Deep Learning Basic

22.08.18 / DSL 7기 전혜령

0. 목차

1. Basics

- AI vs ML
- Machine learning
- Classification
- Regression

2. What is DL?

- Linear
- Input, Output, Hidden Layer
- Nonlinear
- Classification vs Regression

3. How to train DL?

- Loss function
- Gradient Descent
- Backpropagation
- Gradient Vanishing

0. 목차

4. Optimizer

- Problems
- Optimizer

5. Appendix

- Overfitting vs Underfitting
- Solutions

6. Summary

- Summary

AI vs Machine Learning

- **Artificial intelligence (AI)**

사람이 해야 할 일을 기계가 대신할 수 있는 모든 자동화에 해당

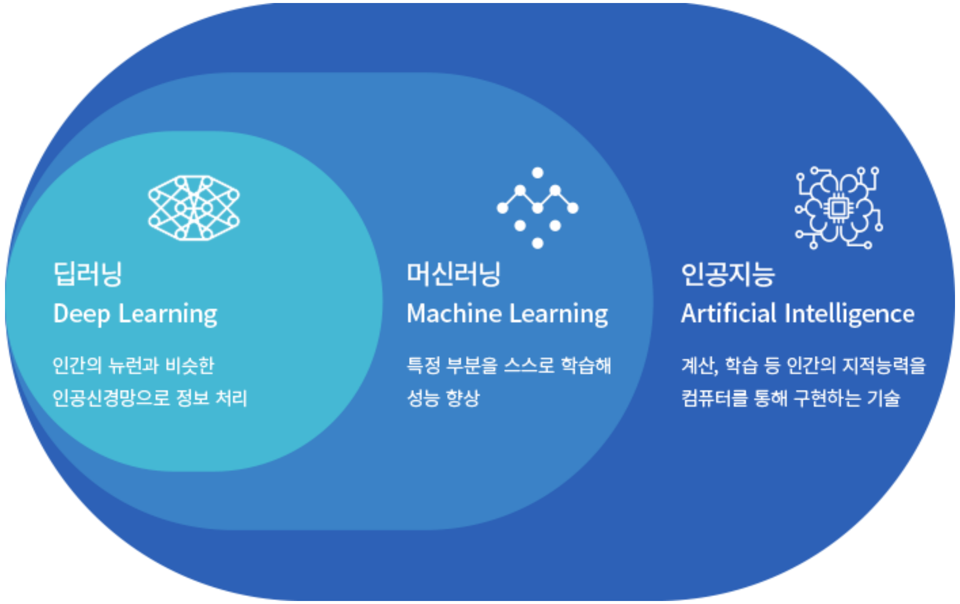
- **Machine learning (ML)**

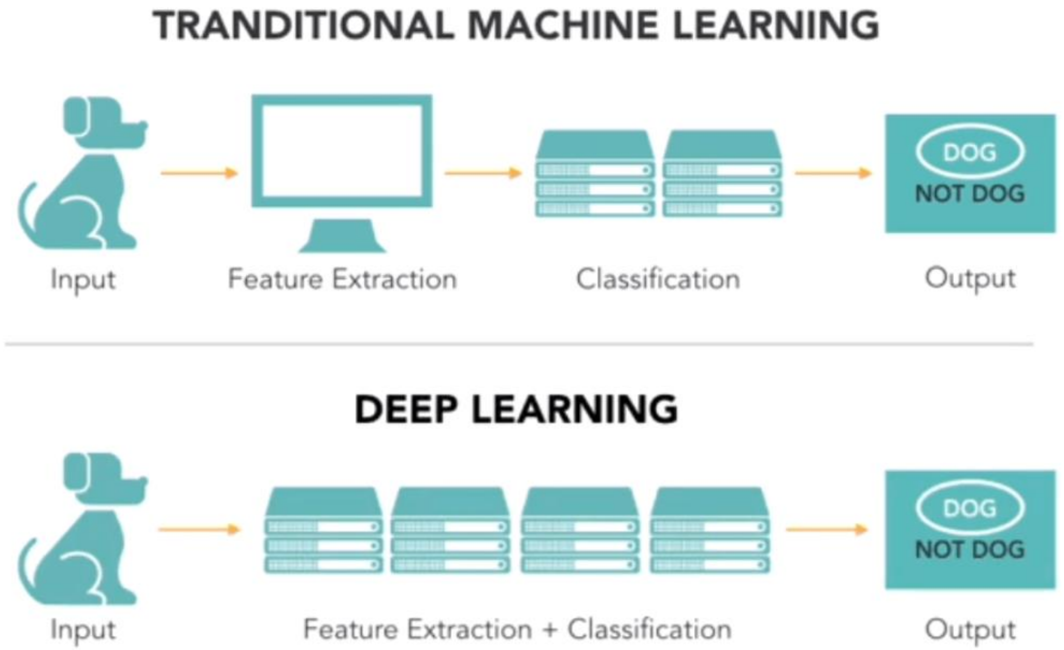
명시적으로 규칙을 프로그래밍하지 않고 데이터로부터 의사결정을 위한 패턴을 기계가 스스로 학습

AI vs Machine Learning

- **(+) Deep learning (DL)**

인공신경망 기반의 모델로, 비정형 데이터로부터 특징 추출 및 판단까지 기계가 한 번에 수행





Machine Learning

- **Supervised Learning (지도 학습)**

➢ 정답이 있는 데이터를 활용해 데이터를 학습

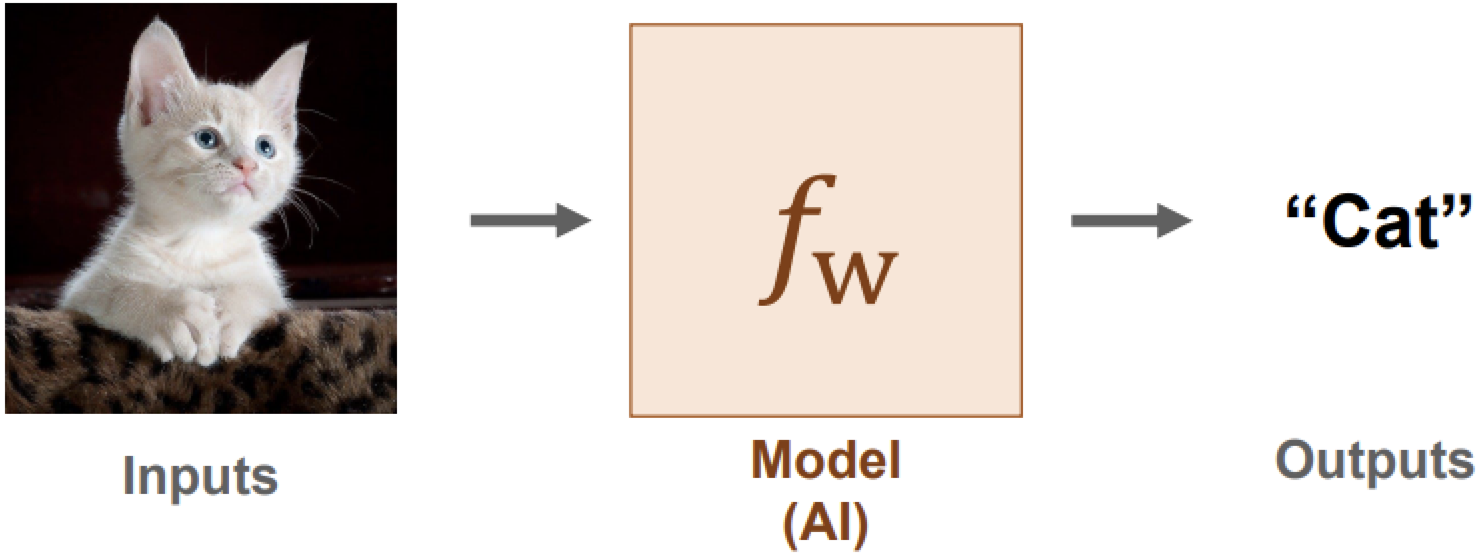
➢ Examples : Classification, regression, structured prediction

- **Unsupervised Learning (비지도 학습)**

➢ 정답 라벨이 없는 데이터를 비슷한 특징끼리 군집화 하여 새로운 데이터에 대한 결과를 예측하는 방법

➢ Examples : Clustering, generative models, self-supervised learning

Machine Learning - Classification



- **Mapping :** $f_W : \mathbb{R}^{W \times H} \rightarrow \{\text{“Cat”}, \text{“Dog”}\}$

1. Basics

Machine Learning - Regression

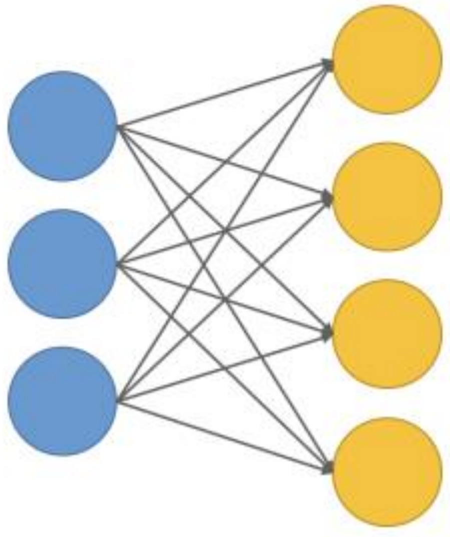
Inputs → f_w Model (AI) → ₩80,000 2022/09/13 Outputs

- **Mapping :** $f_w : \mathbb{R}^N \rightarrow \mathbb{R}$

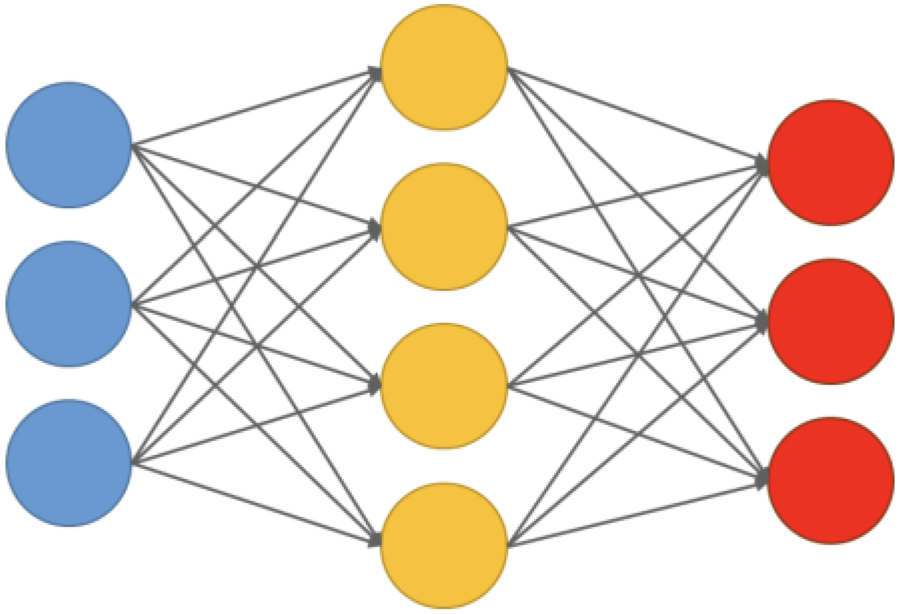
2. What is DL?

