AI Odyssey: Navigating the Depths of Deep Learning

Computer Vision and Pattern Recognition

Byeongjoon Noh

powernoh@sch.ac.kr

SOON CHUN HYANG UNIVERSITY

Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

용어 정리

- 모델링 (modeling)
- 파라미터 (parameter)
- 비용함수, 손실함수 (cost function, loss function)
- 학습, 검증, 테스트 (training, validation, test)

용어 정리

- 평가 지표 (evaluation metric)
- 혼동행렬 (confusion matrix, 오차행렬, 정오행렬)
- TP (True Positive), FN (False Negative), FP (False Positive), TN (True Negative)
- 정확도 (accuracy), 민감도 (sensitive), 특이도 (specificity), 정밀도 (precision), 재현율 (recall)
- RMSE (root mean squared error, 평균 제곱근 오차), MSE (mean squared error, 평균 제곱 오차)
- MAE (mean absolute error, 평균 절대 오차), MAPE (mean absolute percentage error, 평균 절대 비율 오차)
- ROC (receiver operating characteristic) curve, AUC (Area Under the Curve), R2 score
- KL-divergence, cross entropy

용어 정리

- 레이어 (layer)
- 노드 (node)
- 활성화 함수 (activation function)
- 경사하강법 (gradient descent)
- 에포크 (epoch)
- 반복 (iteration)
- 배치 (batch)
- 학습률 (learning rate)
- 옵티마이저 (optimizer)

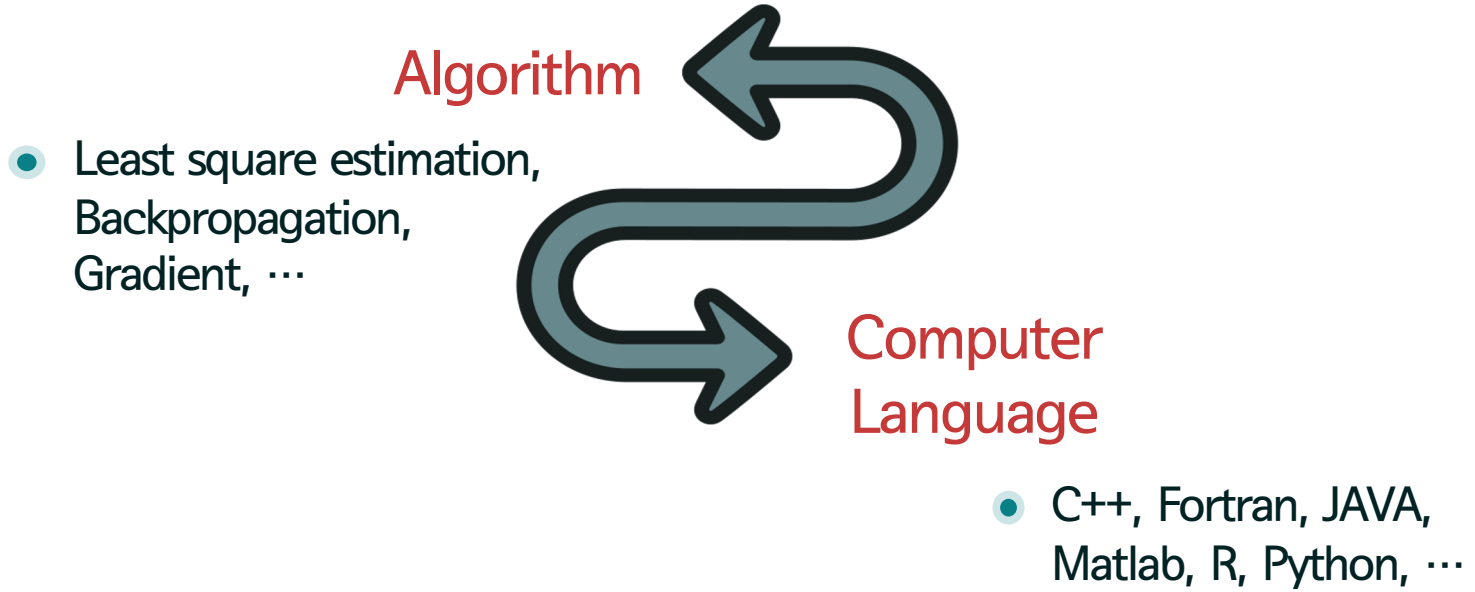
Contents

1. Introduction

기계학습 개요

- 기계학습 (Machine Learning, ML)
  - Machine Learning by computer languages to perform the algorithm created by human
    - algorithm: 문제를 해결하기 위한 방법들의 체계적인 모임

- ➔ 인간이 개발한 알고리즘을 컴퓨터 언어를 통해 기계에게 학습시키는 행위



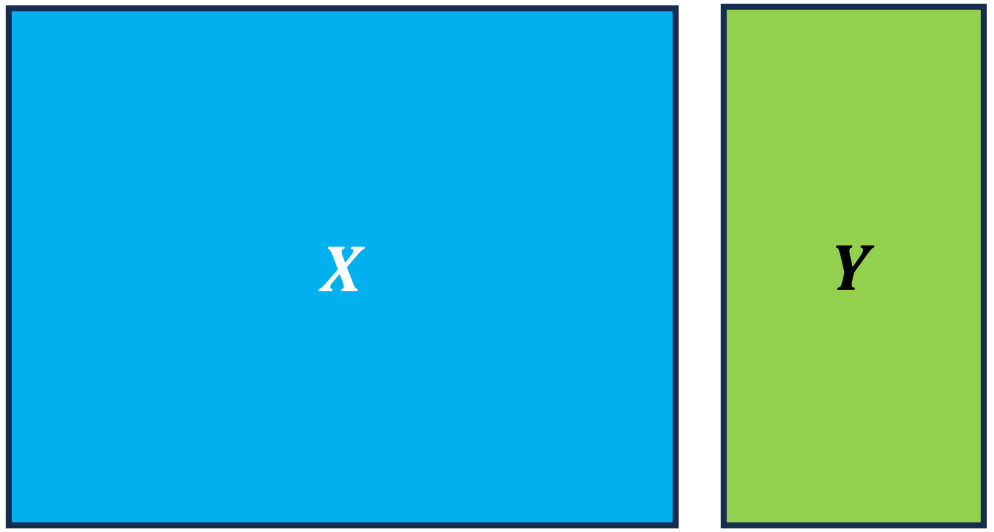
데이터 구조

- 다변량 데이터
  - 관측치: 샘플 (고객, 제품, 청구건, 환자, ...)
  - 변수: 각 관측치의 특성치

변수 / 관측치	X_1	...	X_i	...	X_p
N_1	X_{11}	...	X_{i1}	...	X_{1p}
...	...	...	...	...	...
N_i	X_{i1}	...	X_{ii}	...	X_{ip}
...	...	...	...	...	...
N_n	X_{n1}	...	X_{ni}	...	X_{np}

데이터 구조

- 독립변수와 종속변수



독립변수
예측변수
입력(인풋)변수

종속변수
반응변수
출력(아웃풋)변수

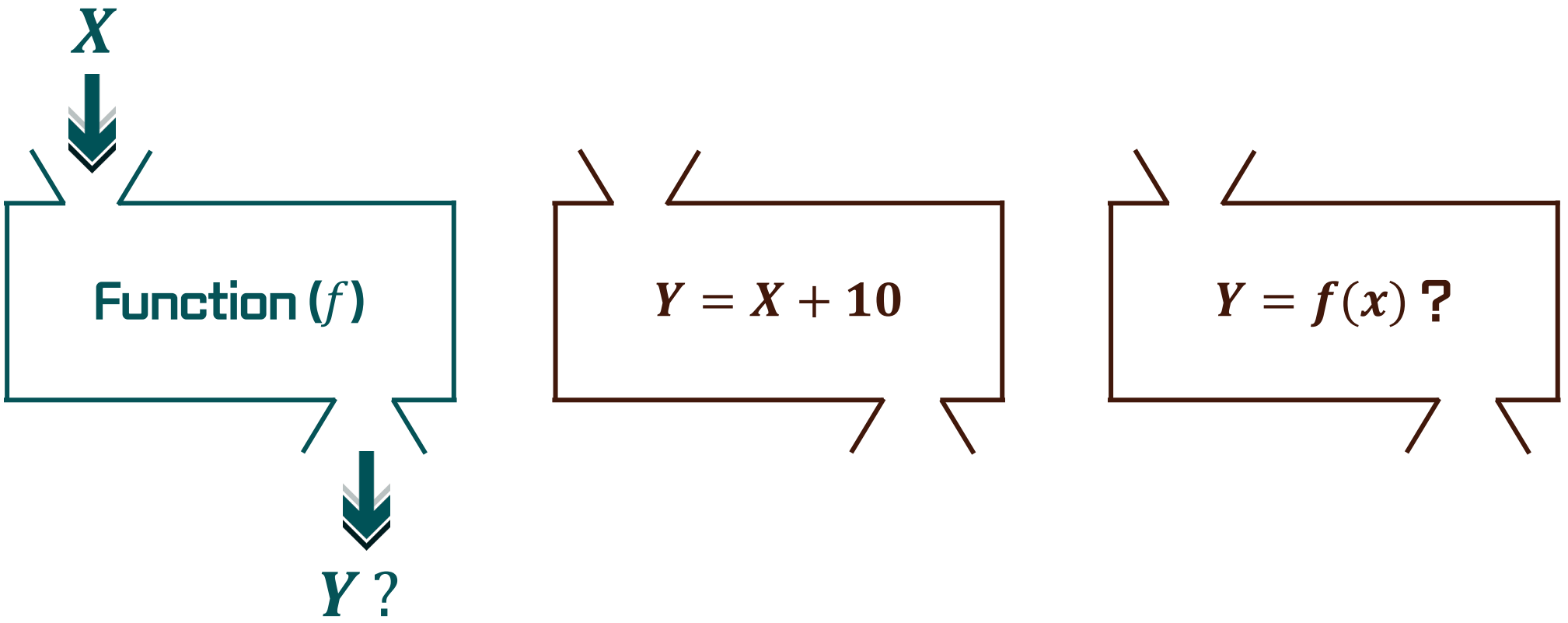
데이터 구조

- 데이터 구조 예시 – 중고차 판매 데이터

		X		Y
모델	주행거리	마력	용량	가격
TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	46,986	90	2,000	13,500
TOYOTA Corolla 1800 T SPORT VVT 12/3-Doors	19,700	192	1,800	21,500
TOYOTA Corolla 1.9 D HATCHB TERRA 2/3-Doors	71,138	69	1,900	12,950
TOYOTA Corolla 1.8 WTL-i T-Sport 3-Drs 2/3-Doors	31,461	192	1,800	20,950
TOYOTA Corolla 1.8 16V VWTLI 3DR T SPORT BNS 2/3-Doors	43,610	192	1,800	19,950
TOYOTA Corolla 1.6 VVTI Linea Terra Comfort 2/3-Doors	21,716	110	1,600	17,950
TOYOTA Corolla 1.6 16v L.SOL 2/3-Doors	25,563	110	1,600	16,750
TOYOTA Corolla 1.6 16V WTI 3DR TERRA 2/3-Doors	64,359	110	1,600	16,950
TOYOTA Corolla 1.6 16V VVTI 3DR SOL AUT4 2/3-Doors	43,905	110	1,600	16,950
TOYOTA Corolla 1.6 16V VVTI 3DR SOL 2/3-Doors	56,349	110	1,600	15,950
TOYOTA Corolla 1.4 VWTI Linea Terra 2/3-Doors	9,750	97	1,400	12,950
TOYOTA Corolla 1.4 16V VVTI 3DR 2/3-Doors	27,500	97	1,400	14,750
TOYOTA Corolla 2.0 D4D 90 SDR SOL 4/5-Doors	79,375	90	2,000	17,950
TOYOTA Corolla 2.0 D4D 90 SDR TERRA 4/5-Doors	79,375	97	2,000	15,800
TOYOTA Corolla 1.4 16V VVTI SDR TERRA COMFORT 4/5-Doors	75,048	97	1,400	15,800

기계학습 모델링

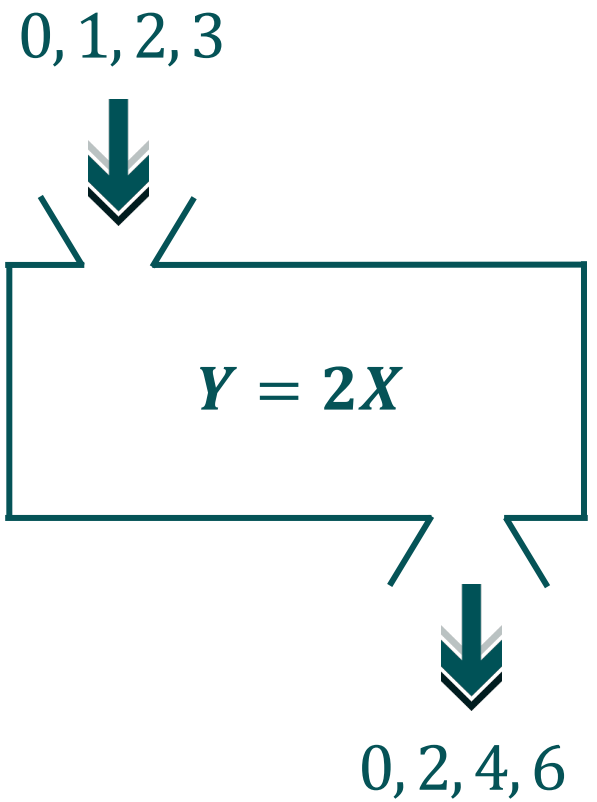
- 모델링의 개념



기계학습 모델링

- 모델링의 개념

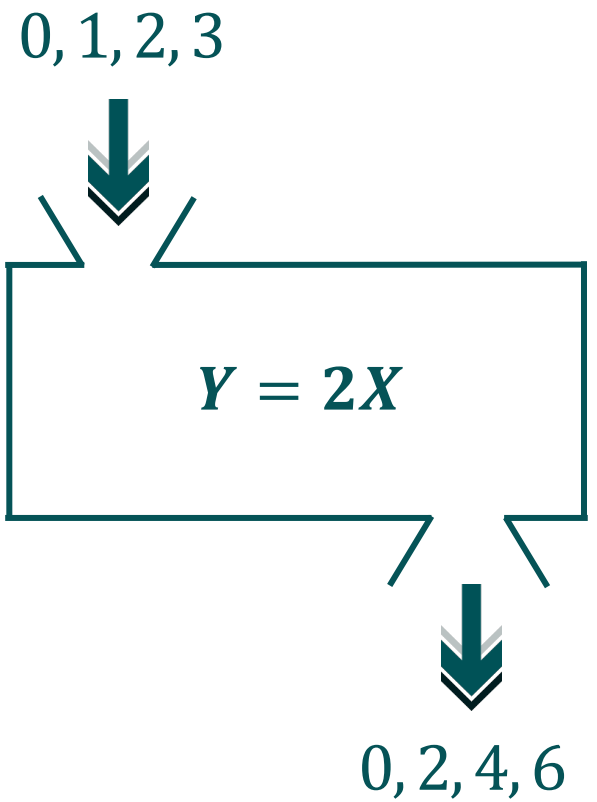
X	Y
0	0
1	2
2	4
3	6



기계학습 모델링

- 모델링의 개념

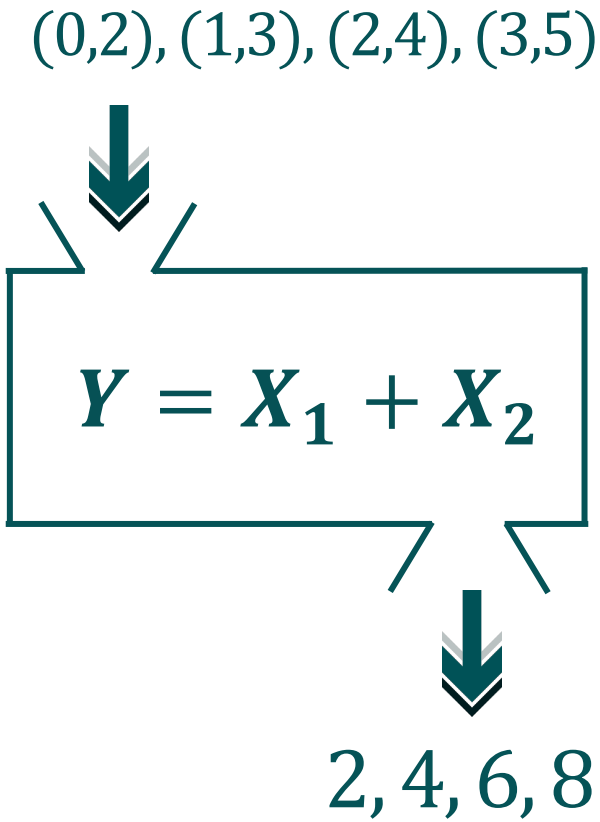
X	Y
0	0
1	2
2	4
3	6



기계학습 모델링

- 모델링의 개념

X1	X2	Y
0	2	2
1	3	4
2	4	6
3	5	8



기계학습 모델링

- 기계학습의 종류
  - 지도학습 (Supervised learning)
  - 비지도학습 (Unsupervised learning)
  - 준지도학습 (Semi-supervised learning)
  - 강화학습 (Reinforcement learning)

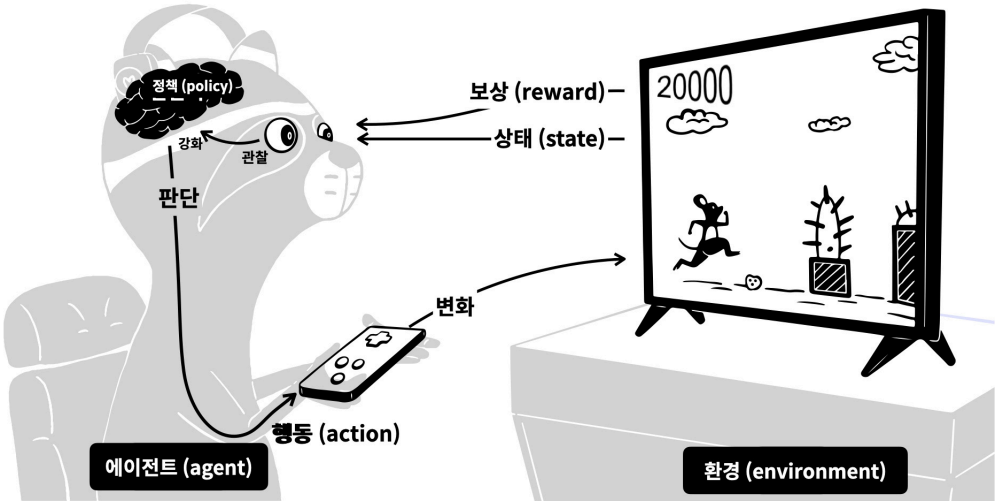
기계학습 모델링

- 지도학습 (Supervised learning)
  - 문제와 정답을 모두 알려주고 모델을 학습하는 방법
  - 예측문제 (Prediction, Regression), 분류문제 (Classification, Softmax problem)
  - X와 Y가 모두 주어진 상황

- 비지도학습 (Unsupervised learning)
  - 정답을 알려주지 않고 모델을 학습하는 방법
  - 군집화 (clustering), 연관규칙 방법 등
  - X는 주어지지만 Y는 주어지지 않는 상황

기계학습 모델링

- 강화학습 (Reinforcement learning)
  - 보상이 높은 방향으로 모델을 학습하는 방법

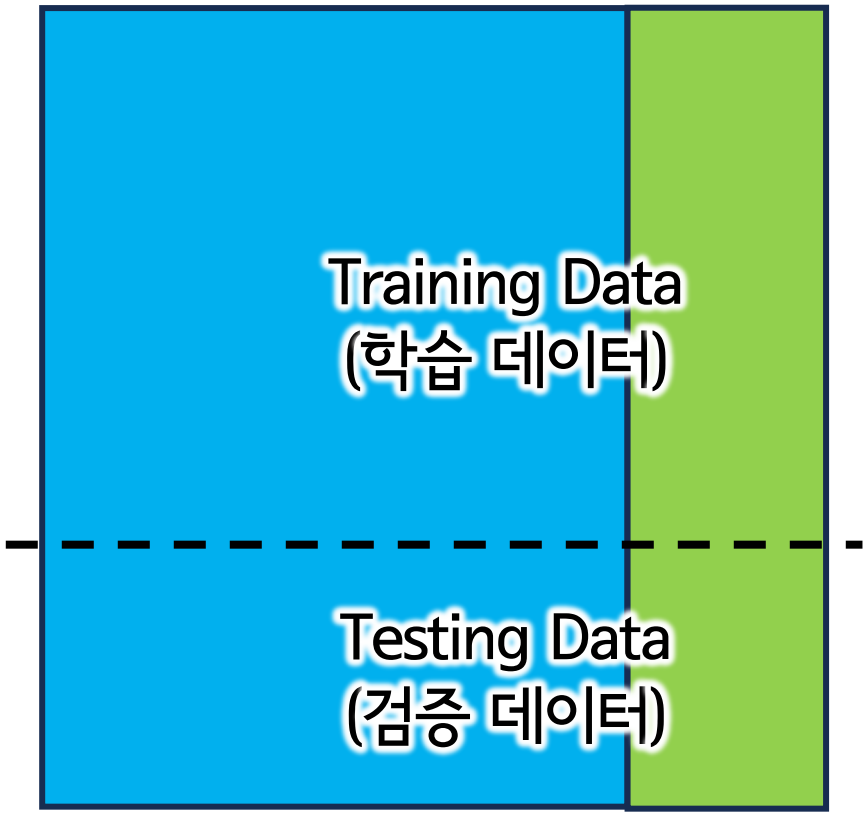


- 준지도학습 (Semi-supervised learning)
  - 주어진 데이터 중 일부만 정답 (Y)가 주어진 경우

    Y가 주어지지 않은 데이터와 함께 사용하여 모델을 학습하는 방법

기계학습 모델링

- 학습데이터 (training data)와 테스트데이터 (test data)

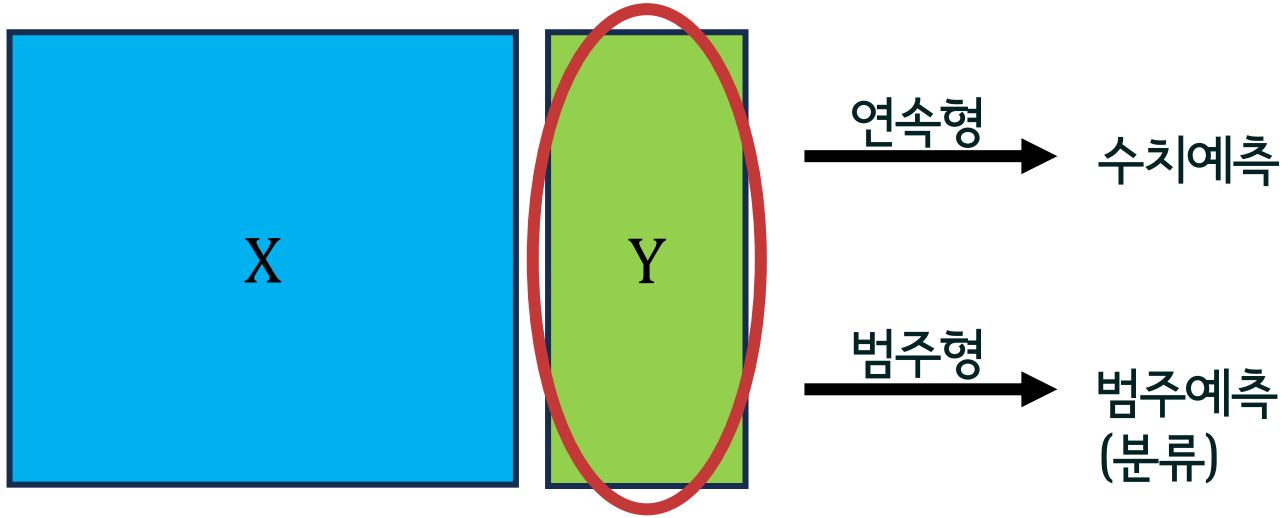


모델 구축 시 사용되는 데이터

구축된 모델을 검증하는데
사용되는 데이터

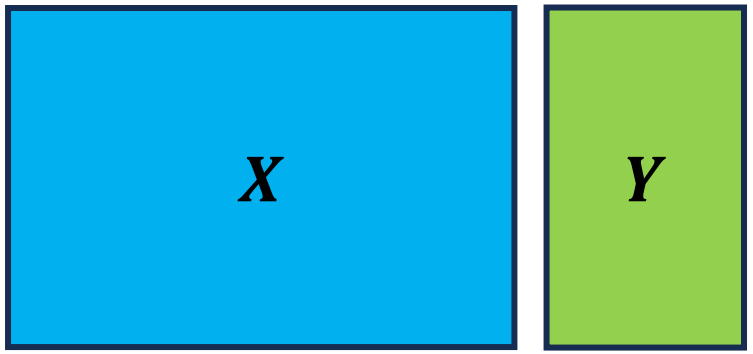
기계학습 모델링

- 수치예측과 범주예측(분류)
  - 연속형 데이터: 데이터 자체를 숫자로 표현
    - ex) 가격, 길이, 압력, 두께, ...
  - 범주형 데이터: 원칙적으로 숫자로 표현할 수 없는 데이터
    - ex) 제품불량 여부 (양품/불량), 보험사기여부(정상/비정상)



