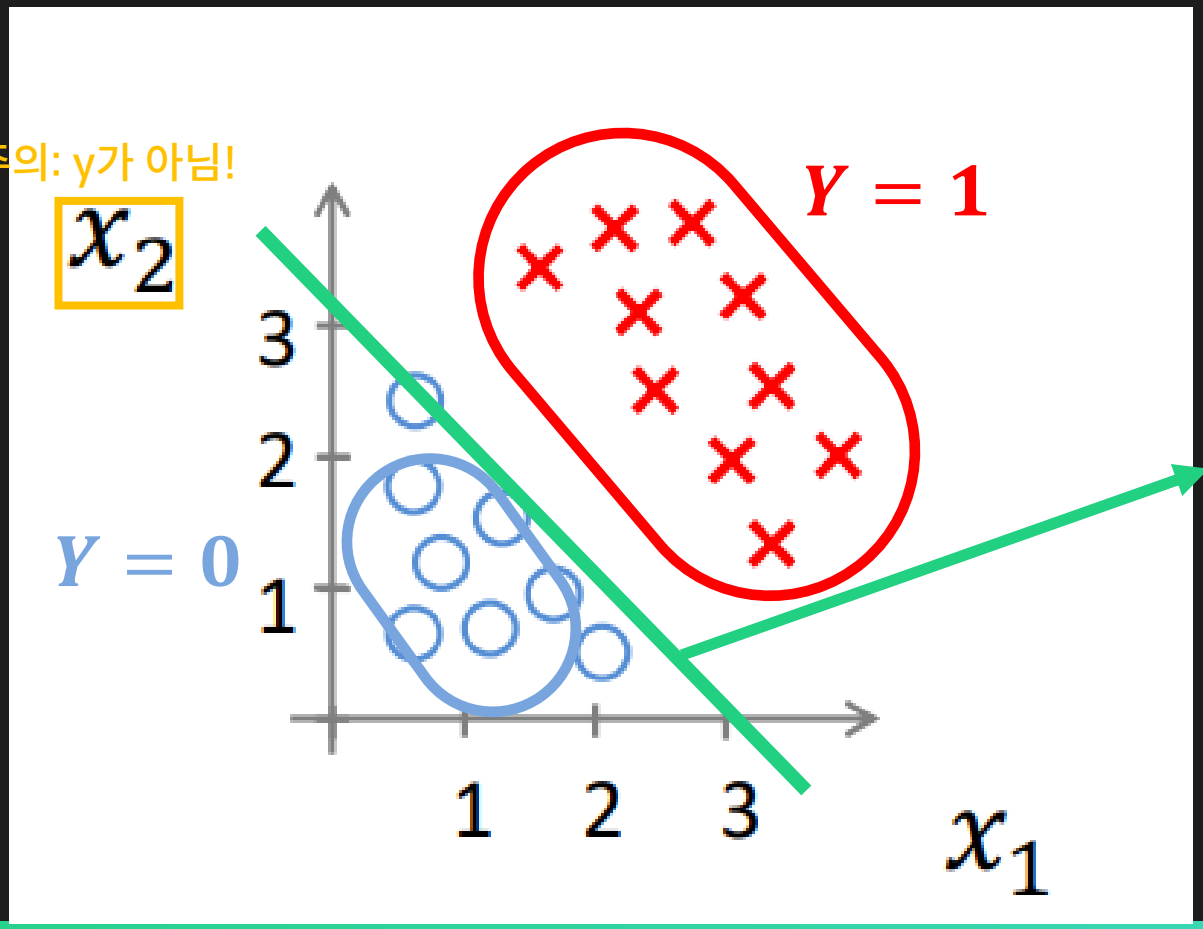
$w_0 = 3$
$w_1 = 1$
$w_2 = 1$

- Predict $y = 1$ if $-3 + x_1 + x_2 \geq 0$
- Predict $y = 0$ if $-3 + x_1 + x_2 < 0$

Logistic Regression

$$f_w(x) = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2)}$$

$w_0 = 3$
$w_1 = 1$
$w_2 = 1$



$$\hat{y} = f_w(x) = 0.5$$

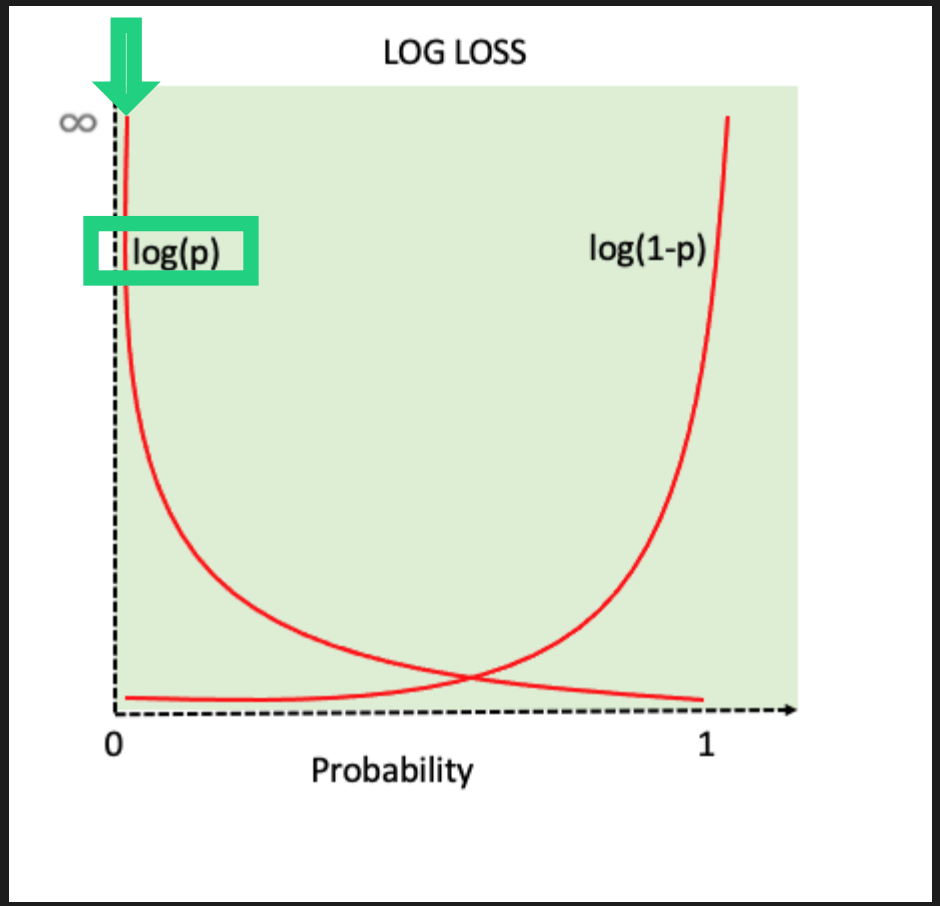
Decision Boundary

Loss function of Logistic Regression

- **Binary Cross Entrophy Loss (BCE Loss)**

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

- 예측값($\hat{y}$) : 0 / 실제값 (y) : 1 : +inf

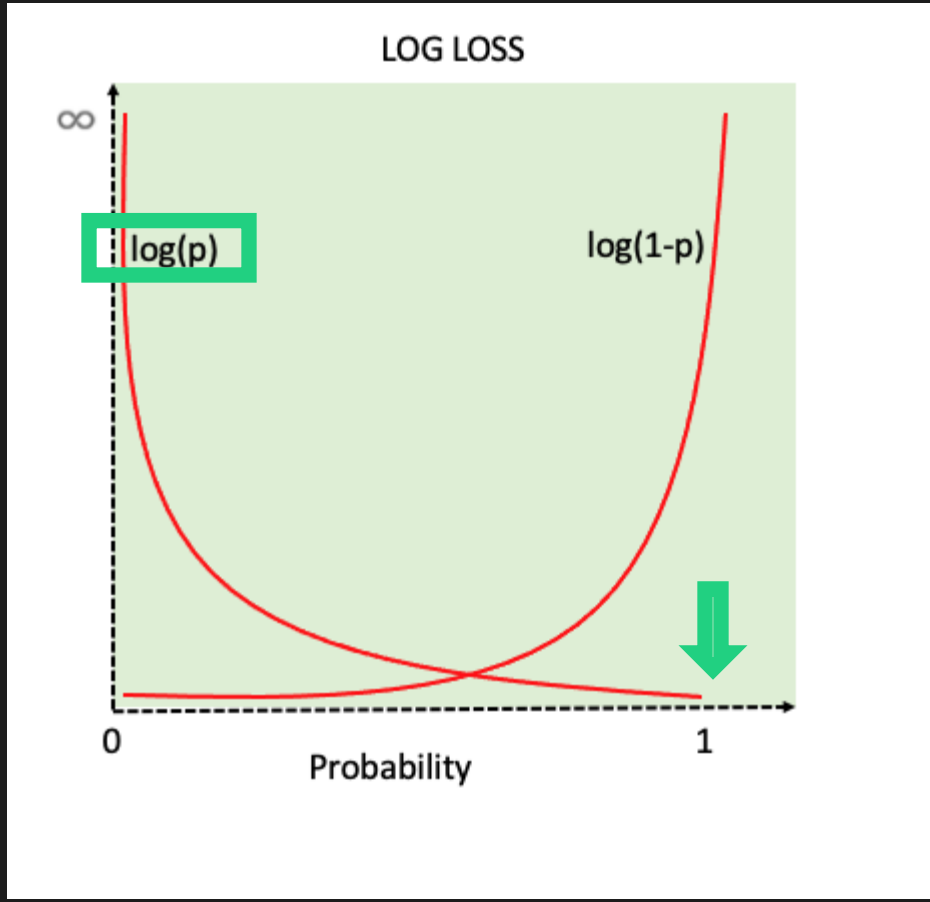


Loss function of Logistic Regression

- **Binary Cross Entrophy Loss (BCE Loss)**

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

- 예측값($\hat{y}$) : 0 / 실제값 (y) : 1 : +inf
- 예측값($\hat{y}$) : 0 / 실제값 (y) : 0 : 0

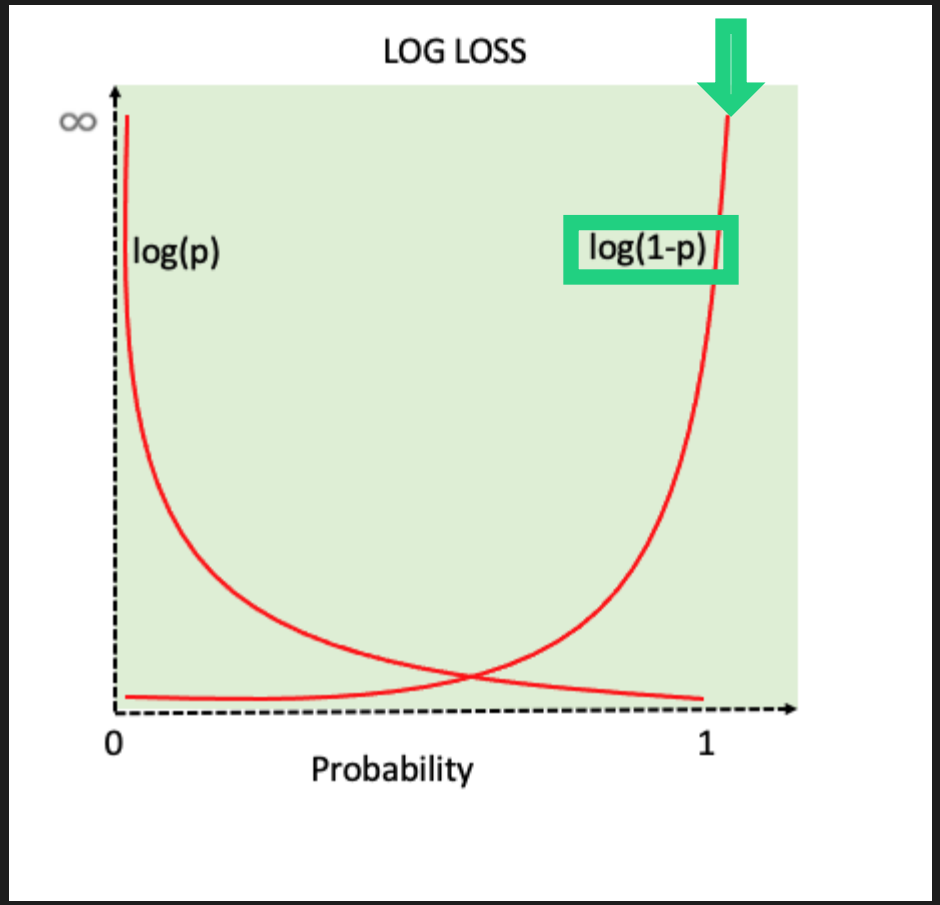


Loss function of Logistic Regression

- **Binary Cross Entrophy Loss (BCE Loss)**

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

- 예측값($\hat{y}$) : 0 / 실제값 (y) : 1 : +inf
- 예측값($\hat{y}$) : 0 / 실제값 (y) : 0 : 0
- 예측값($\hat{y}$) : 1 / 실제값 (y) : 0 : +inf

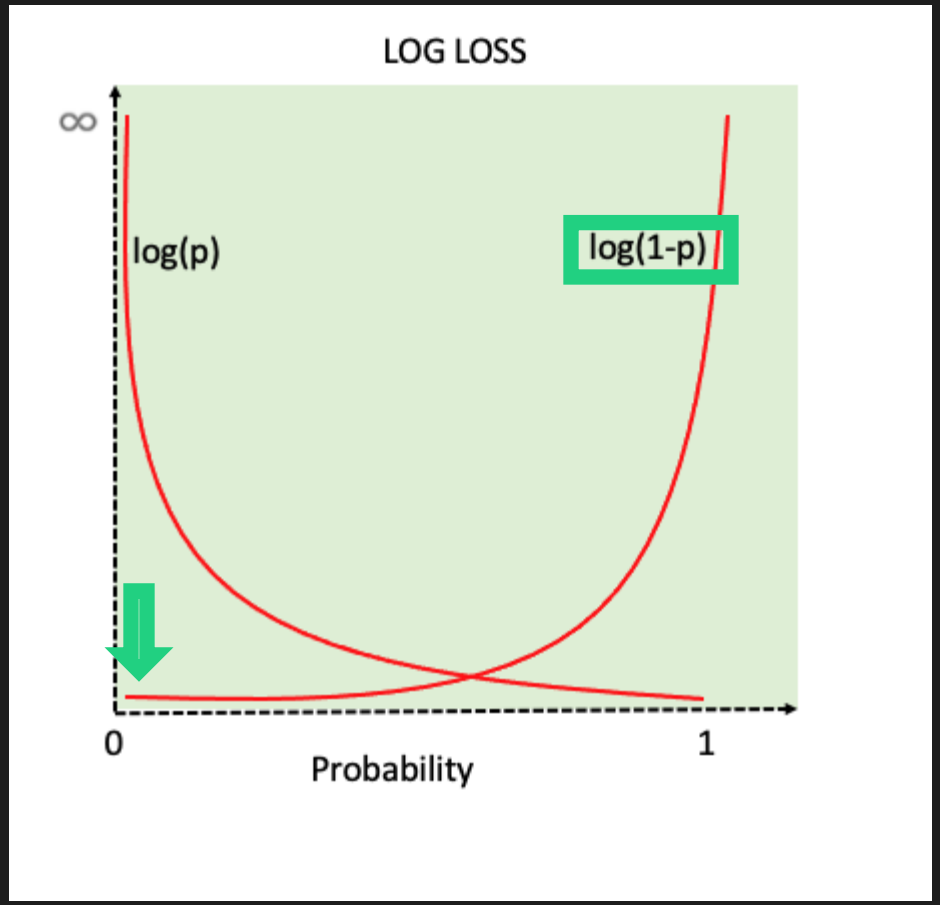


Loss function of Logistic Regression

- **Binary Cross Entrophy Loss (BCE Loss)**

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

- 예측값($\hat{y}$) : 0 / 실제값 (y) : 1 : +inf
- 예측값($\hat{y}$) : 0 / 실제값 (y) : 0 : 0
- 예측값($\hat{y}$) : 1 / 실제값 (y) : 0 : +inf
- 예측값($\hat{y}$) : 1 / 실제값 (y) : 1 : 0