DL : 여러 층의 비선형 모델

$$f = \mathbf{W}\mathbf{x}$$



$$f = \mathbf{W}_2 max(0, \mathbf{W}_1\mathbf{x})$$



2. What is DL?

1. DL : 여러 층의 비선형 모델

▪ 선형 결합 (linear combination)

벡터공간 V에 속한 부분집합 $S = \{v_1, v_2, \dots, v_n\}$ 의 원소인 벡터 $v_1, v_2, \dots, v_n$와 어떤 스칼라 $a_1, a_2, \dots, a_n$에 대하여 다음을 만족시키는 벡터 $v \in V$를 S의 선형 결합(linear combination) 이라 한다.

$$v = a_1 v_1 + a_2 v_2 + \cdots + a_n v_n$$

▪ 선형 모델

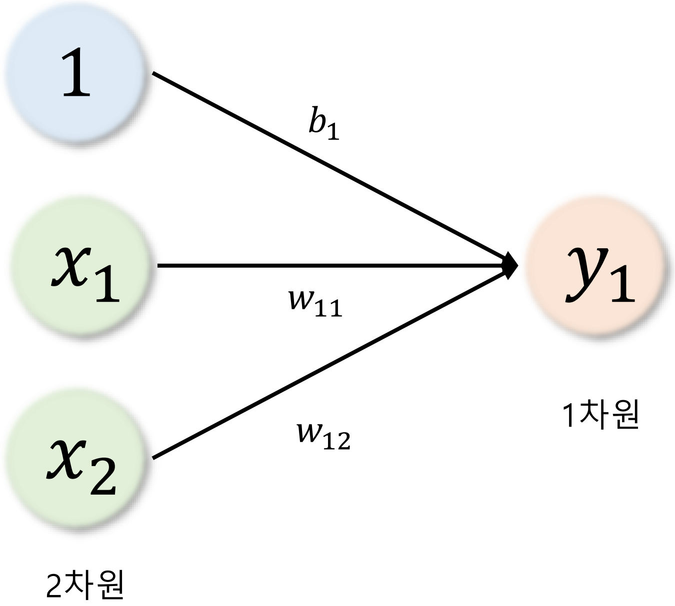
parameter를 계수로 하여 변수(samples)들을 선형결합한 것

$$y = b + w_1 x_1 + \cdots + w_k x_k \qquad \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ \dots \\ w_k \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_k \end{bmatrix}$$

2. What is DL?

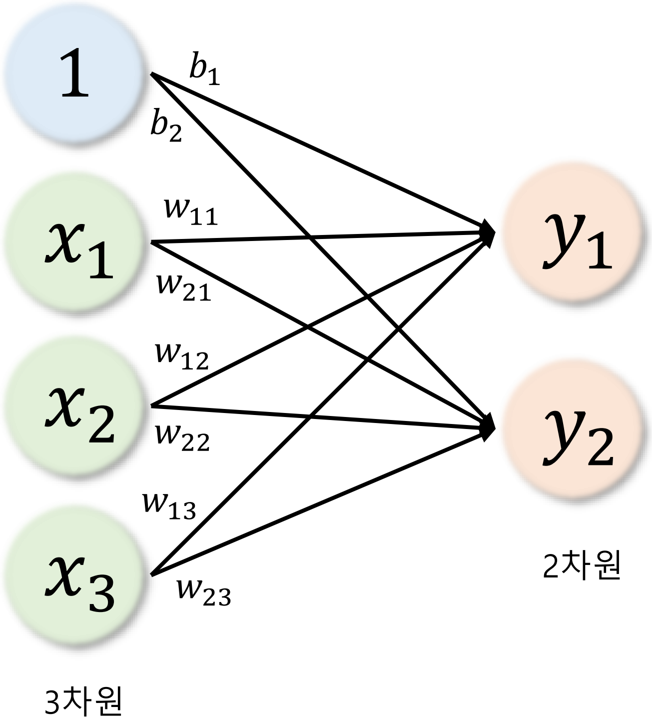
1. DL : 여러 층의 비선형 모델

▪ 선형 모델



1. DL : 여러 층의 비선형 모델

▪ 선형 모델



$$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1$$

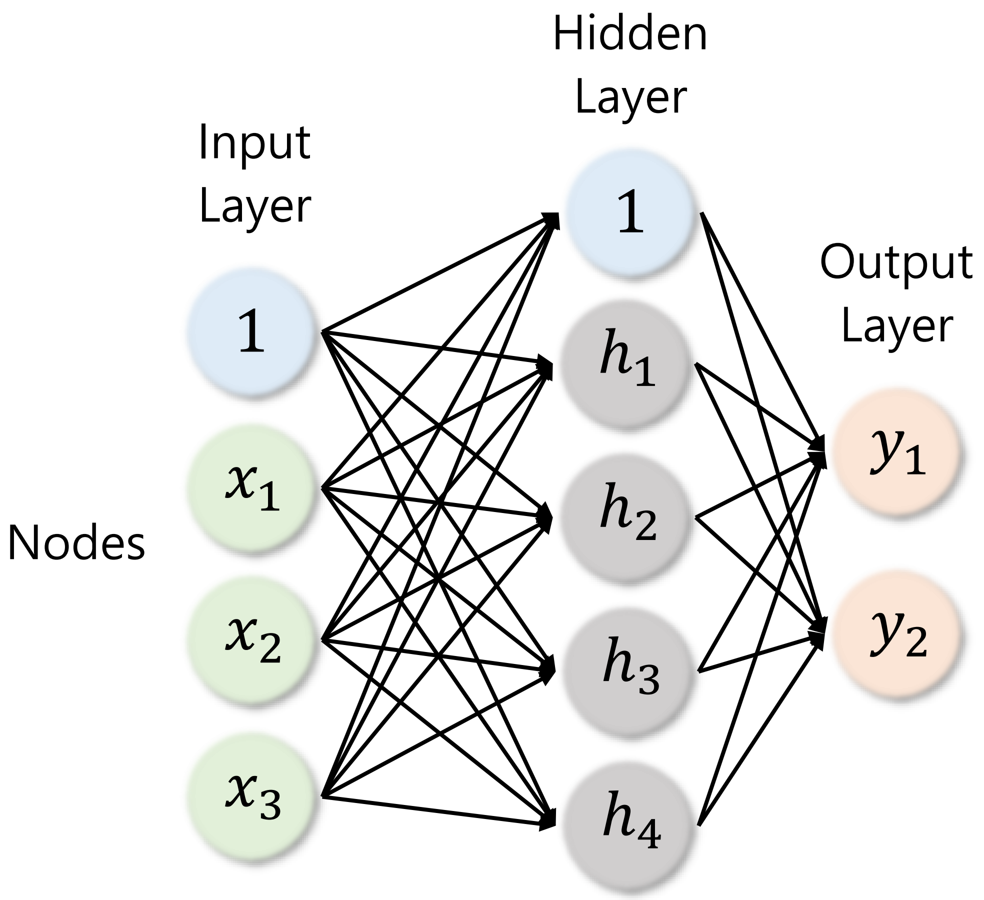
$$y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2$$

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

2. What is DL?

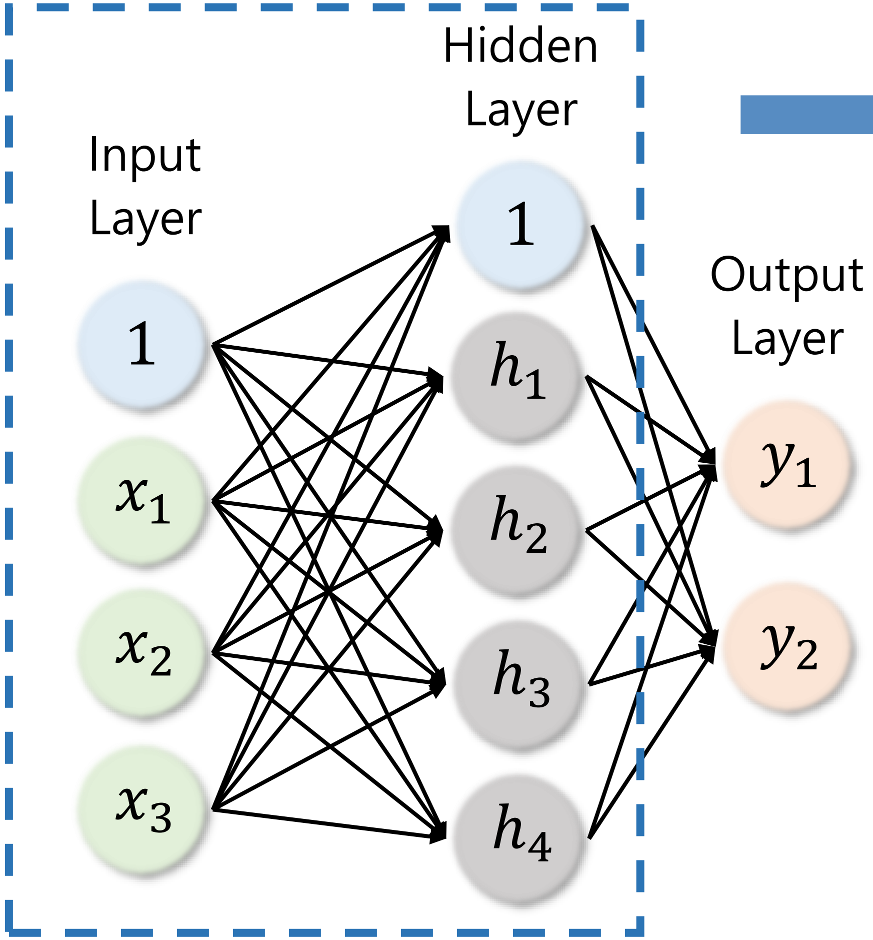
2. DL : 여러 층의 비선형 모델

- **입력층(Input Layer)**
- **출력층 (Output Layer)**
- **은닉층 (Hidden Layer)**
- **노드(Node)**