기계학습 모델링

- 수치예측

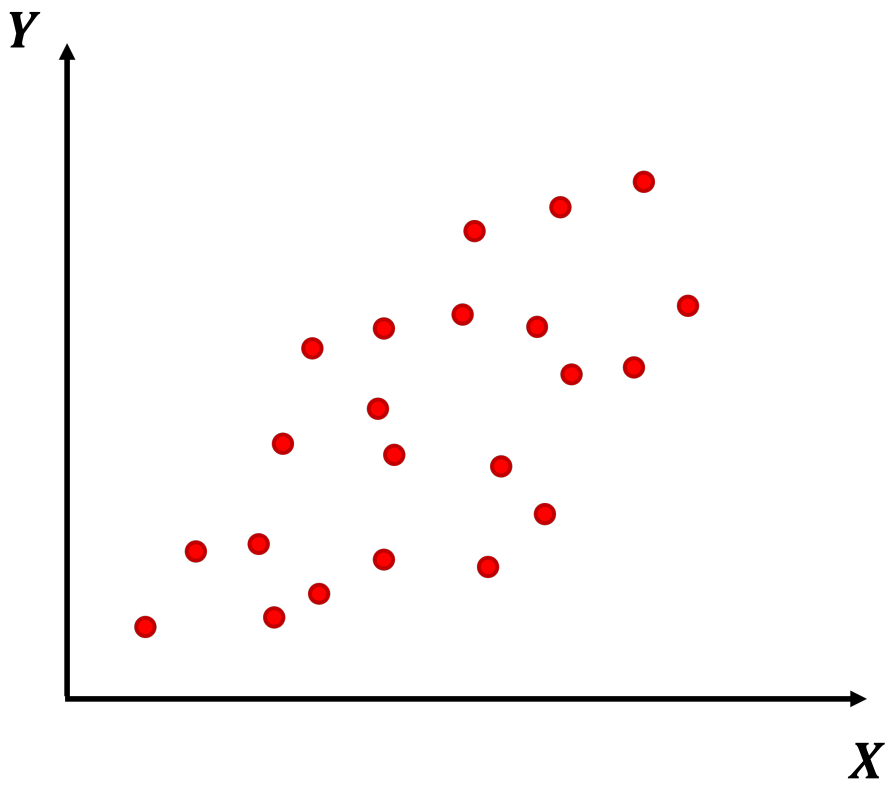


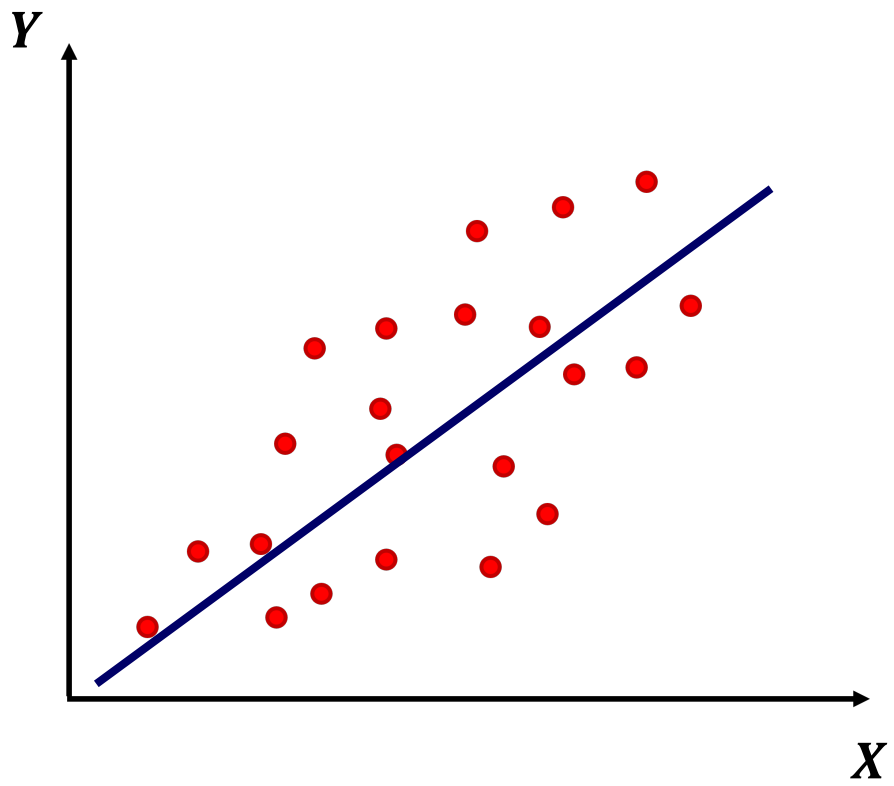
연속형 → 수치예측 (Regression)

인자(변수) / 관측치	X_1	…	X_i	…	X_p	Y
N_1	X_{11}	…	X_{i1}	…	X_{1p}	20.5
…	…	…	…	…	…	22.2
N_i	X_{i1}	…	X_{ii}	…	X_{ip}	…
…	…	…	…	…	…	72.3
N_n	X_{n1}	…	X_{ni}	…	X_{np}	82.8

기계학습 모델링

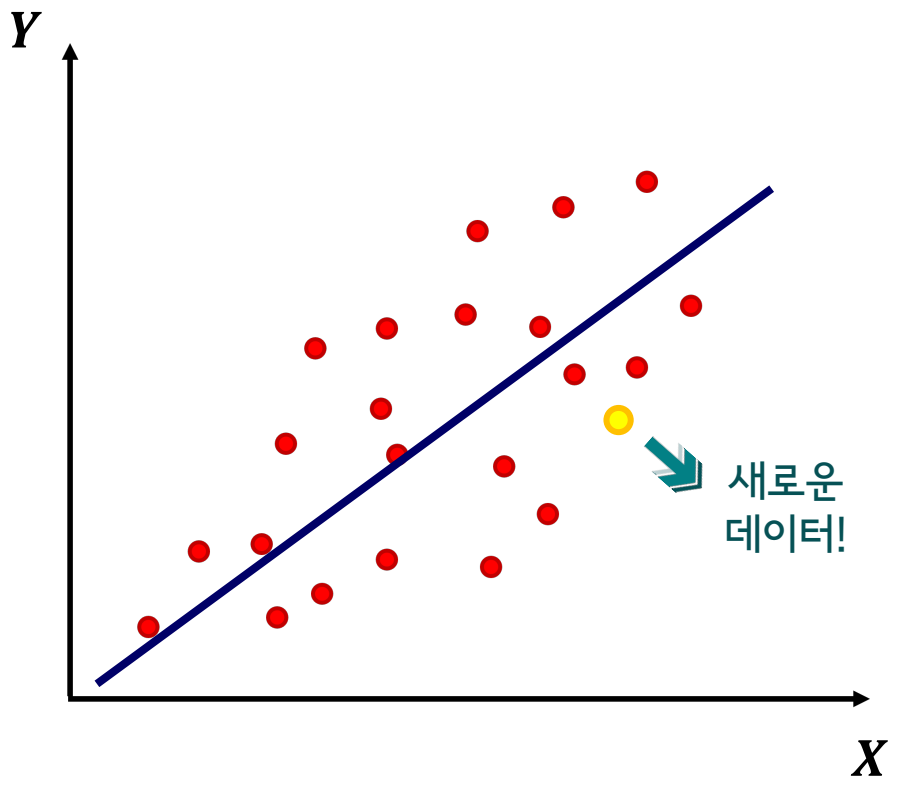
- 수치예측

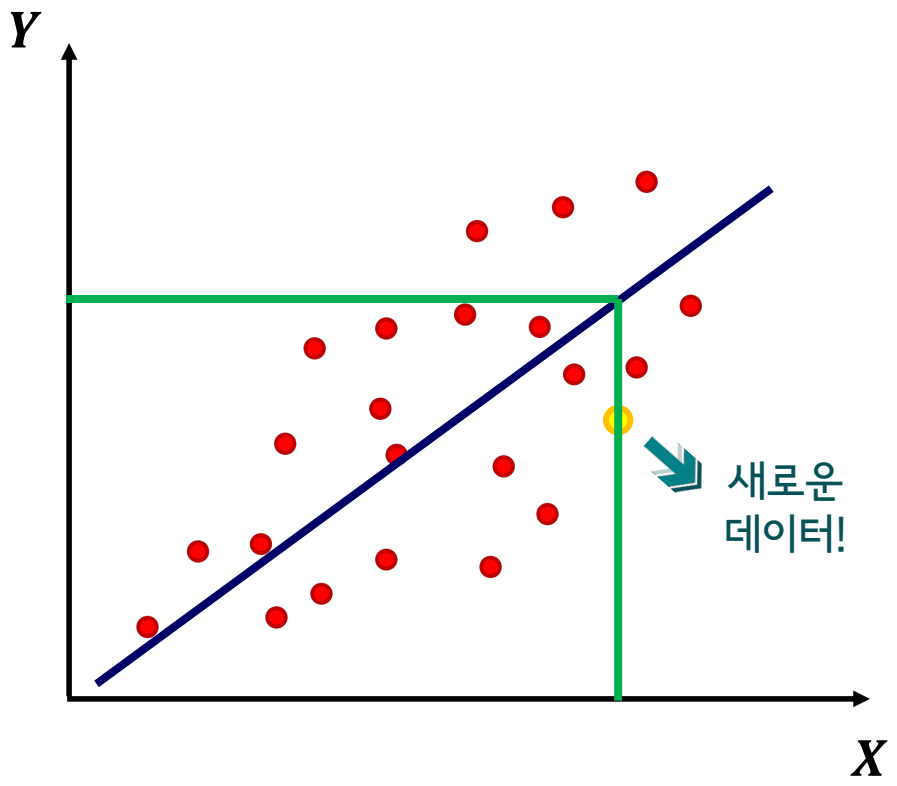




기계학습 모델링

- 수치예측





기계학습 모델링

- 수치예측 예제 – 중고차 가격 예측

		X		Y
모델	주행거리	마력	용량	가격
TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	46,986	90	2,000	13,500
TOYOTA Corolla 1800 T SPORT VVT 12/3-Doors	19,700	192	1,800	21,500
TOYOTA Corolla 1.9 D HATCHB TERRA 2/3-Doors	71,138	69	1,900	12,950
TOYOTA Corolla 1.8 WTL-i T-Sport 3-Drs 2/3-Doors	31,461	192	1,800	20,950
TOYOTA Corolla 1.8 16V VWTLI 3DR T SPORT BNS 2/3-Doors	43,610	192	1,800	19,950
TOYOTA Corolla 1.6 VVTI Linea Terra Comfort 2/3-Doors	21,716	110	1,600	17,950
TOYOTA Corolla 1.6 16v L.SOL 2/3-Doors	25,563	110	1,600	16,750
TOYOTA Corolla 1.6 16V WTI 3DR TERRA 2/3-Doors	64,359	110	1,600	16,950
TOYOTA Corolla 1.6 16V VVTI 3DR SOL AUT4 2/3-Doors	43,905	110	1,600	16,950
TOYOTA Corolla 1.6 16V VVTI 3DR SOL 2/3-Doors	56,349	110	1,600	15,950
TOYOTA Corolla 1.4 16V VVTI 3DR 2/3-Doors	27,500	97	1,400	14,750
TOYOTA Corolla 2.0 D4D 90 SDR SOL 4/5-Doors	79,375	90	2,000	17,950
TOYOTA Corolla 2.0 D4D 90 SDR TERRA 4/5-Doors	79,375	97	2,000	15,800
TOYOTA Corolla 1.4 16V VVTI SDR TERRA COMFORT 4/5-Doors	75,048	97	1,400	15,800

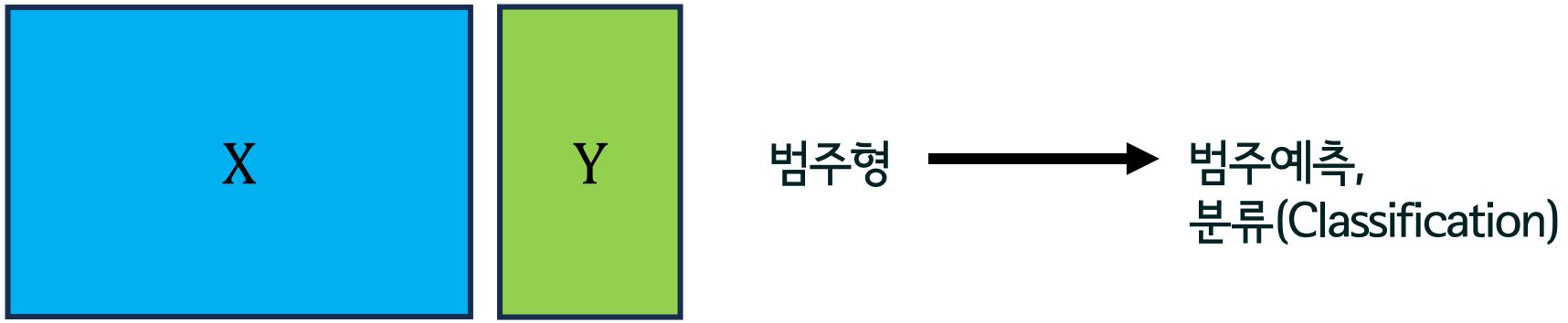
기계학습 모델링

- 수치예측 예제 – 중고차 가격 예측

		X		Y
모델	주행거리	마력	용량	가격
TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	46,986	90	2,000	13,500
TOYOTA Corolla 1800 T SPORT VVT 12/3-Doors	19,700	192	1,800	21,500
TOYOTA Corolla 1.9 D HATCHB TERRA 2/3-Doors	71,138	69	1,900	12,950
TOYOTA Corolla 1.8 WTL-i T-Sport 3-Drs 2/3-Doors	31,461	192	1,800	20,950
TOYOTA Corolla 1.8 16V VWTLI 3DR T SPORT BNS 2/3-Doors	43,610	192	1,800	19,950
TOYOTA Corolla 1.6 VVTI Linea Terra Comfort 2/3-Doors	21,716	110	1,600	17,950
TOYOTA Corolla 1.6 16v L.SOL 2/3-Doors	25,563	110	1,600	16,750
TOYOTA Corolla 1.6 16V WTI 3DR TERRA 2/3-Doors	64,359	110	1,600	16,950
TOYOTA Corolla 1.6 16V VVTI 3DR SOL AUT4 2/3-Doors	43,905	110	1,600	16,950
TOYOTA Corolla 1.6 16V VVTI 3DR SOL 2/3-Doors	56,349	110	1,600	15,950
TOYOTA Corolla 1.4 16V VVTI 3DR 2/3-Doors	27,500	97	1,400	14,750
TOYOTA Corolla 2.0 D4D 90 SDR SOL 4/5-Doors	79,375	90	2,000	17,950
TOYOTA Corolla 2.0 D4D 90 SDR TERRA 4/5-Doors	79,375	97	2,000	15,800
TOYOTA Corolla 1.4 16V VVTI SDR TERRA COMFORT 4/5-Doors	75,048	97	1,400	15,800
TOYOTA Corolla 1.4 16V VVTI SDR TERRA COMFORT 4/5-Doors	132151	110	1600	????

기계학습 모델링

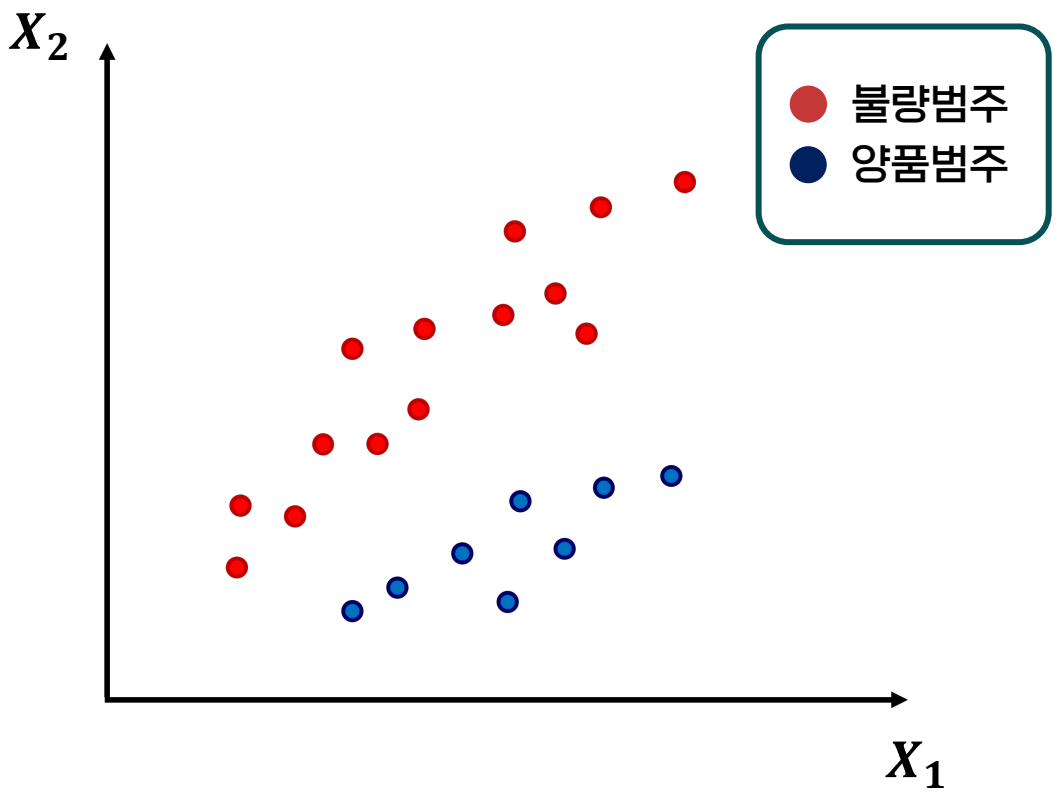
- 수치예측

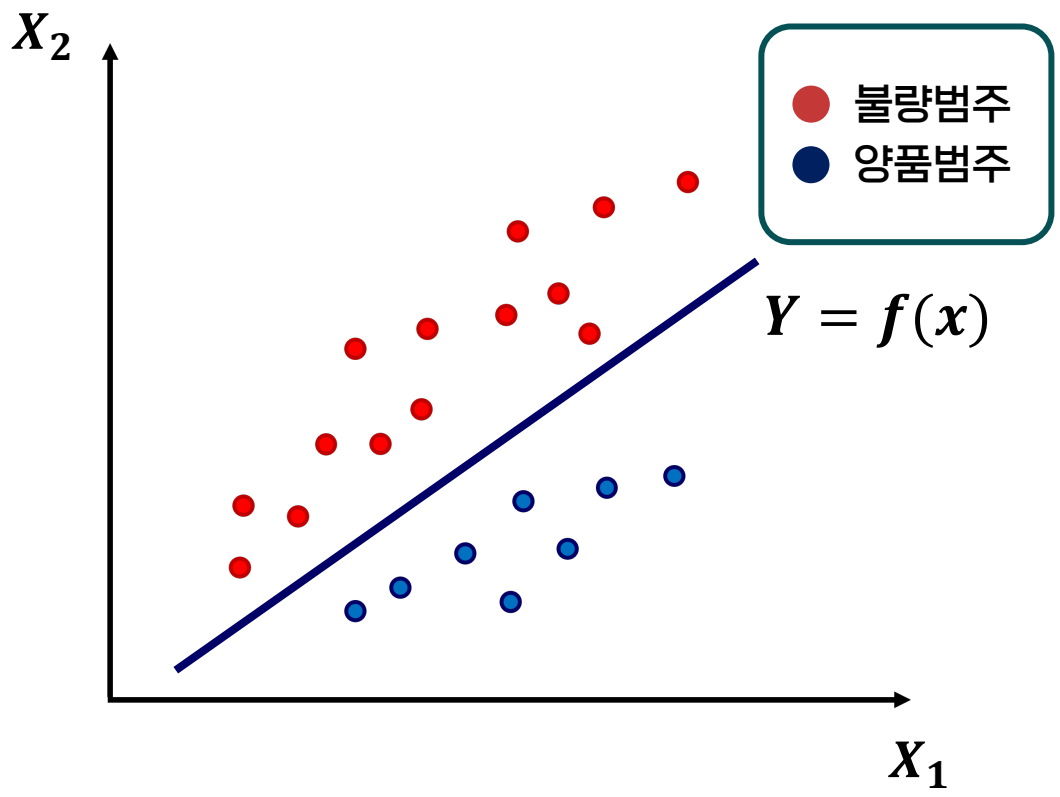


인자(변수) / 관측치	X_1	…	X_i	…	X_p	Y
N_1	X_{11}	…	X_{i1}	…	X_{1p}	0(정상)
…	…	…	…	…	…	0(정상)
N_i	X_{i1}	…	X_{ii}	…	X_{ip}	…
…	…	…	…	…	…	1(불량)
N_n	X_{n1}	…	X_{ni}	…	X_{np}	1(불량)