PyTorch Session #1

Basic Tensor Manipulation & Simple Linear Regression

ML&DL Basics

Model Training & Selection

Overall pipline of Model Training

```
# Pseudocode for Model Training Pipeline

# Step 1.  Data Preprocessing
data =  ##Preprocessing
train_data, val_data, test_data =
       split_data(data, train_ratio, val_ratio, test_ratio)

# Step 2: Set Model, Optimizer, Scheduler
model = set_model(model_type)
optimizer = set_optimizer(optimizer_type)
scheduler = set_scheduler(scheduler_type) #optional
```

Overall pipline of Model Training

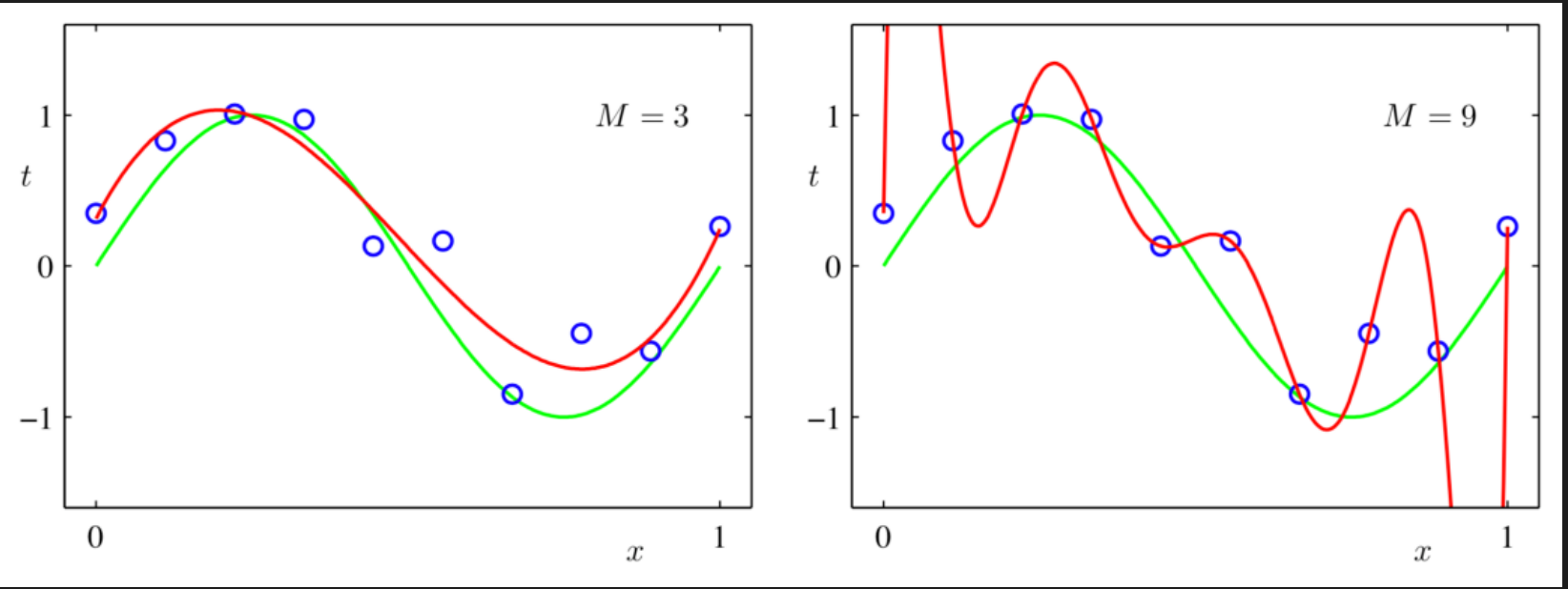
```
# Step 3: Training
for epoch in range(num_epochs):
    for batch in get_batches(train_data, batch_size):
        predictions = model.forward(batch.inputs)
        loss = calculate_loss(predictions, batch.targets)
        gradients = model.backward(loss)
        optimizer.update_parameters(gradients, learning_rate)

    # Step 4: Evaluation on Validation Set
    val_predictions = model.forward(val_data.inputs)
    val_loss = calculate_loss(val_predictions, val_data.targets)
    print(f'Epoch {epoch+1}, Validation Loss: {val_loss}')
```

Dataset Split

```
train_data, val_data, test_data = split_data(data …)
```

모델은 Training Set에 대해서 '잘 되도록' 학습되나. 실제로 잘 '일반화'되는 지는 Training Set 만으로는 확인 불가

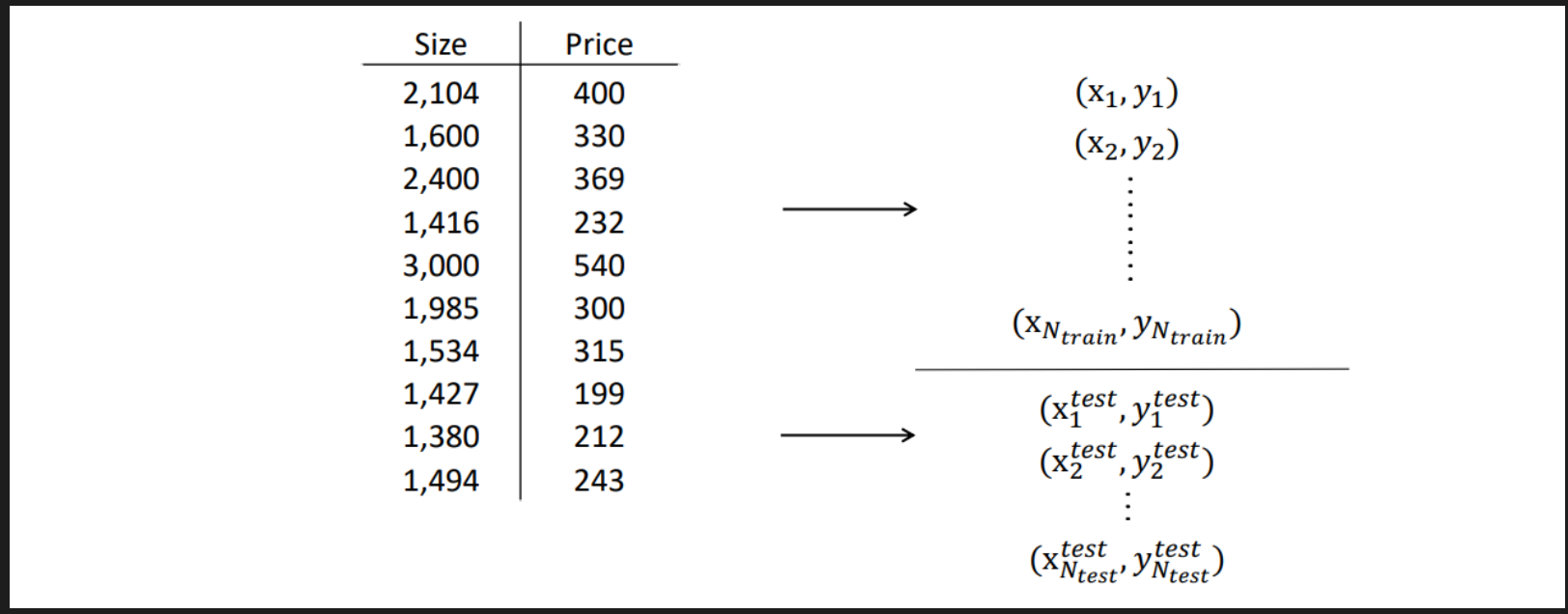


Dataset Split

```
train_data, val_data, test_data = split_data(data …)
```

모델은 Training Set에 대해서 '잘 되도록' 학습되나. 실제로 잘 '일반화'되는 지는 Training Set 만으로는 확인 불가

Training Set을 나누어, 일부 데이터를 훈련에 사용하지 않고, 평가(Test/Validation)에 사용하자!



Dataset Split

```
train_data, val_data, test_data = split_data(data …)
```

모델은 Training Set에 대해서 '잘 되도록' 학습되나. 실제로 잘 '일반화'되는 지는 Training Set 만으로는 확인 불가

Training Set을 나누어, 일부 데이터를 훈련에 사용하지 않고, 평가(Validation)에 사용하자!

여러 개의 Model 후보를 만들어 놓고 훈련시킨 후,

가장 Validation Score가 높은 모델 선택 -> Model Selection

Model Selection

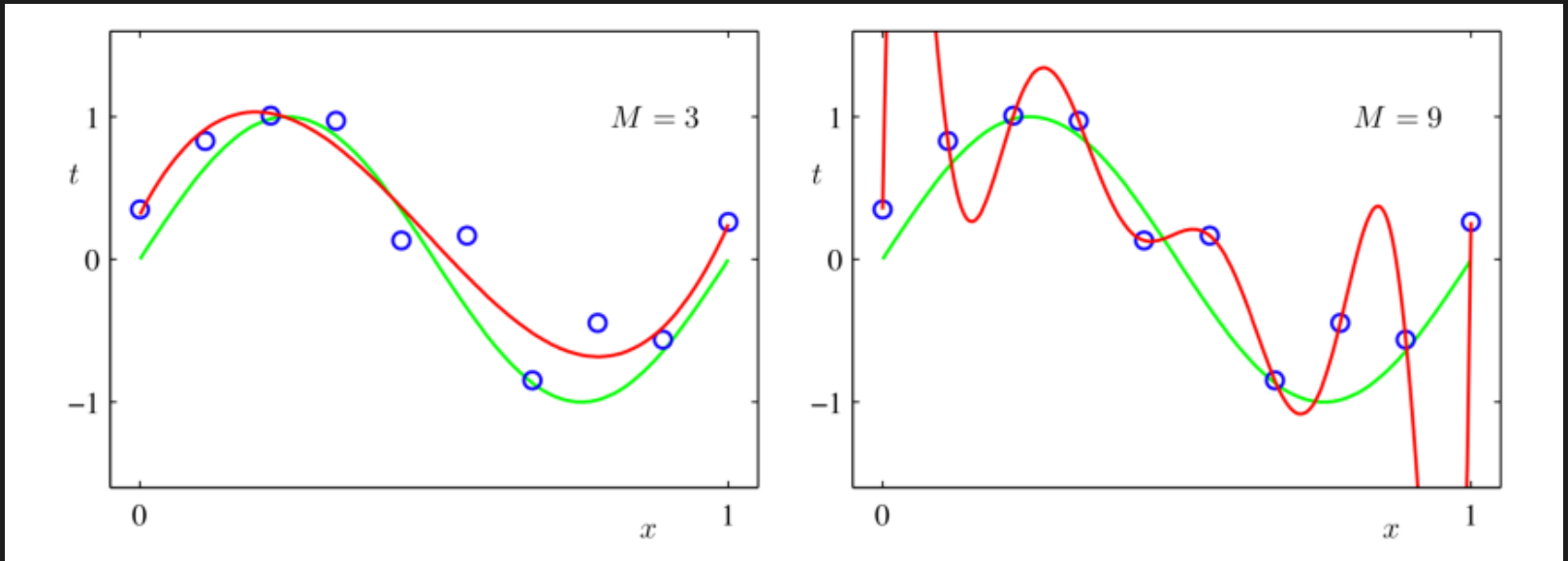
Training Set을 나누어, 일부 데이터를 훈련에 사용하지 않고, 평가(Validation)에 사용하자!

Q. 왜 Training Score를 모델 선택에 사용하면 안될까?

Model Selection

Training Set을 나누어, 일부 데이터를 훈련에 사용하지 않고, 평가(Validation)에 사용하자!

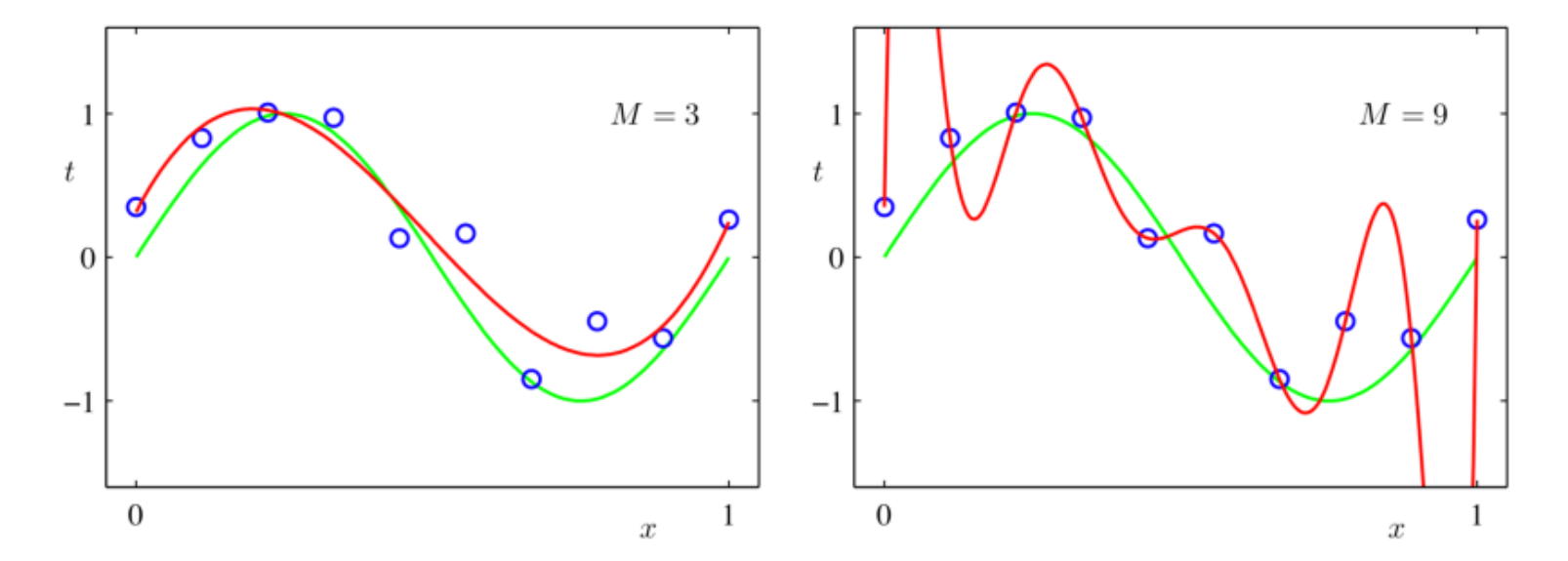
Q. 왜 Training Score를 모델 선택에 사용하면 안될까?