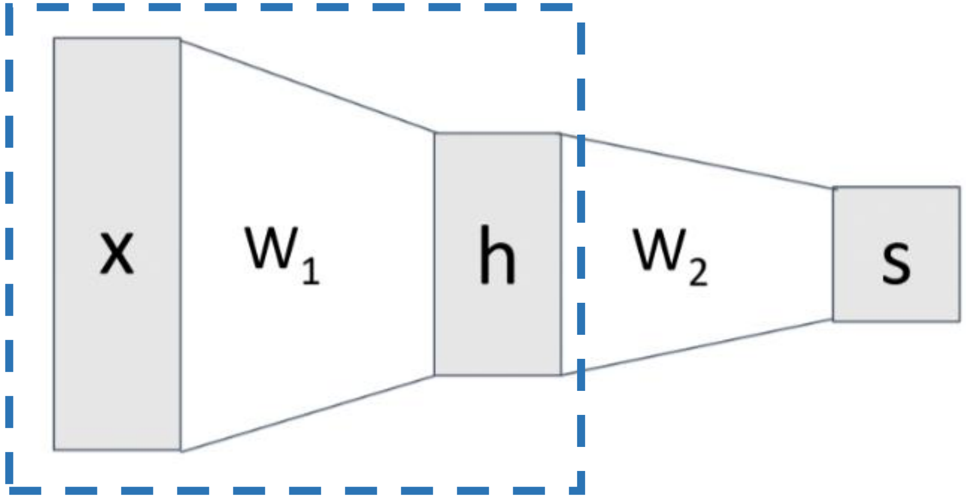


2. What is DL?

2. DL : 여러 층의 비선형 모델

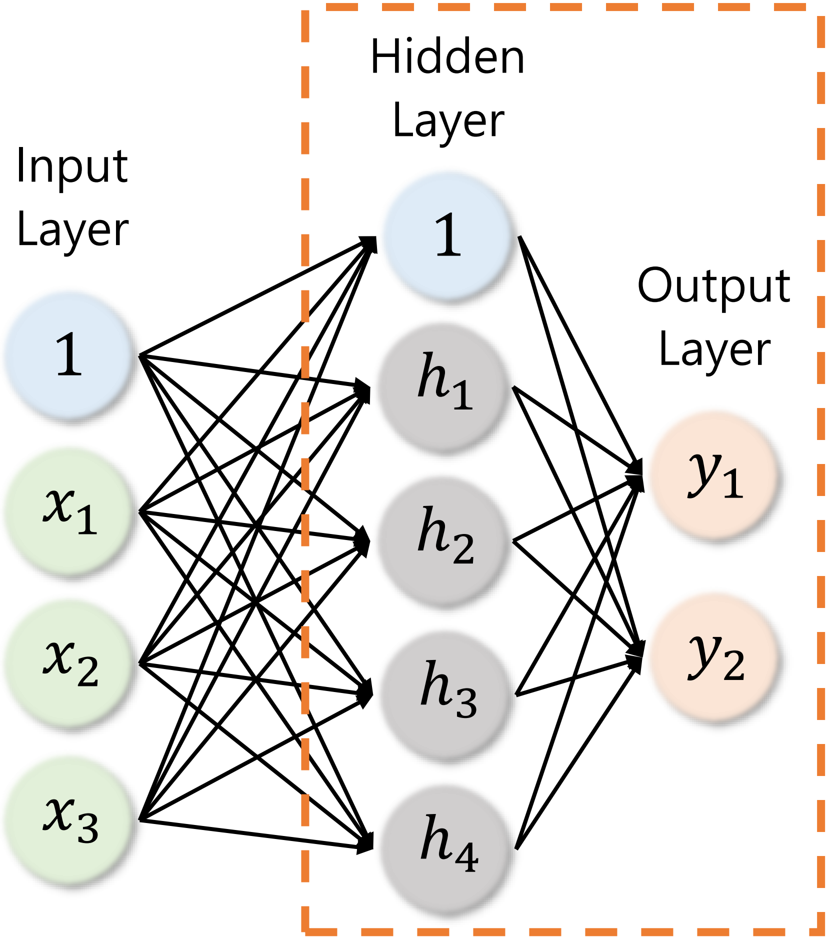


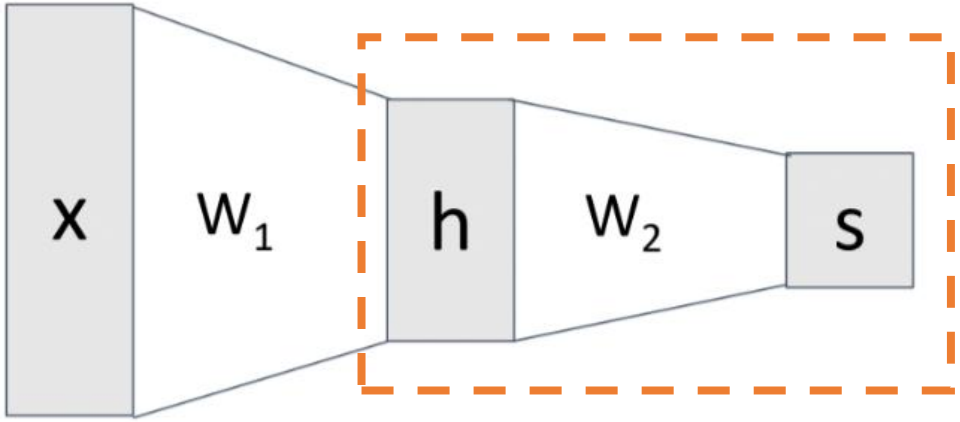
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}$$



2. What is DL?

2. DL : 여러 층의 비선형 모델

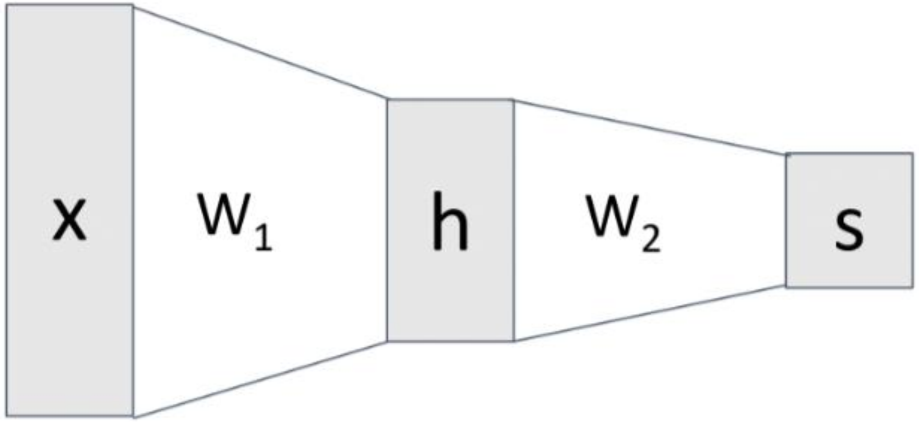




$$\begin{bmatrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} + \begin{bmatrix} b'_1 \\ b'_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

2. What is DL?

2. DL : 여러 층의 비선형 모델



Q: What happens if we build a neural network with no activation function?

$$s = W_2 W_1 x$$

$$\begin{aligned} y_1 = {} & w'_{11}(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1) \\ & +w'_{12}(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2) \\ & +w'_{13}(w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + b_3) \\ & +w'_{14}(w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + b_4) \\ & \quad +b'_1 \end{aligned}$$

$$\begin{aligned} y_2 = {} & w'_{21}(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1) \\ & +w'_{22}(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2) \\ & +w'_{23}(w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + b_3) \\ & +w'_{24}(w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + b_4) \\ & \quad +b'_2 \end{aligned}$$

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \alpha$$

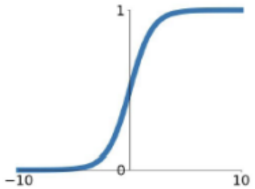
: 결국 선형 모델 → 층 깊이 할 이유 X

3. DL : 여러 층의 비선형 모델

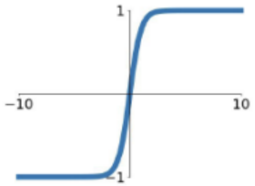
▪ 어떻게 비선형성을 부여할 수 있을까?

활성화 함수(Activation function)을 중간중간에 껴주자!

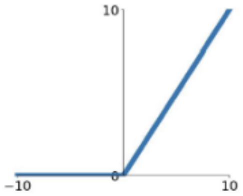
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



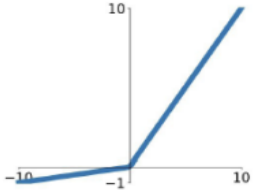
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

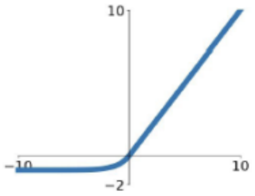


Leaky ReLU
$\max(0.1x, x)$



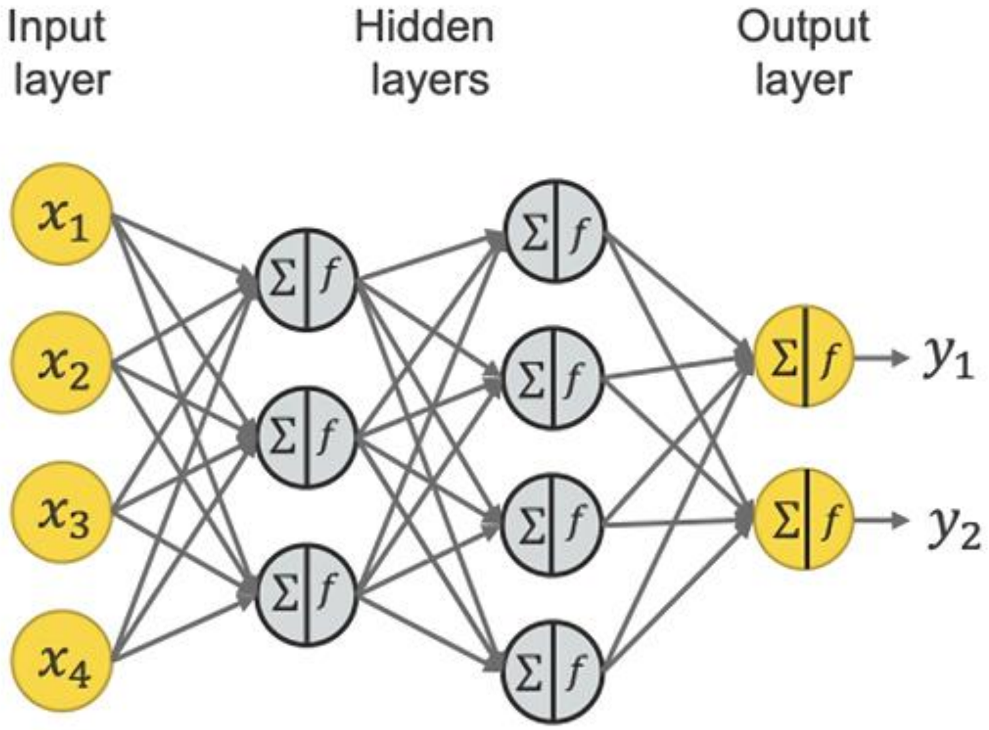
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

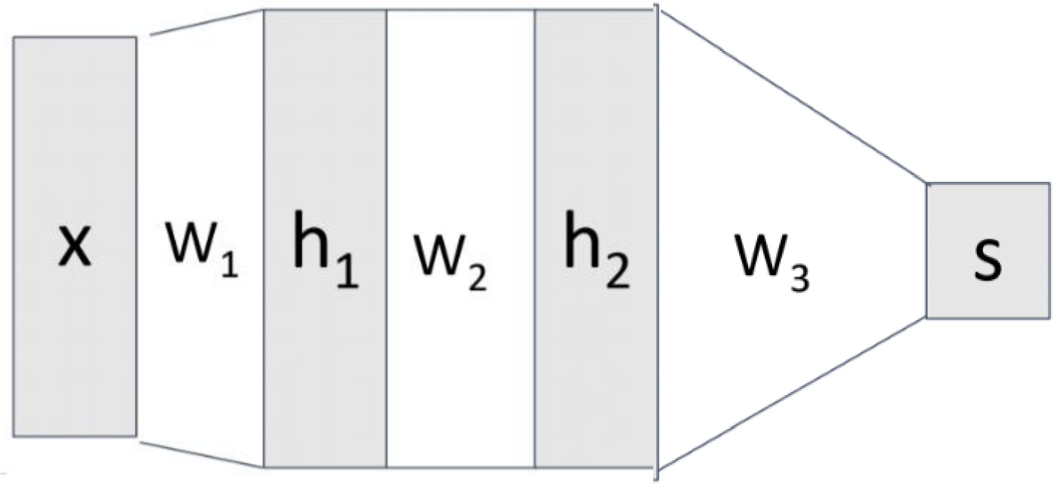
ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



2. What is DL?

3. DL : 여러 층의 비선형 모델





$$s = W_3 \max(0, W_2 \max(0, W_1 x))$$

3. DL : 여러 층의 **비선형** 모델

- **Why is max operator important?**

$$f = W_2 \boldsymbol{max}(0, W_1 x) \longleftrightarrow f = W_2 W_1 x$$ (without max operator)

$$f = W_3 x$$

Linear Score function

Activation Functions

$$\boldsymbol{max}(0, z)$$

2. What is DL?

4. Classification & Regression

- **Regression (회귀 모델)**

: 특정 숫자를 예측하려는 모델이므로 마지막에 활성화 함수 적용 X

- **Classification (분류 모델)**

: 마지막에 적용 O

➢ 이진 분류, 다중 레이블 분류: Sigmoid

➢ 다중 분류: Softmax → 총 합을 1로 만들어서 각각 확률로 생각

4. Classification & Regression - Softmax

▪ Softmax 함수

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}} \qquad j = 1,2,\dots,k$$

I. 값을 지수화시킨다.

II. 합이 1이 되도록 정규화

Cat	3.2	24.5	0.13
Car	5.1	164.0	0.87
Frog	-1.7	-0.18	0.00

1. 손실 함수 (Loss function)

▪ 손실 함수 (Loss function)

: 모델이 예측한 것과 실제 정답과의 차이를 이야기하는 함수

➢ 회귀: MSE, MAE

➢ 분류: Cross Entropy, Binary Cross Entropy

- Mean Squared Error (MSE)

$$E = \frac{1}{n}\frac{1}{k}\sum_{i=1}^{n}\sum_{j=1}^{k}\left(y_{i,j}^{true} - y_{i,j}^{pred}\right)^2$$

- Mean Absolute Error (MAE)

$$E = \frac{1}{n}\frac{1}{k}\sum_{i=1}^{n}\sum_{j=1}^{k}\left|y_{i,j}^{true} - y_{i,j}^{pred}\right|$$

(n은 출력 벡터의 개수, k는 예측 벡터의 차원)

1. 손실 함수 (Loss function)

▪ 손실 함수 (Loss function)

: 모델이 예측한 것과 실제 정답과의 차이를 이야기하는 함수

➢ 회귀: MSE, MAE

➢ 분류: Cross Entropy, Binary Cross Entropy

- Categorical Cross Entropy Error

$$E = -\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{k} t_i \log\left(y_i^{pred}\right)$$

- Binary Cross Entropy Error

$$E = -\frac{1}{n}\sum_{j=1}^{n}\left[y_j^{true}\log\left(y_j^{pred}\right) + \left(1 - y_j^{true}\right)\log\left(1 - y_j^{pred}\right)\right]$$