기계학습 모델링

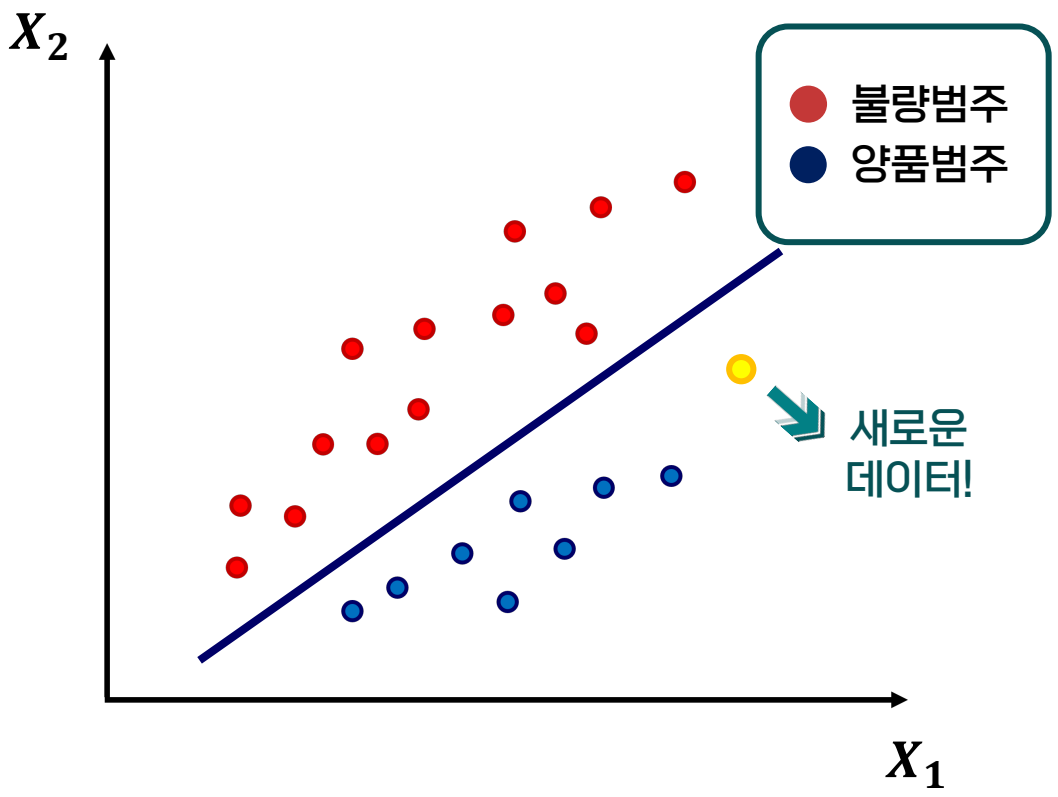
- 범주예측

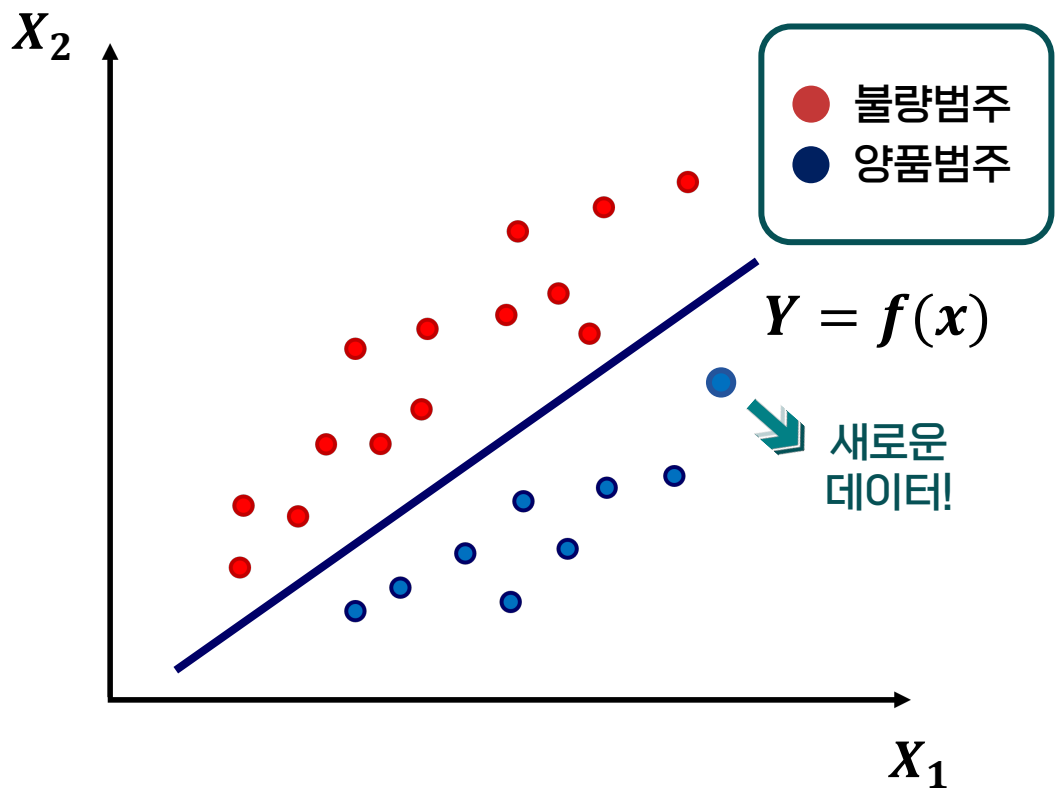




기계학습 모델링

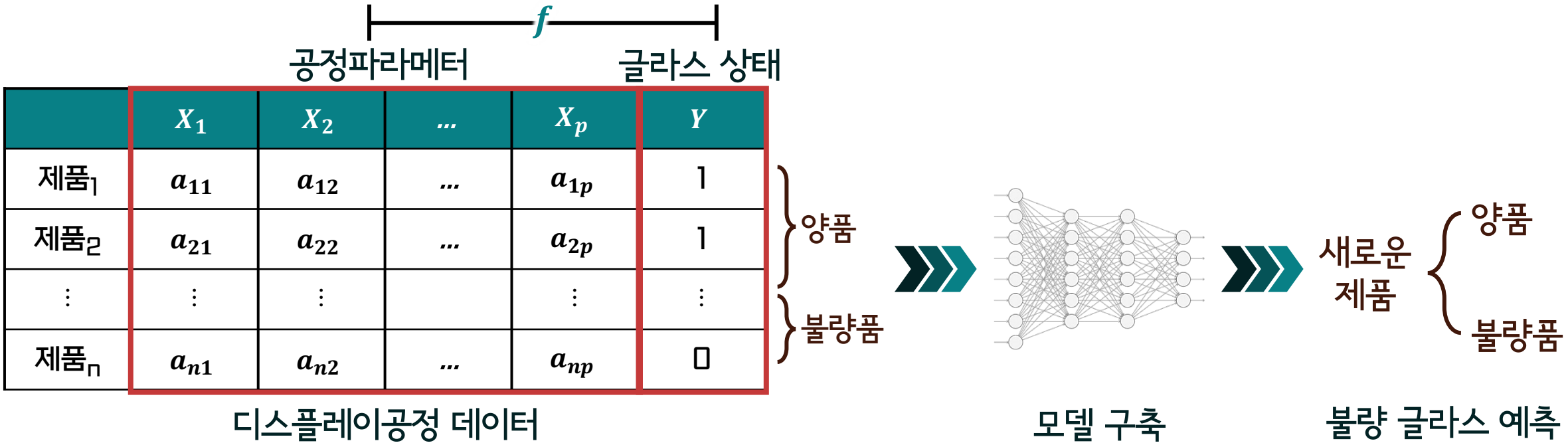
- 범주예측





기계학습 모델링

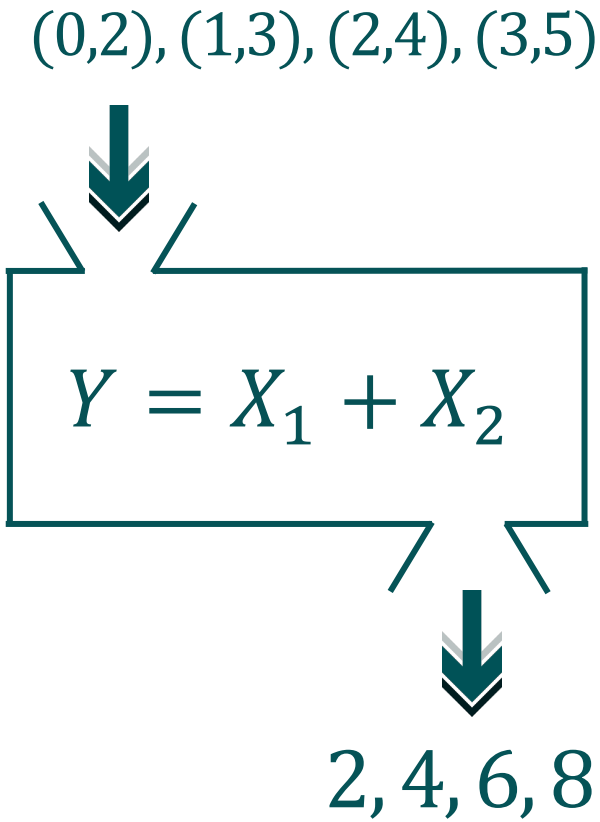
- 범주예측 예제 – 디스플레이 공정 내 양품/불량품 여부 예측



기계학습 모델의 학습 프로세스

- X와 Y의 관계 찾기

X1	X2	Y
0	2	2
1	3	4
2	4	6
3	5	8

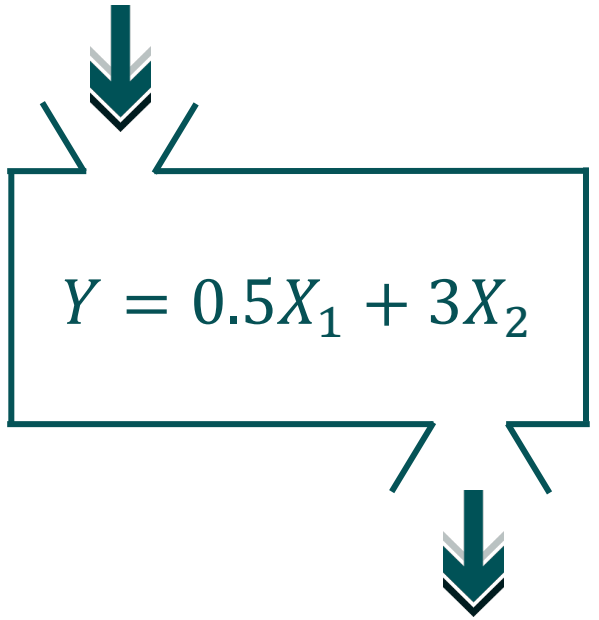


기계학습 모델의 학습 프로세스

- X와 Y의 관계 찾기

X1	X2	Y
0	0	6
1	1	9.5
2	2	13
3	3	16.5

$(0,2), (1,3), (2,4), (3,5)$



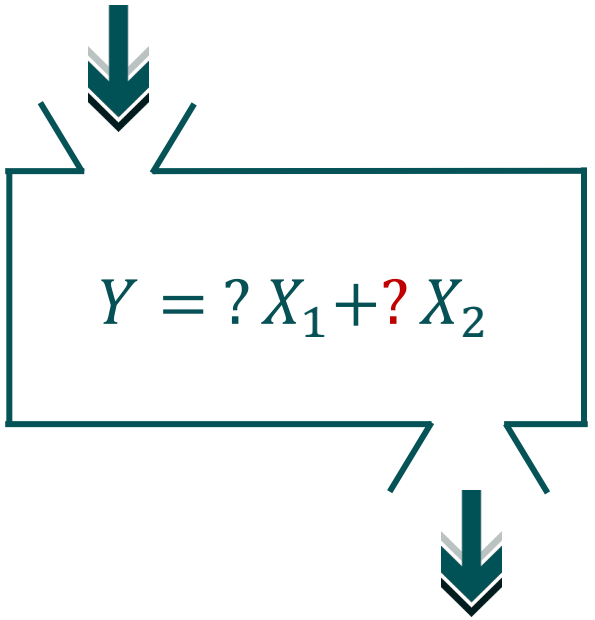
$6, 9.5, 13, 16.5$

기계학습 모델의 학습 프로세스

- X와 Y의 관계 찾기

X1	X2	Y
0	0	6
1	1	9.5
2	2	13
3	3	16.5

$(0,2), (1,3), (2,4), (3,5)$



$6, 9.5, 13, 16.5$

기계학습 모델의 학습 프로세스

- X와 Y의 관계 찾기

$$Y = ? X_1 + ? X_2 + ? X_3 + \varepsilon$$

X로 설명되는 부분 / 그렇지 않은 부분

	X_1	X_2	X_3	Y
모델	주행거리	마력	용량	가격
TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	46,986	90	2,000	13,500
TOYOTA Corolla 1800 T SPORT VVT 12/3-Doors	19,700	192	1,800	21,500
TOYOTA Corolla 1.9 D HATCHB TERRA 2/3-Doors	71,138	69	1,900	12,950
TOYOTA Corolla 1.8 WTL-i T-Sport 3-Drs 2/3-Doors	31,461	192	1,800	20,950
TOYOTA Corolla 1.8 16V VVTLI 3DR T SPORT 2/3-Doors	43,610	192	1,800	19,950
TOYOTA Corolla 1.6 Terra 2/3-Doors	21,716	110	1,600	17,950
TOYOTA Corolla 1.6 16V LSOL 2/3-Doors	25,563	110	1,600	16,750
[illegible]	64,359	110	1,600	16,950
TOYOTA Corolla 1.6 16V VVTI 3DR SOL AUT4 2/3-Doors	43,905	110	1,600	16,950
[illegible] 2/3-Doors	56,349	110	1,600	15,950
TOYOTA Corolla 1.4 VVTI Linea Terra 2/3-Doors	9,750	97	1,400	12,950
TOYOTA Corolla 1.4 16V VVTI 3DR 2/3-Doors	27,500	97	1,400	14,750
TOYOTA Corolla 2.0 D4D 90 SDR SOL 4/5-Doors	79,375	90	2,000	17,950
TOYOTA Corolla 2.0 D4D 90 SDR TERRA 4/5-Doors	79,375	97	2,000	15,800
TOYOTA Corolla 1.4 16V VVTI SDR TERRA COMFORT 4/5-Doors	75,048	97	1,400	15,800

기계학습 모델의 학습 프로세스

- 기계학습 모델 학습의 핵심

$$Y = w_1 X_1 + w_2 X_2 + \varepsilon$$

파라미터 (parameter, 母數, 媒介變數)

데이터가 주어졌을 때 모델의 파라미터 찾기 ! ➤ 파라미터 추정

기계학습 모델의 학습 프로세스

- 파라미터 추정

$$Y = w_1 X_1 + w_2 X_2 + \varepsilon$$
$$= f(X) + \varepsilon$$

$$\varepsilon = Y - f(X) \Rightarrow \text{오차}$$

Loss function
(손실함수)

$$Y - f(X) = 0, \quad \varepsilon = 0$$

기계학습 모델의 학습 프로세스

- 파라미터 추정

$$\varepsilon = Y - f(x)$$ ⇒ Loss function (손실함수)

$$\varepsilon = Y - (w_1 X_1 + w_2 X_2)$$

$$f(X) = w_1 X_1 + w_2 X_2$$

$$\varepsilon_i = Y_i - \left(w_1 X_{1_i} + w_2 X_{2_i}\right), i = 1, 2, \ldots, n$$

기계학습 모델의 학습 프로세스

- 파라미터 추정

$$\varepsilon_i = Y_i - \left(w_1 X_{1_i} + w_2 X_{2_i}\right), i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n} \{Y_i - (w_1 X_{1i} + w_2 X_{2i})\} = 0$$

$$\sum_{i=1}^{n} \{Y_i - (w_1 X_{1i} + w_2 X_{2i})\}^2$$

Cost function
(비용함수)

기계학습 모델의 학습 프로세스

- 파라미터 추정

$$\sum_{i=1}^{n} \{Y_i - (w_1 X_{1i} + w_2 X_{2i})\}^2$$

Cost function
(비용함수)

 비용함수를 최소로 하는 w_1와 w_2를 찾자!

$$\min_{w_1, w_2} \sum_{i=1}^{n} \{Y_i - (w_1 X_{1i} + w_2 X_{2i})\}^2$$

기계학습 모델의 학습 프로세스

- 파라미터 추정

$$\min_{w_1, w_2} \sum_{i=1}^{n} \{Y_i - (w_1 X_{1i} + w_2 X_{2i})\}^2$$

➢ 답 : $\hat{w}_1$, $\hat{w}_2$

$$\hat{f}(X) = \hat{w}_1 X_{1i} + \hat{w}_2 X_{2i}$$

기계학습 모델의 학습 프로세스

- 모델 결정 ➔ 파라미터 추정

$$\min_{w_1,w_2} \sum_{i=1}^{n} \{Y_i - (w_1 X_{1i} + w_2 X_{2i})\}^2 \rightarrow f(x)$$

$$f(x) = w_0 + w_1 X_1 + w_2 X_2$$ 다중선형회귀 모델

$$f(x) = \frac{1}{1 + e^{-(w_0 + w_1 X_1 + w_2 X_2)}}$$ 로지스틱회귀 모델

$$f(x) = \sum_{m=1}^{n} k(m) I\{(x_1, x_2) \in R_m\}$$ 의사결정나무 모델

$$f(x) = \frac{1}{1 + e^{\left(-\left(w_0 + w_1\left(\frac{1}{1+e^{-(w_{01}+w_{11}x_1+w_{21}x_2)}}\right)\right)+w_2\left(\frac{1}{1+e^{-(w_{02}+w_{12}x_1+w_{22}x_2)}}\right)\right)}}$$ 뉴럴네트워크 모델

기계학습 모델의 학습 프로세스

- 파라미터 추정 예제 – 다중선형회귀 모델
  - Y가 연속형 변수일 때 활용
  - 입력변수 X의 선형결합으로 출력변수 Y를 표현
    - 선형결합 : 변수들을 (상수 배와)더하기, 빼기를 통해 결합

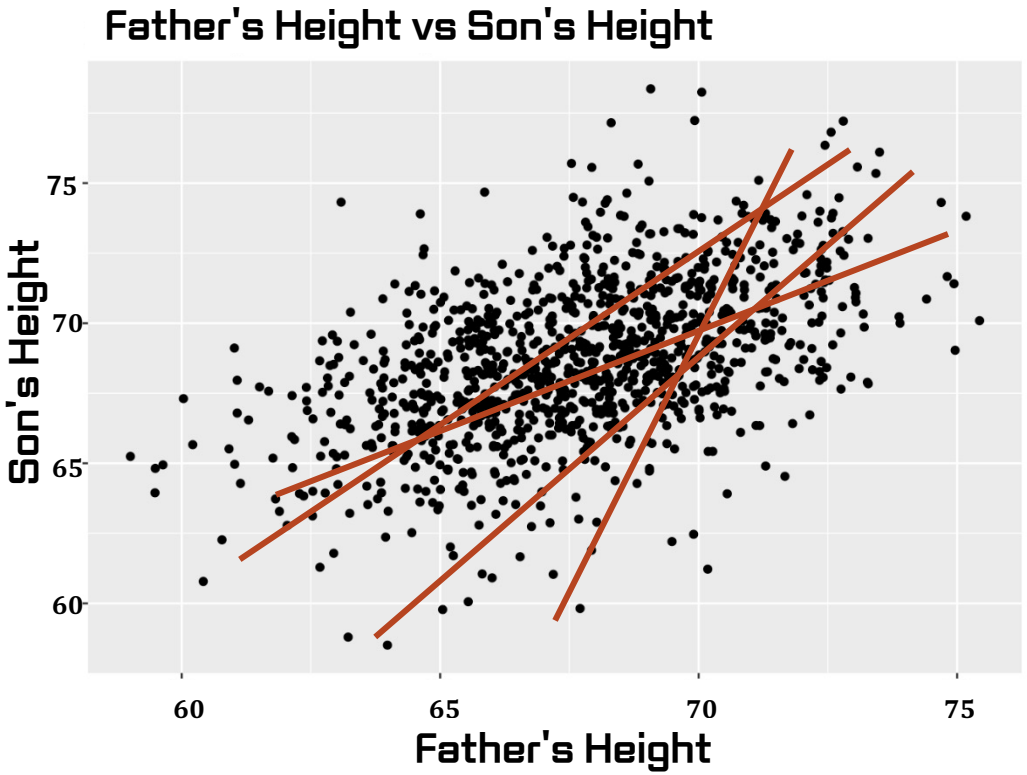
$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = \beta_0 + \sum_{i=1}^{p} \beta_i X_i$$

$$Y = w_0 + w_1 X_1 + w_2 X_2 + \cdots + w_p X_p = w_0 + \sum_{i=1}^{p} w_i X_i$$

기계학습 모델의 학습 프로세스

- 파라미터 추정 예제 – 다중선형회귀 모델

$$Y = f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$



무수히 많은 직선 중 단 하나의 직선을 찾는 것

파라미터 (Parameter)

- 파라미터를 찾자 (추정하자)

  가지고 있는 데이터들의 함수식으로!

기계학습 모델의 학습 프로세스

- 파라미터 추정 예제 – 다중선형회귀 모델

$$\min_{W} \sum_{i=1}^{n} \{Y_i - f(X)\}^2$$

$$f(x) = w_0 + w_1 X_1 + w_2 X_2 \quad \text{다중선형회귀 모델}$$

$$\min_{w_0, w_1, w_2} \sum_{i=1}^{n} \{Y_i - (w_0 + w_1 X_1 + w_2 X_2)\}^2$$

Least square estimation algorithm

$$\hat{f}(X) = \hat{w}_0 + \hat{w}_1 X_1 + \hat{w}_2 X_2$$

기계학습 모델의 학습 프로세스

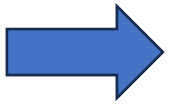
- 파라미터 추정 예제 – 다중선형회귀 모델
  - Least squared estimation algorithm

$$\min_{w_0, w_1} \sum_{i=1}^{n} \left(Y_i - (w_0 + w_1 X_i)\right)^2$$

  - cost function is convex ➔ globally optimal solution exists (전역 최적해가 반드시 존재)

$$\frac{\partial C(w_0, w_1)}{\partial w_0} = -2\left(\sum_{i=1}^{n}\left(Y_i - (w_0 + w_1 X_i)\right)\right) = 0$$

$$\frac{\partial C(w_0, w_1)}{\partial w_1} = -2\left(\sum_{i=1}^{n}(Y_i - (w_0 + \beta_1 X_i)X_i)\right) = 0$$



$$\hat{w}_0 = \bar{Y} - \hat{w}_1 \bar{X}$$

$$\hat{w}_1 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

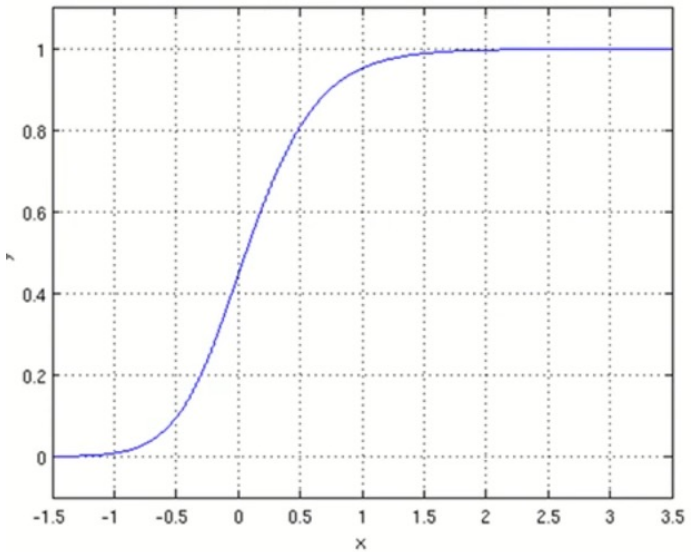
$$f(X) = \hat{Y} = \hat{w}_0 + \hat{w}_1 X$$

기계학습 모델의 학습 프로세스

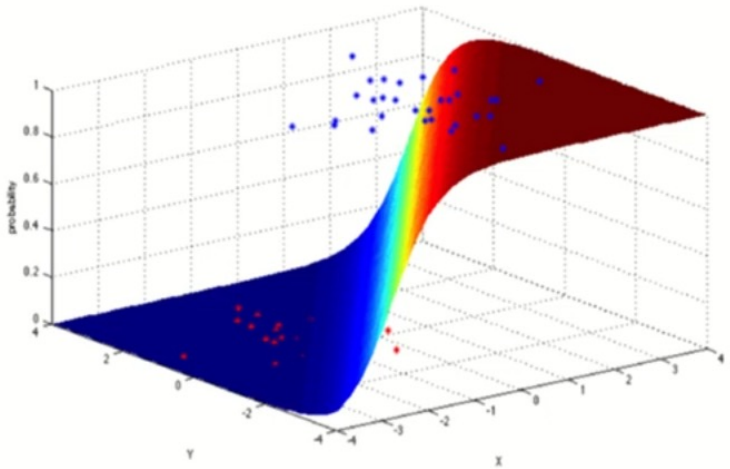
- 파라미터 추정 예제 – 로지스틱회귀 모델
  - Y가 범주형 변수일 때 활용
  - 입력변수 X의 비선형결합(로지스틱 함수 형태)으로 출력변수 Y를 표현

$$f(X) = \frac{1}{1+e^{-(w_0+w_1X_1+\cdots+w_pX_p)}}$$

$$f(X) = \frac{1}{1+e^{-(w_0+w_1X_1)}}$$

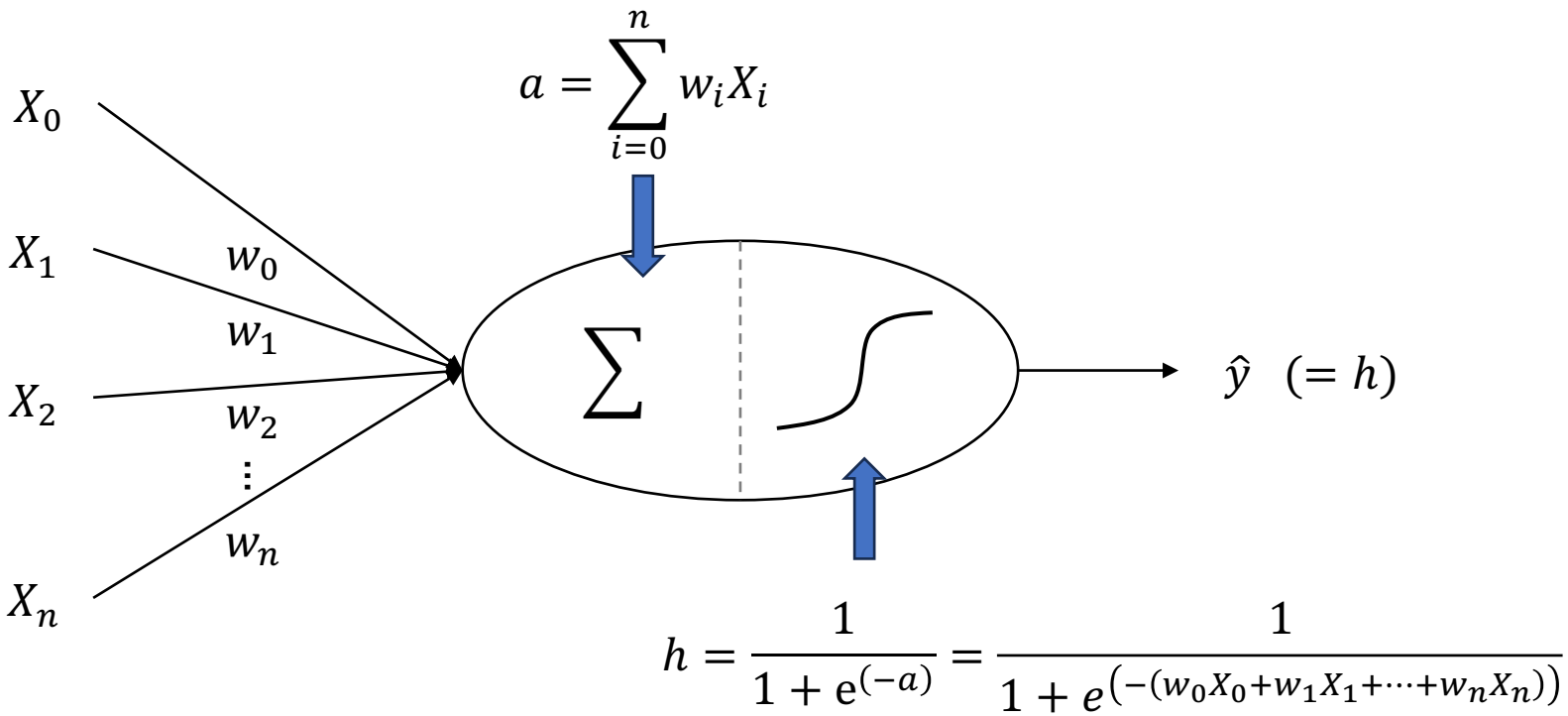


$$f(X) = \frac{1}{1+e^{-(w_0+w_1X_1+w_2X_2)}}$$



기계학습 모델의 학습 프로세스

- 파라미터 추정 예제 – 로지스틱회귀 모델
  - 비선형결합
    - 1) 입력변수의 선형결합
    - 2) 선형결합 값의 비선형 변환 (non-linear transformation)



기계학습 모델의 학습 프로세스

- 파라미터 추정 예제 – 로지스틱회귀 모델

$$\min_{W} \sum_{i=1}^{n} \{Y_i - f(X)\}^2$$

$$f(x) = \frac{1}{1 + e^{-(w_0 + w_1 X_1 + w_2 X_2)}}$$ 로지스틱회귀 모델

$$\min_{w_0, w_1, w_2} \sum_{i=1}^{n} \left\{ Y_i - \left(\frac{1}{1 + e^{-(w_0 + w_1 X_1 + w_2 X_2)}} \right) \right\}^2$$

Conjugate gradient algorithm

$$\hat{f}(X) = \frac{1}{1 + e^{-(\hat{w}_0 + \hat{w}_1 X_1 + \hat{w}_2 X_2)}}$$

기계학습 모델의 학습 프로세스

- 파라미터 추정 예제 – 뉴럴네트워크 모델

$$\min_{W} \sum_{i=1}^{n} \{Y_i - f(X)\}^2$$

뉴럴네트워크 모델

$$f(X) = \frac{1}{1 + e^{-\left(w_0 + w_1\left(\frac{1}{1+e^{-(w_{01}+w_{11}X_1+w_{21}X_2)}}\right) + w_2\left(\frac{1}{1+e^{-(w_{02}+w_{12}X_1+w_{22}X_2)}}\right)\right)}}$$

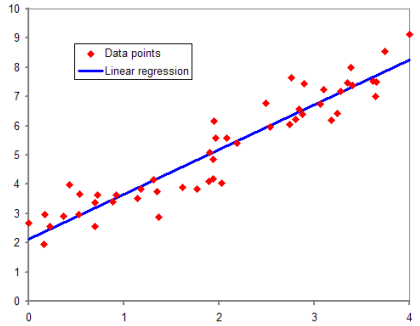
$$\min_{w_0 \cdots w_{22}} \sum_{i=1}^{n} \left\{ Y_i - \left(\frac{1}{1 + e^{\left(-\left(w_0 + w_1\left(\frac{1}{1+e^{-(w_{01}+w_{11}X_1+w_{21}X_2)}}\right)\right) + w_2\left(\frac{1}{1+e^{-(w_{02}+w_{12}X_1+w_{22}X_2)}}\right)\right)}} \right) \right\}^2$$