Training Loss	0.5	0
Validation Loss	0.5	10

Model Selection

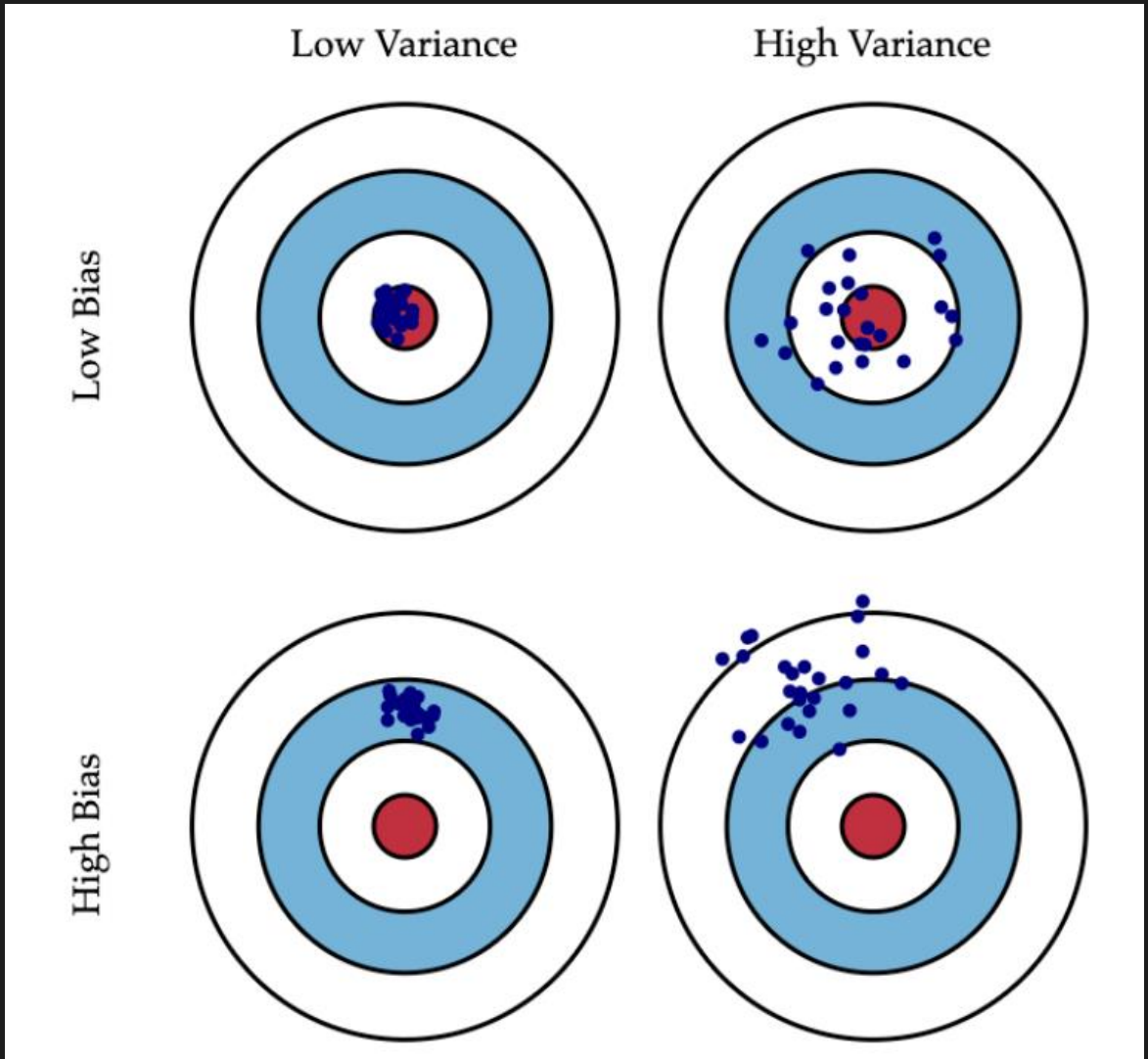
Q. 왜 Training Score를 모델 선택에 사용하면 안될까?



데이터의 실제 분포에 비해 모델의 가정이 너무 복잡한 경우, Overfitting 문제가 발생할 수 있다.

*Underfitting : 데이터에 비해 모델의 가정이 너무 단순하여 제대로 Fit할 수 없는 경우

Bias-Variance Problem



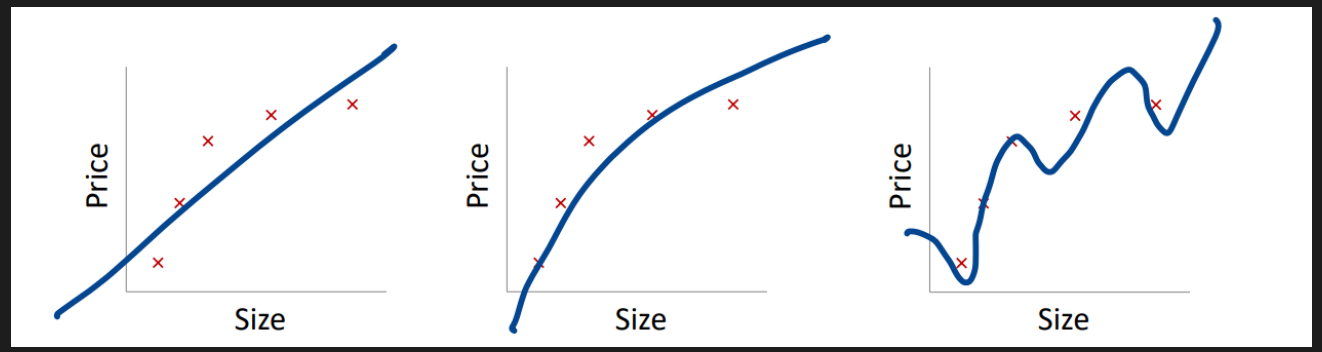
Bias-Variance Problem

Bias² is the squared difference between the model's average prediction and the true values

Irreducible Error is the inherent noise in the data, which cannot be reduced by improving the model

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Variance measures the model's inconsistency across different training sets



High Bias "Just Right" High Variance

모델이 복잡해질수록,
Bias는 감소하지만
Variance는 증가한다.
Bias-Variance Trade-off

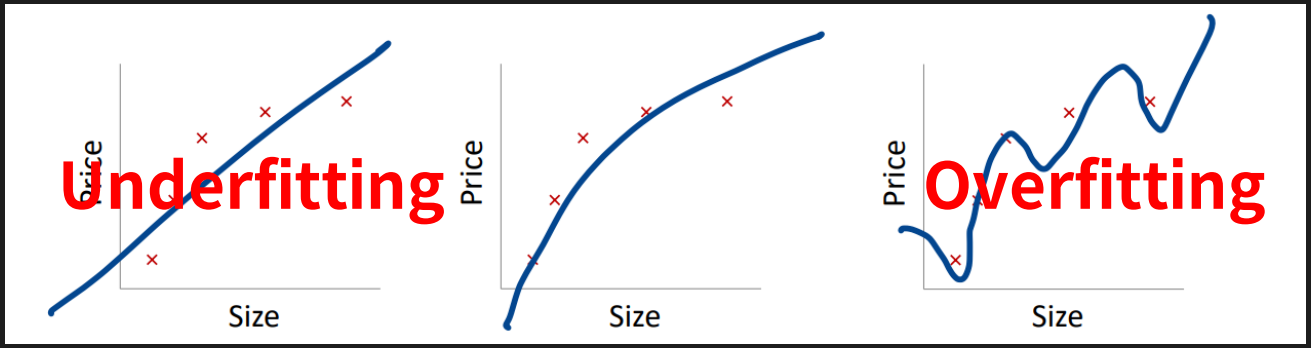
Bias-Variance Problem

Bias² is the squared difference between the model's average prediction and the true values

Irreducible Error is the inherent noise in the data, which cannot be reduced by improving the model

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Variance measures the model's inconsistency across different training sets



High Bias "Just Right" High Variance

모델이 복잡해질수록,
Bias는 감소하지만
Variance는 증가한다.
Bias-Variance Trade-off

ML&DL Basics

Optimization Technique

(Recap) Gradient Descent

- Model:

$$f_{\mathrm{w}}(x) = w_0 + w_1 x$$

- Parameters:

$$\mathrm{w} = (w_0, w_1)$$

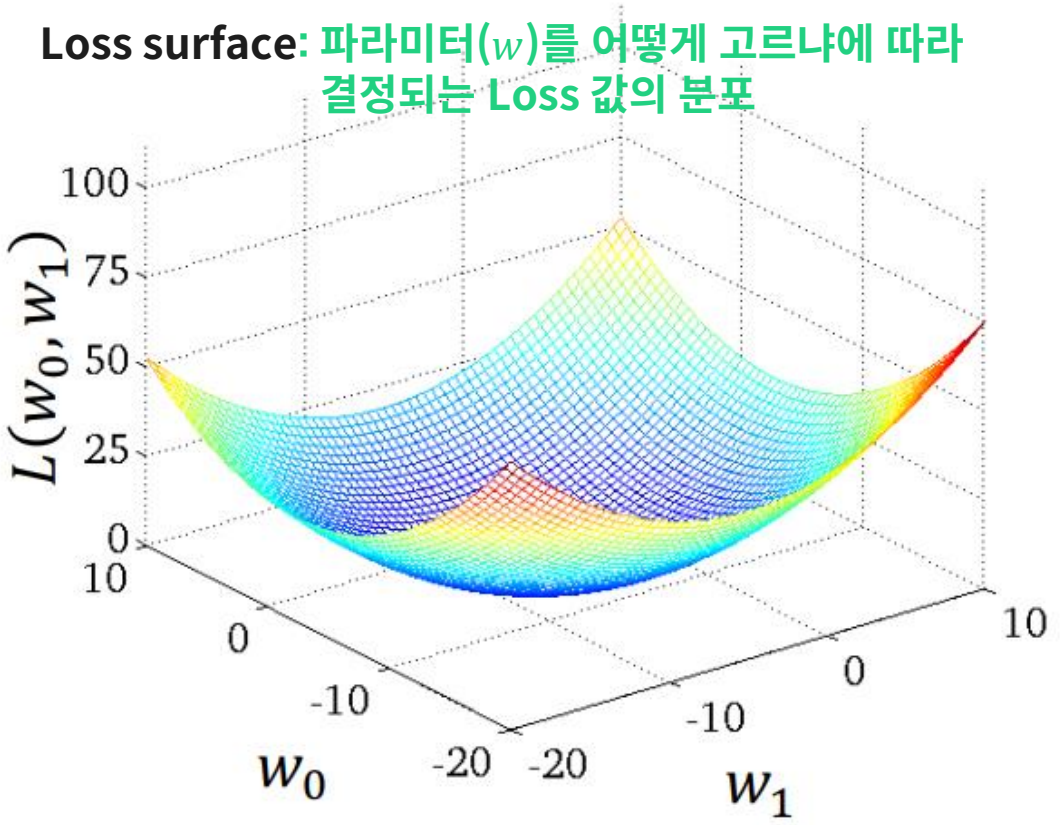
- Cost function:

$$L(\mathrm{w}) = \frac{1}{N}\sum_{n=1}^{N}(f_{\mathrm{w}}(x_n) - y_n)^2$$

- Goal:

$$\mathrm{w}^* = \underset{\mathrm{w}}{\operatorname{argmin}}\, L(\mathrm{w})$$

Loss surface: 파라미터(w)를 어떻게 고르냐에 따라 결정되는 Loss 값의 분포



어떻게 찾을 것인가? : Gradient Descent

(Recap) Gradient Descent

- Model:

$$f_{\mathrm{w}}(x) = w_0 + w_1 x$$

- Parameters:

$$\mathrm{w} = (w_0, w_1)$$

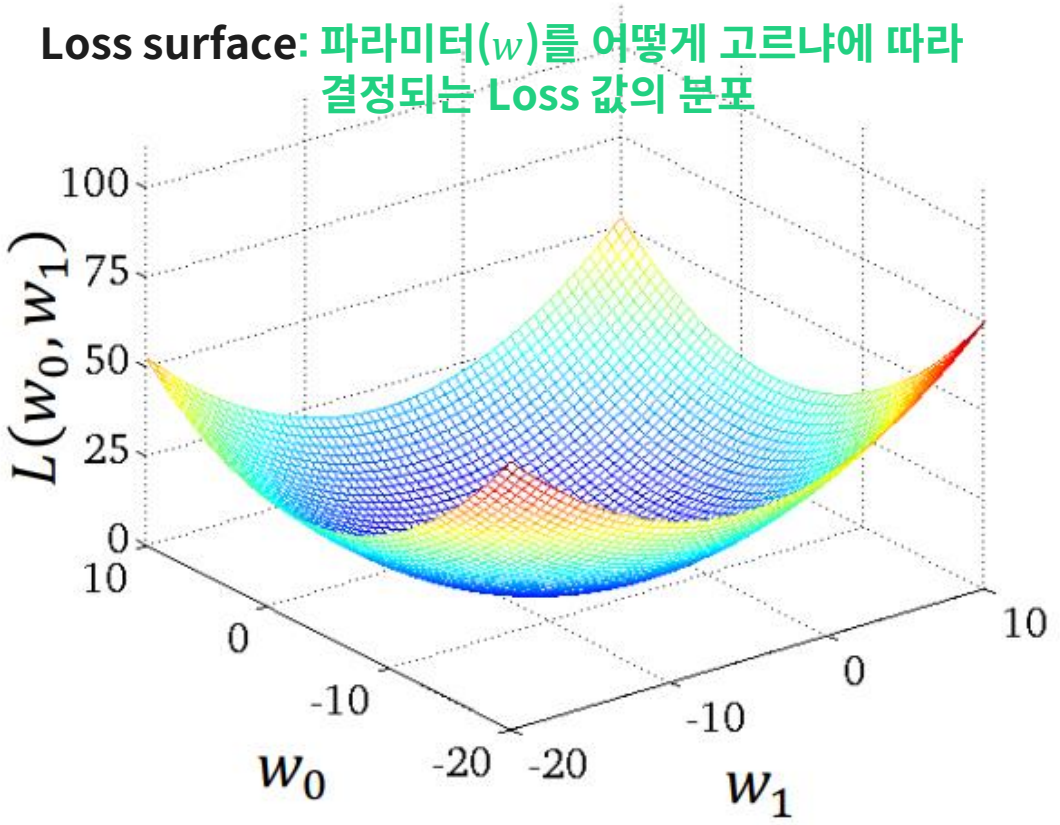
- Cost function:

$$L(\mathrm{w}) = \frac{1}{N}\sum_{n=1}^{N}(f_{\mathrm{w}}(x_n) - y_n)^2$$

- Goal:

$$\mathrm{w}^* = \underset{\mathrm{w}}{\operatorname{argmin}}\, L(\mathrm{w})$$

Loss surface: 파라미터(w)를 어떻게 고르냐에 따라 결정되는 Loss 값의 분포



가지고 있는 모든 데이터셋에 대해 평가한 후, Loss 계산이 이루어짐

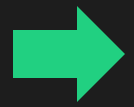
Pros & Cons of Gradient Descent

장점 :

- 모든 데이터셋에 대해 Loss를 계산 한 후 Gradient를 계산한다.
- 한 번의 Step이 더 정확할 것이므로, 더 적은 Step 수로 Global/Local Minima를 찾을 수 있을 것이다.