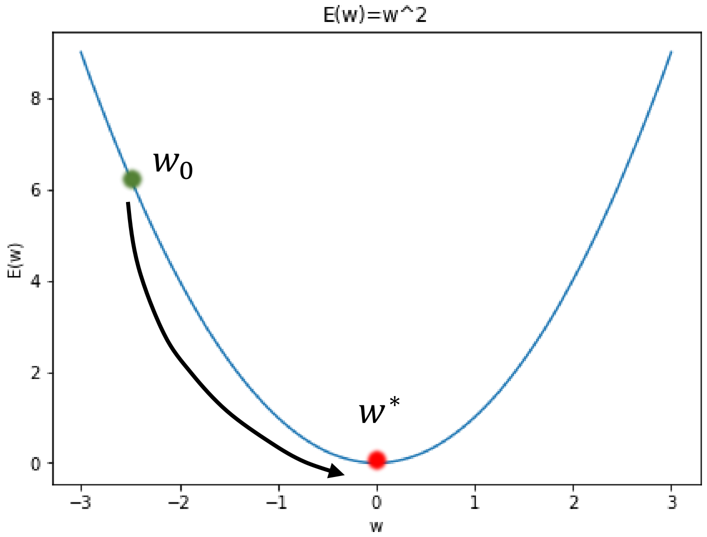
(n은 출력 벡터의 개수, k는 예측 벡터의 차원)

2. 경사 하강법 (Gradient Descent)

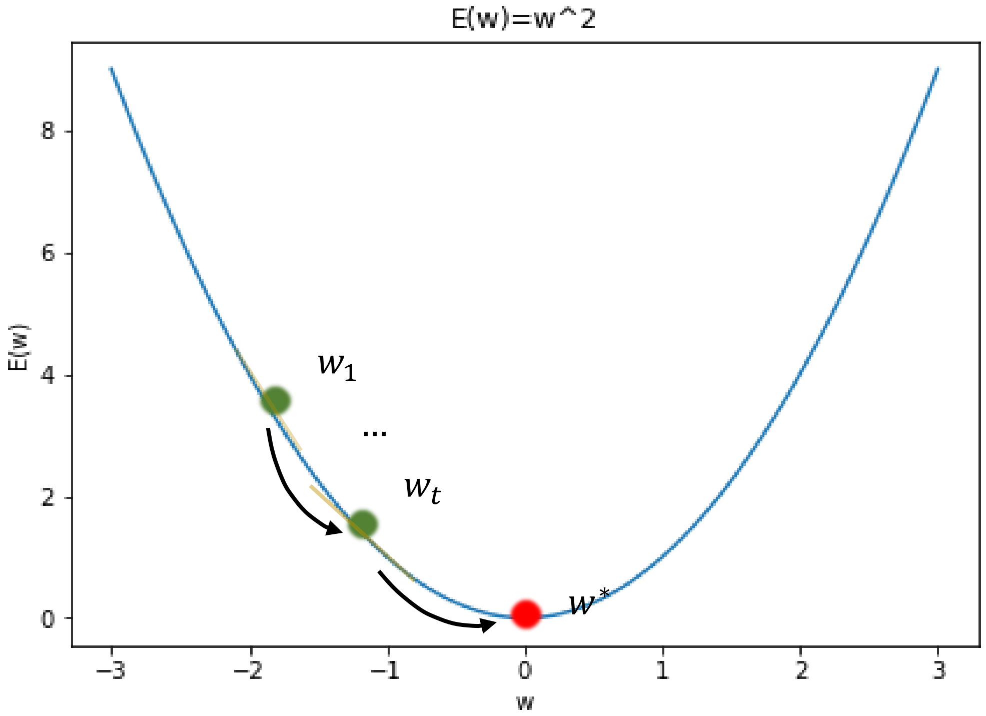
▪ 경사 하강법 (Gradient Descent)

: 1차 미분계수를 이용해 함수의 최소값을 찾아가는 iterative한 방법

$$E(w) = (0 - w)^2 = w^2$$



2. 경사 하강법 (Gradient Descent)



I. Loss를 최소화 시키는 파라미터를 찾는 것이 우리의 목적

II. 시작점은 랜덤 배정

III. 이후 접선의 기울기를 구하여
기울기 양수면 음의 방향 / 기울기 음수면 양의 방향

$$w_{t+1} = w_t - \gamma \frac{\partial E}{\partial w}$$

- γ는 학습률 (Learning rate) 로 hyperparameter
- 정해진 수만큼 이동 반복, 반복할 횟수가 epoch

3. How to train DL

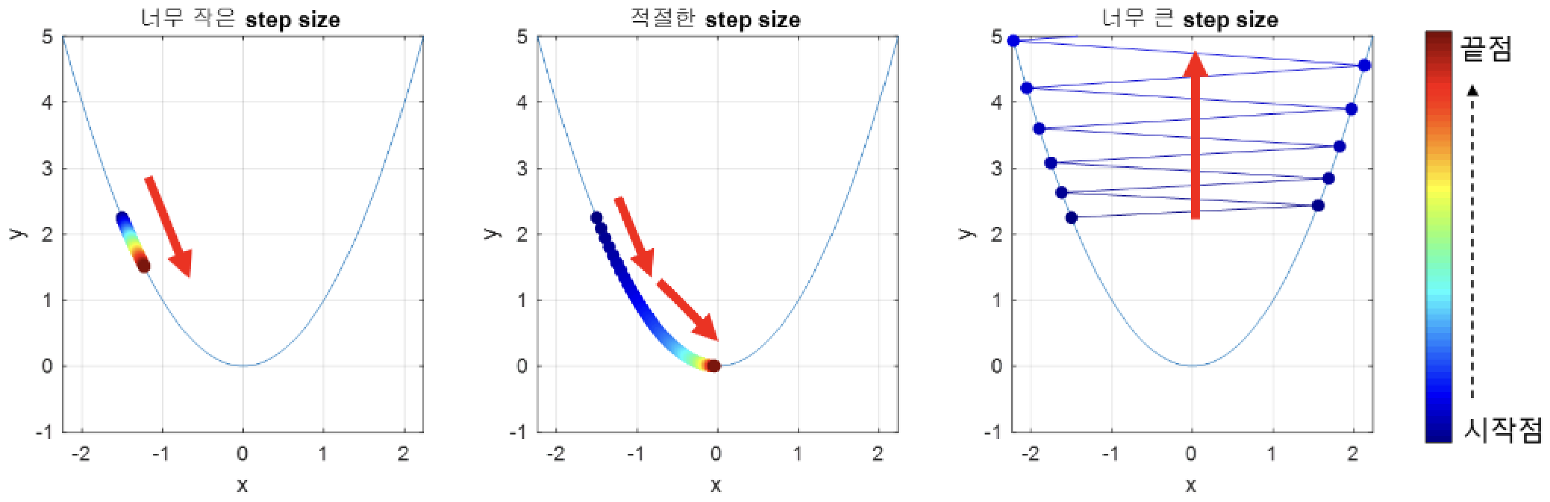
2. 경사 하강법 (Gradient Descent)

▪ 학습률 (Learning rate)

: 경사하강법에서 파라미터를 업데이트하는 정도를 조절하기 위한 변수

▪ Parameter vs Hyperparameter

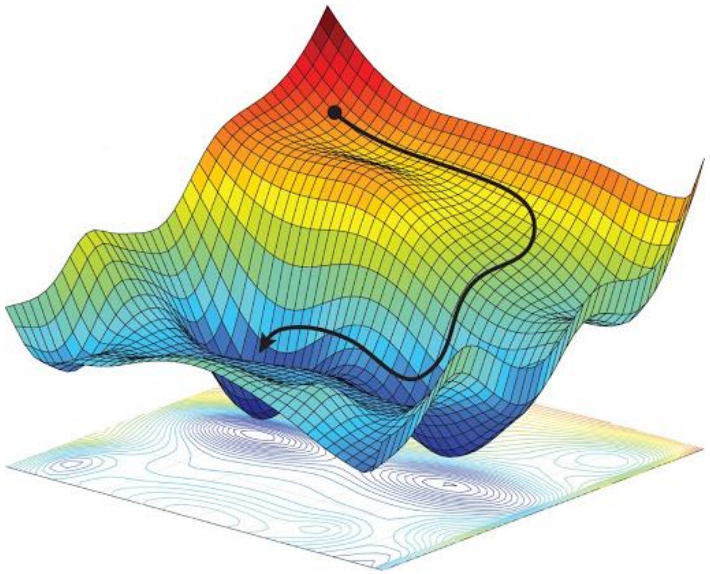
: 모델 혹은 데이터에 의해 결정되면 파라미터, 사용자가 직접 설정하면 하이퍼 파라미터

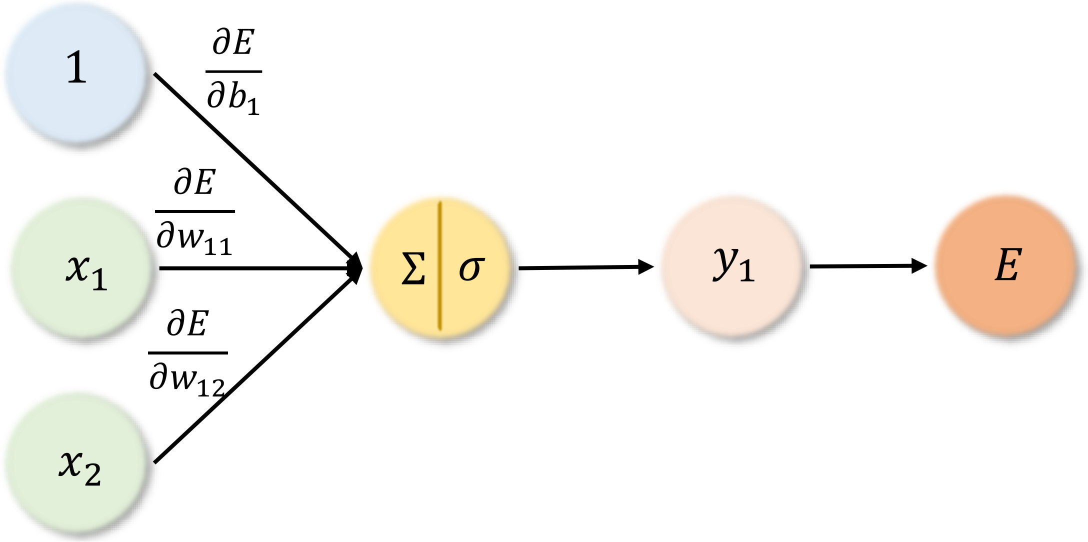


3. How to train DL

3. 오차 역전파 (Backpropagation)

- 결국 우리는 가중치에 대한 손실을 미분해야 함: $\frac{\partial E}{\partial W}$
- 실제 손실함수는 앞의 예시처럼 간단하지 않으므로, 기울기 계산 쉽지 않음 → 오차 역전파 방식 이용





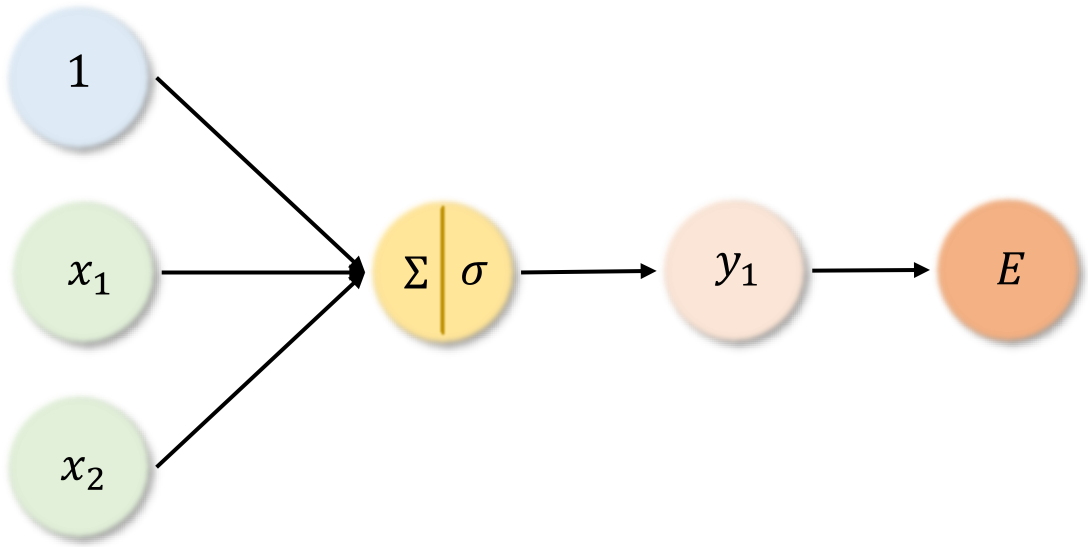
3. How to train DL

3. 오차 역전파 (Backpropagation)