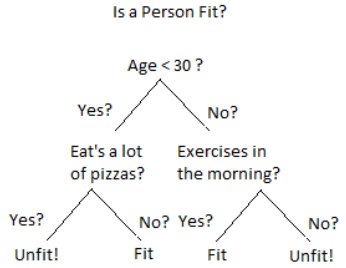
Backpropagation algorithm

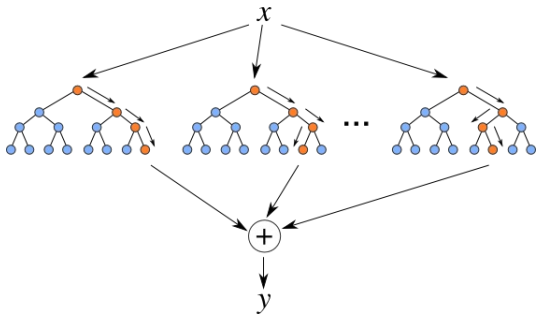
$$\hat{f}(X) = \frac{1}{1 + e^{\left(-\left(\hat{w}_0 + \hat{w}_1\left(\frac{1}{1+e^{-(\hat{w}_{01}+\hat{w}_{11}x_1+\hat{w}_{21}x_2)}}\right)\right) + \hat{w}_2\left(\frac{1}{1+e^{-(\hat{w}_{02}+\hat{w}_{12}x_1+\hat{w}_{22}x_2)}}\right)\right)}}$$

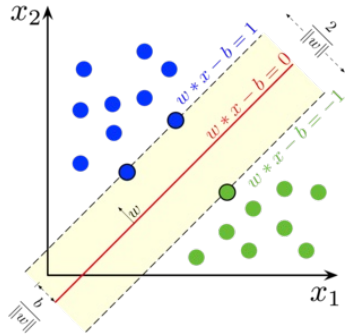
기계학습 모델의 학습 프로세스

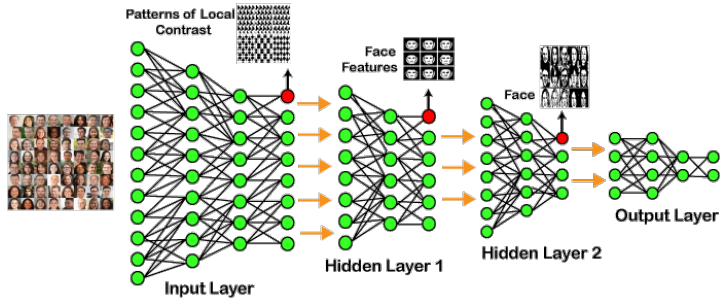
- 기계학습 모델 학습 과정 요약
  - 모델 결정하기 (Y를 표현하기 위한 X들의 조합 방식 결정)
  - 모델을 구성하는 파라미터 찾기
    - 어떻게? ➔ 가지고 있는 데이터를 이용해서
    - 무엇을 추구하며? ➔ 실제 데이터의 값과 최대한 같게 나오도록







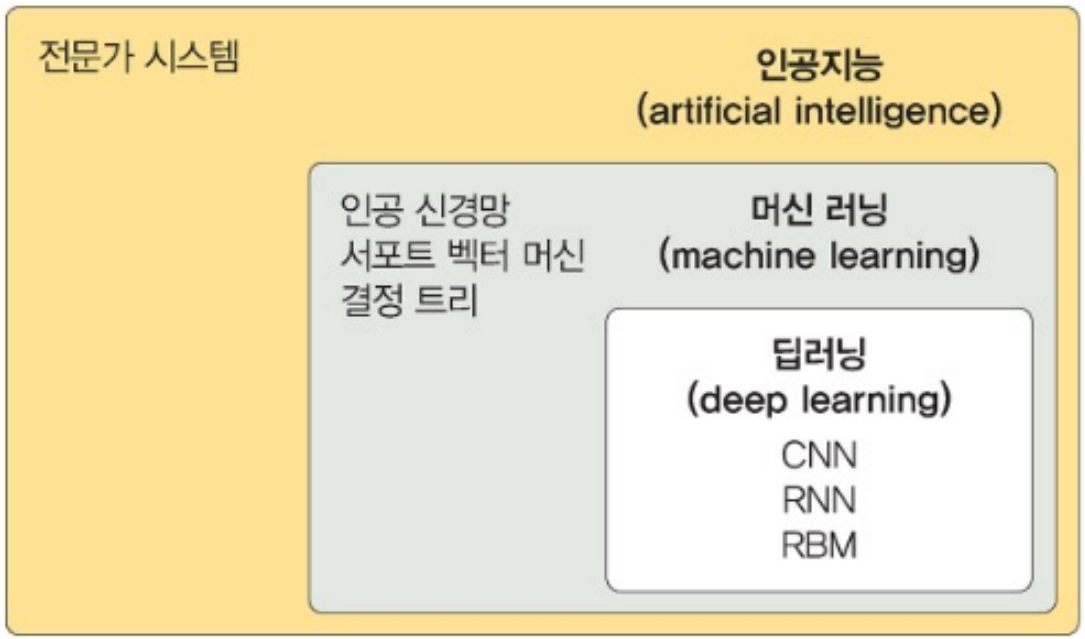


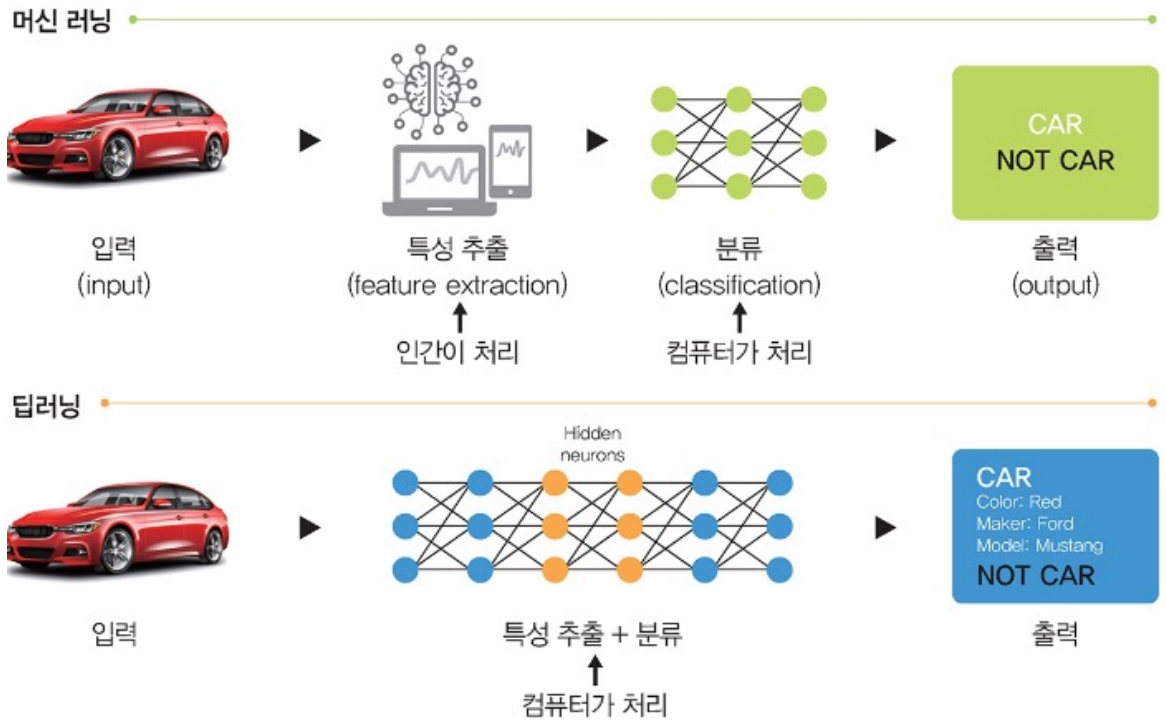


2. Multi Layer Perceptron (MLP)

Introduction

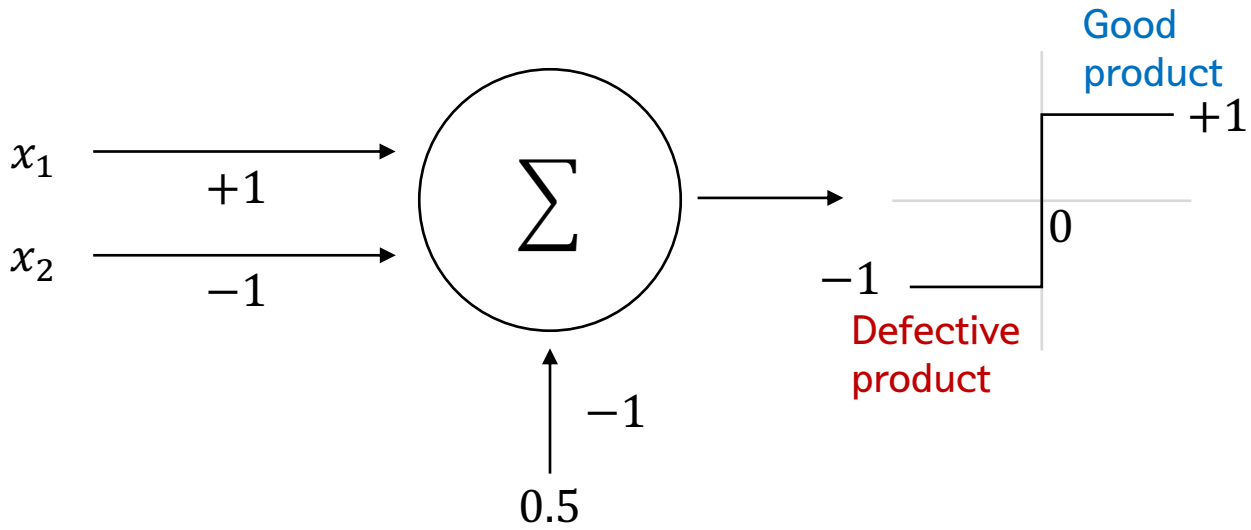
- 인공지능, 머신러닝, 딥러닝의 범주와 차이점





Perceptron

- A concept of perceptron
  - Single perceptron (1975) by Frank Rosenblatt
  - Linear combination of the input values and classify it by whether the value is greater than 0



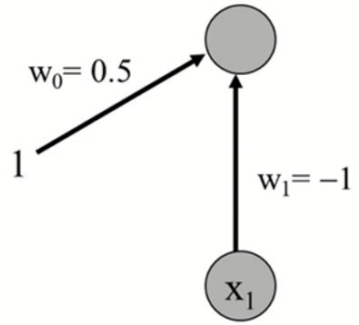
$x_1 - x_2 - 0.5 > 0$ ➔ Good product

$x_1 - x_2 - 0.5 \leq 0$ ➔ Defective product

Perceptron

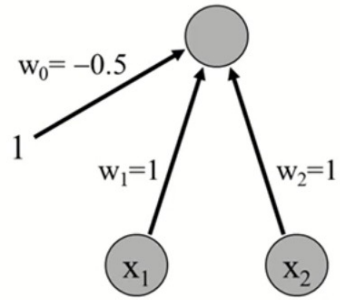
- Single layer perceptron

input x1	output
0	1
1	0



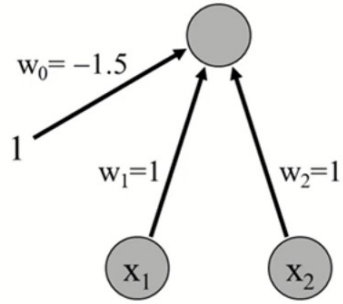
변환기

input x1	input x2	output
0	0	0
0	1	1
1	0	1
1	1	1



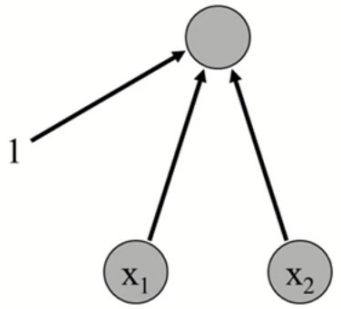
Boolean OR

input x1	input x2	output
0	0	0
0	1	0
1	0	0
1	1	1



Boolean AND

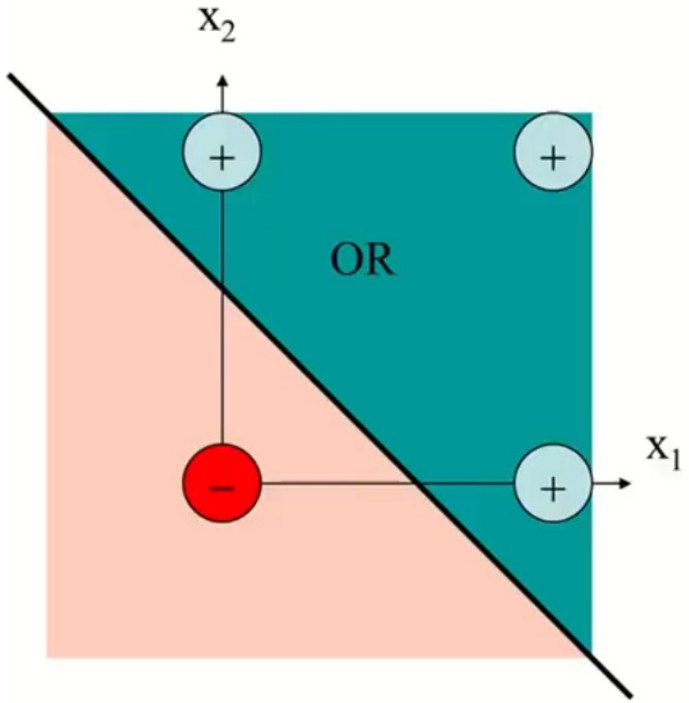
input x1	input x2	output
0	0	0
0	1	1
1	0	1
1	1	0



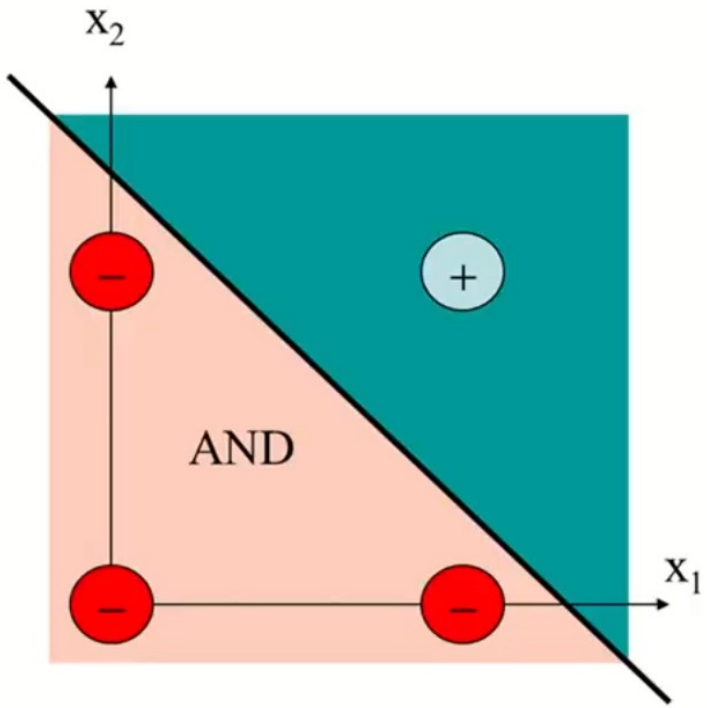
Boolean XOR
Value = 1 iff x1 ≠ x2

Perceptron

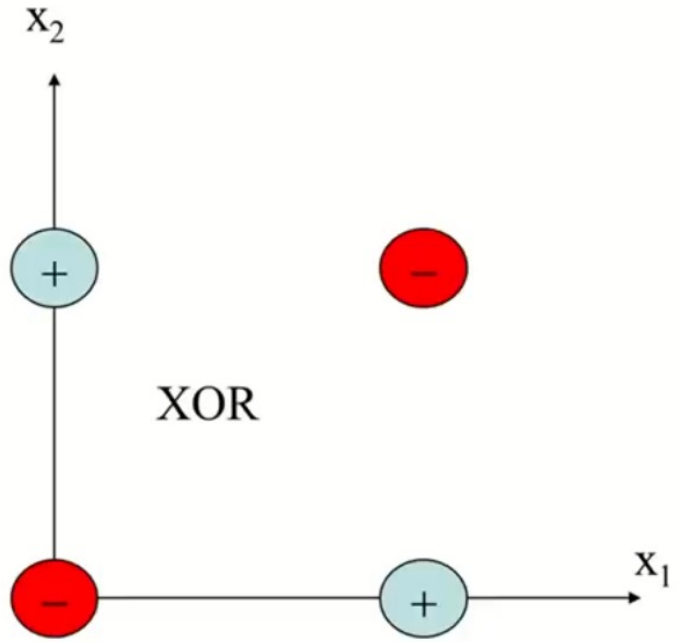
- Single layer perceptron



Boolean OR



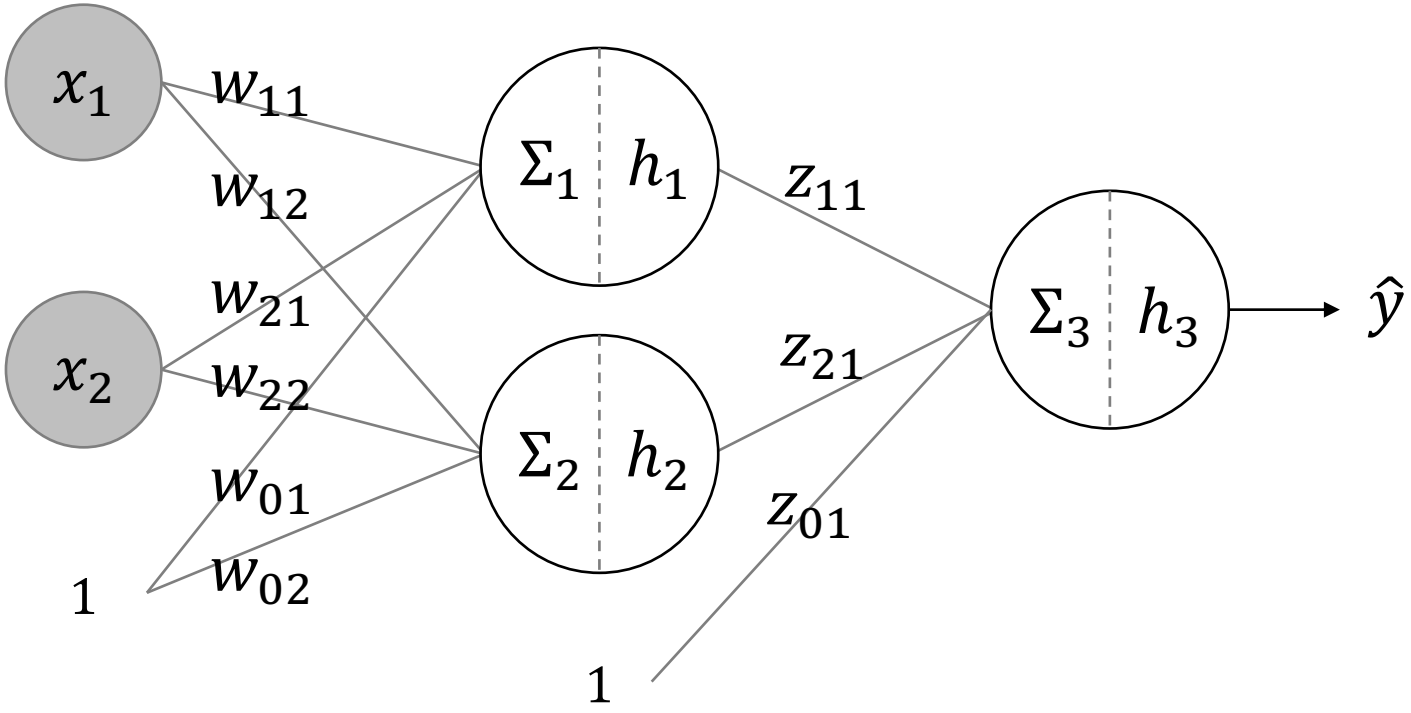
Boolean AND



Boolean XOR

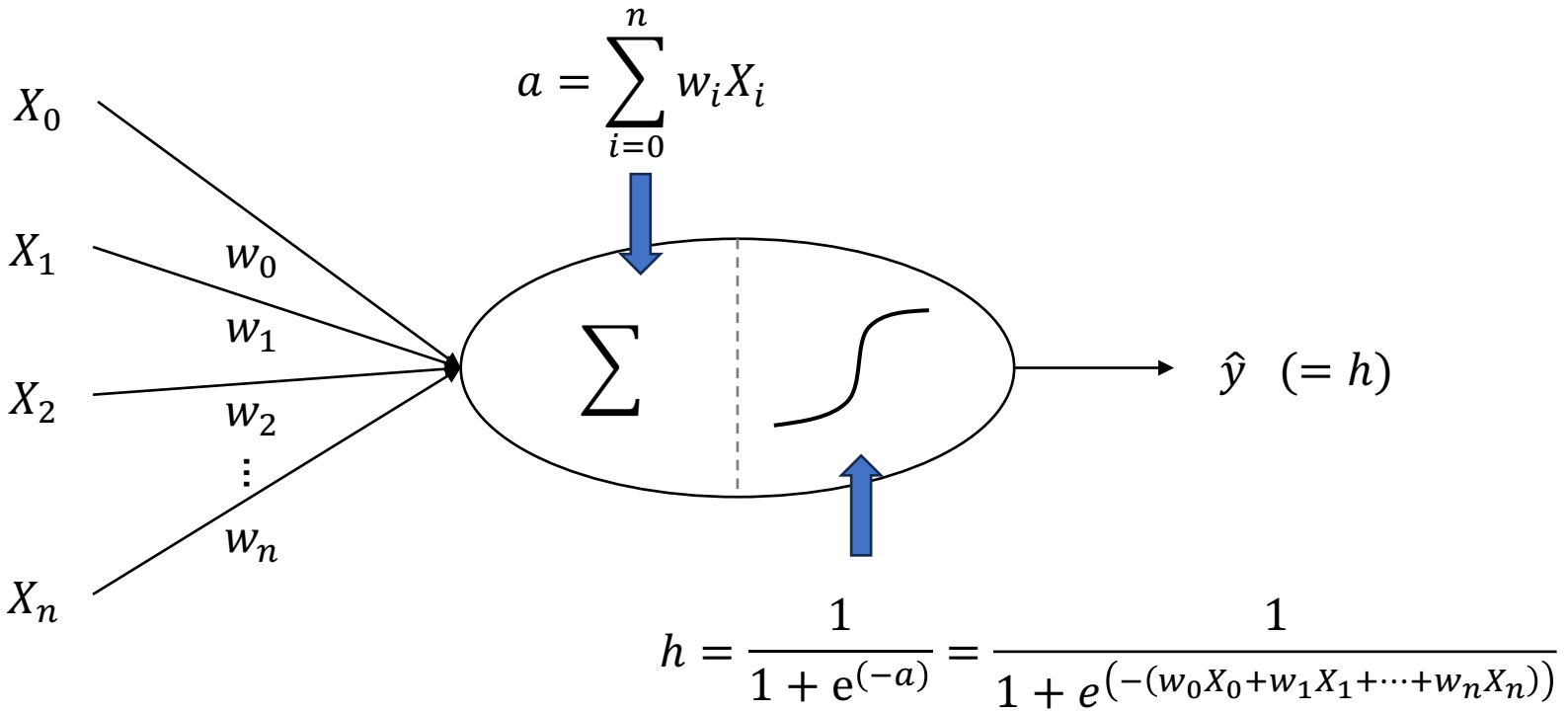
Multi layer perceptron

- 2-layer perceptron
  - 두 개의 perceptron을 결합
    - 두 개의 입력변수와 한개의 출력변수를 갖는 구조



RECALL: 비선형 결합

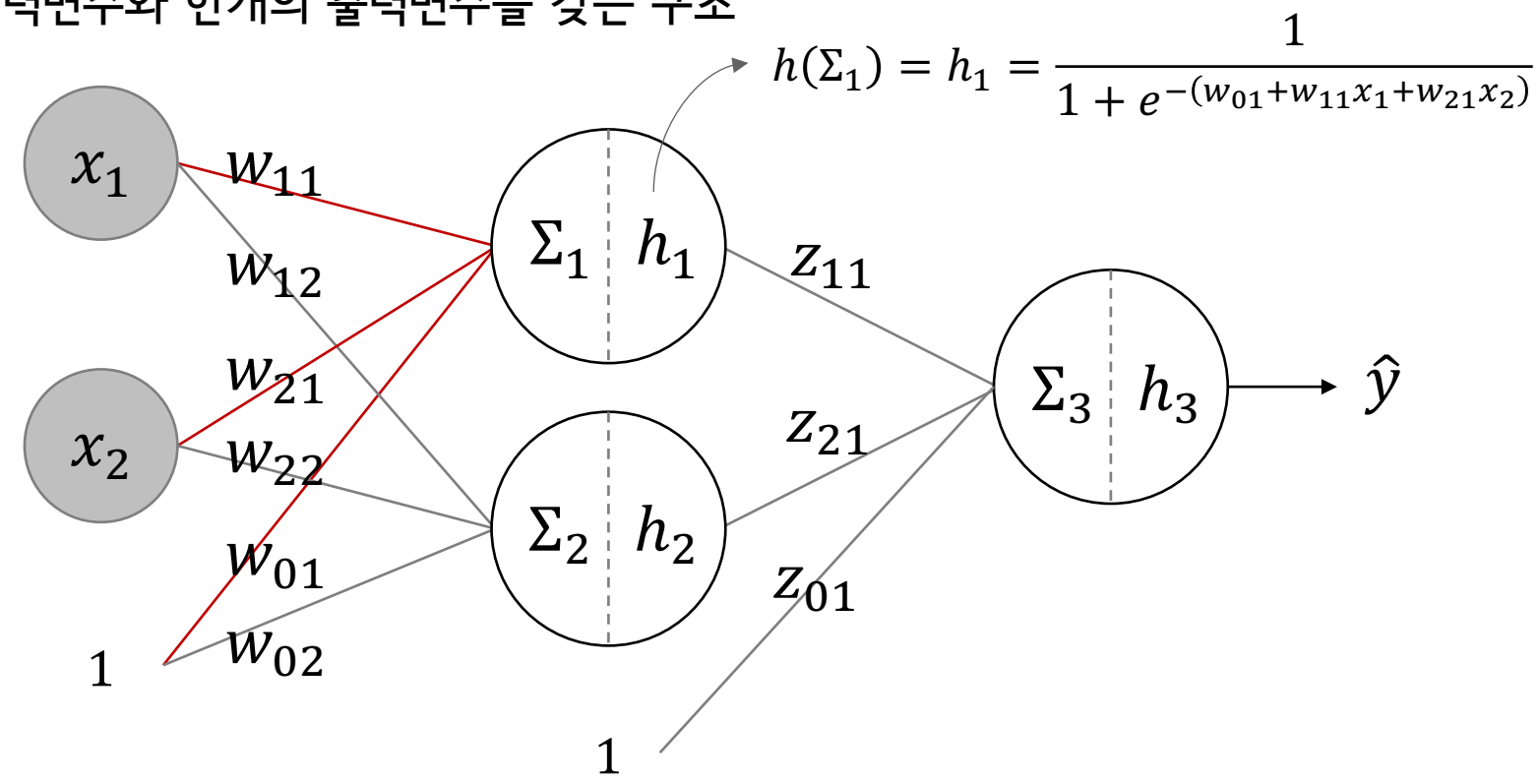
- 비선형결합
  - 1) 입력변수의 선형결합
  - 2) 선형결합 값의 비선형 변환 (non-linear transformation)