단점 :

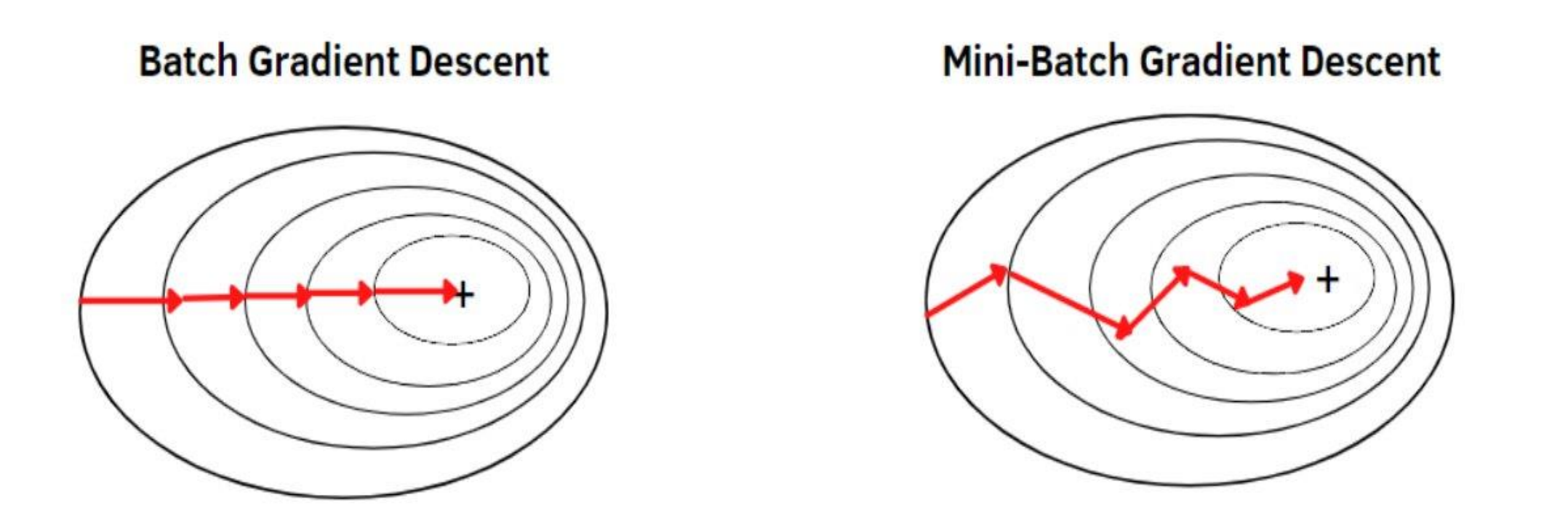
- 모든 데이터셋에 대해 Loss를 계산 해야 한 후 Gradient를 계산한다.
- 만약 데이터셋의 크기가 아주 크다면 GPU가 데이터를 모두 처리하지 못할 수도 있다. (OOM)
- 다른 테크닉을 사용하더라도, 매우 많은 횟수의 Inference를 거친 후에야 한 Step의 Update가 가능하다.



데이터셋의 '일부(Mini-Batch)'만 보고, 파라미터를 업데이트 하자!

Stochastic(Mini-Batch) Gradient Descent

SGD



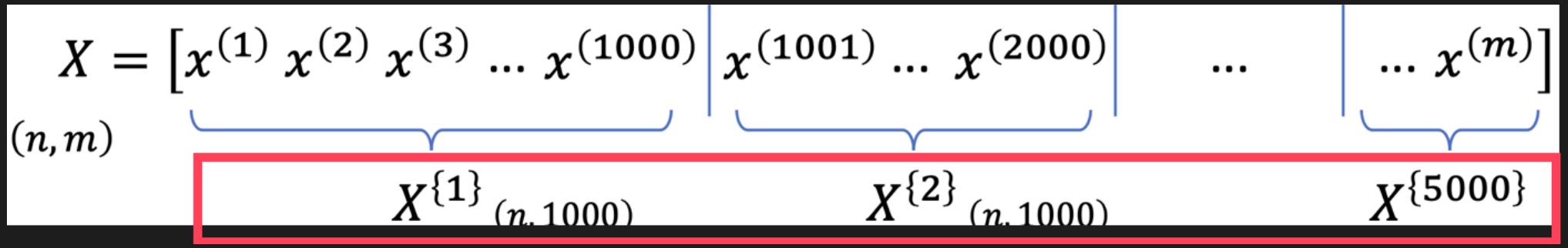
데이터셋의 '일부(Mini-Batch)'만 보고, 파라미터를 업데이트 하자!

Stochastic(Mini-Batch) Gradient Descent

SGD

```
# Step 3: Training
for epoch in range(num_epochs):
    for batch in get_batches(train_data, batch_size):
```

…

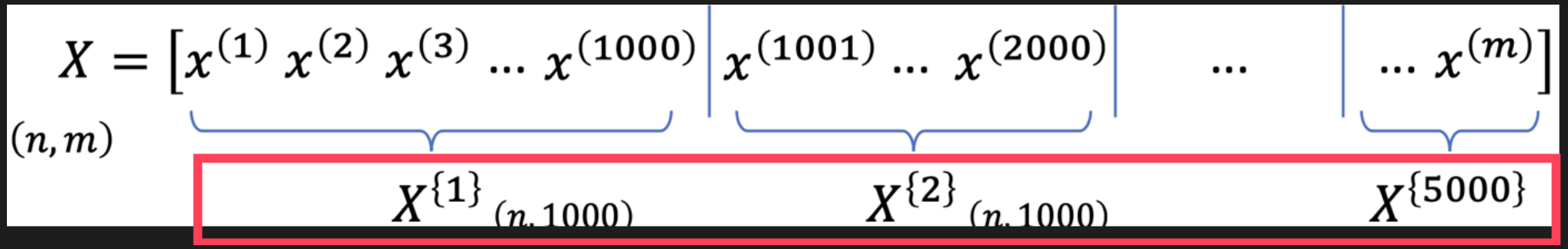


or “Epoch”

batch

mini-batch

SGD



or "Epoch"

batch

mini-batch

$$\text{SGD} \Rightarrow \ w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

Advanced Optimizers

Momentum

- 물리학에서의 관성의 개념 도입, 진동 방향을 상쇄해 더 효율적으로 Update 가능

SGD

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

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

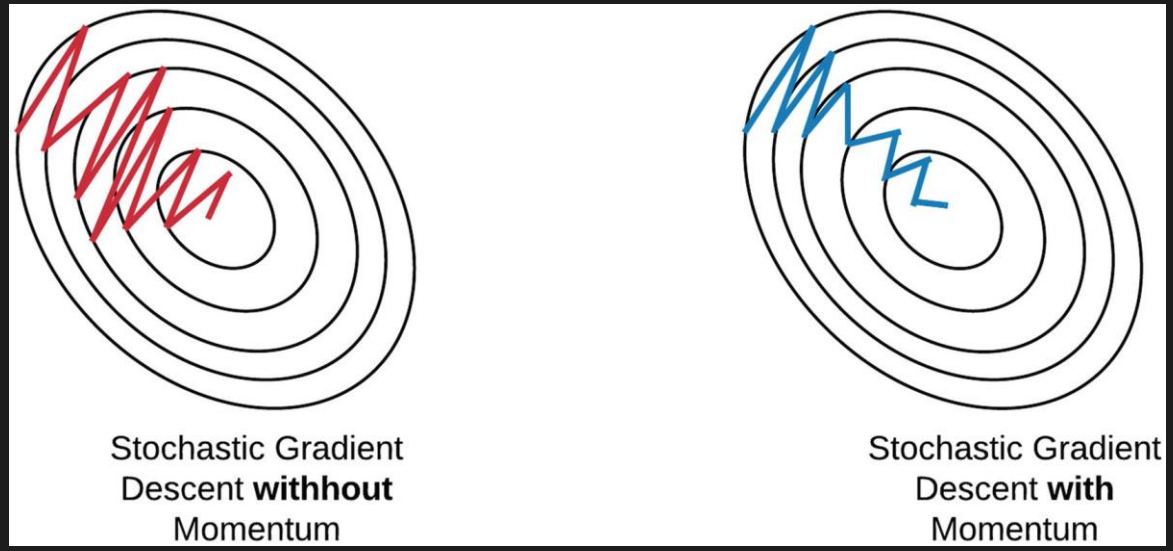
SGD+Momentum

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

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013



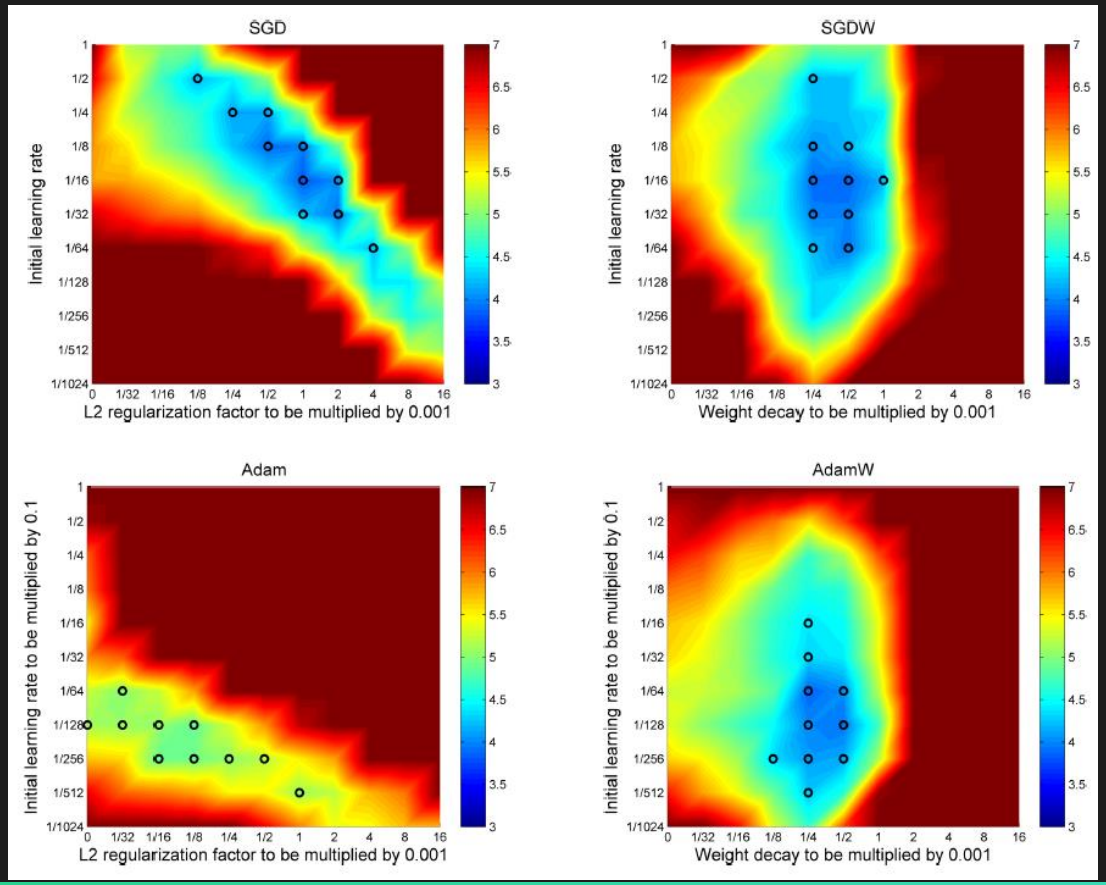
Stochastic Gradient Descent **without** Momentum

Stochastic Gradient Descent **with** Momentum

Advanced Optimizers

And more…

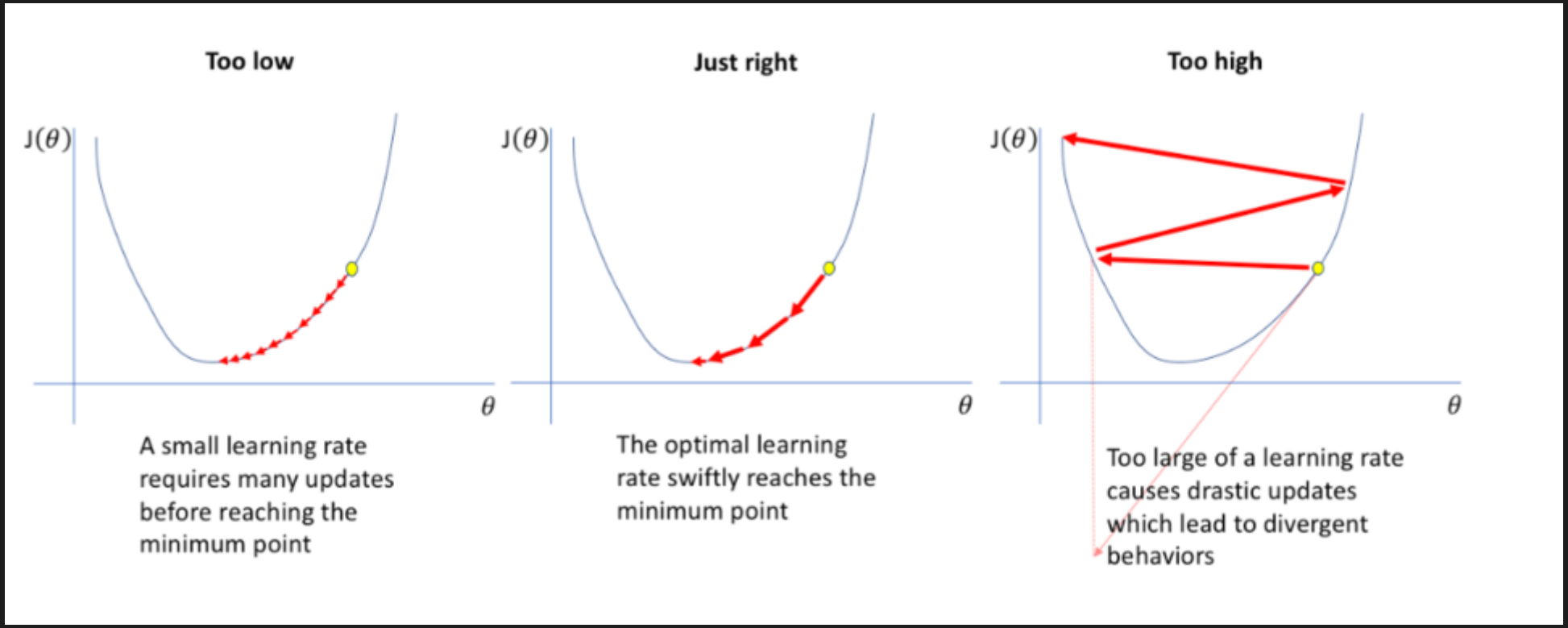
- 많은 Optimizer가 있으며, SGD, Adam과 AdamW가 대중적으로 가장 많이 사용됨.



Learning Rate Scheduler

Learning Rate

- LR이 잘못 설정되면, 제대로 된 학습이 이루어지지 않을 수 있음.