▪ **Forward pass**

: Computational graph를 기준으로 왼쪽부터 오른쪽으로 계산해 나가는 것, 즉 정상적으로 식을 계산하는 것

: Forward pass에서는 각 노드의 계산 결과를 일시적으로 저장

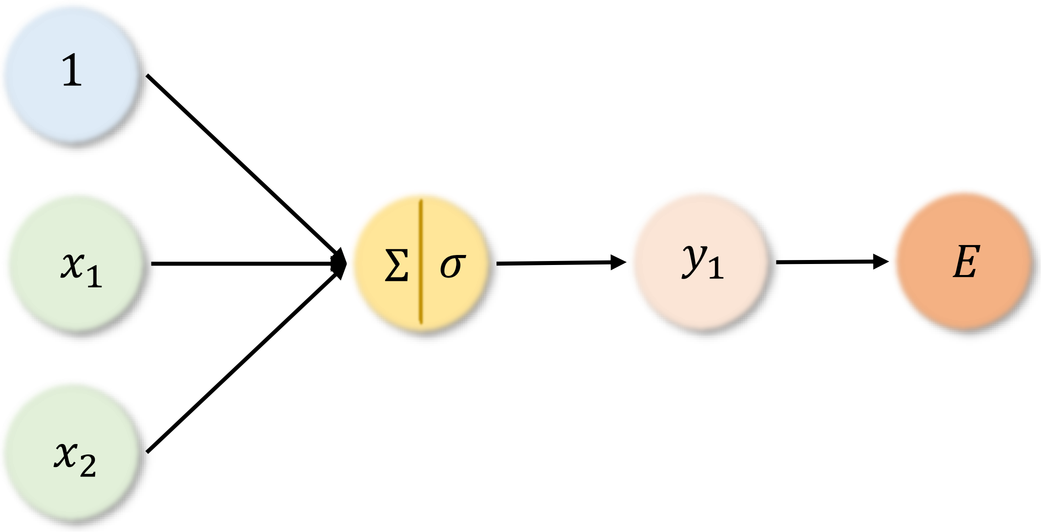


3. 오차 역전파 (Backpropagation)

▪ **Backward pass**

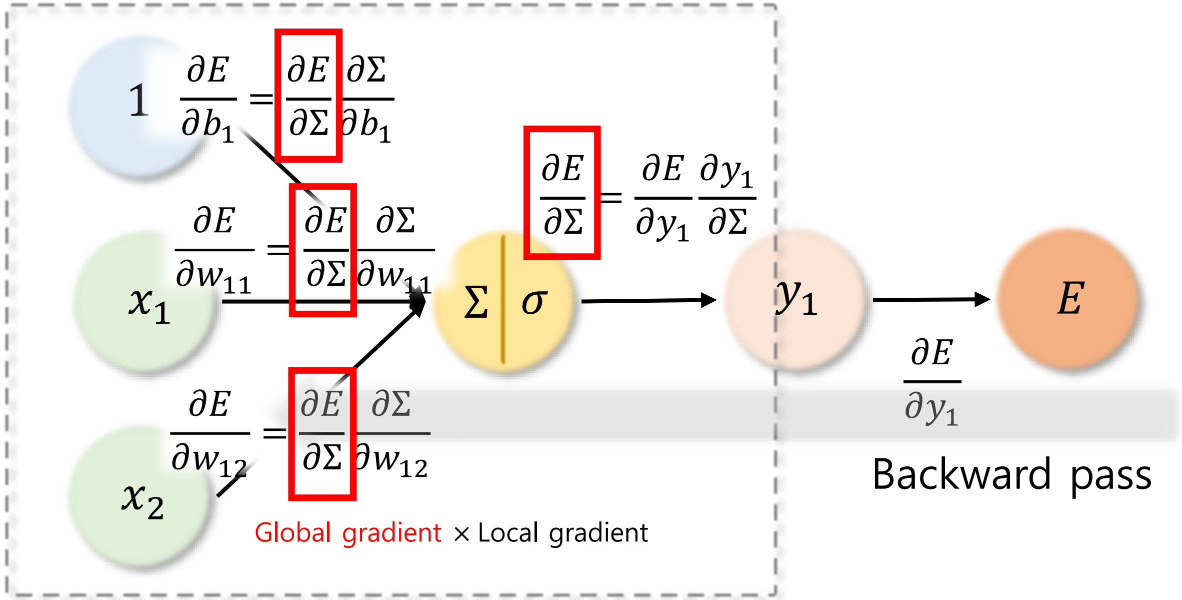
: Forward pass에서 정상적인 식의 계산으로 최종 출력값을 얻었다면, 이제 그 값이 나온 오른쪽 끝에서부터 다시 거슬러 오는 Backward pass를 수행

: global gradient (upstream gradient) vs local gradient



3. 오차 역전파 (Backpropagation)

- **Forwardpass – Backwardpass – Gradient descent – W를 update**



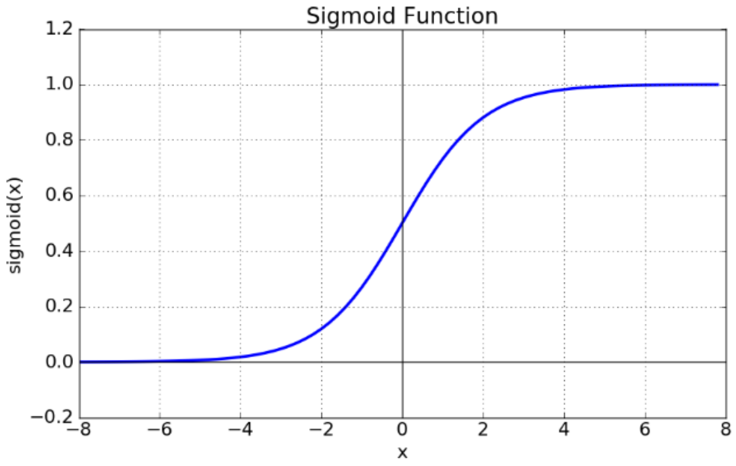
$$\frac{\partial E}{\partial b_1} = \frac{\partial E}{\partial \Sigma}\frac{\partial \Sigma}{\partial b_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \Sigma}\frac{\partial \Sigma}{\partial b_1} = -2(y - y_1)\sigma(1-\sigma) \times 1$$

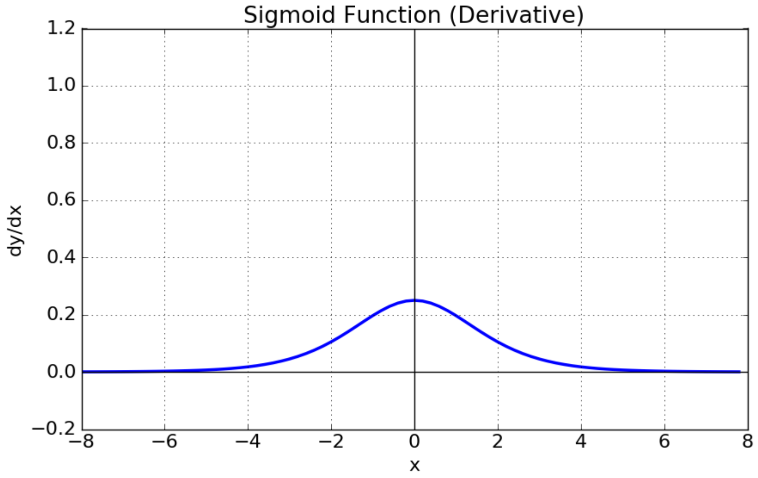
$$\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial \Sigma}\frac{\partial \Sigma}{\partial w_{11}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \Sigma}\frac{\partial \Sigma}{\partial w_{11}} = -2(y - y_1)\sigma(1-\sigma)x_1$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial \Sigma}\frac{\partial \Sigma}{\partial w_{12}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial \Sigma}\frac{\partial \Sigma}{\partial w_{12}} = -2(y - y_1)\sigma(1-\sigma)x_2$$

4. Gradient Vanishing

- local X global 과정에서 기울기 값들이 계속 0과 1 사이의 값을 갖게 된다면?





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

3. How to train DL

4. Gradient Vanishing

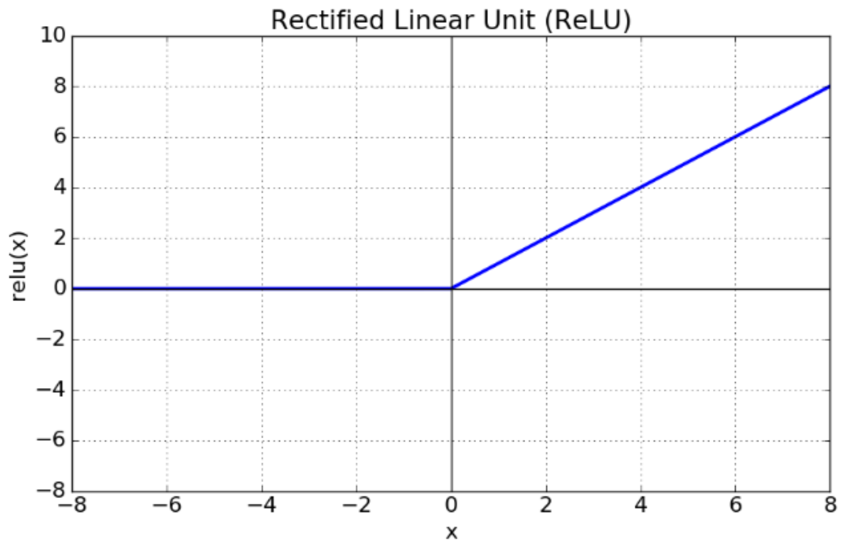
▪ **Problem Solving 1 : ReLU**

: 0보다 작은 값은 0으로 반환하고, 0보다 큰 값이 나올 경우 그대로 반환

: 도함수의 값 1 또는 0이므로 기울기 소실 발생 X

: but Dying ReLU 문제 발생 → ㄱㅊ

: **Leaky ReLU**



$\mathrm{ReLu}(x) = \max(0, x)$

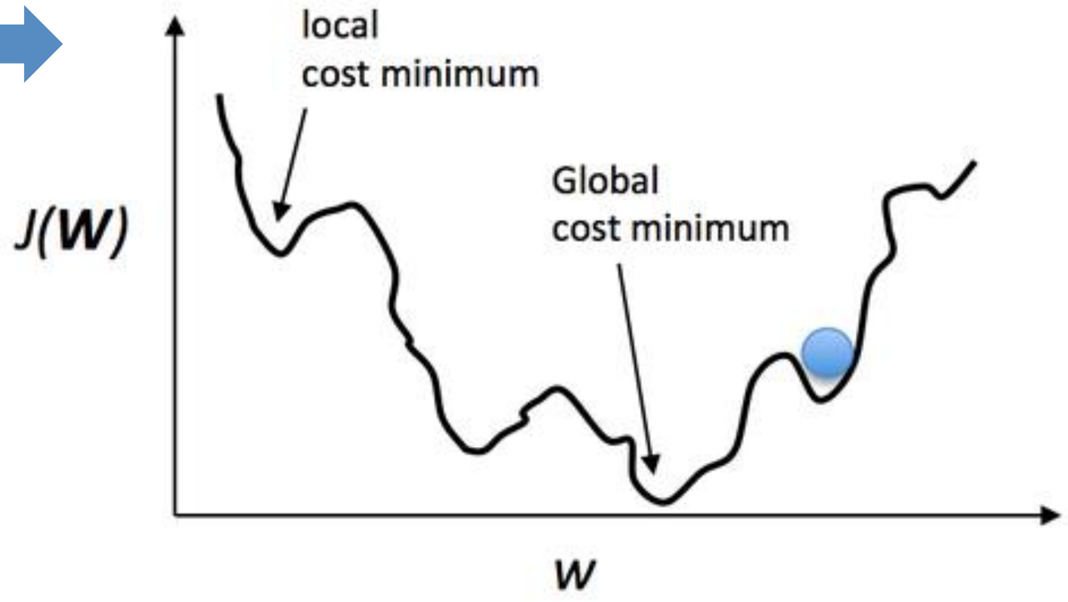
▪ **Problem Solving 2 : Weight initialization**