Multi layer perceptron

- 2-layer perceptron
  - 두 개의 perceptron을 결합
    - 두 개의 입력변수와 한개의 출력변수를 갖는 구조

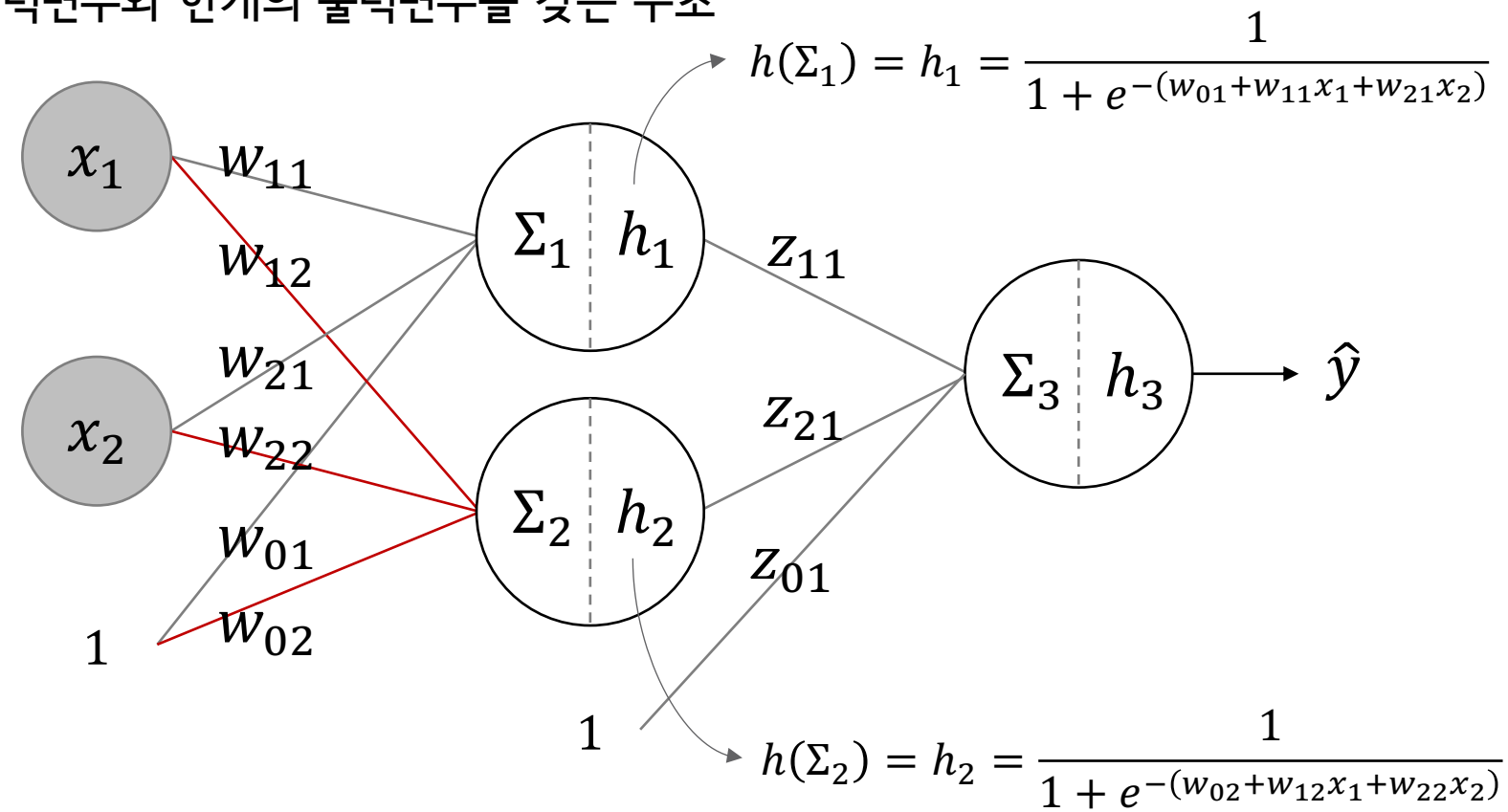
$$h = \frac{1}{1 + \exp(-\Sigma)}$$



Multi layer perceptron

- 2-layer perceptron
  - 두 개의 perceptron을 결합
    - 두 개의 입력변수와 한개의 출력변수를 갖는 구조

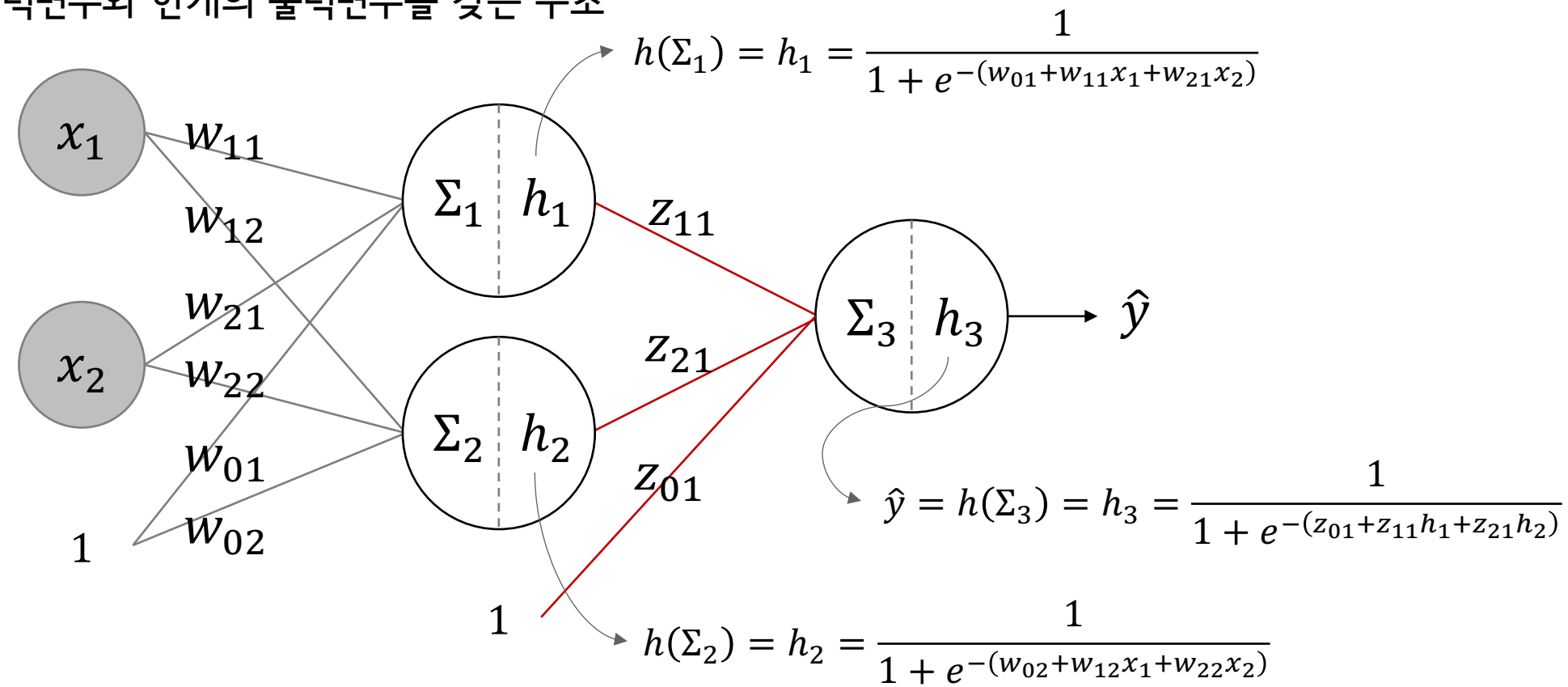
$$h = \frac{1}{1 + \exp(-\Sigma)}$$



Multi layer perceptron

- 2-layer perceptron
  - 두 개의 perceptron을 결합
    - 두 개의 입력변수와 한개의 출력변수를 갖는 구조

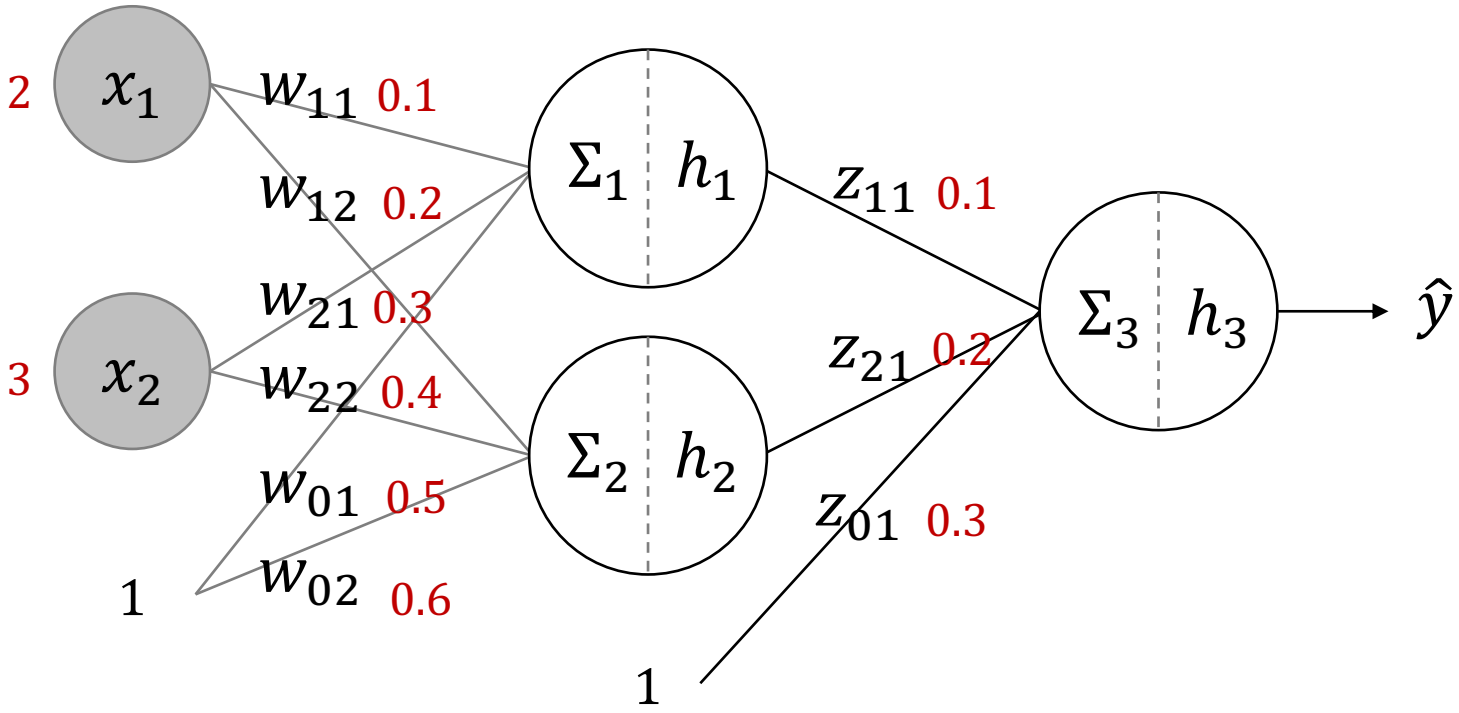
$$h = \frac{1}{1 + \exp(-\Sigma)}$$



Multi layer perceptron

- 예제

$$h = \frac{1}{1 + \exp(-\Sigma)}$$



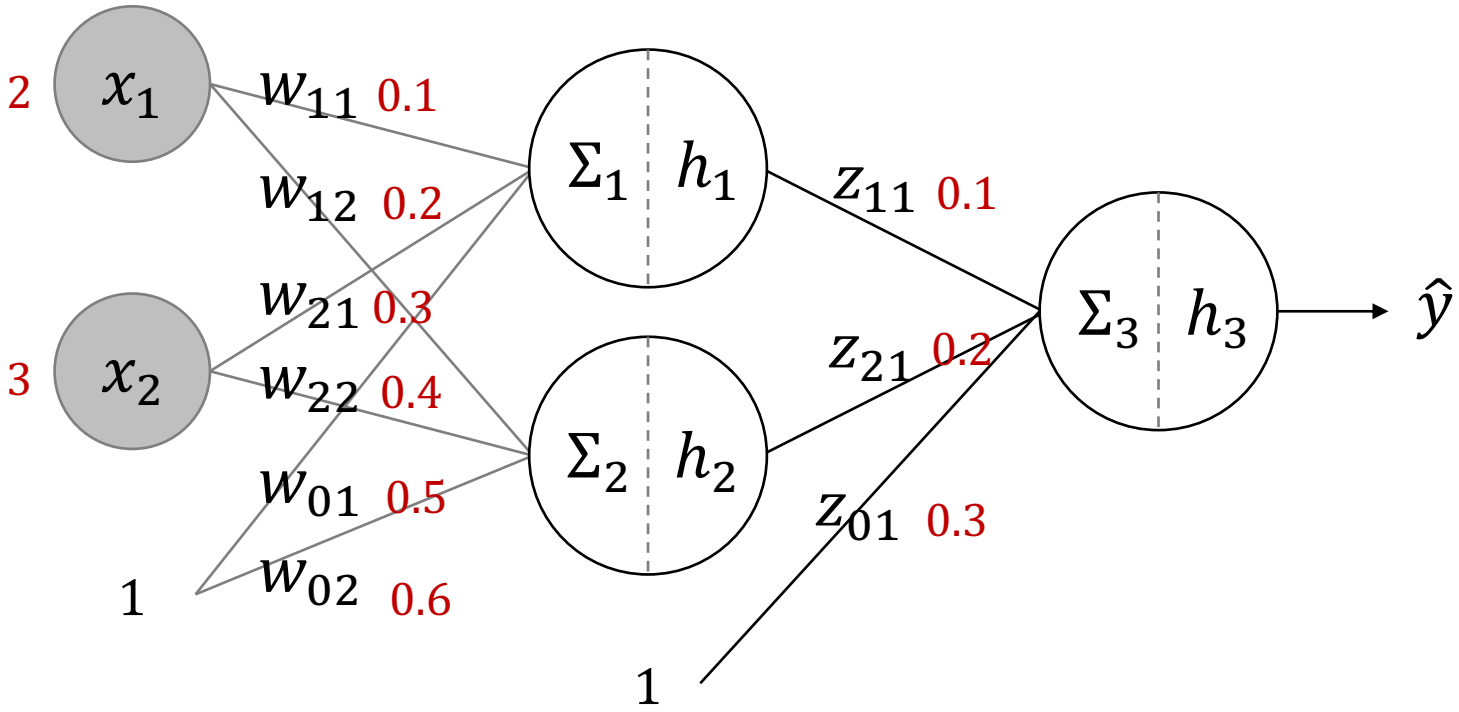
$$\Sigma_1 = 2 * 0.1 + 3 * 0.3 + 1 * 0.5 = 1.6 \qquad h_1 = \frac{1}{1 + e^{-1.6}} = 0.472$$

$$\Sigma_2 = 2 * 0.2 + 3 * 0.4 + 1 * 0.6 = 2.2 \qquad h_2 = \frac{1}{1 + e^{-2.2}} = 0.659$$

Multi layer perceptron

- 예제

$$h = \frac{1}{1 + \exp(-\Sigma)}$$



$$h_1 = \frac{1}{1 + e^{-1.6}} = 0.472$$

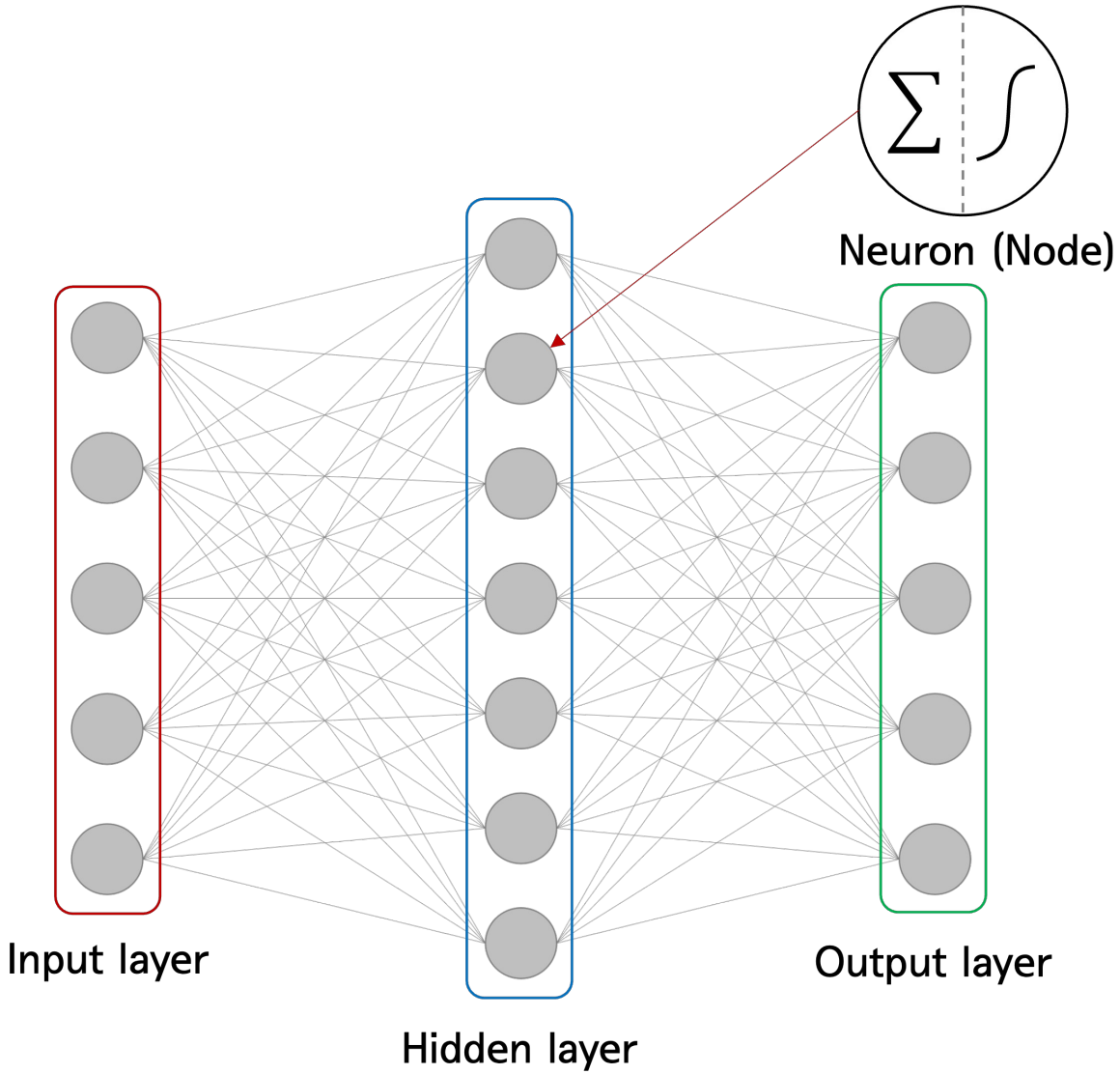
$$h_2 = \frac{1}{1 + e^{-2.2}} = 0.659$$

$$\Sigma_3 = 0.1 * 0.474 + 0.2 * 0.659 + 0.3 * 1 = 0.432$$

$$h_3 = \frac{1}{1 + e^{-0.432}} = 0.304$$

MLP의 구조

- MLP ➔ 인공신경망 (Artificial Neural Network) ➔ Deep Neural Network (DNN)



입력층 (input layer)

- 입력변수의 값이 들어오는 곳
- 입력 변수의 수 == 입력 노드의 수

은닉층 (hidden layer)

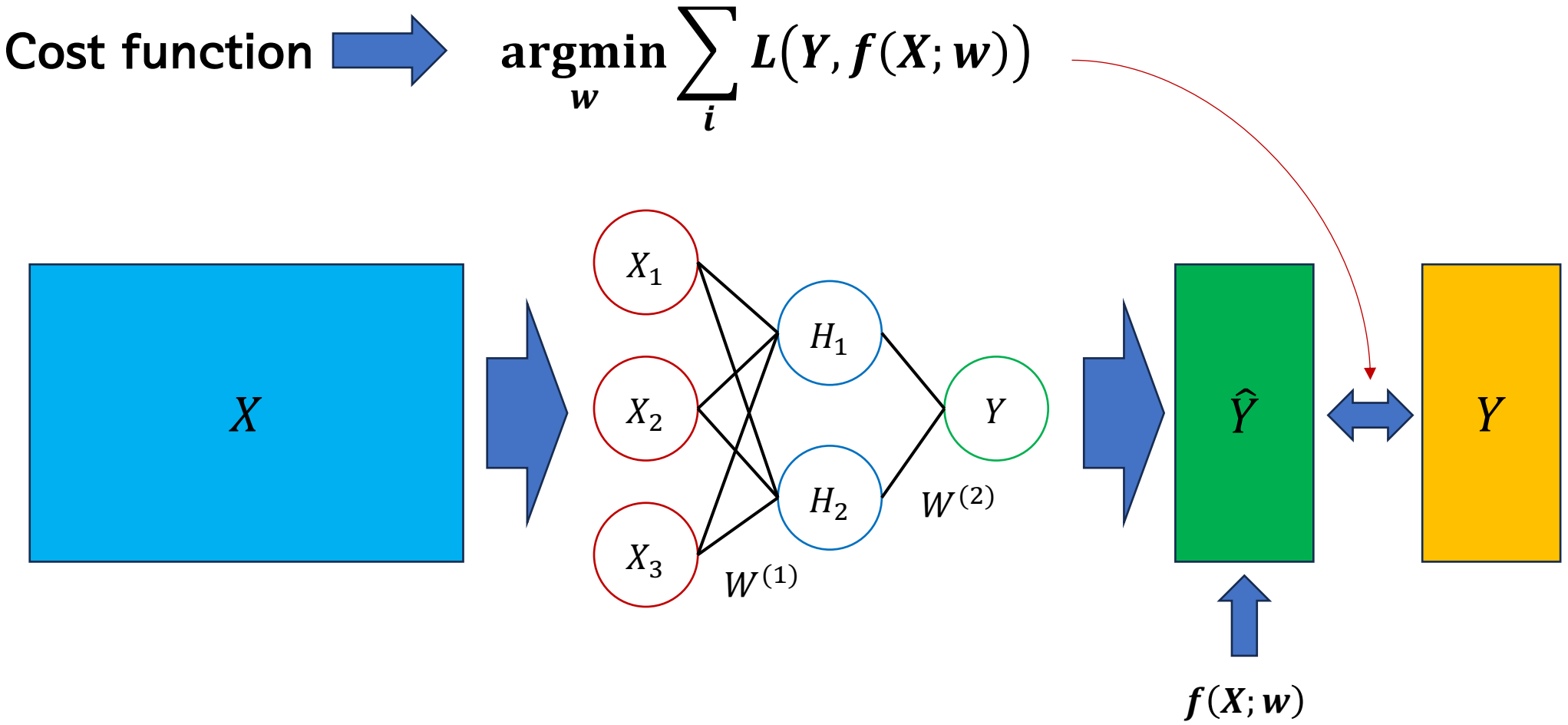
- 은닉층에는 다수 노드 포함 가능
- 다수의 은닉층을 형성 가능

출력층 (output layer)

- (범주형) 출력노드의 수 = 출력변수의 범주 개수
- (연속형) 출력노드의 수 = 출력변수의 개수 (1개)