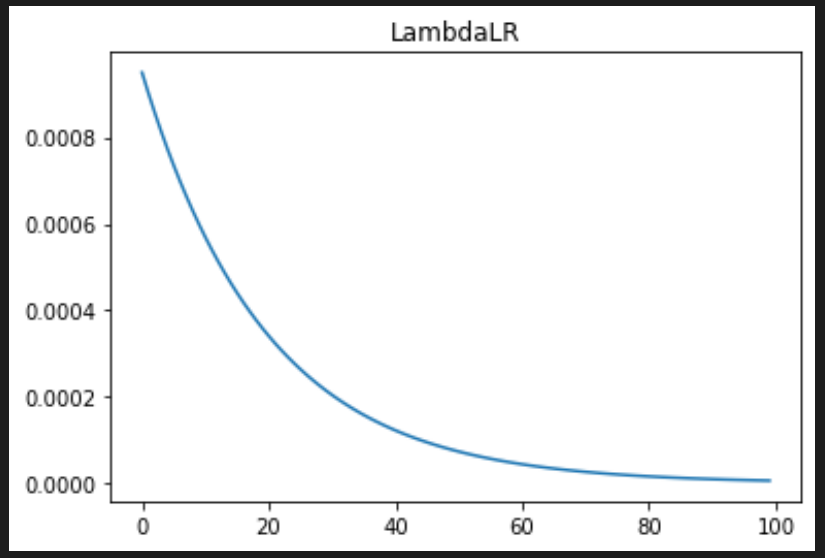


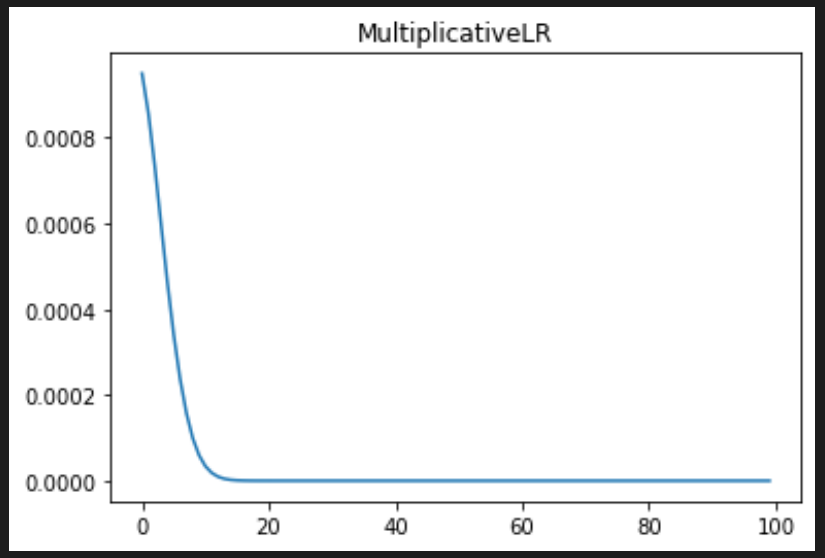
Learning Rate Scheduler

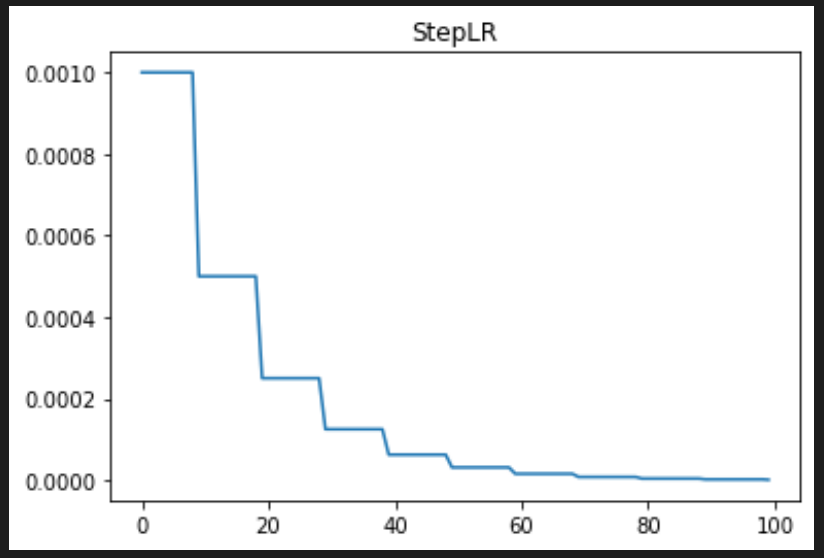
Learning Rate

- LR이 잘못 설정되면, 제대로 된 학습이 이루어지지 않을 수 있음.
- 학습 초반에는 LR를 크게, 중~후반에는 작게 가져가자!

(Note) 수학적 배경은 "Simulated Annealing" 참조





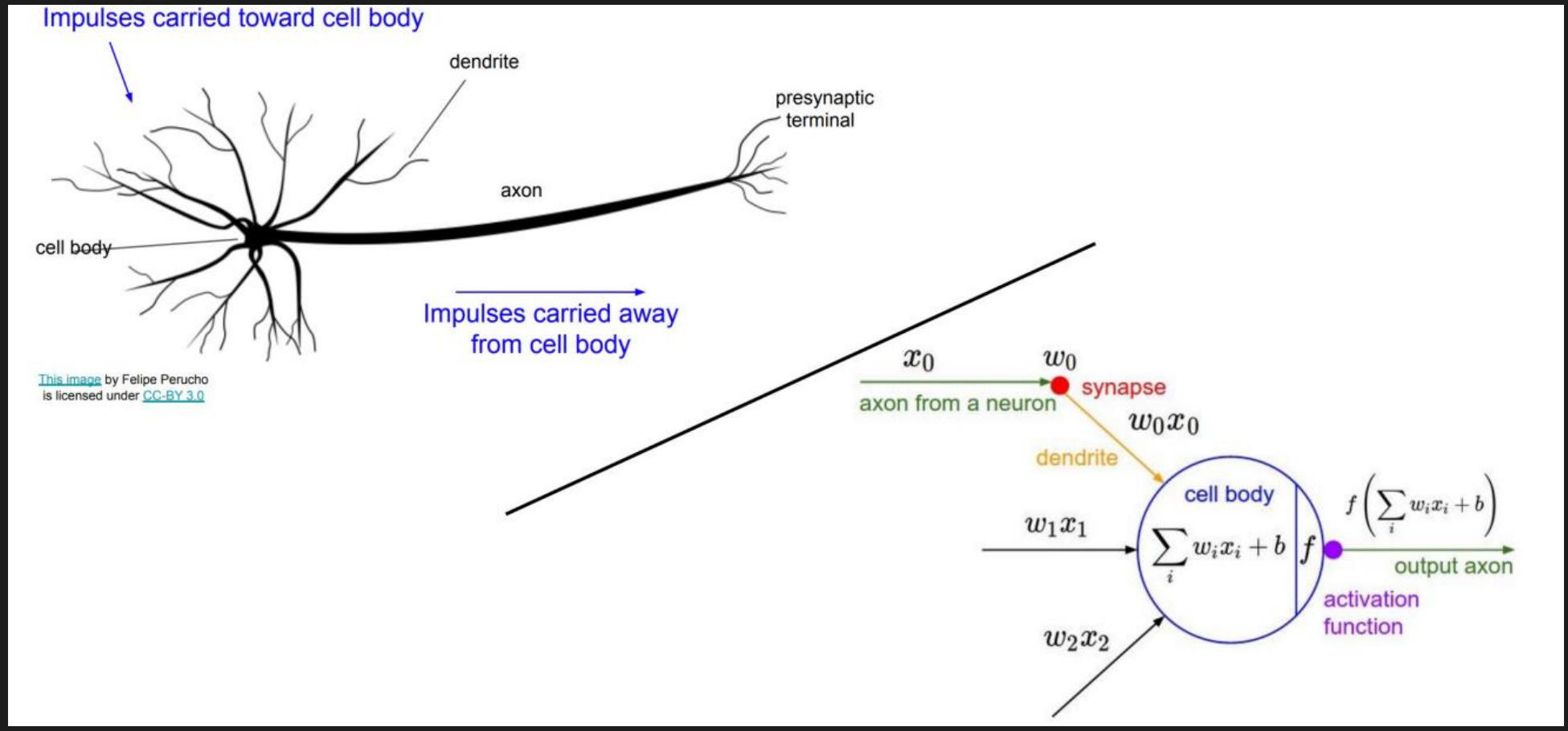


ML&DL Basics

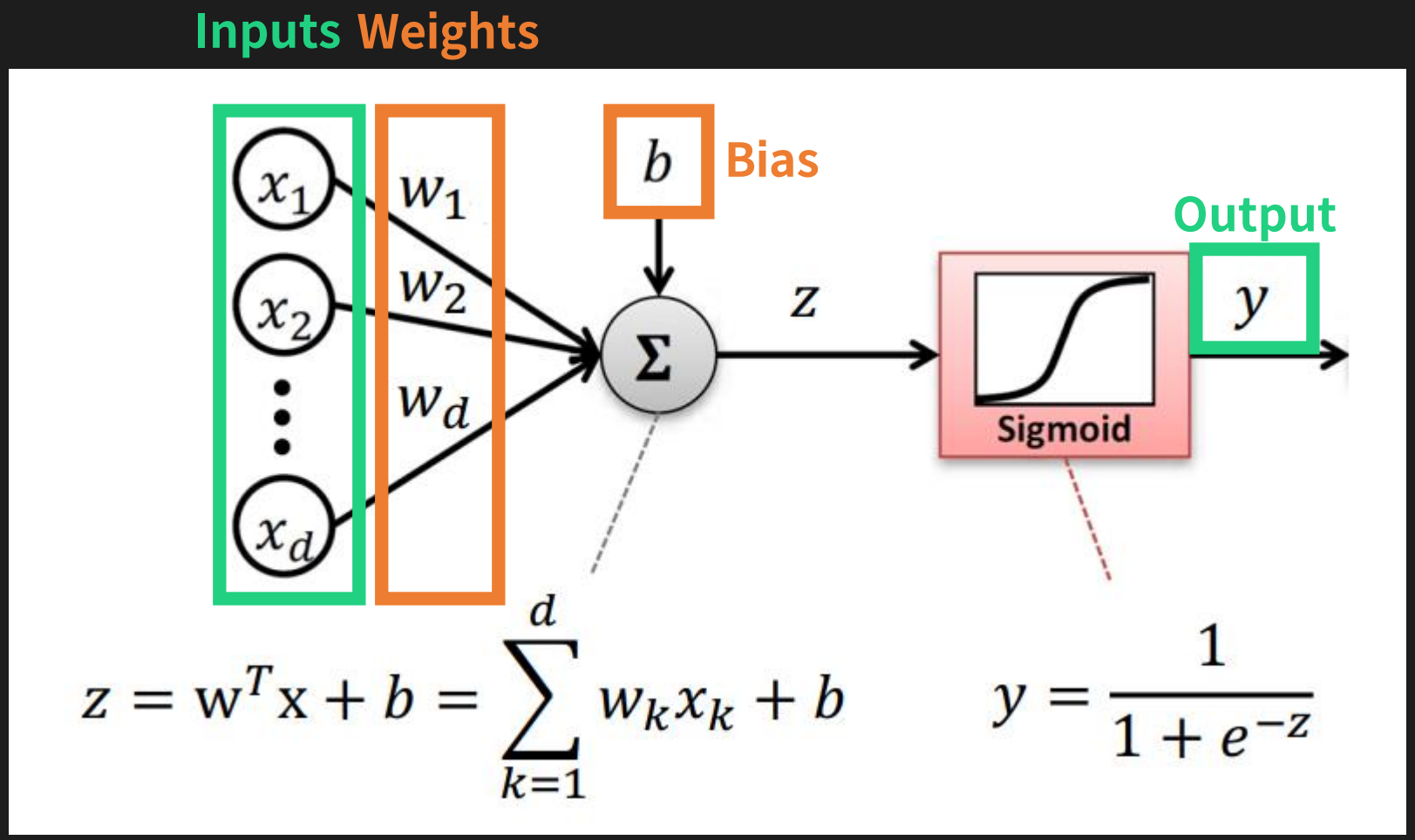
Introduction to Neural Network

The Perceptron

- Inspired by Neurobiology



Single Layer Peceptron

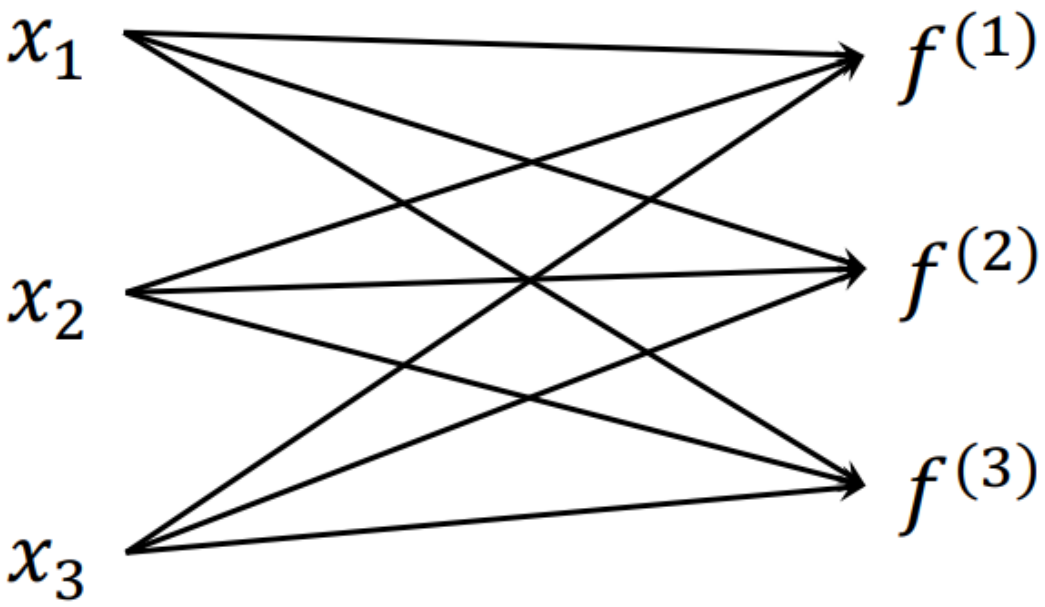


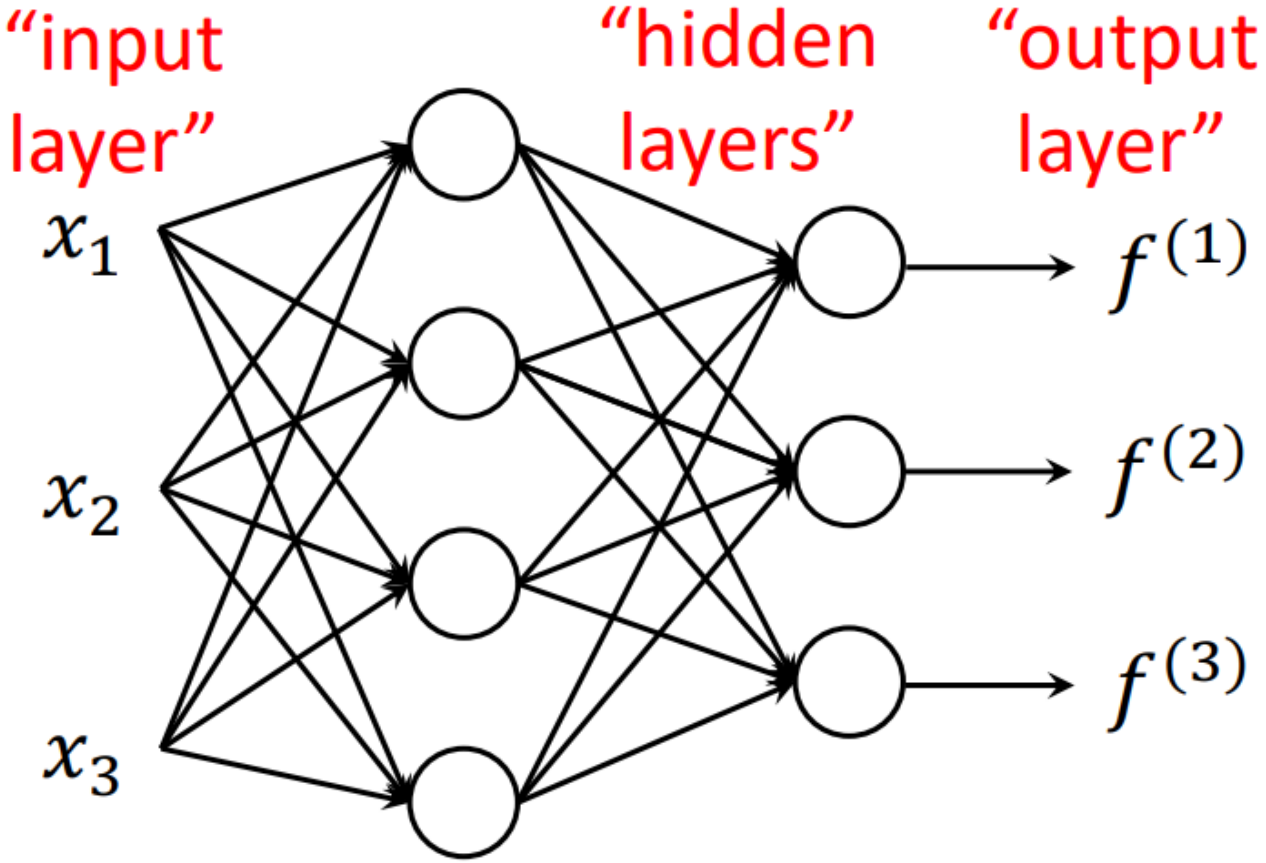
- Logistic Regression Model의 형태와 동일
- Linear Decision Boundary
  -> 더 복잡한 문제 풀기 어려움

Soulution :
Stack More Layers!