▪ **Problem Solving 3 : Batch Normalization**

4. Optimizer

1. Problems

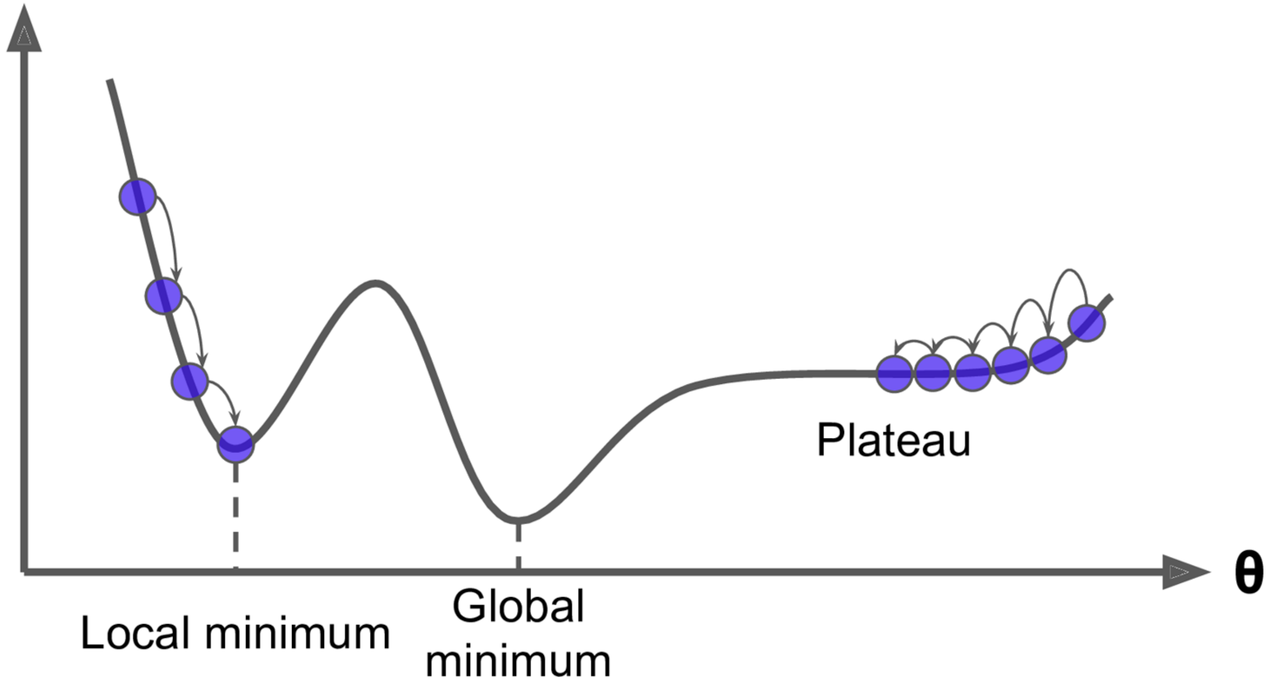
- Problem 1 : 데이터 수 증가 → 계산량 증가
- Problem 2 : Local minimum
- Problem 3 : Pleateau
- Problem 4 : Zigzag



4. Optimizer

1. Problems

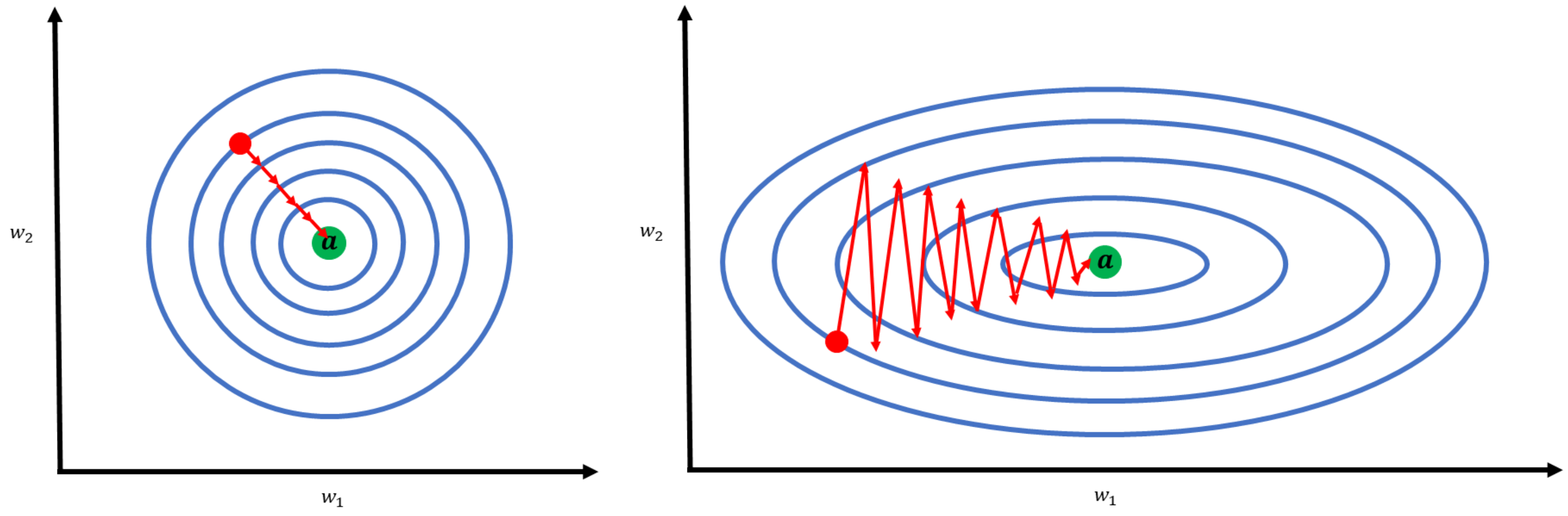
- Problem 1 : 데이터 수 증가 → 계산량 증가
- Problem 2 : Local minimum
- Problem 3 : Plateau
- Problem 4 : Zigzag



4. Optimizer

1. Problems

- Problem 1 : 데이터 수 증가 → 계산량 증가
- Problem 2 : Local minimum
- Problem 3 : Pleateau
- Problem 4 : Zigzag



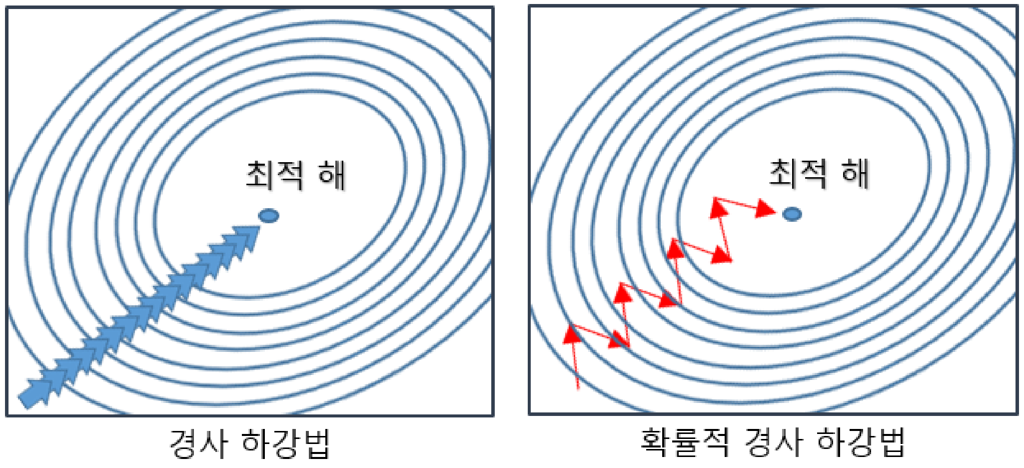
2. Optimizers

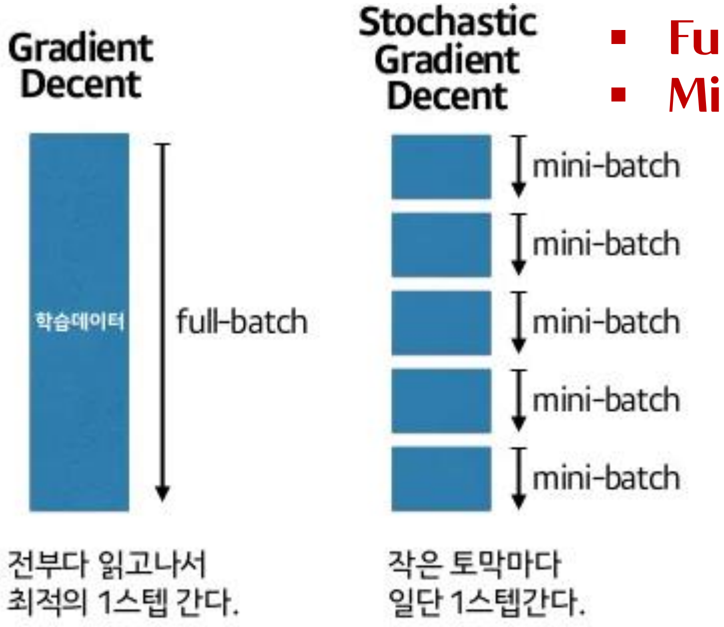
- **SGD (Stochastic Gradient Descent ; 확률적 경사하강법)**

: 기본적인 gradient descent 알고리즘은 batch gradient descent

: SGD 알고리즘은 한 번의 파라미터 업데이트를 위해 하나의 훈련 데이터를 사용

: **Mini-Batch Stochastic Gradient Descent (SGD, MSGD)**





- Full-Batch?
- Mini-Batch?

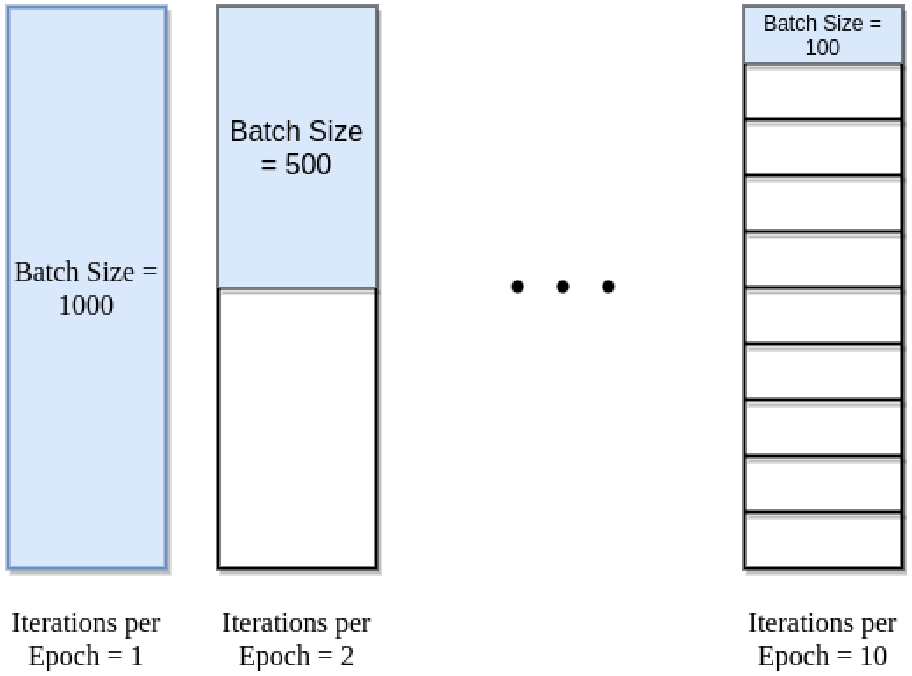
4. Optimizer

2. Optimizers

- **Batch** : 모델의 가중치를 한번 업데이트 시킬 때 사용되는 샘플들의 묶음 수
- **Full-batch** : 훈련데이터를 한번에 모두 훈련시킬 때
- **Mini-batch** : 훈련데이터를 나눠서 훈련시킬 때, 훈련하기 위해 나눈 그룹 1개
- **Batch size** : 전체 트레이닝 데이터 셋을 여러 작은 그룹을 나누었을 때 하나의 소그룹에 속하는 데이터 수

  = 미니배치 1개 안에 들어있는 데이터 개수
- **epoch** : 학습의 횟수 – 한 번의 epoch는 전체 데이터 셋에 대해 한 번 학습을 완료한 상태
- **iteration** : 1-epoch를 마치는데 필요한 미니배치 개수