뉴럴네트워크 모델의 학습 방법

- 개념 1) cost function: 뉴럴네트워크 모델로부터 나온 Y값과 실제 Y값의 차이를 최소로 하는 가중치를 찾자



뉴럴네트워크 모델의 학습 방법

- 개념 1) Cost function – 수치형 데이터의 예측 문제 (regression)
  - 예측값과 실제값의 차이를 측정하는 방법: MSE (mean squared error)
  - $L = \frac{1}{n}\sum_{i=1}^{n}(o_i - t_i)^2$
  - o_i: 예측값, t_i: 실제값

예측값	실제값
11	10
19	20
30	30
43	40
47	50

$$L = \frac{1}{5}\sum_{i=15}(y_i - \hat{y}_i)^2$$

$$= \frac{1}{5}[(10-11)^2 + (20-19)^2 + (30-30)^2 + (40-43)^2 + (50-47)^2] = 4$$

뉴럴네트워크 모델의 학습 방법

- 개념 1) Cost function – 범주형 데이터의 분류 문제 (classification, softmax problem)
  - 예측값과 실제값의 확률 차이를 측정하는 방법: cross entropy
  - $L = -\sum_i t_i \log p_i$
  - p_i: 예측값, t_i: 실제값

예측값

Class 1	Class 2	Class 3
0.3	0.2	0.5
0.1	0.8	0.1
0.6	0.2	0.2
0.1	0.5	0.4

실제값

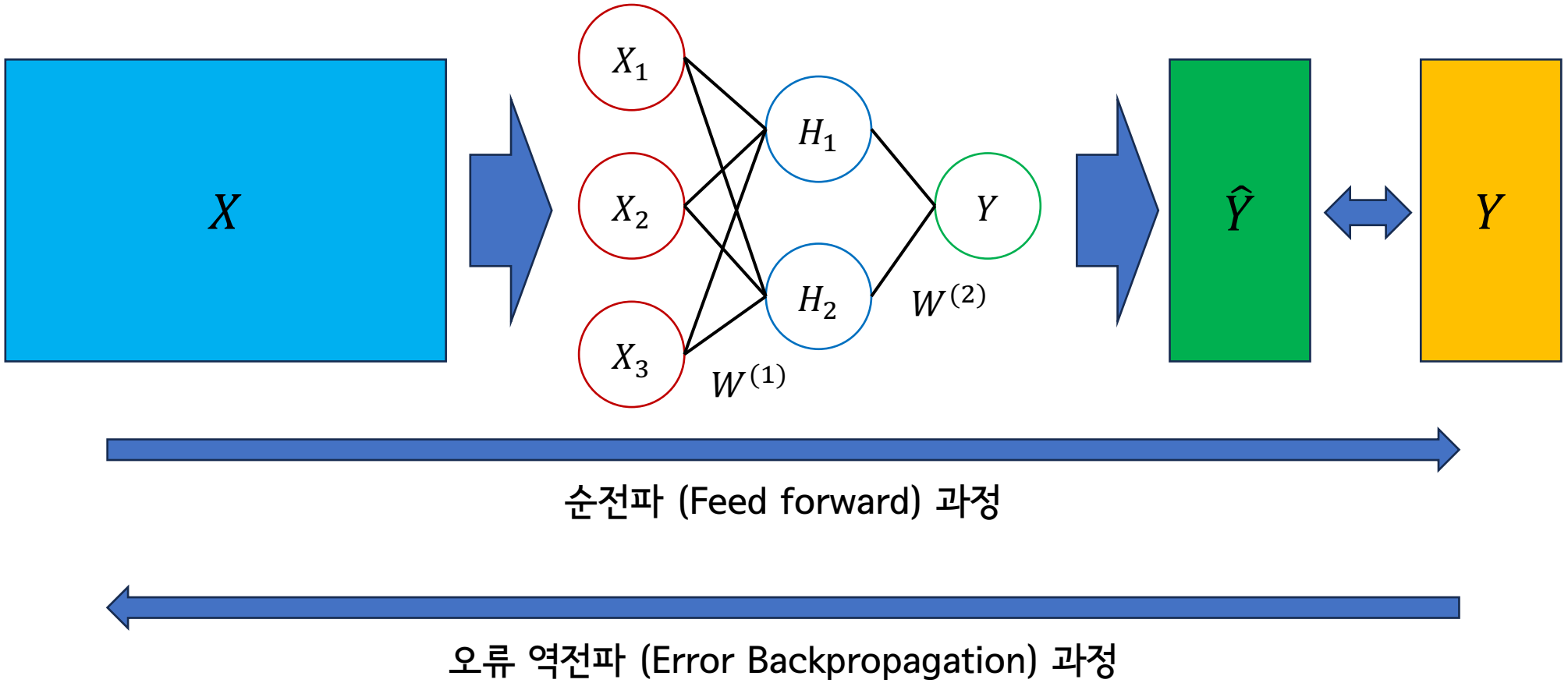
Class 1	Class 2	Class 3
1	0	0
0	1	0
1	0	0
0	0	1

$$L = -\sum_i t_i \log p_i$$

$$= -[\ln(0.3) + \ln(0.8) + \ln(0.6) + \ln(0.4)] = 2.85$$

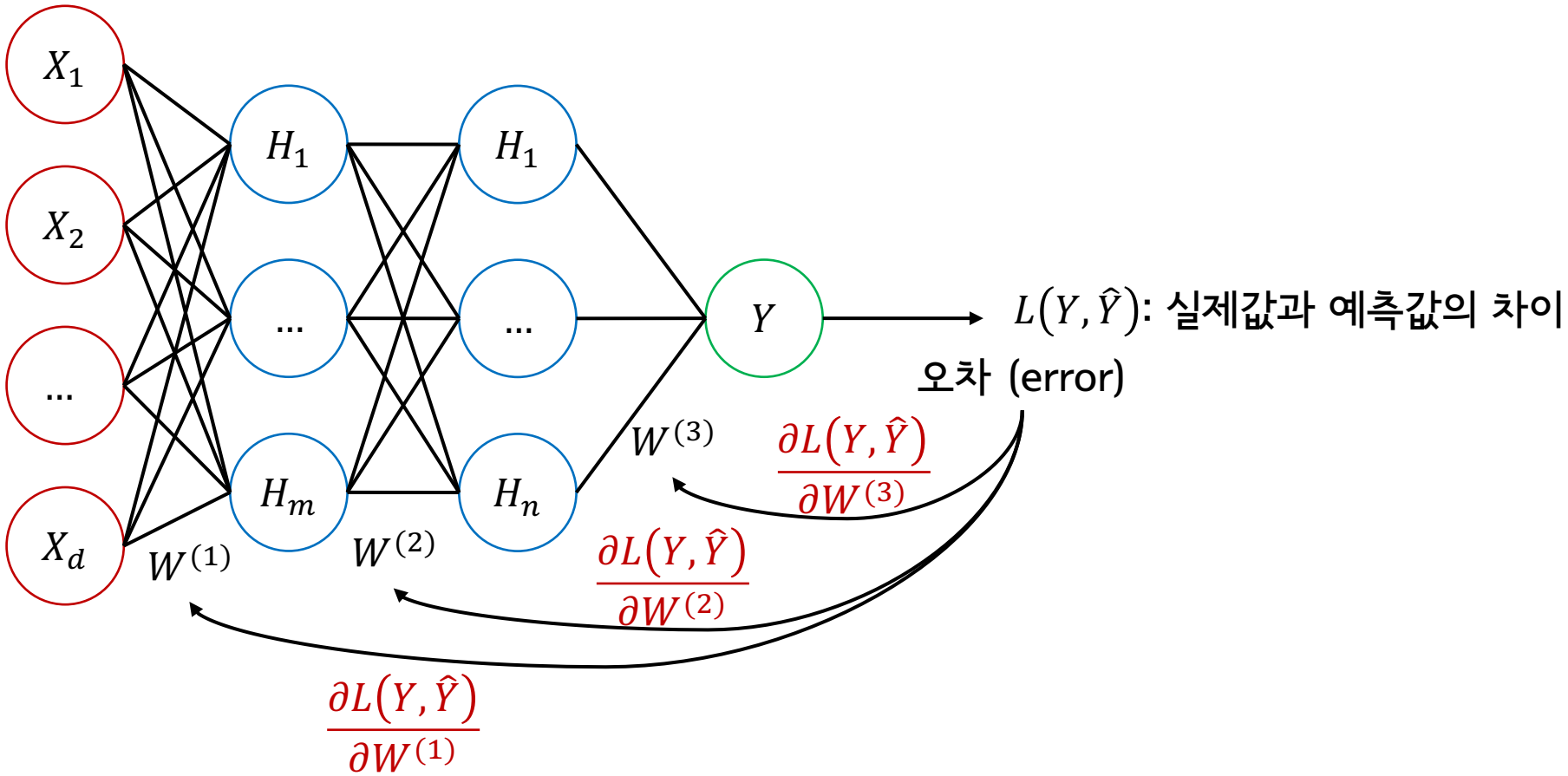
뉴럴네트워크 모델의 학습 방법

- 개념 2) 뉴럴네트워크 모델의 오류 역전파 (error backpropagation)



뉴럴네트워크 모델의 학습 방법

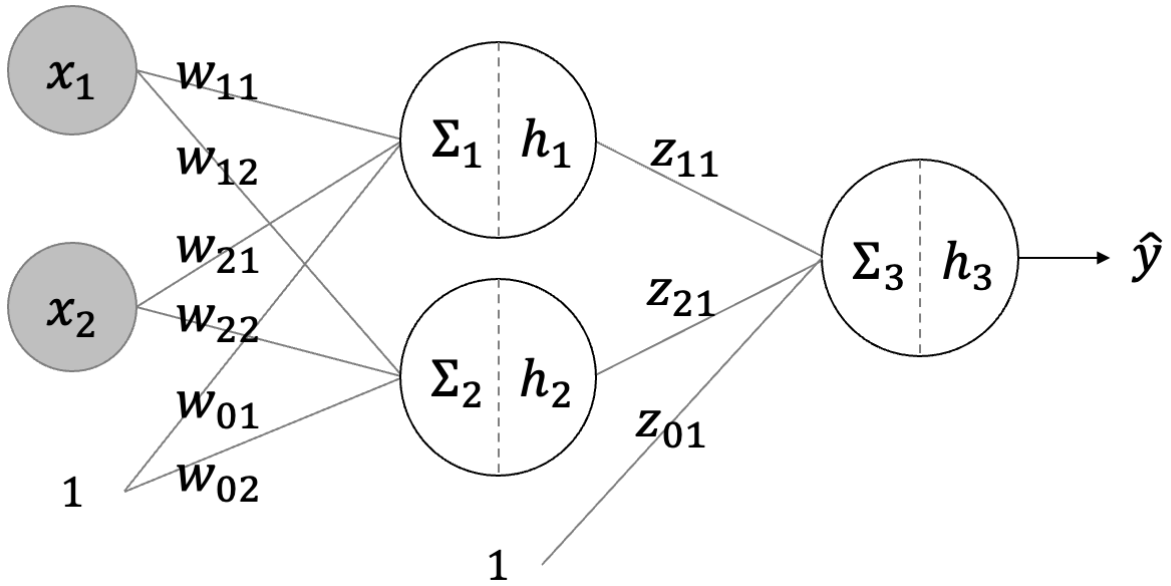
- 개념 2) 뉴럴네트워크 모델의 오류 역전파 (error backpropagation)



$$W_{new} = W_{old} \pm error$$

뉴럴네트워크 모델의 학습 방법

- 개념 3) 파라미터 (parameter)와 하이퍼파라미터 (hyperparameter)

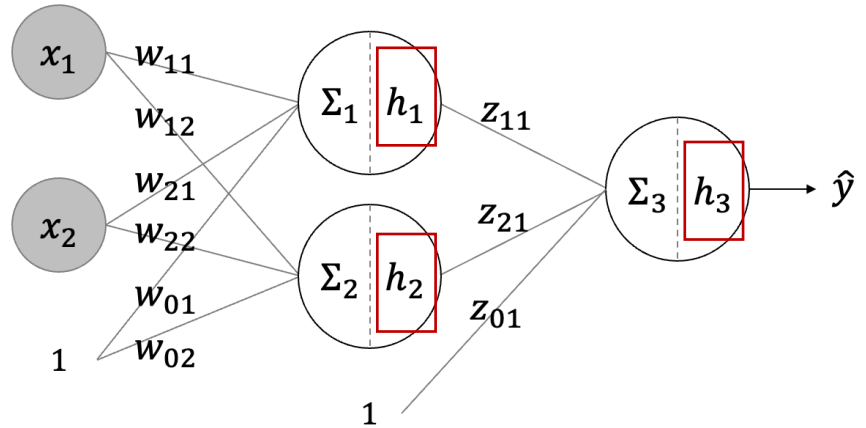


- 파라미터 (Parameters)
  - 층 간 노드를 연결하는 가중치 ($w_{11}, w_{12}, \ldots, z_{21}$) ➔ 알고리즘으로 결정

- 하이퍼파라미터 (Hyperparameters)
  - 은닉층 개수, 은닉노드 개수, acivation function, optimizer, learning rate, etc. ➔ 사용자가 임의로 결정

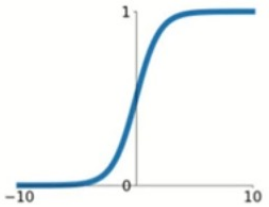
뉴럴네트워크 모델의 학습 방법

- 개념 4) 활성화 함수 (activation function)
  - 선형결합을 비선형 형태로 변환하는 과정
    - 데이터로부터 풍부한 특징정보를 획득하는 가장 기초적인 방법



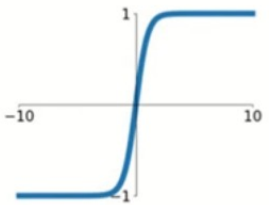
Sigmoid

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



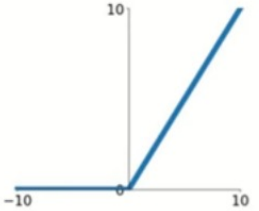
tanh

$$\tanh(x)$$



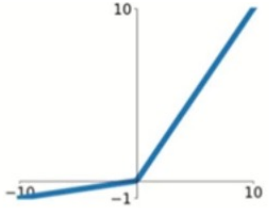
ReLU

$$\max(0, x)$$



Leaky ReLU

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

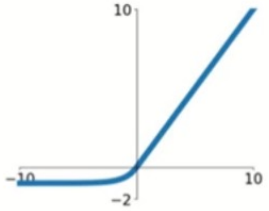


Maxout

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

ELU

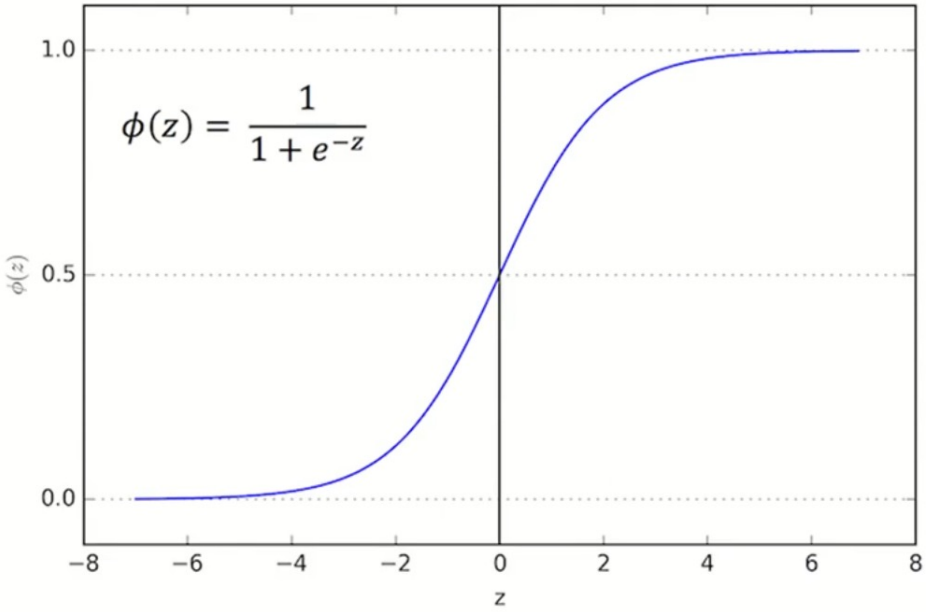
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



뉴럴네트워크 모델의 학습 방법

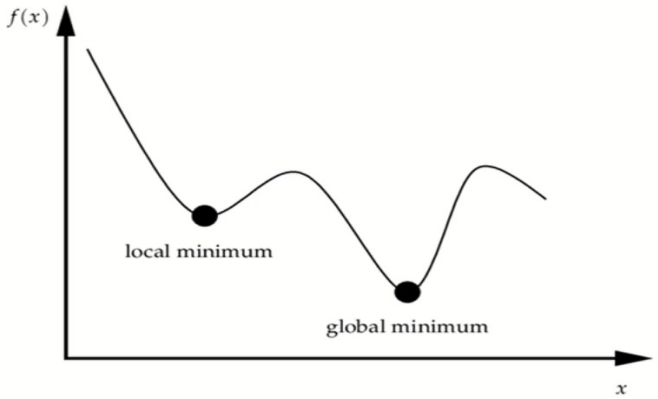
- 개념 4) 활성화 함수 (activation function)
  - Sigmoid function
    - logistic function, squashing function
    - large input ➔ small output
    - Output range: 0 ~ 1
    - input 값에 대해 단조 증가(감소)하는 형태
    - 미분 결과를 output의 함수로 표현 가능
      - ➔ gradient learning method에 유용하게 사용

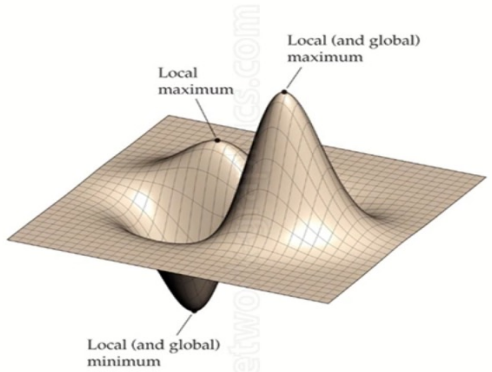
$$\frac{d\phi(z)}{dz} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = \phi(z)\big(1-\phi(z)\big)$$



뉴럴네트워크 모델의 학습 방법

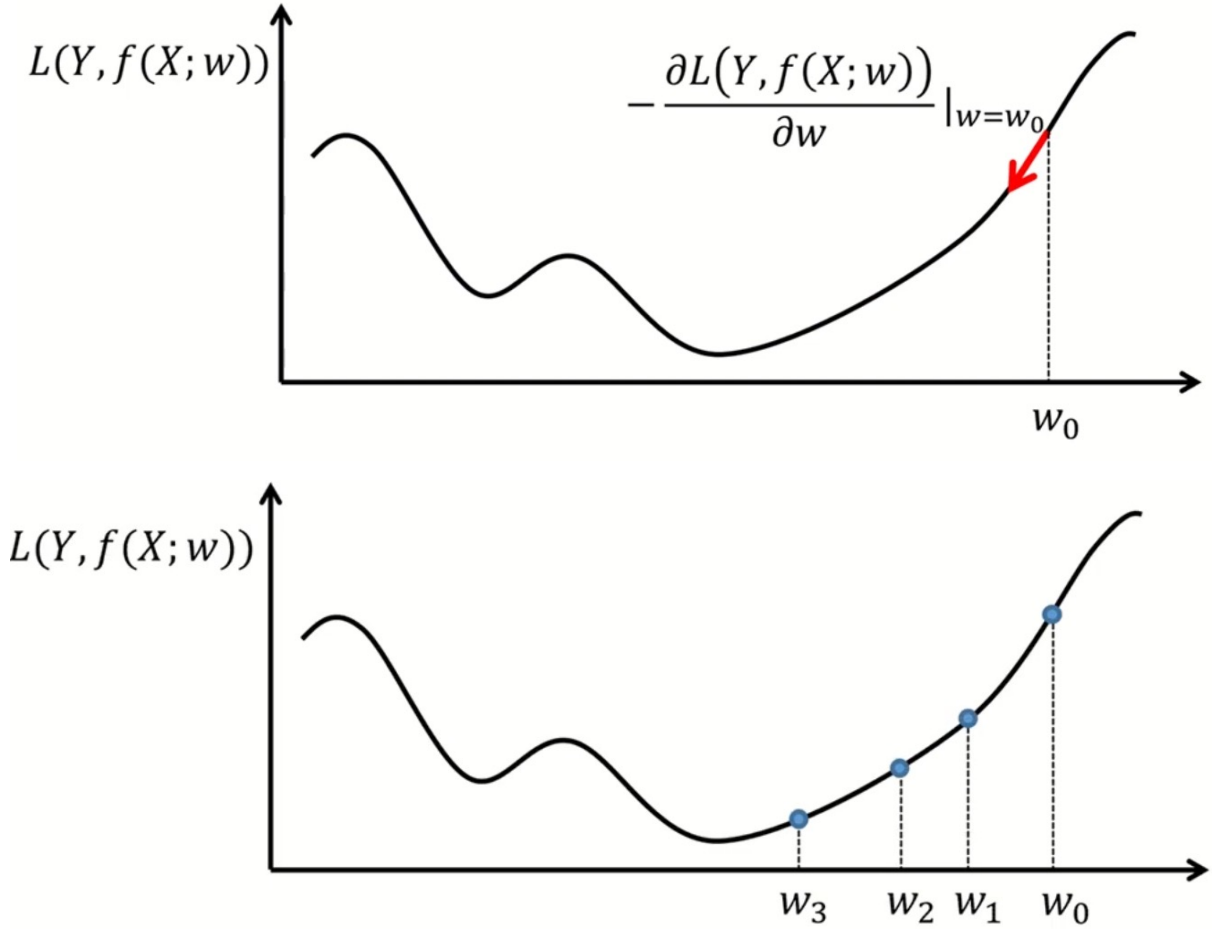
- 개념 5) Gradient descent (optimizer)
  - Gradient (경사): 함수의 기울기
  - Optimization (최적화): 함수의 최솟값 혹은 최댓값을 찾는 과정
    - “turning points”의 개수는 함수의 차수에 의해 결정
      - 모든 turning point가 최솟값이나 최댓값을 갖지는 않음
    - 최솟값들 중 가장 작은 최솟값: 전역 최솟값 (global minimum)
    - 지역적 최솟값: 지역 최솟값 (local minimum)



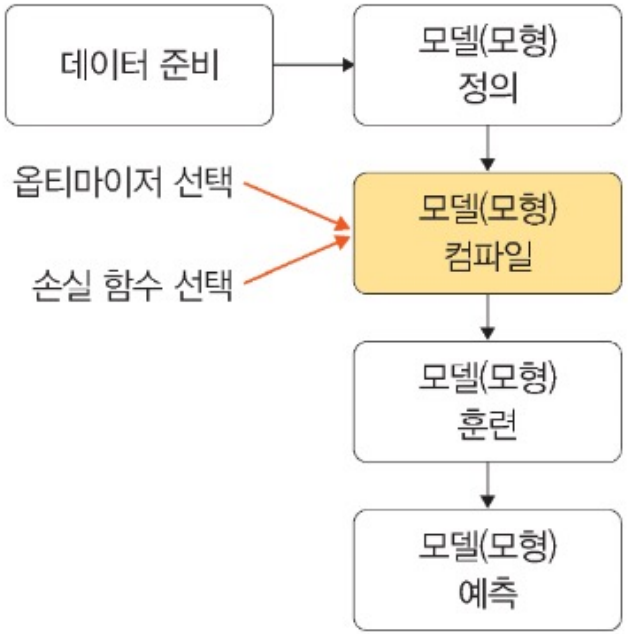


뉴럴네트워크 모델의 학습 방법

- 개념 5) Gradient descent (optimizer)
  - Gradient descent method



뉴럴네트워크 모델의 학습 방법



- 개념 5) Gradient descent (optimizer)
  - Optimizer
    - 최적화를 수행하는 알고리즘 ➔ 학습속도 향상, 성능 향상을 목표로 함
    - 다양한 optimizer
      - gradient descent (GD), stochastic gradient descent (SGD)
        - GD: 전체 학습 데이터 셋에 대해 (1 epoch) 가중치 1번 업데이트
        - SCD: batch에 따라 가중치 업데이트
      - Adagrad
      - Root Mean Square Propagation (RMSProp)
      - Adaptive Moment Estimation (Adam)
        - 진행하던 속도에 관성(momentum)을 주고 그 변화량에 따라 적응적으로 learning rate를 결정