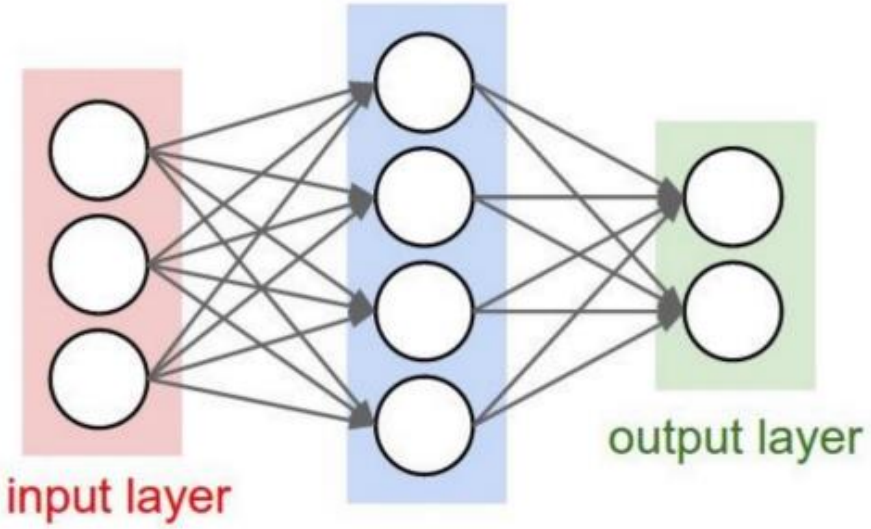
Multi-Layer Perceptron (MLP)

$$x_n = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}$$

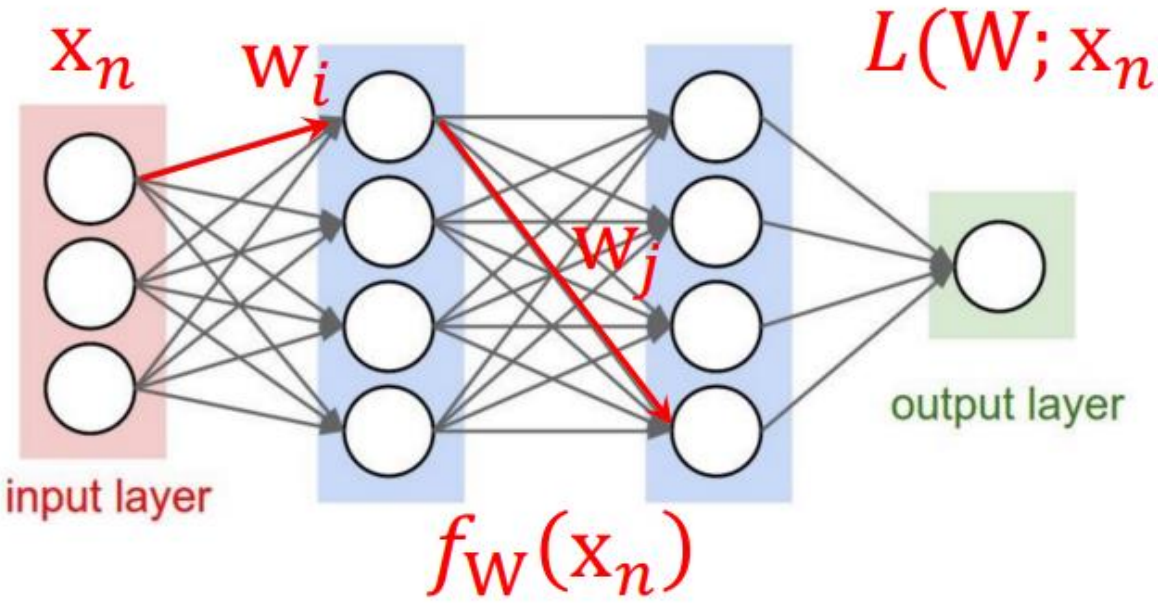




Multi-Layer Perceptron (MLP)



"2-layer Neural Net."
or "1-hidden-layer Neural Net."



"3-layer Neural Net."
or "2-hidden-layer Neural Net."

Backpropagation

- Neural Network에서의 Gradient Descent

- $L(\mathrm{W}) = \mathrm{loss}(f_{\mathrm{W}}(\mathrm{x}_n), y_n)$

Want $\underset{\mathrm{W}}{\mathrm{argmin}}\, L(\mathrm{W})$

Repeat{

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} L(\mathrm{W})$$

(simultaneously update all w_j)

}

$\mathrm{W} = \{\mathrm{W}_1, \mathrm{W}_2\}$

$\mathrm{W}_1 \in \mathbb{R}^{4\times 3}, \mathrm{W}_2 \in \mathbb{R}^{4\times 1}$

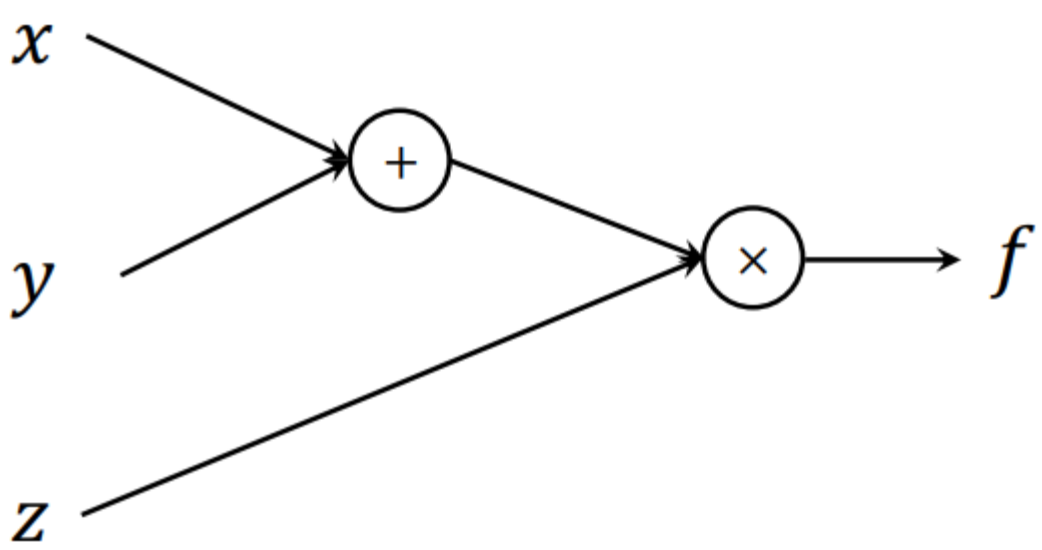
문제점

- 아주 많은 행렬 연산 필요
- Loss를 바꾸고 싶다면, 모든 Weight의 Gradient를 다시 계산 필요
- 아주 복잡한 모델에 대해서는 현실적으로 불가능

- Gradient를 더 효율적으로 계산할 수 없을까?

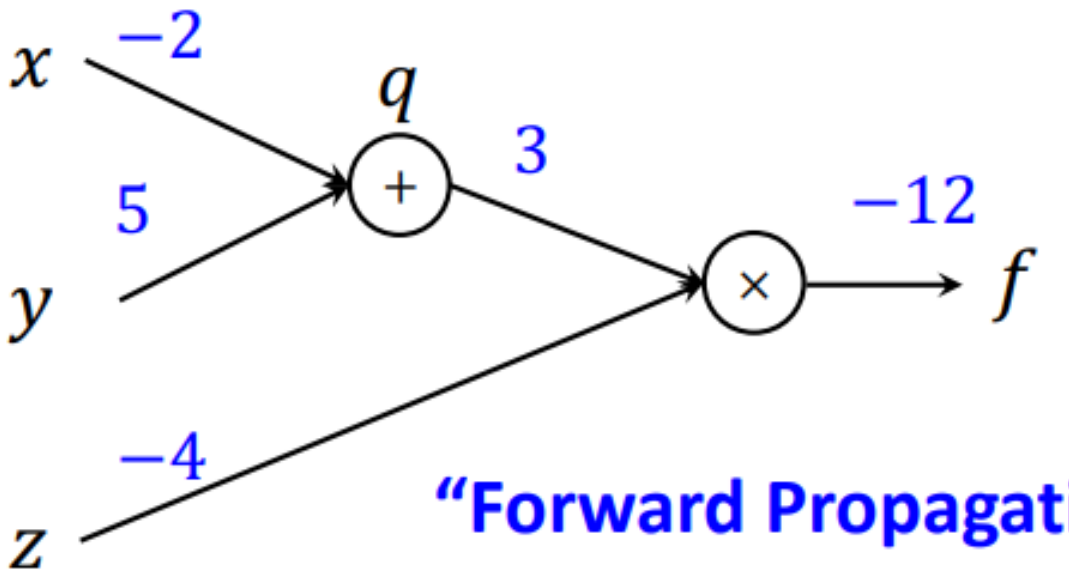
Backpropagation

- $f(x, y, z) = (x + y)z$
  e.g.,
  $x = -2, y = 5, z = -4$
- Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
- $q = x + y \rightarrow \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$
- $f = q * z \rightarrow \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$



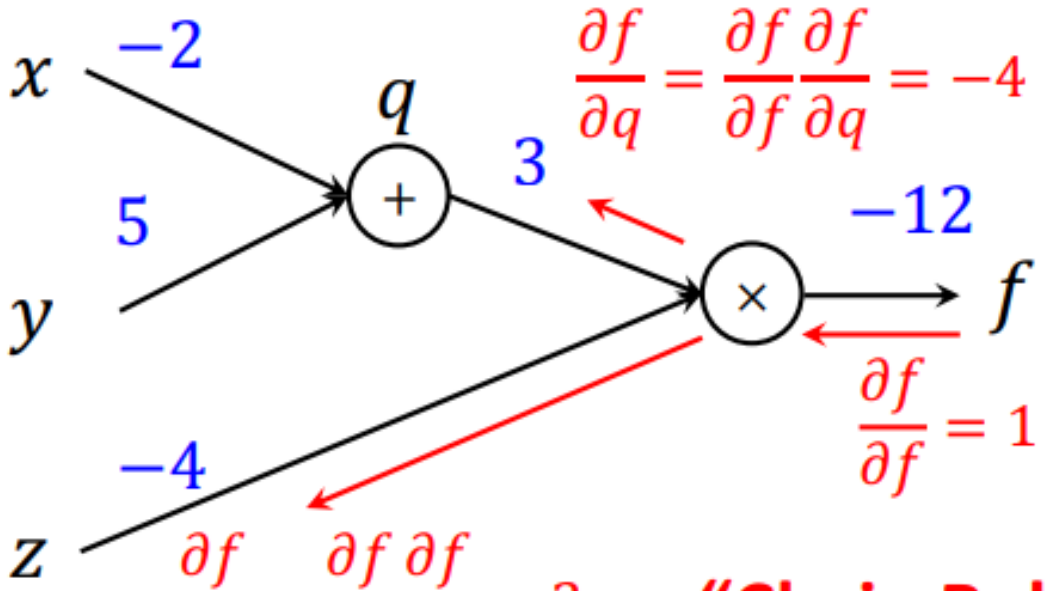
Backpropagation

- $f(x, y, z) = (x + y)z$
  e.g.,
  $x = -2, y = 5, z = -4$
- Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
- $q = x + y \rightarrow \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$
- $f = q * z \rightarrow \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$



Backpropagation

- $f(x, y, z) = (x + y)z$
  e.g.,
  $x = -2, y = 5, z = -4$
- Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
- $q = x + y \rightarrow \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$
- $f = q * z \rightarrow \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

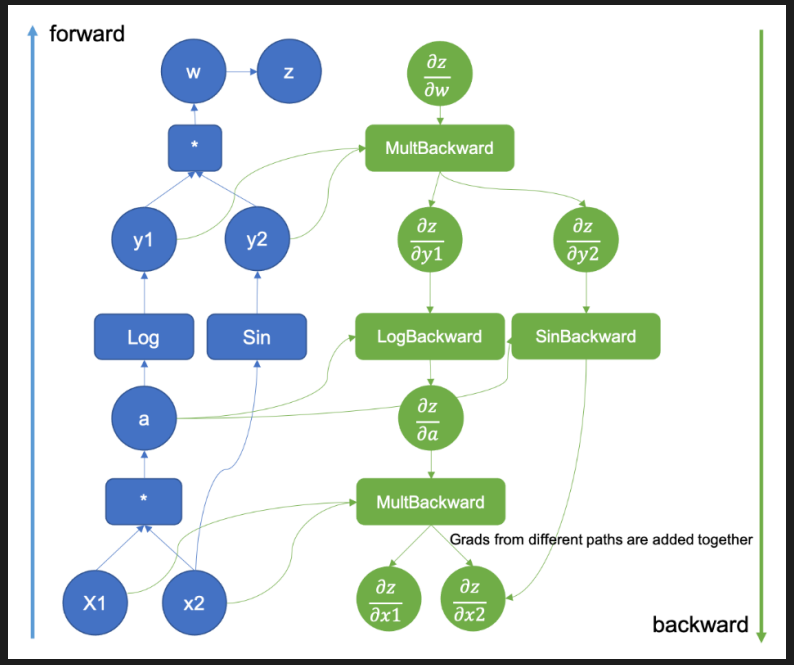


"Backward Propagation"
or **"Backpropagation"**

Backpropagation in PyTorch

- Autograd : Automatic gradient calculating API

  해당 변수가 계산되는 데에 사용되었던 모든 변수들과 관련된 계산 그래프를 자동으로 저장, Backpropagation이 효율적으로 진행될 수 있도록 해줌.



PyTorch에서 autograd 동작 이해하기 (velog.io)

Backpropagation in PyTorch

```
# requires_grad=True로 해야 x의 연산들을 추적하기 시작한다.

x = torch.ones(2,2, requires_grad=True)
y = x+1
z = 2*y**2
res = z.mean()
res.backward() # 역전파(Backpropagation) 계산
```

```
tensor([[1., 1.],
        [1., 1.]],
requires_grad=True)
```

Backpropagation in PyTorch

```
# requires_grad=True로 해야 x의 연산들을 추적하기 시작한다.


x = torch.ones(2,2, requires_grad=True)
y = x+1
z = 2*y**2
res = z.mean()
res.backward() # 역전파(Backpropagation) 계산
```

```
tensor([[2., 2.],
        [2., 2.]],
grad_fn=<AddBackward0>)
```

Backpropagation in PyTorch

```
# requires_grad=True로 해야 x의 연산들을 추적하기 시작한다.

x = torch.ones(2,2, requires_grad=True)
y = x+1
z = 2*y**2
res = z.mean()
res.backward() # 역전파(Backpropagation) 계산
```

```
tensor(8.,
grad_fn=<MeanBackward0
>)
```

Backpropagation in PyTorch

```
# requires_grad=True로 해야 x의 연산들을 추적하기 시작한다.

x = torch.ones(2,2, requires_grad=True)
y = x+1
z = 2*y**2
res = z.mean()
res.backward() # 역전파(Backpropagation) 계산
```

```
x.grad =>
tensor([[2., 2.],
        [2., 2.]])
```

Multi-Layer Perceptron (MLP)

- $f = W_3 g(W_2 g(W_1 x_n)$
- "**Neural Network**" is a very broad term;
  these are more accurately called "**Fully-Connected(FC) Networks**"
  or sometimes "**Multi-Layer Perceptron (MLP)**".
- $g(z)$: Activation Function / 네트워크에 **Nonlinearity**를 부여해줌.