2. Optimizers



1 Epoch : 모든 데이터 셋을 한 번 학습
1 iteration : 1회 학습

minibatch : 데이터 셋을 batch size 크기로 쪼개서 학습

ex) 총 데이터가 100개, batch size가 10이면,
1 iteration = 10개 데이터에 대해서 학습
1 Epoch = 100/batch size = 10 iteration

4. Optimizer

2. Optimizers

▪ Momentum

: 학습 방향을 유지하려는 성질

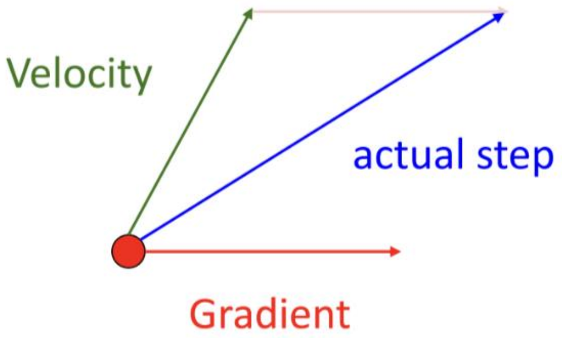
: 같은 방향의 학습이 진행된다면 가속을 가지며 더 빠른 학습을 기대할 수 있음

Velocity term

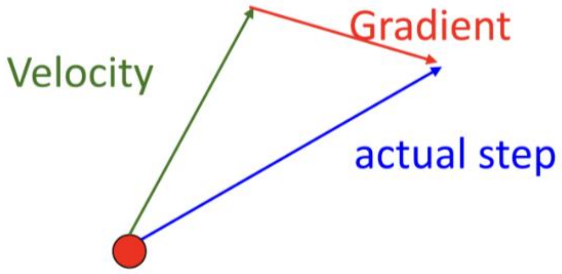
$$v_{t+1} = \rho v_t + \boxed{\nabla f(x_t)}$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Momentum update:



Nesterov Momentum



2. Optimizers

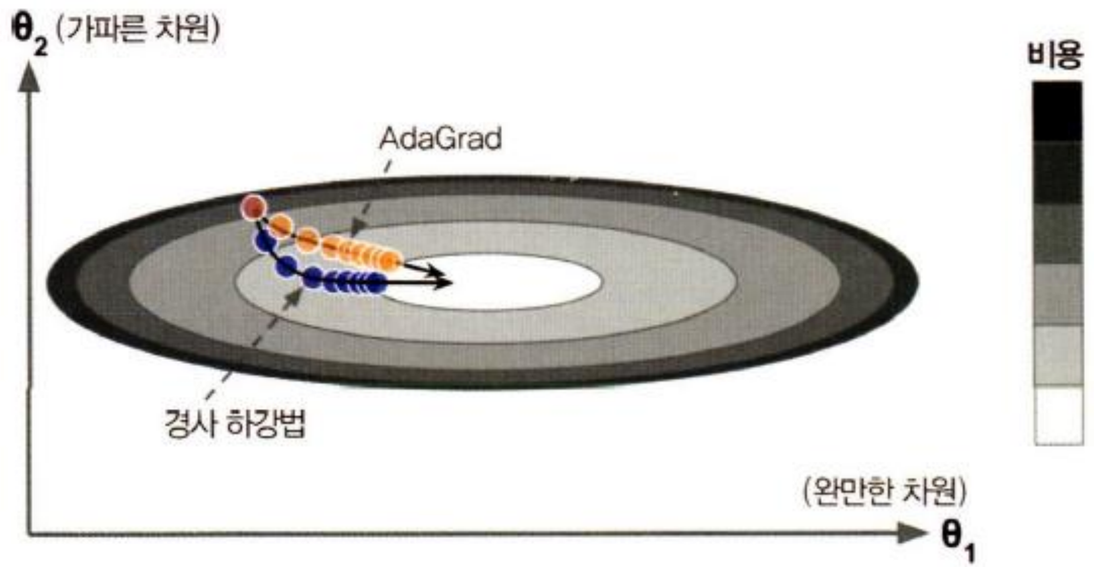
▪ Adagrad

: 각 파라미터와 각 단계마다 학습률을 변경

: 지금까지 많이 변화하지 않은 변수들은 step size를 크게 하고, 지금까지 많이 변화했던 변수들은 step size를 작게 하자

$$h \leftarrow h + \underbrace{\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}}_{\text{Squared gradient}}$$

$$W \leftarrow W - \eta \boxed{\frac{1}{\sqrt{h}}} \frac{\partial L}{\partial W}$$



2. Optimizers

▪ Adagrad

: 각 파라미터와 각 단계마다 학습률을 변경

: 지금까지 많이 변화하지 않은 변수들은 step size를 크게 하고, 지금까지 많이 변화했던 변수들은 step size를 작게 하자

$$h \leftarrow h + \underbrace{\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}}_{\text{Squared gradient}}$$

$$W \leftarrow W - \eta \boxed{\frac{1}{\sqrt{h}}} \frac{\partial L}{\partial W}$$

▪ RMSProp

: 기울기를 단순 누적하지 않고 지수 가중 이동 평균을 사용하여 최신 기울기들이 더 크게 반영되도록

: 먼 과거의 기울기는 조금 반영, 최신의 기울기 많이 반영

$$h \leftarrow \overbrace{\boxed{\rho}}^{\text{Decay rate}} h + \boxed{(1-\rho)} \underbrace{\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}}_{\text{Squared gradient}}$$

$$W \leftarrow W - \eta \boxed{\frac{1}{\sqrt{h}}} \frac{\partial L}{\partial W}$$

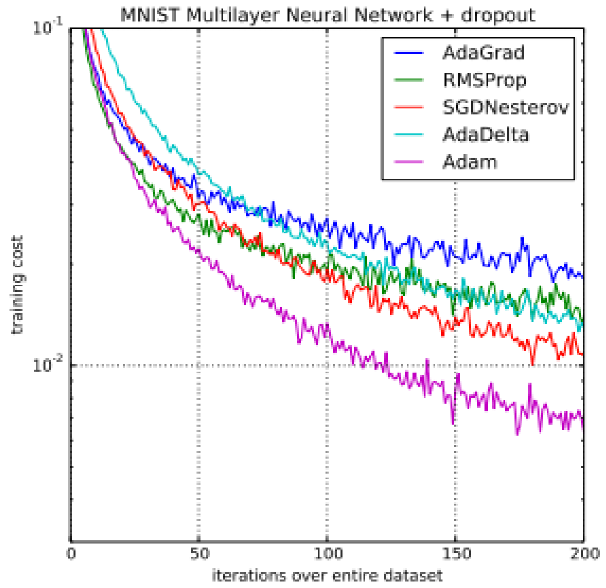
2. Optimizers

▪ Adam (Adaptive Moment Esimation)

: Momentum 와 RMSProp 두가지를 섞어 쓴 알고리즘

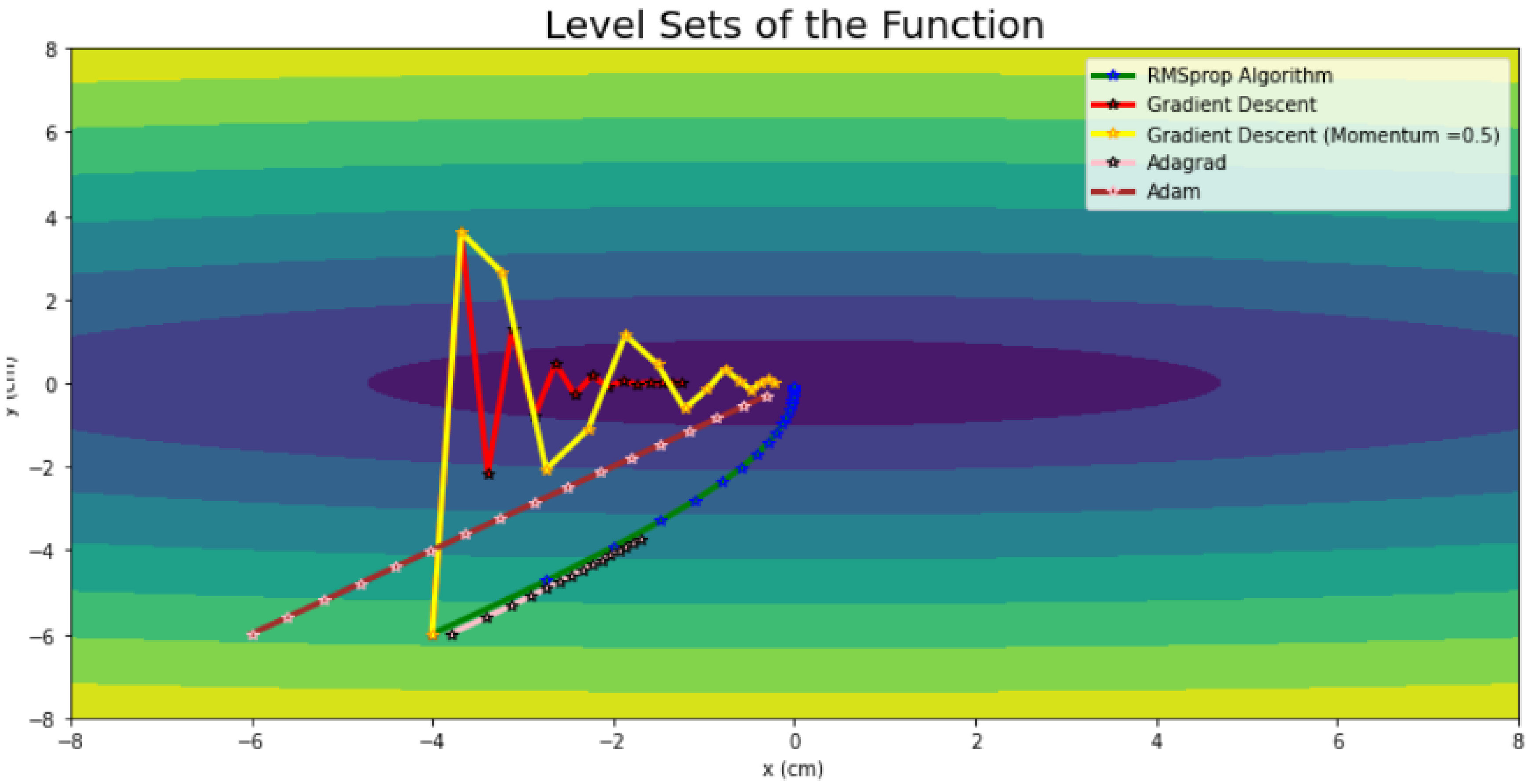
: 즉, 진행하던 속도에 관성을 주고, 최근 경로의 곡면의 변화량에 따른 적응적 학습률을 갖는 알고리즘