뉴럴네트워크 모델의 학습 방법

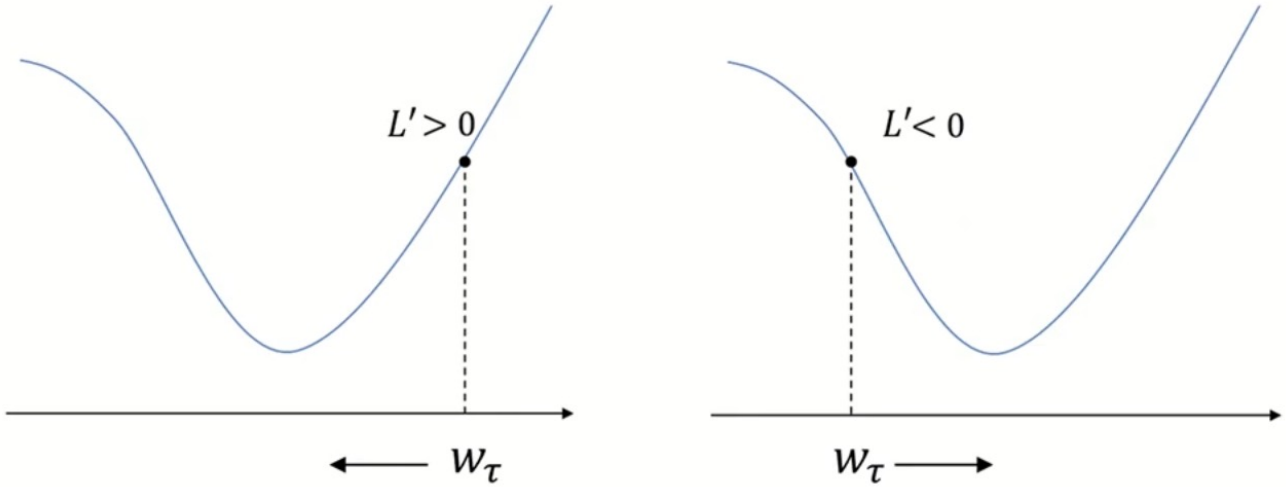
- 개념 6) Learning rate (학습률)

$$w_{\tau+1} = w_\tau - \alpha \cdot L'(w_\tau)$$

$$0 < \alpha < 1$$

좀 더 섬세하고 촘촘히? ➔ Small α

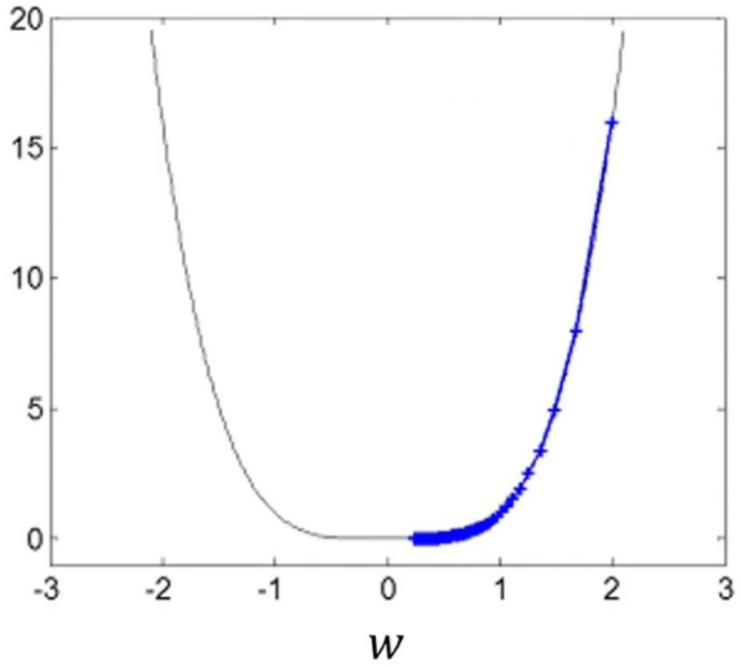
좀 더 빠르게? ➔ Large α



뉴럴네트워크 모델의 학습 방법

- 개념 6) Learning rate (학습률)

$$L(w) = w^4$$



α: learning rate

$w_0 = 2 \qquad \alpha = 0.01$

$w_1 = w_0 - \alpha L'(w_0) = 1.6800$

$w_2 = w_1 - \alpha L'(w_1) = 1.4903$

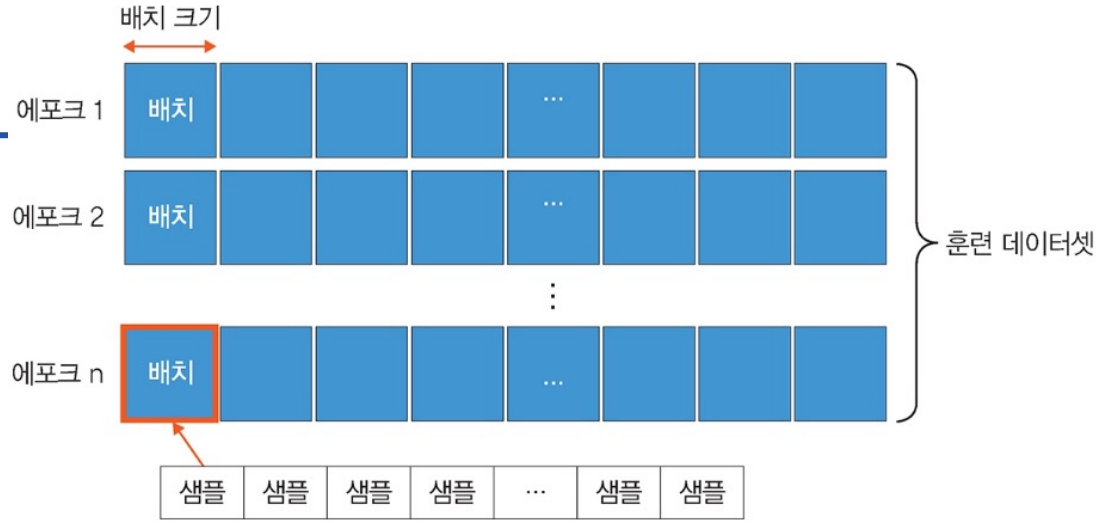
$w_3 = w_2 - \alpha L'(w_2) = 1.3579$

…

$w_{200} = w_{199} - \alpha L'(w_{199}) = 0.2461$

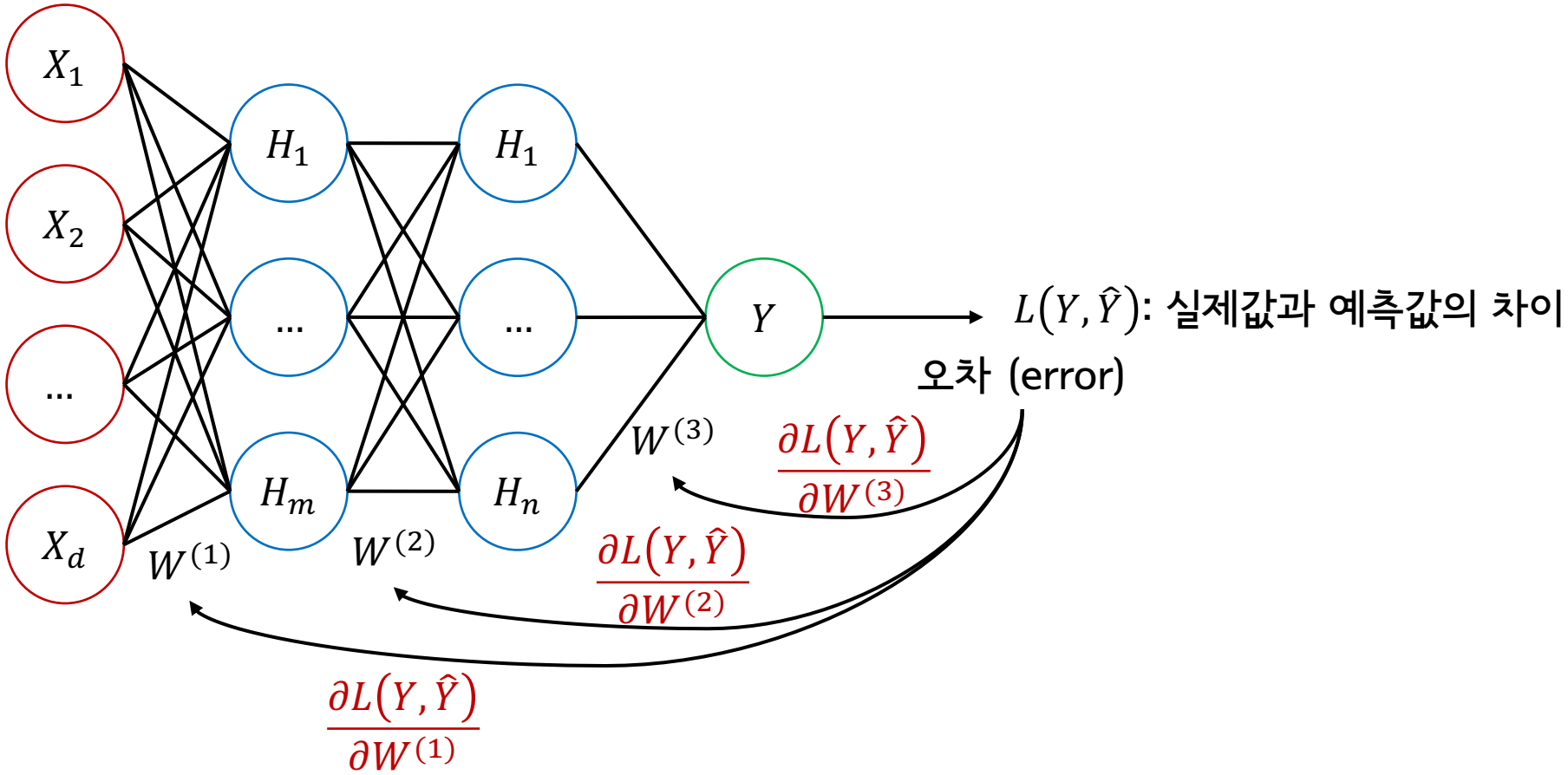
$\alpha = 0.02$ 라면?

뉴럴네트워크 모델의 학습 방법



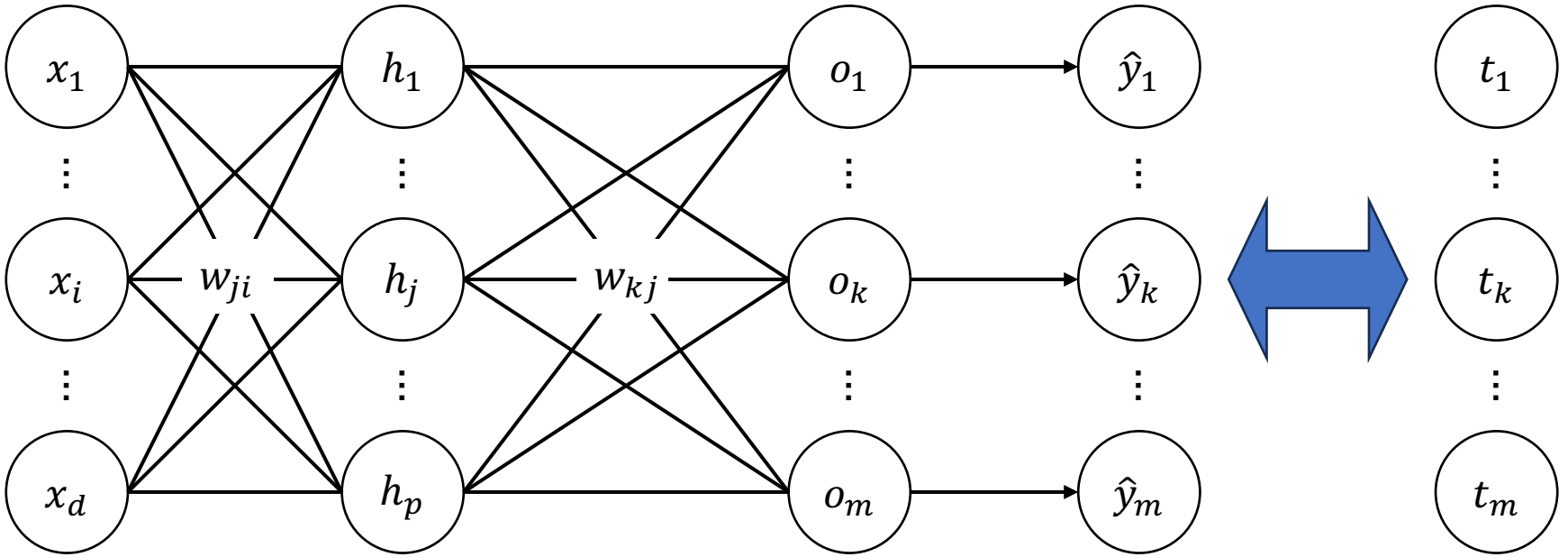
- 개념 7) Batch, epoch, iteration
  - Batch: 일괄적으로 처리되는 집단
    - batch size: 소그룹의 크기
    - 1000개의 학습데이터에 대해 batch size = 64인 경우
      - ➔ 64개씩 묶어서 16개의 batch로 나뉨 (≈ 1000)
  - Epoch: (사전적 의미) 중요한 사건이 일어난 시기/시대
    - 전체 데이터를 한 번 훑는 과정 = 1 epoch
    - epoch = 300 ➔ 전체 데이터를 300번 반복 학습
  - Iteration: 계산 및 처리의 반복
    - 500개의 데이터를 batch size = 10으로 나누었을 때
      - ➔ batch의 개수 50, 1 epoch = 50 iteration (parameter 업데이트 50번 진행)

오류 역전파 알고리즘



$$W_{new} = W_{old} \pm error$$

오류 역전파 알고리즘



입력 X, 출력 Y, 예측 O

$$D_1 = (x_{11}, x_{12}, \dots, x_{1d}, t_{11}, t_{12}, \dots, t_{1m}) \qquad (o_{11}, o_{12}, \dots, o_{1m})$$

$$D_2 = (x_{21}, x_{22}, \dots, x_{2d}, t_{21}, t_{22}, \dots, t_{2m}) \qquad (o_{21}, o_{22}, \dots, o_{2m})$$

$$\dots$$

$$D_N = (x_{N1}, x_{N2}, \dots, x_{Nd}, t_{N1}, t_{N2}, \dots, t_{Nm}) \qquad (o_{N1}, o_{N2}, \dots, o_{Nm})$$

오류 역전파 알고리즘

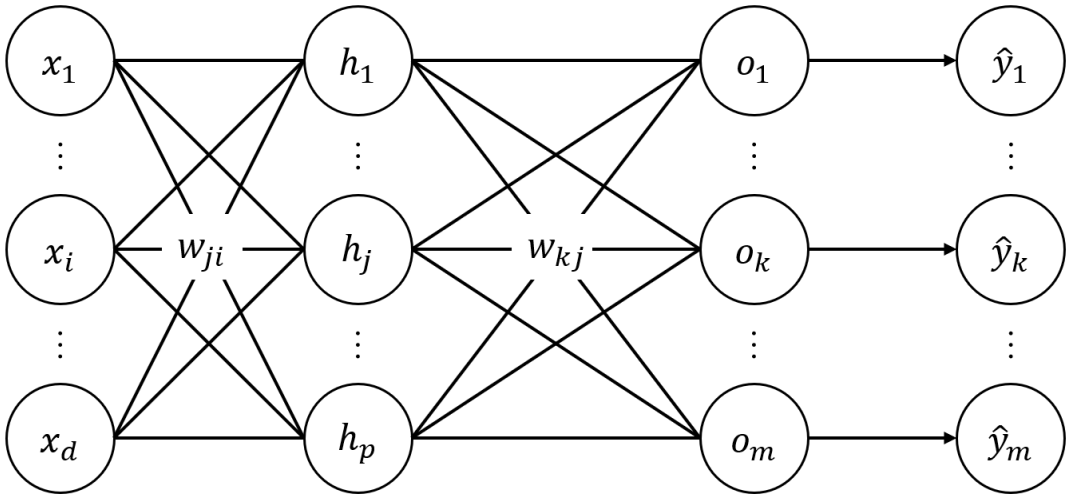
- 목표: 비용함수 $E(w)$를 최소로 하는 w를 찾자 (*Y가 연속형)

$$E(w) = \sum_{n=1}^{N} E_n(w) \qquad E_n(w) = \frac{1}{2}\sum_{n=1}^{m}(t_{nk} - o_{nk})^2$$

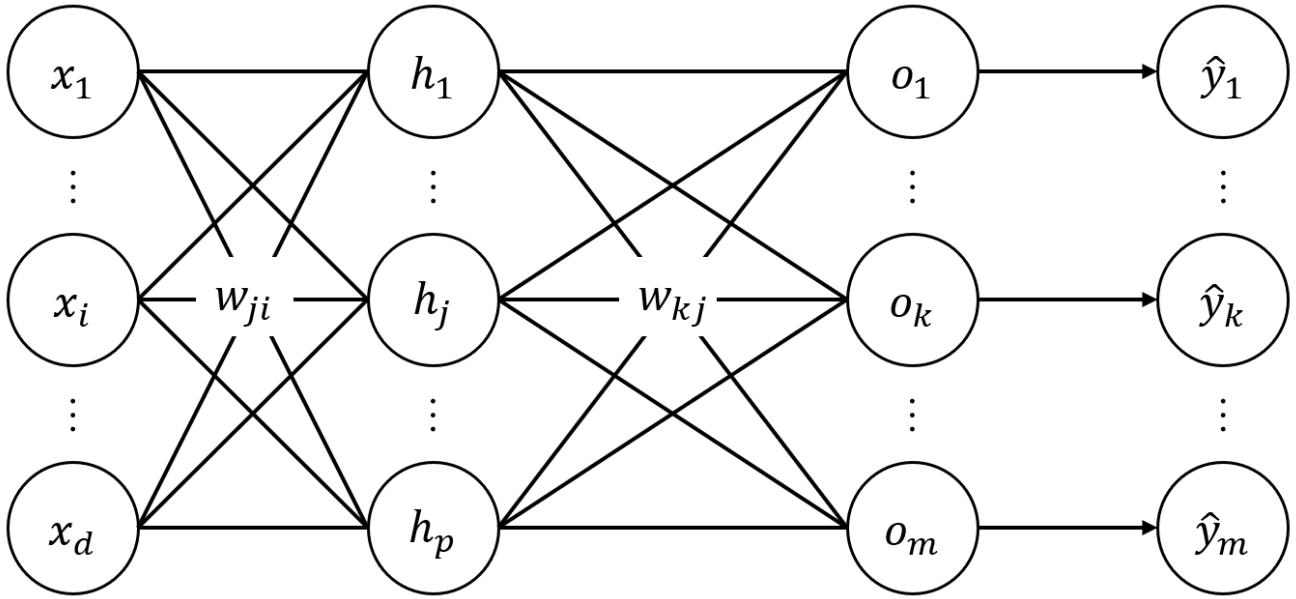
출력층-은닉층

- w에 대해서 미분 가능 ➔ $\frac{\partial E_n}{\partial w_{kj}}$와 $\frac{\partial E_n}{\partial w_{ji}}$를 구할 수 있음

은닉층-입력층



오류 역전파 알고리즘



$$h_j = \frac{1}{1 + \exp\left(-\left(w_{j0} + \sum_{i=1}^{d} w_{ji} x_i\right)\right)}$$

$$o_k = \frac{1}{1 + \exp\left(-\left(w_{k0} + \sum_{j=1}^{p} w_{kj} h_j\right)\right)}$$

오류 역전파 알고리즘

$$E_n(w) = \frac{1}{2}\sum_{n=1}^{m}(t_{nk} - o_{nk})^2$$

$$o_k = \frac{1}{1 + \exp\left(-\left(w_{k0} + \sum_{j=1}^{p} w_{kj} h_j\right)\right)}$$

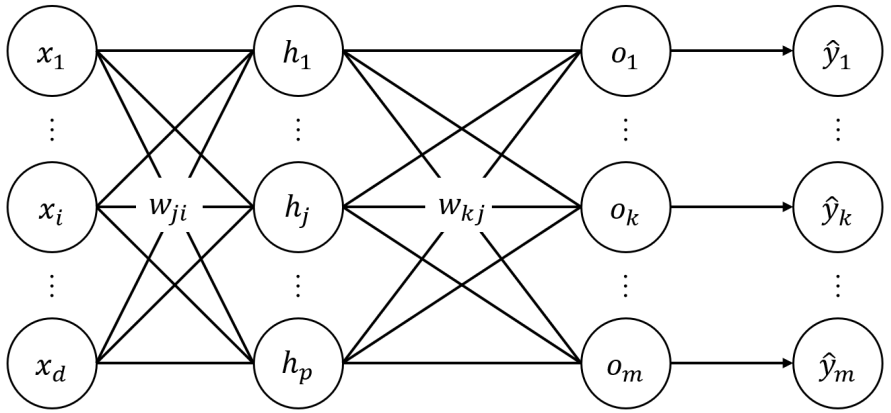
$$h_j = \frac{1}{1 + \exp\left(-\left(w_{j0} + \sum_{i=1}^{d} w_{ji} x_i\right)\right)}$$

오류 역전파 알고리즘

$$E_n(w) = \frac{1}{2}\sum_{n=1}^{m}\left(t_k - \left(\frac{1}{1+\exp\left(-\left(w_{k0} + \sum_{j=1}^{p} w_{kj}\left(\frac{1}{1+\exp\left(-\left(w_{j0} + \sum_{i=1}^{d} w_{ji}x_i\right)\right)}\right)\right)\right)}\right)\right)$$

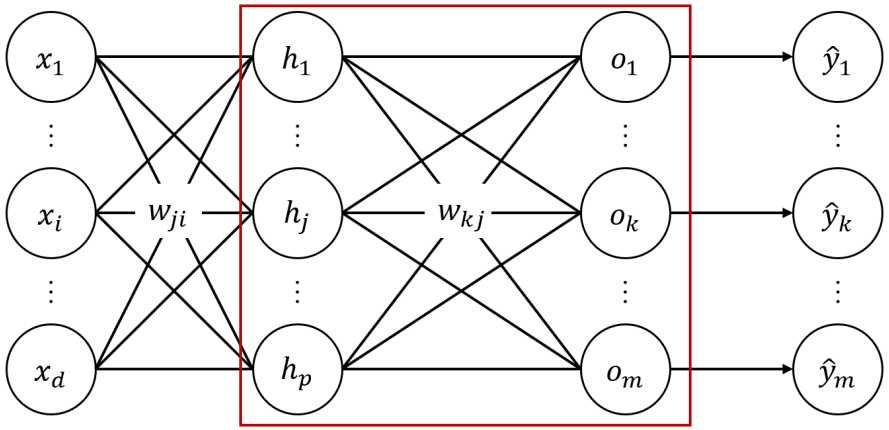
미분을 바로 하기에 너무 복잡

- 1) 출력층-은닉층 사이
- 2) 은닉층-입력층 사이



오류 역전파 알고리즘

- 1) 출력층-은닉층 사이



$$net_k = h_1 w_{k1} + h_2 w_{k2} + \cdots + h_j w_{kj} + \cdots + h_p w_{kp} + w_{k0}$$

- h_j: 은닉층의 j번째 노드값
- $o_k = sigmoid(net_k) = \frac{1}{1+\exp(-net_k)}$
- $E_n(w) = \frac{1}{2}\sum_{n=1}^{m}(t_k - o_k)^2$

경사하강법을 이용하여 w_{kj}를 계산하기 위해 $\Delta w_{kj} = -\alpha \frac{\partial E_n}{\partial w_{kj}}$

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E_n}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E_n}{\partial o_k}\frac{\partial o_k}{\partial net_k}h_j$$

오류 역전파 알고리즘

- 1) 출력층-은닉층 사이

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial o_k}\frac{\partial o_k}{\partial net_k}h_j$$

$$\frac{\partial E_n}{\partial o_k} = \frac{\partial}{\partial o_k}\frac{1}{2}\sum_{k=1}^{m}(t_k - o_k)^2 = \frac{\partial}{\partial o_k}\frac{1}{2}(t_k - o_k)^2 = -(t_k - o_k)$$

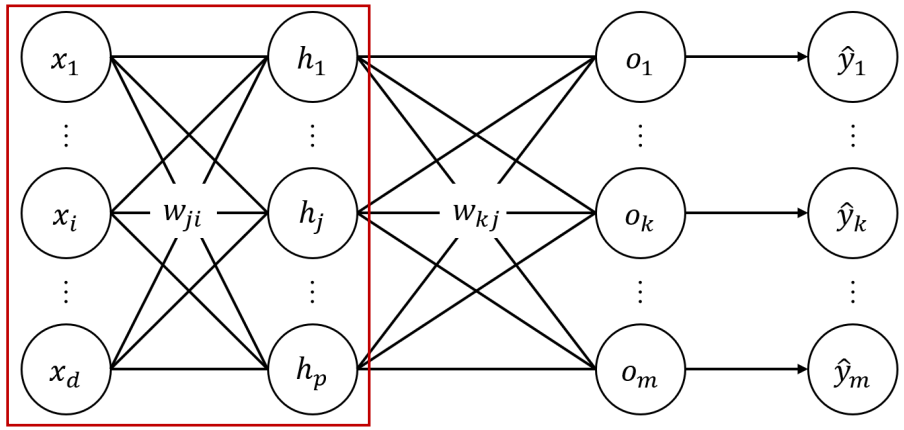
$$\frac{\partial o_k}{\partial net_k} = \frac{\partial}{\partial net_k}\frac{1}{1+\exp(-net_k)} = \frac{-(-\exp(-net_k))}{(1+\exp(-net_k))^2} = \frac{\exp(-net_k)}{(1+\exp(-net_k))^2}$$

$$= \frac{1}{1+\exp(-net_k)}\frac{\exp(-net_k)}{1+\exp(-net_k)} = o_k(1-o_k)$$

$$\Delta w_{kj} = -\alpha\frac{\partial E_n}{\partial w_{kj}} = \alpha(t_k - o_k)o_k(1-o_k)h_j$$

오류 역전파 알고리즘

- 2) 은닉층-입력층 사이



$$net_j = x_1 w_{j1} + x_2 w_{j2} + \cdots + x_i w_{ji} + \cdots + x_d w_{jd} + w_{j0}$$

- $h_j = sigmoid(net_j) = \frac{1}{1+\exp(-net_j)}$

$$net_k = h_1 w_{k1} + h_2 w_{k2} + \cdots + h_j w_{kj} + \cdots + h_p w_{kp} + w_{k0}$$

- $o_k = sigmoid(net_k) = \frac{1}{1+\exp(-net_k)}$

$$E_n(w) = \frac{1}{2}\sum_{n=1}^{m}(t_k - o_k)^2$$

경사하강법을 이용하여 w_{ji}를 계산하기 위해 $\Delta w_{ji} = -\alpha \frac{\partial E_n}{\partial w_{ji}}$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_n}{\partial net_j} x_i$$

오류 역전파 알고리즘

- 2) 은닉층-입력층 사이

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial net_j} x_i$$

$$\frac{\partial E_n}{\partial net_j} = \frac{\partial}{\partial net_j} \frac{1}{2} \sum_{k=1}^{m} (t_k - o_k)^2 = \frac{1}{2} \sum_{k=1}^{m} \frac{\partial}{\partial net_j} (t_k - o_k)^2$$