Q. 만약 Activation Function이 없다면?

Multi-Layer Perceptron (MLP)

- $f = W_3 g(W_2 g(W_1 x_n)$
- "**Neural Network**" is a very broad term;
  these are more accurately called "**Fully-Connected(FC) Networks**"
  or sometimes "**Multi-Layer Perceptron (MLP)**".
- $g(z)$: Activation Function / 네트워크에 **Nonlinearity**를 부여해줌.

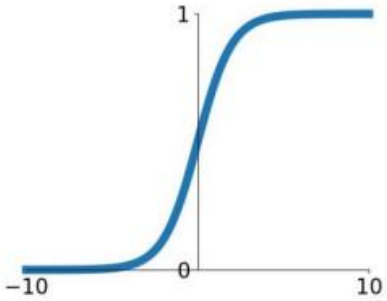
Q. 만약 Activation Function이 없다면?

$$f = W_3 W_2 W_1 x_n = W x_n$$

Activation Functions

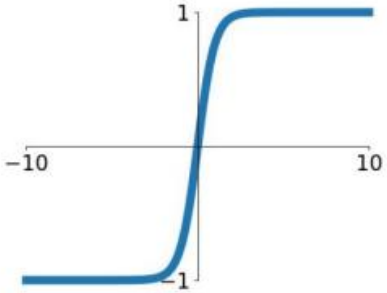
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



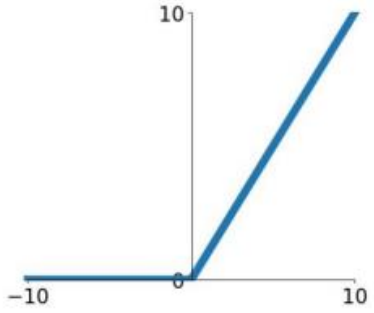
tanh

$$\tanh(x)$$



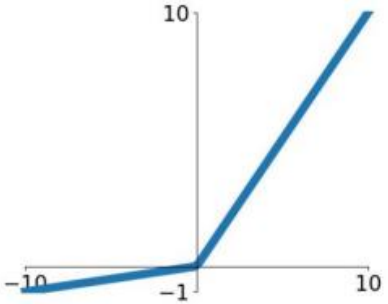
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

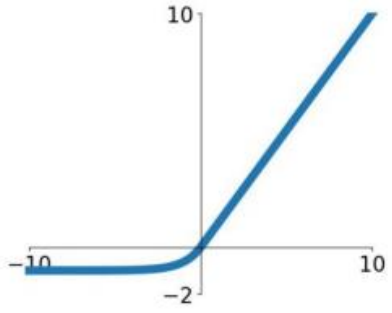


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

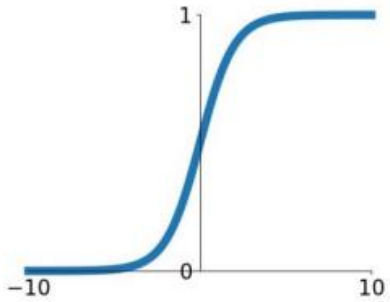
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmoid Function

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Problem

1. Gradient vanishing problem
2. Not zero-centered outputs
3. Exp() is a bit compute expensive

Tanh Function

- Squashes numbers to range [0,1]
- Zero-centered outputs

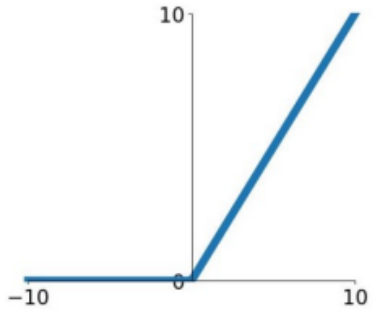
Problem

1. Gradient vanishing problem
2. Exp() is a bit compute expensive

Activation Functions

ReLU

$\max(0, x)$



- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

Problem

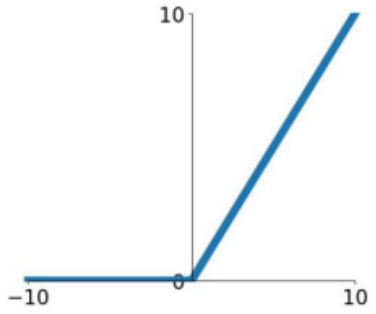
1. Not zero-centered output
2. "Dead ReLU" Problem

$\frac{\partial L}{\partial w_j} = \left(\frac{\partial L}{\partial s}\right)_j \mathrm{x} = g s_j \mathrm{x}$ will be never updated if $s_j \leq 0$

Activation Functions

ReLU

$\max(0, x)$



- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

Problem

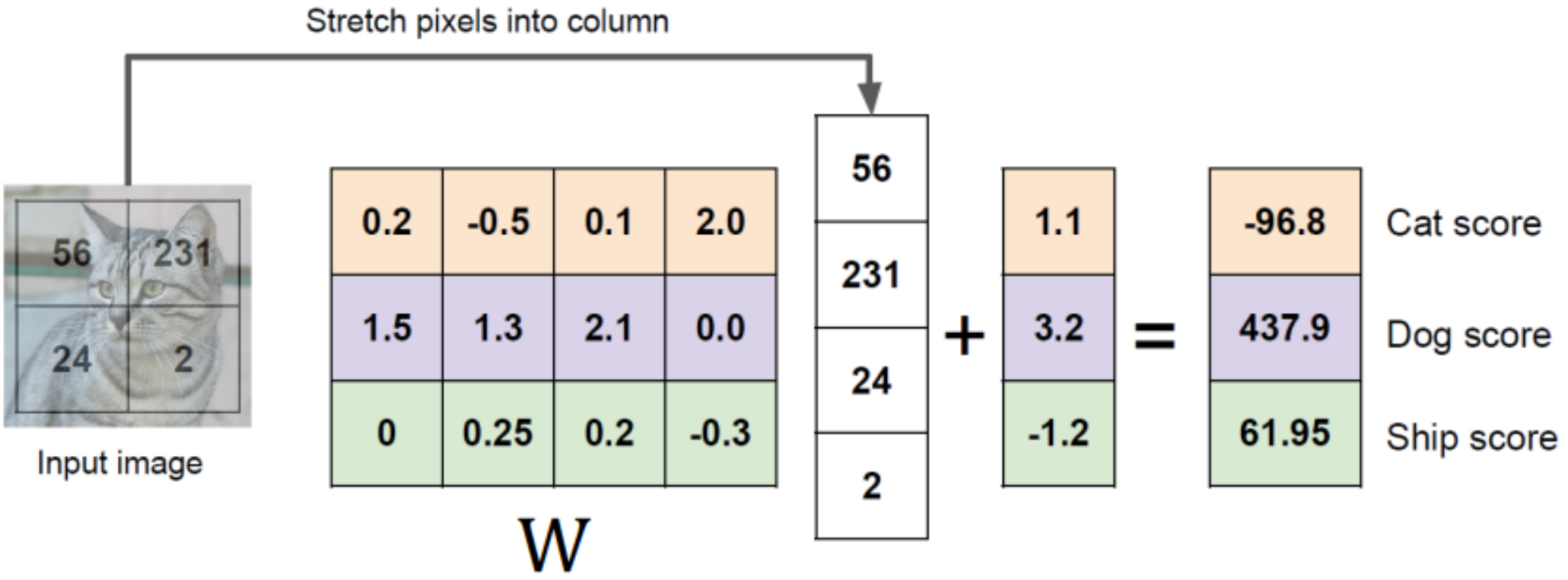
1. Not zero-centered output
2. "Dead ReLU" Problem
   ➡ Leaky ReLU

$$\frac{\partial L}{\partial w_j} = \left(\frac{\partial L}{\partial s}\right)_j \mathrm{x} = g s_j \mathrm{x} \text{ will be never updated if } s_j \leq 0$$

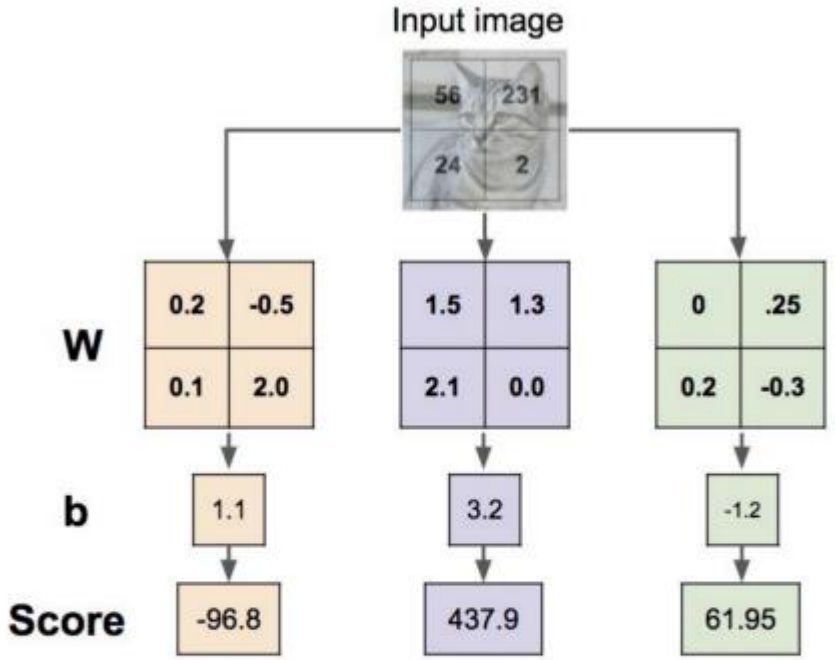
Training Neural Network

Ex) An image with 4 pixels and 3 classes (cat/dog/ship)

$$f_W(x_n) = Wx_n + b$$



Training Neural Network



Training Neural Network

$$f_W(x_n) = Wx_n + b$$

How can we tell whether this W is good or bad?

Loss function
(quantifying what it means to have a "good" W)

Optimization
(start with random W and find a W that minimizes the loss)

Example class scores for 3 images for *some* W:



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

(Recap) Logistic Regression for Binary Classification

확률 대신, Logit을 Regress해보자! (단, 여전히 우리의 목표는 확률 구하기)

$$\text{logit}(p) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

반대로, Sigmoid 함수의 역할이 Regression된 값을 0과 1의 확률값 범위로 mapping하는 것이라고도 생각할 수 있음.

해당 모델로부터 데이터 x가 어떤 클래스에 속할 확률(p)을 역으로 계산할 수 있다.

Logistic Regression Model

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

Sigmoid Function

Multi-class Classification

Example class scores for 3 images for *some* W:



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

한 개의 Sample에 대해서 여러 개의 Class Score Output 존재

➡ 이를 일종의 예측된 Class Score 확률 분포로 변환하자!

Multi-class Classification

Example class scores for 3 images for *some* W:



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

한 개의 Sample에 대해서 여러 개의 Class Score Output 존재

➡ 이를 일종의 예측된 Class Score 확률 분포로 변환하자!

➡ Softmax Function

$$p = \frac{\exp(z)}{\sum_j \exp(z_j)}$$

Multi-class Classification

- Want to interpret raw classifier scores as **probabilities**

x_n

$$p_{n,k} = P(Y = k|X = x_n) = \frac{\exp(z_{n,k})}{\sum_j \exp(z_{n,j})}$$

Softmax function

Probabilities must be >= 0

Probabilities must sum to 1

y_n

cat	**3.2**	exp →	**24.5**	norm. →	**0.13**	→ **Compare** ← **1.00**
car	5.1		164.0		0.87	0.00
frog	-1.7		0.18		0.00	0.00

Unnormalized Probabilities

Probabilities

Correct prob.

$$z_{n,k} = f_W(x_n)$$

$$L(W) = -\log p_{n,y_n} = -\log P(Y = y_n|X = x_n)$$

Multi-class Classification

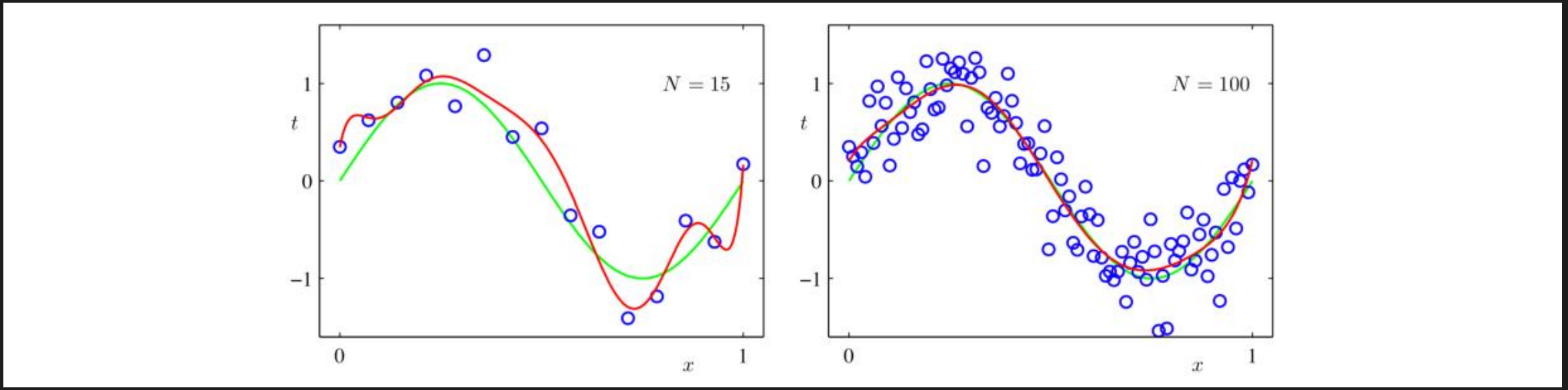
ML&DL Basics

Data Processing

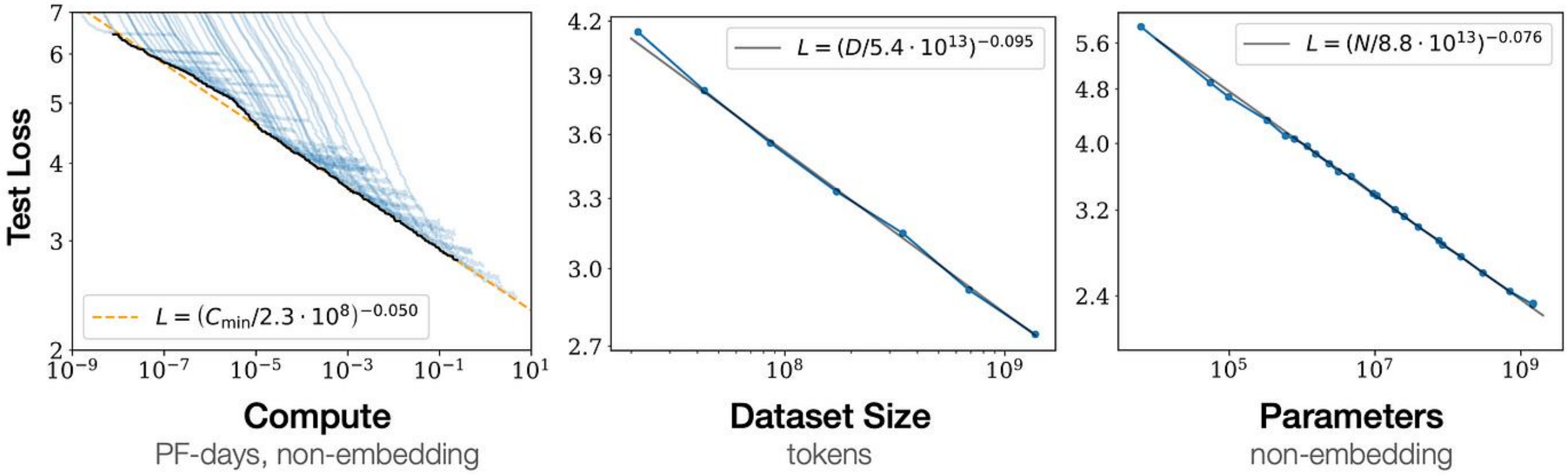
Why Data is Important? : Scaling Law

모델의 Capacity가 충분하다면,

더 많은 (좋은) 데이터로 학습된 모델은 반드시 더 뛰어난 성능을 낼 수 있다.