: 일반적 알고리즘에 현재 가장 많이 사용



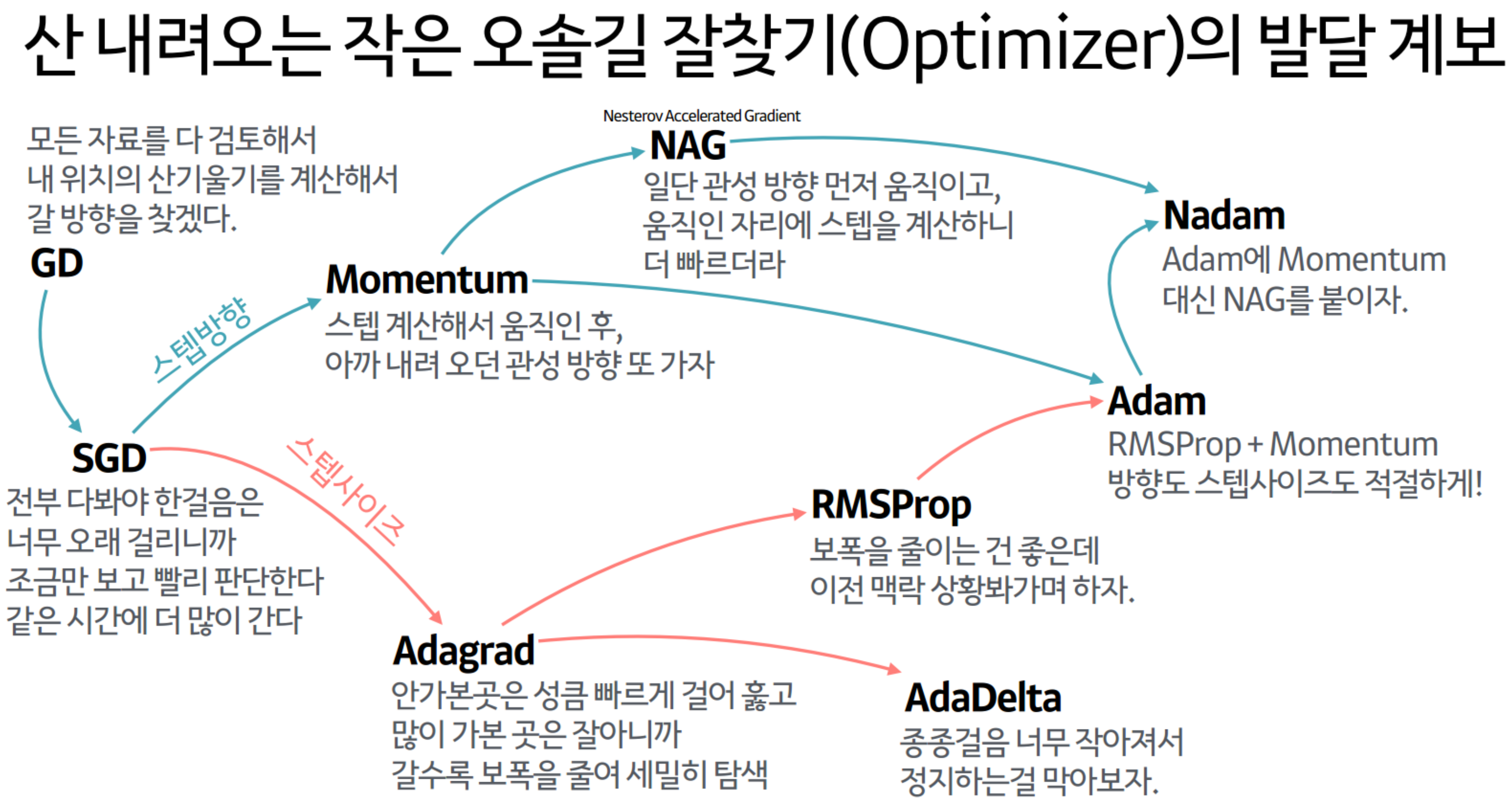
4. Optimizer

2. Optimizers



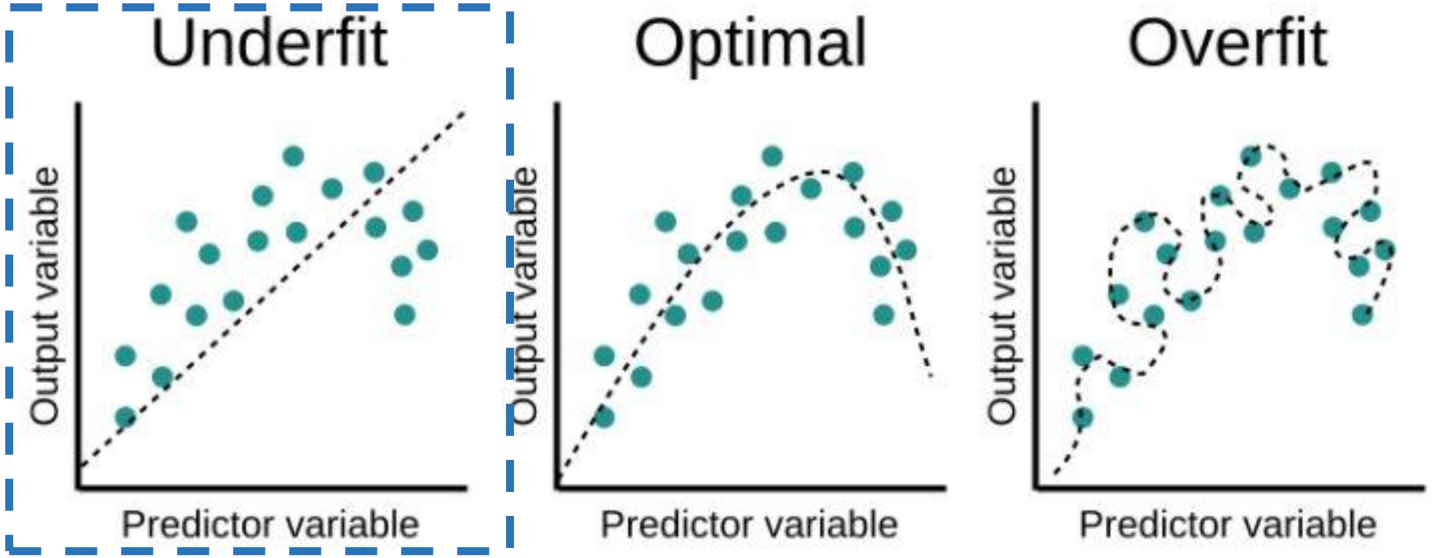
4. Optimizer

2. Optimizers



5. Appendix

1. Overfitting vs Underfitting



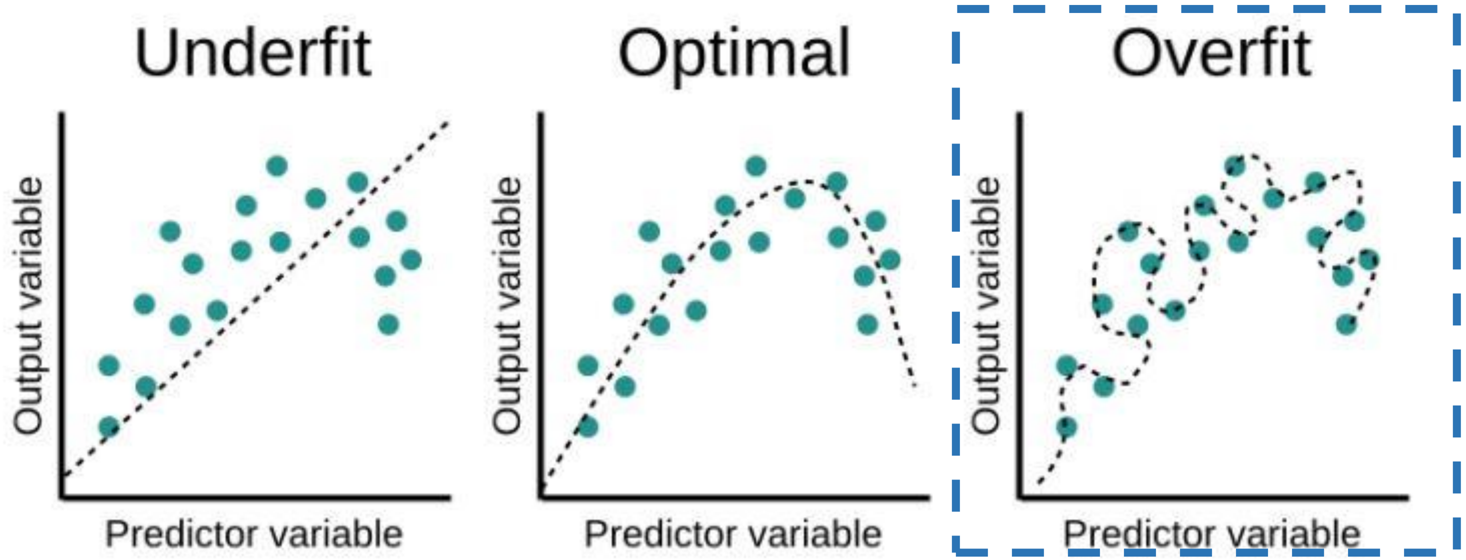
- **과소적합 (Underfitting)**

이미 있는 Train set도 학습을 하지 못한 상태

모델이 너무 단순해서 데이터의 내재된 구조를 학습하지 못할 때 발생

5. Appendix

1. Overfitting vs Underfitting



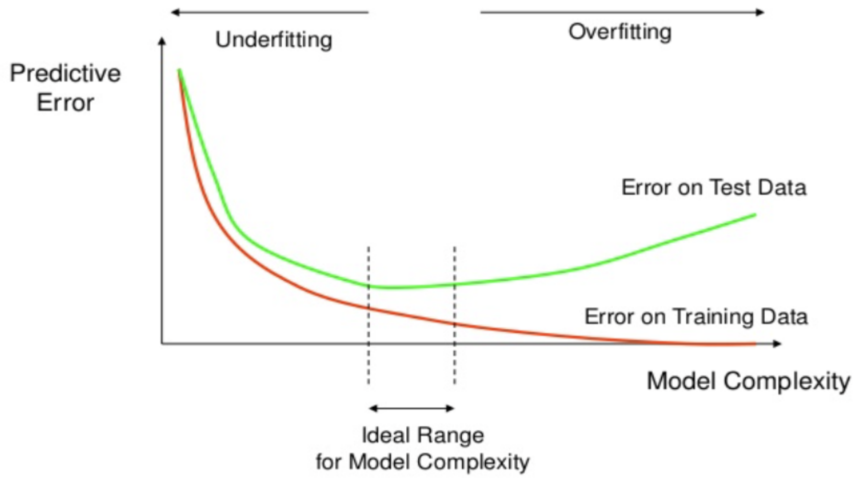
- **과대적합 (Overfitting)**

너무 과도하게 데이터 모델을 학습(learning)을 한 경우를 의미

학습 데이터에는 잘 맞지만 검증 데이터(테스트 데이터)에 잘 맞지 않는 것

2. Solutions

- **Model complexity 낮추기**



- **L1/L2 Regularization**

손실함수(Loss function)에 람다항을 추가하여 페널티 부여하기

$$cost(W,b) = \frac{1}{m}\sum_{i}^{m} L(\hat{y}_i, y_i) + \lambda\frac{1}{2}|w|$$

($L1$ 정규화)

$$cost(W,b) = \frac{1}{m}\sum_{i}^{m} L(\hat{y}_i, y_i) + \lambda\frac{1}{2}|w|^2$$

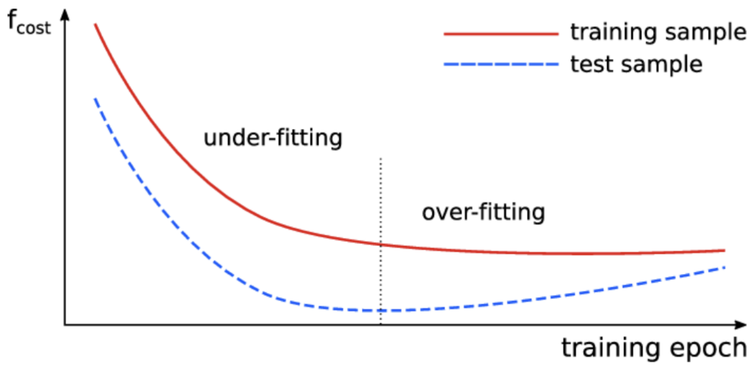
($L2$ 정규화)

5. Appendix

2. Solutions (그 외)

▪ **test loss가 가장 작은 곳에서 epoch stop (사실 val)**

▪ **학습 데이터 늘리기 (Data augmentation)**



▪ Dropout (드롭아웃)

▪ Batch normalization

3. Summary

Summary

- **What is DL?**

여러 층의 비선형모델 : 활성화 함수를 도입함으로써 비선형성을 부여

- **How to train DL model**

Backprop, GD 통해 Loss를 최소화 시켜보자.

Gradient vanishing 문제?! – optimizer의 발전

- **Optimizer**

GD ("full batch" vs "stochastic"), Momemtum, Adaptive learning rate(Ada-grad, RMS prop), Adam (Momemtum + RMS prop)

Reference

6기 박준우님 Deep Learning Basic 강의 자료

6기 손예진님 BatchNorm, Regularization, WeightInit, Dropout 강의 자료

https://light-tree.tistory.com/133

https://needjarvis.tistory.com/694?category=933540

https://velog.io/@yookyungkho/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EC%98%B5%ED%8B%B0%EB%A7%88%EC%9D%B4%EC%A0%80-%EC%A0%95%EB%B3%B5%EA%B8%B0%EB%B6%80%EC%A0%9C-CS231n-Lecture7-Review

https://hiddenbeginner.github.io/deeplearning/2019/09/22/optimization_algorithms_in_deep_learning.html

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html