Why Data is Important? : Scaling Law



Why Data is Important? : Scaling Law

어떻게 더 많은 데이터를 얻을 수 있을까?

1. Human Annotator -> 데이터셋의 크기가 작을 때는 가능, 클 때는 비용 문제 발생

Why Data is Important? : Scaling Law

어떻게 더 많은 데이터를 얻을 수 있을까?

1. Human Annotator -> 데이터셋의 크기가 작을 때는 가능, 클 때는 비용 문제 발생
2. Unsupervised / Semi-supervised / Self-Supervised Learning

Why Data is Important? : Scaling Law

어떻게 더 많은 데이터를 얻을 수 있을까?

1. Human Annotator -> 데이터셋의 크기가 작을 때는 가능, 클 때는 비용 문제 발생
2. Unsupervised / Semi-supervised / Self-Supervised Learning
3. Data Augmentation

Data Augmentation

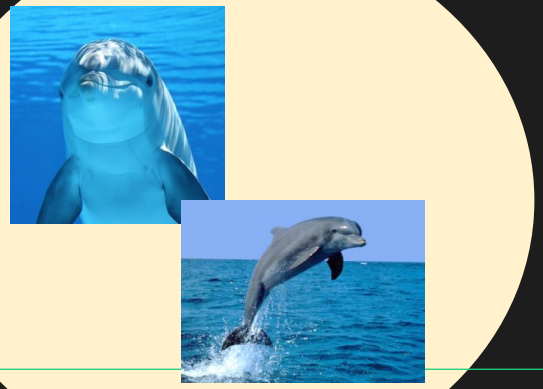
인간은 개의 사진을 뒤집어도, 색을 반전시키고, 일부를 지우더라도
여전히 “개”로 인식할 수 있다. 하지만 AI는?

Training Set

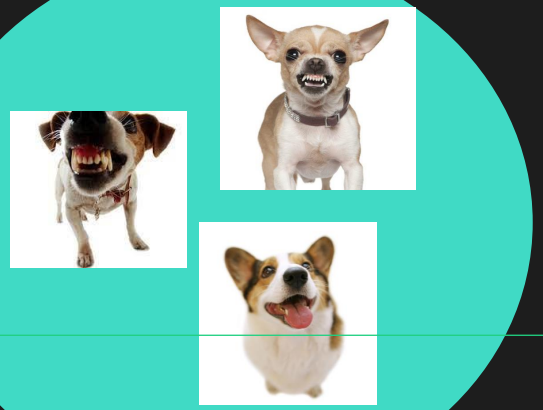
Test Set

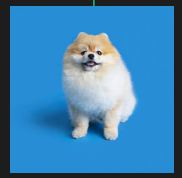
Data Augmentation

모델이 인식하는 "돌고래"의 범위



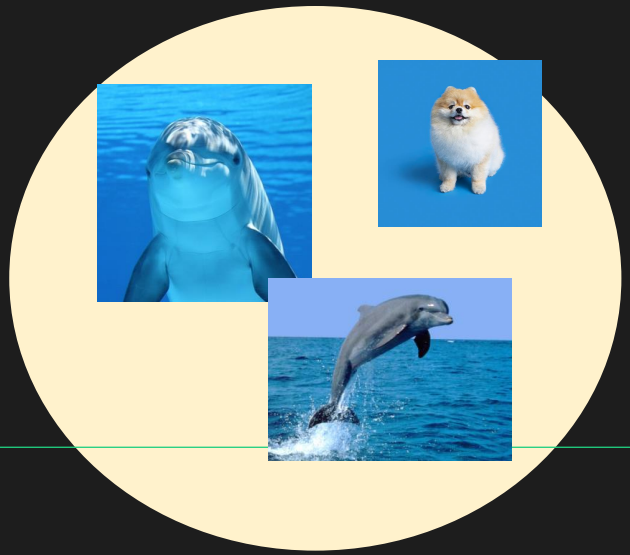
모델이 인식하는 "개"의 범위



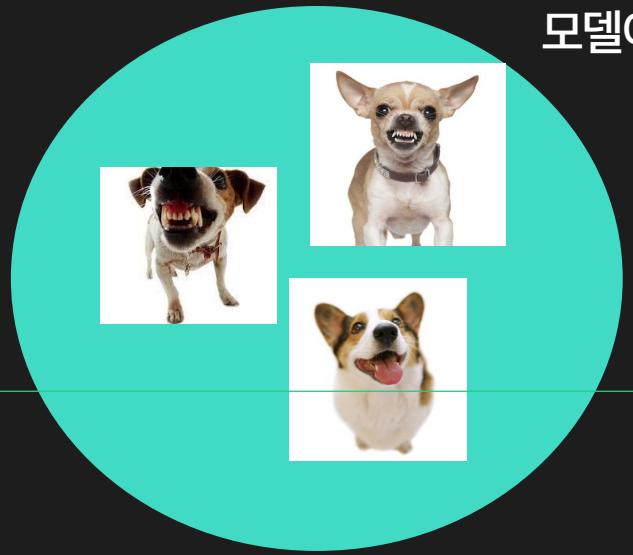


Data Augmentation

모델이 인식하는 "돌고래"의 범위



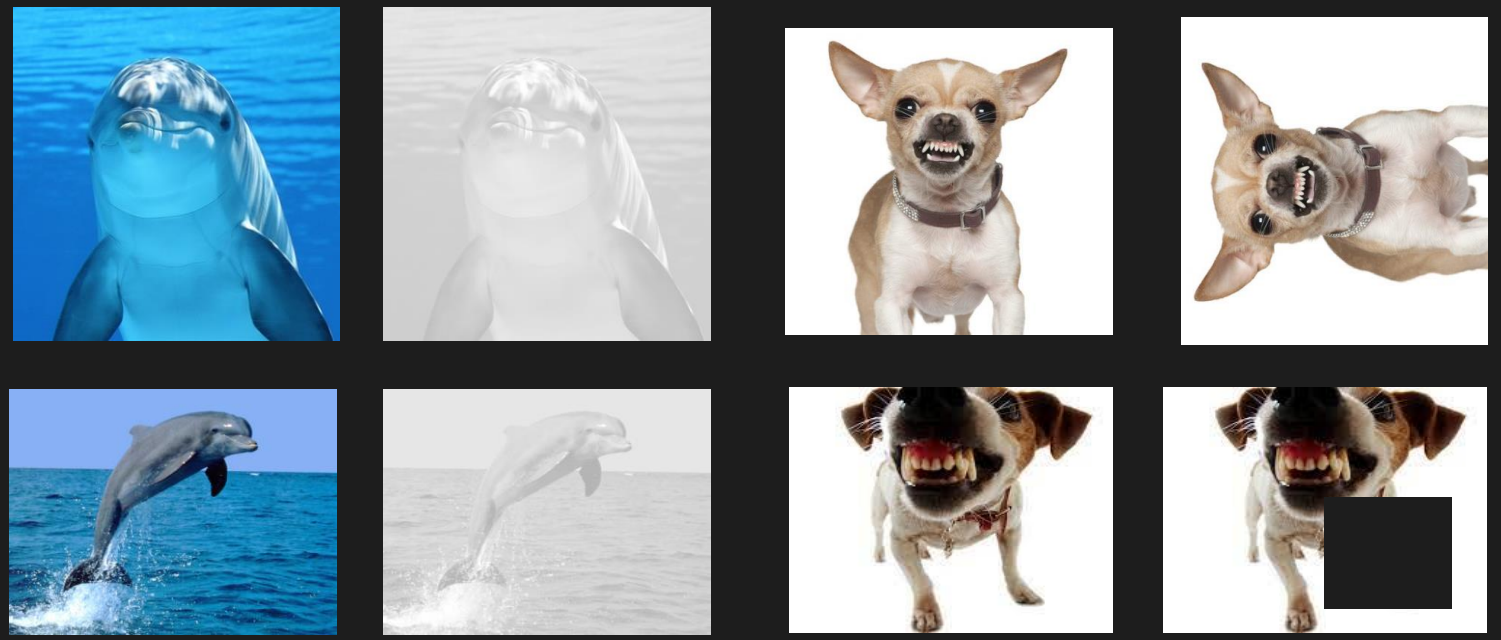
모델이 인식하는 "개"의 범위



Data Augmentation

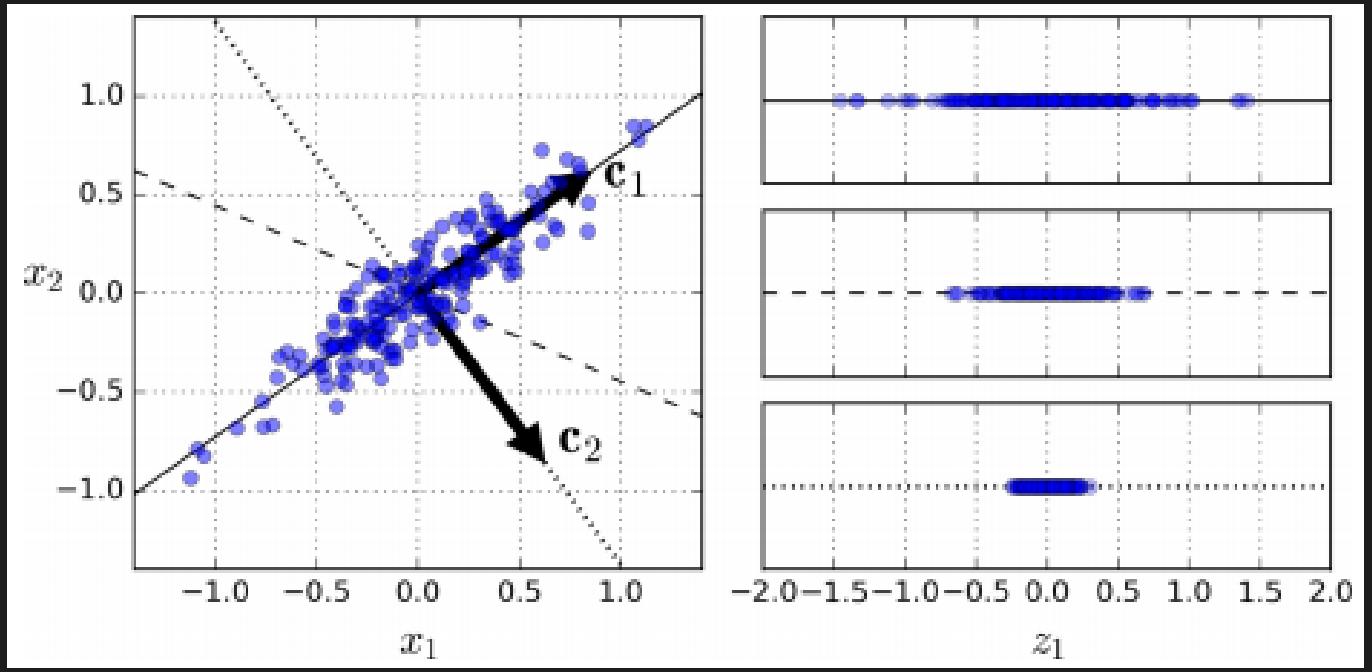
Input 데이터의 일부를 자르고, 색칠하는 등 변형하여 데이터를 “증강“한다.

모델이 데이터에서 “더 중요한 요소”에 더 집중할 수 있도록 도와주며, Overfitting을 방지한다.



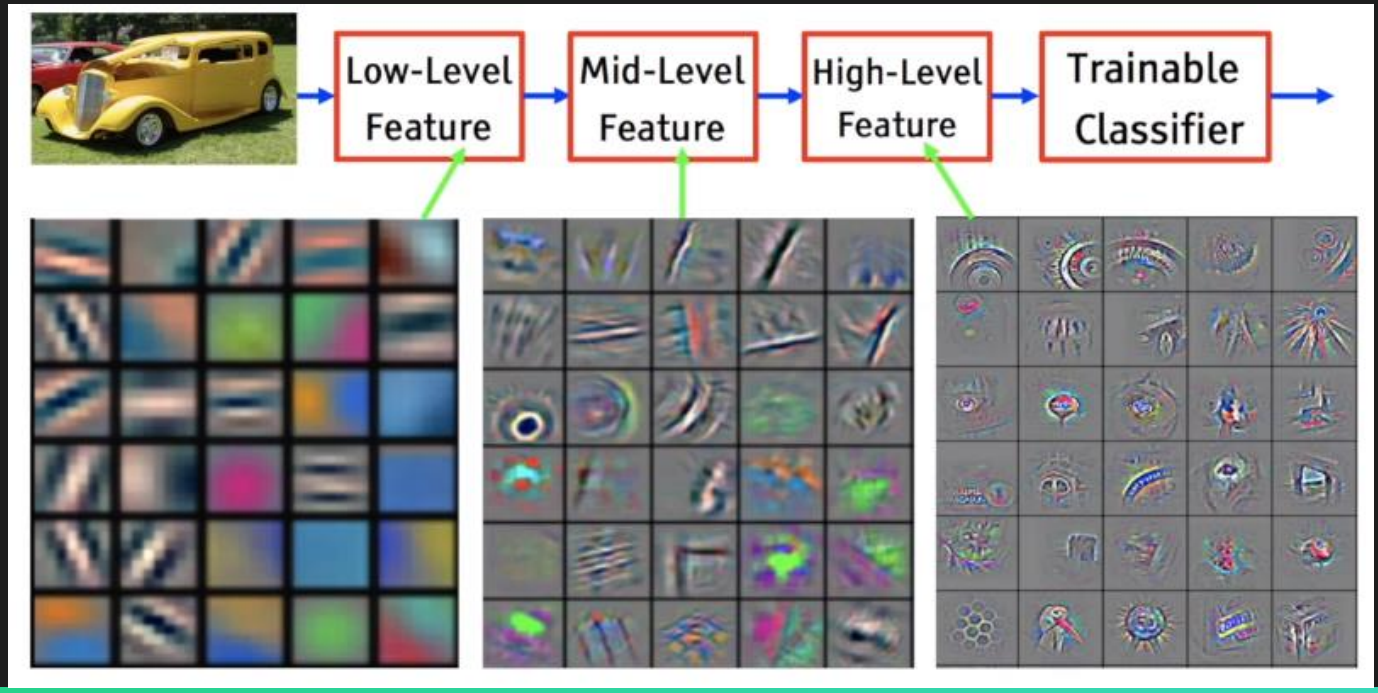
Why Data is Important? : Preprocessing

가지고 있는 데이터의 대부분의 정보(feature)는, 그렇게 중요하지 않을 수도 있다.
우리가 가진 데이터 중에서, 정말 중요한 특징들만 추출할 수 있다면?
데이터를 훨씬 더 효율적으로 저장하고 모델링할 수 있다.



Why Data is Important? : Preprocessing

가지고 있는 데이터의 대부분의 정보(feature)는, 그렇게 중요하지 않을 수도 있다.
우리가 가진 데이터 중에서, 정말 중요한 특징들만 추출할 수 있다면?
데이터를 훨씬 더 효율적으로 저장하고 모델링할 수 있다.
현대의 딥러닝에서는, DNN 그 자체가 특징 추출기 (Feature Extractor)의 역할을 한다.



PyTorch Session #2

Multi-layer Perceptron
And Training Pipeline

감사합니다.