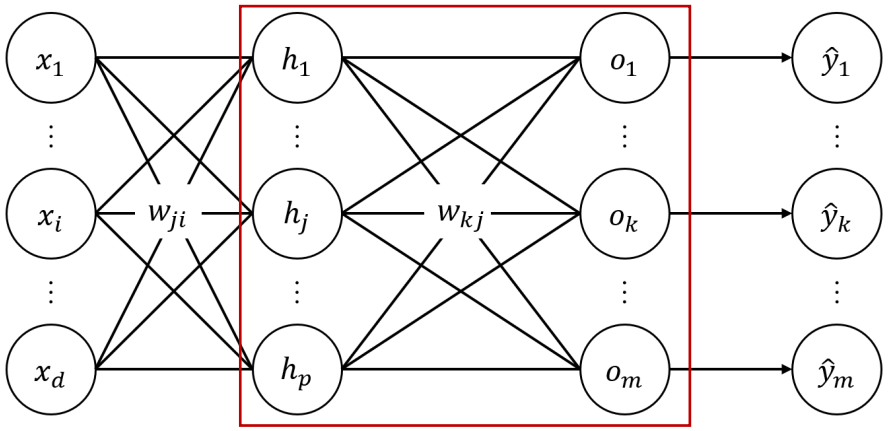
$$= \frac{1}{2} \sum_{k=1}^{m} \frac{\partial h_j}{\partial net_j} \frac{\partial net_k}{\partial h_j} \frac{\partial o_k}{\partial net_k} \frac{\partial (t_k - o_k)^2}{\partial o_k}$$

$$\frac{\partial h_j}{\partial net_j} = h_j(1 - h_j) \qquad \frac{\partial net_k}{\partial h_j} = w_{kj} \qquad \frac{\partial o_k}{\partial net_k} = o_k(1 - o_k) \qquad \frac{\partial (t_k - o_k)^2}{\partial o_k} = -2(t_k - o_k)$$

$$\Delta w_{jh} = -\alpha \frac{\partial E_n}{\partial w_{ji}} = \alpha x_i h_j(1 - h_j) \sum_{k=1}^{m} w_{kj}\, o_k(1 - o_k)(t_k - o_k)$$

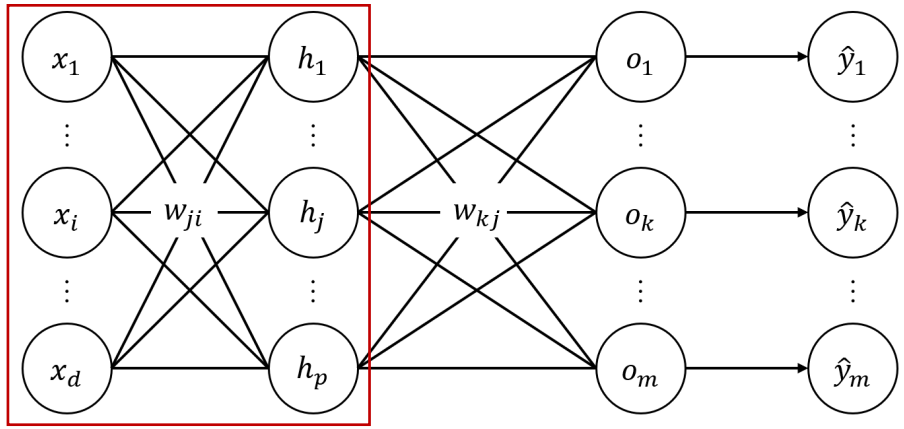
오류 역전파 알고리즘

- 1) 출력층-은닉층 사이



$$\Delta w_{kj} = -\alpha \frac{\partial E_n}{\partial w_{kj}} = \alpha(t_k - o_k)o_k(1 - o_k)h_j$$

- 2) 은닉층-입력층 사이



$$\Delta w_{jh} = -\alpha \frac{\partial E_n}{\partial w_{ji}} = \alpha x_i h_j (1 - h_j) \sum_{k=1}^{m} w_{kj}\, o_k(1 - o_k)(t_k - o_k)$$

오류 역전파 알고리즘

- Step 1: 모든 가중치 w를 임의로 생성
- Step 2: 입력변수 값과 입력층과 은닉층 사이의 w값을 이용하여 은닉노드의 값을 계산
- ➔ 선형 결합 후 activation
- Step 3: 은닉노드의 값과 은닉층과 출력층 사이의 w값을 이용하여 출력노드의 값을 계산
- ➔ 선형 결합 후 activation
- Step 4: 계산된 출력노드의 값과 실제 출력변수의 값의 차이를 줄일 수 있도록 은닉층-출력층 사이의 w값 업데이트

- Step 5: 계산된 출력노드의 값과 실제 출력변수의 값의 차이를 줄일 수 있도록 입력층-은닉층 사이의 w값 업데이트
- Step 6: 에러가 충분히 줄어 들 때 까지 Step2 ~ Step 6반복
- **➔ 오류 역전파 알고리즘 (Error Backpropagation Algorithm, EBPA)**

오류 역전파 알고리즘

- Step 1: 모든 가중치 w를 임의로 생성
- Step 2: 입력변수 값과 입력층과 은닉층 사이의 w값을 이용하여 은닉노드의 값을 계산 $h_j = sigmoid\left(w_{j0} + \sum_{i=1}^{d} w_{ji} x_i\right)$
- ➔ 선형 결합 후 activation
- Step 3: 은닉노드의 값과 은닉층과 출력층 사이의 w값을 이용하여 출력노드의 값을 계산 $o_k = sigmoid\left(w_{k0} + \sum_{j=1}^{p} w_{kj} h_j\right)$
- ➔ 선형 결합 후 activation
- Step 4: 계산된 출력노드의 값과 실제 출력변수의 값의 차이를 줄일 수 있도록 은닉층-출력층 사이의 w값 업데이트

$$w_{kj} = w_{kj} - \alpha \frac{\partial E_n}{\partial w_{kj}}$$

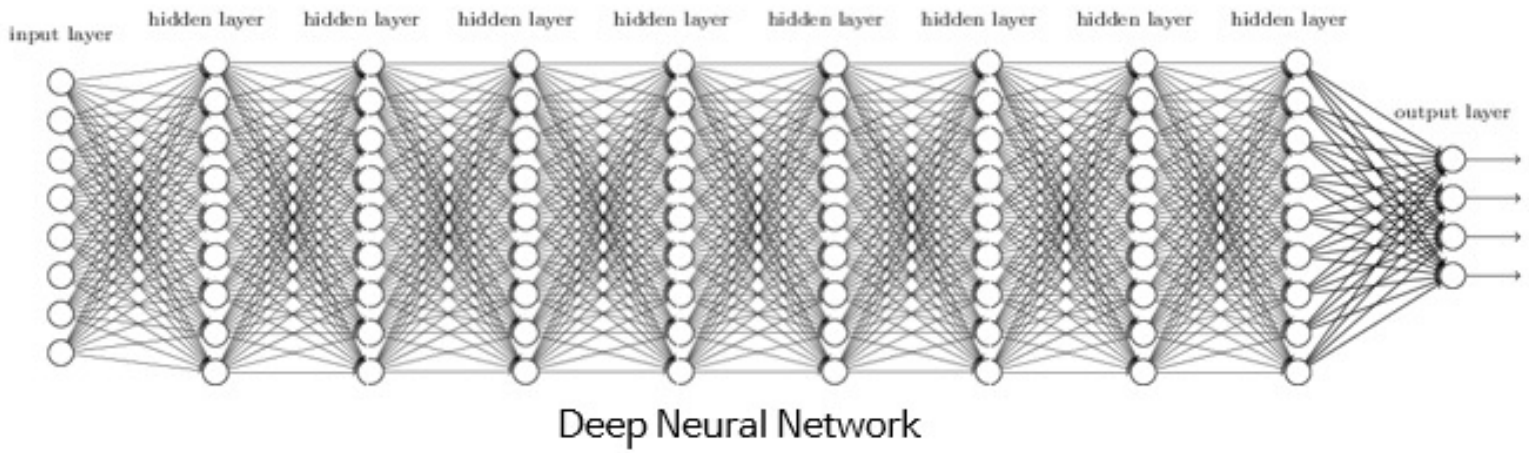
- Step 5: 계산된 출력노드의 값과 실제 출력변수의 값의 차이를 줄일 수 있도록 입력층-은닉층 사이의 w값 업데이트
- Step 6: 에러가 충분히 줄어 들 때 까지 Step2 ~ Step 6반복

$$w_{ji} = w_{ji} - \alpha \frac{\partial E_n}{\partial w_{ji}}$$

- **➔ 오류 역전파 알고리즘 (Error Backpropagation Algorithm, EBPA)**

Deep Neural Network

- Backpropagation을 계기로 인공신경망 분야에 한동안 붐이 일었음 (‘90~)
  - 그러나, 신경망의 깊이가 깊어질수록 학습이 잘 이루어 지지 않는 문제가 발생 (두번째 빙하기)



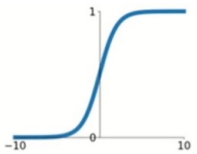
- 기울기 소실 (Gradient vanishing)
  - 신경망이 깊어질 수록 역전파 과정에서 입력층으로 가까워질 수록 기울기가 점점 작아지는 현상이 발생

Deep Neural Network

- Deep의 출현
  - 2006년 Geoffrey Hinton (A fast learning algorithm for deep belief nets)
    - ➔ weight의 초기값을 잘 설정하면 된다 ➔ 층 단위의 학습을 수행하여 더 나은 초기값을 얻을 수 있다
    - 사전학습 (pre-training)의 개념 등장
  - Activation function ➔ ReLU 활용
  - Batch normalization
  - Dropout
  - Early stopping
  - Residual connection (잔차연결)
  - Gradient clipping
  - etc.

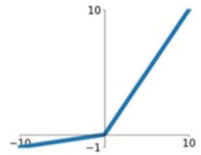
Sigmoid

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



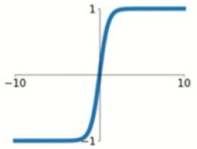
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

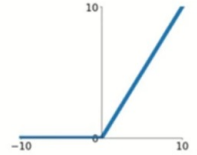


Maxout

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

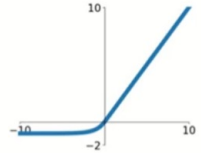
ReLU

$\max(0, x)$



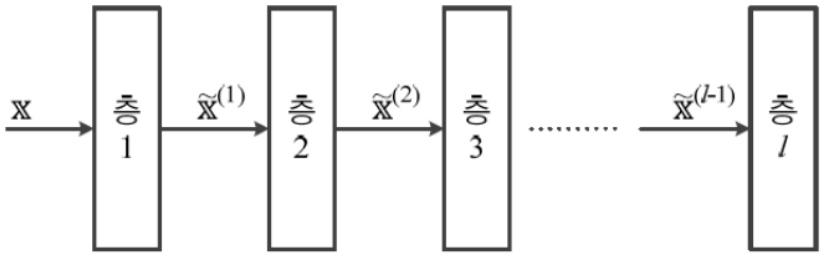
ELU

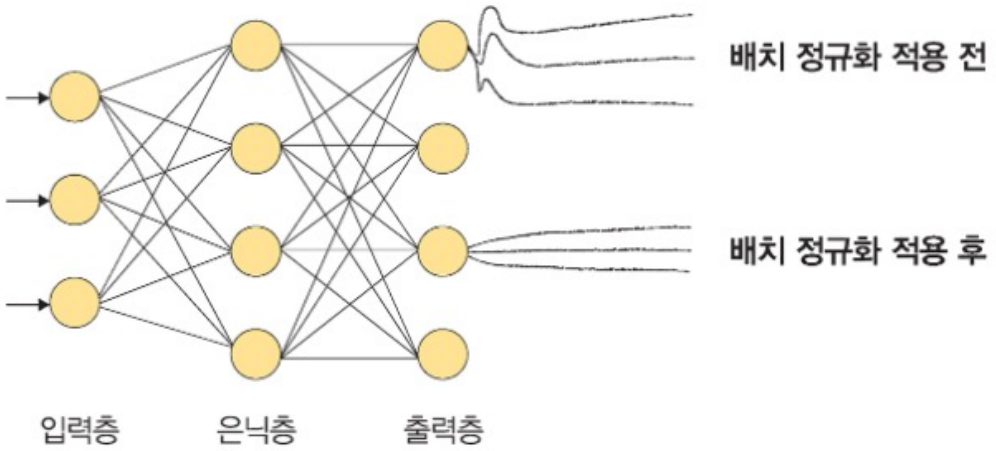
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

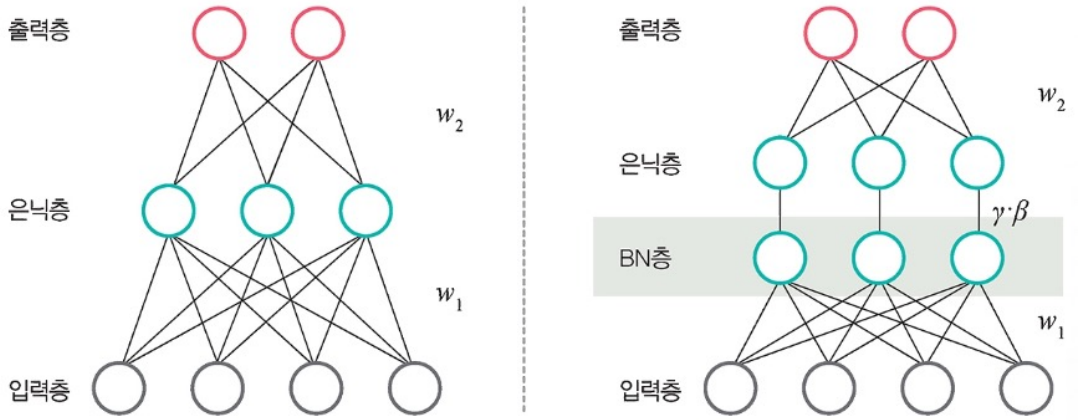


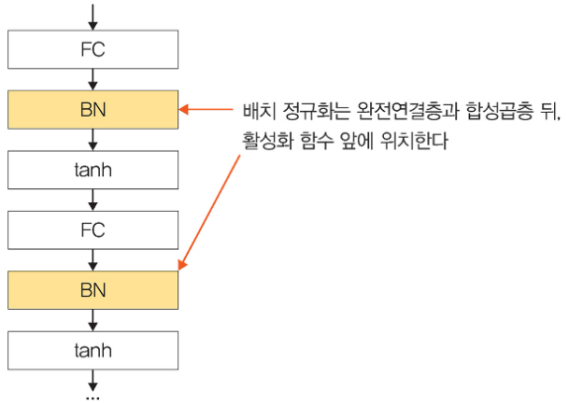
Deep Neural Network

- Batch normalization
  - Covariate shift (공변량 시프트) 현상
    - 학습이 진행되면서 이전 layer의 매개변수가 바뀜 ➔ 다음 layer의 입력 데이터 분포가 바뀌는 결과
    - ➔ 학습을 방해하는 요인으로 작용



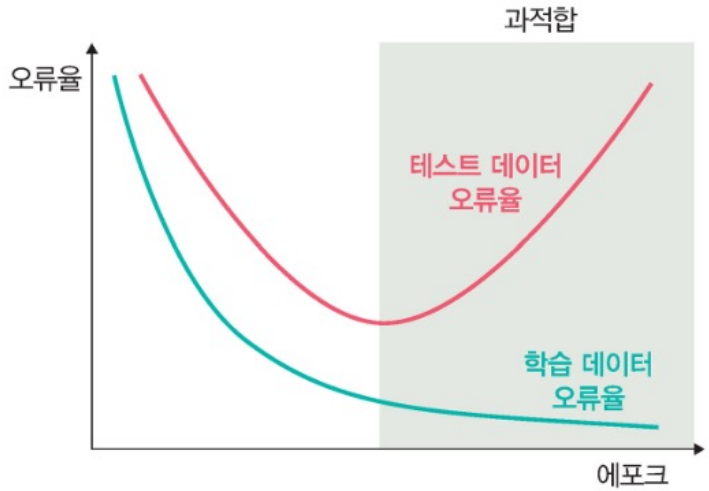


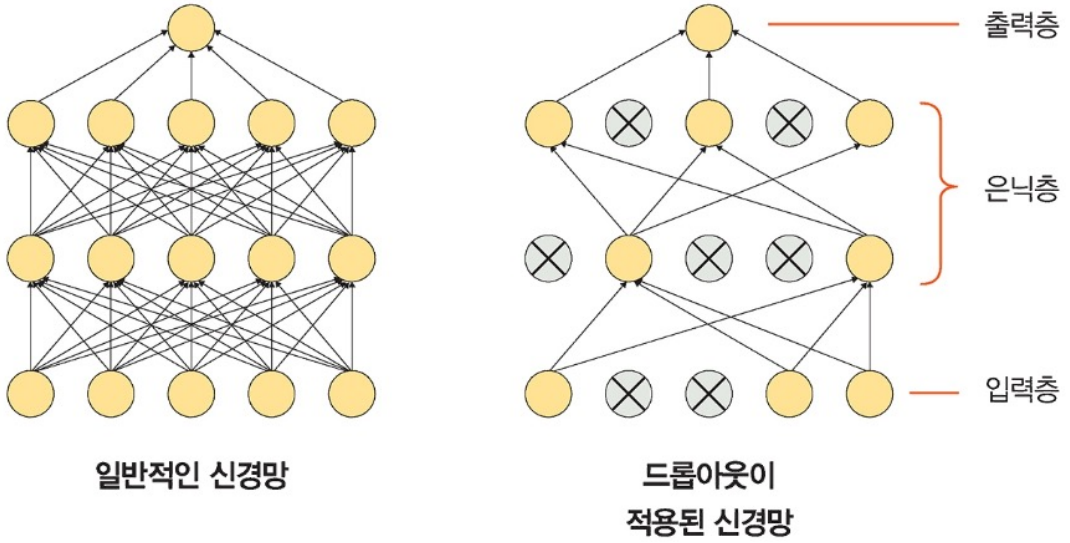




Deep Neural Network

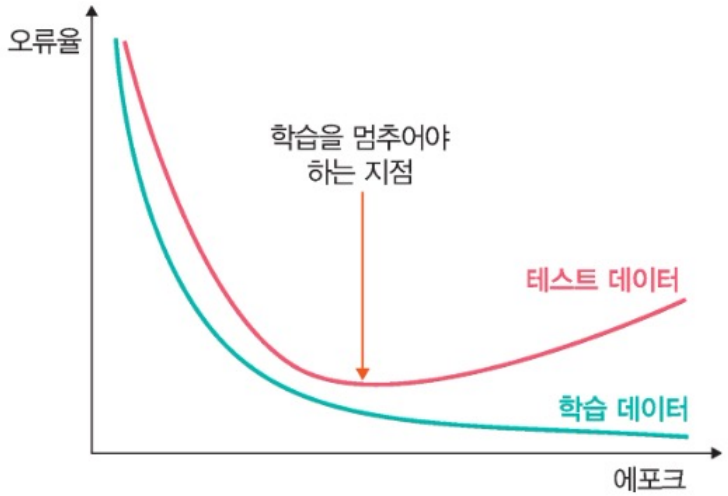
- Dropout
  - Overfitting ➔ Training 데이터의 과도한 학습이 원인 중 하나
  - Dropout 기법
    - Training 시 일정 비율의 node(neuron)만 활용 ➔ weight update X
    - Test 시에는 모든 node 사용





Deep Neural Network

- Early stopping
  - Overfitting ➔ Training 데이터의 과도한 학습이 원인 중 하나



  - 특정 지점에서 학습을 종료하는 방법
    - 매 epoch마다 검증 데이터에 대한 오차 (validation loss)를 측정하여 종료 시점 결정
    - 검증 데이터의 오차가 증가 ➔ Overfitting 발생 ➔ 학습 중지

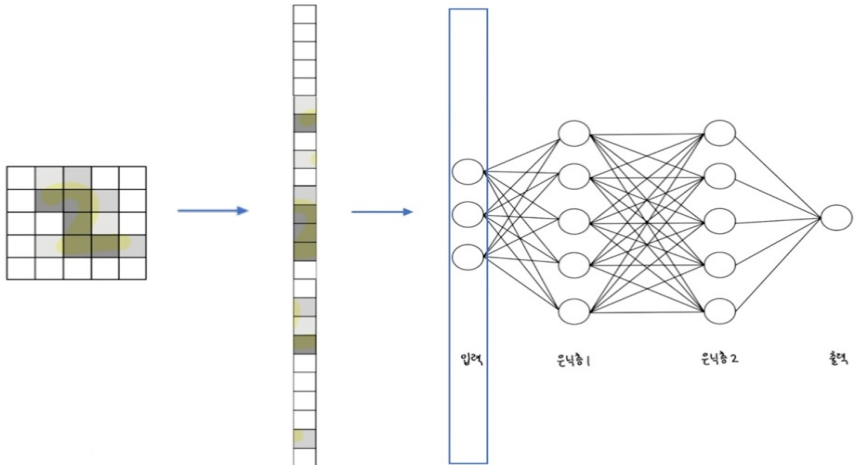
뉴럴네트워크 모델 구축 실습

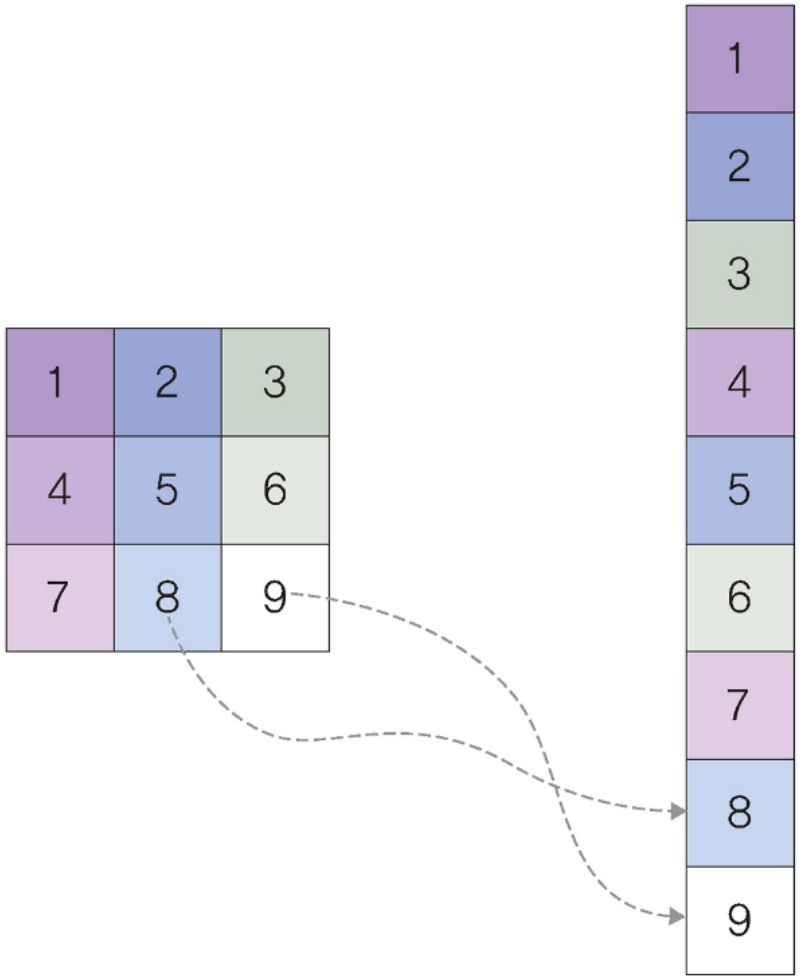
- 1) Boston house price prediction – regression problem
- 2) MNIST handwritten digit recognition – classification problem

3. Convolutional Neural Network (CNN)

Getting started

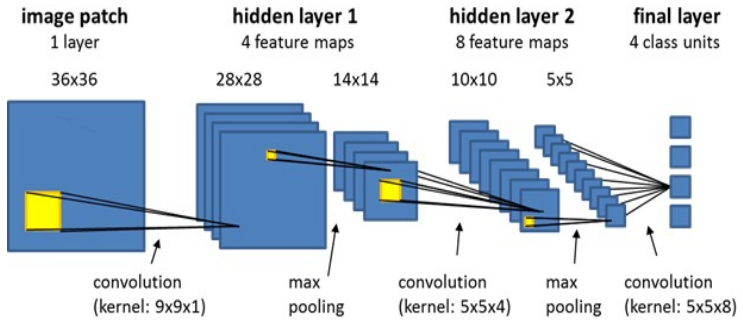
- **Primary objectives of CNN**
  - To process the data with efficient time/resource cost by calculating partial region in the given data, not the entire data
    - The mainly used data = image, video, etc. (regional featur
  - To consider the spatial features (partial region) in image
- .vs typical neural network model
  - Loss the regional information





CNN의 기본 구조

- Basic format of CNN



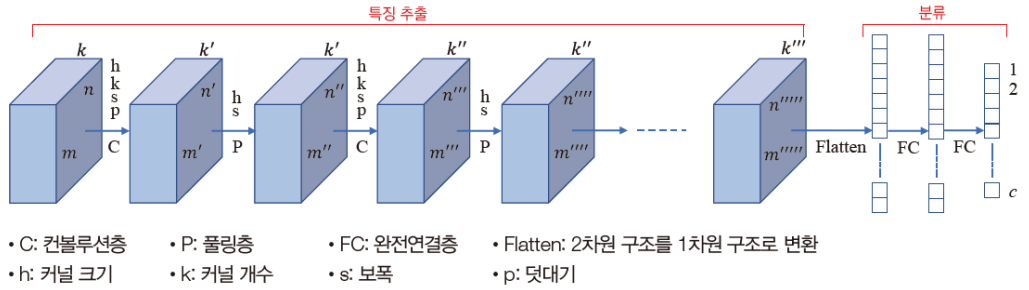
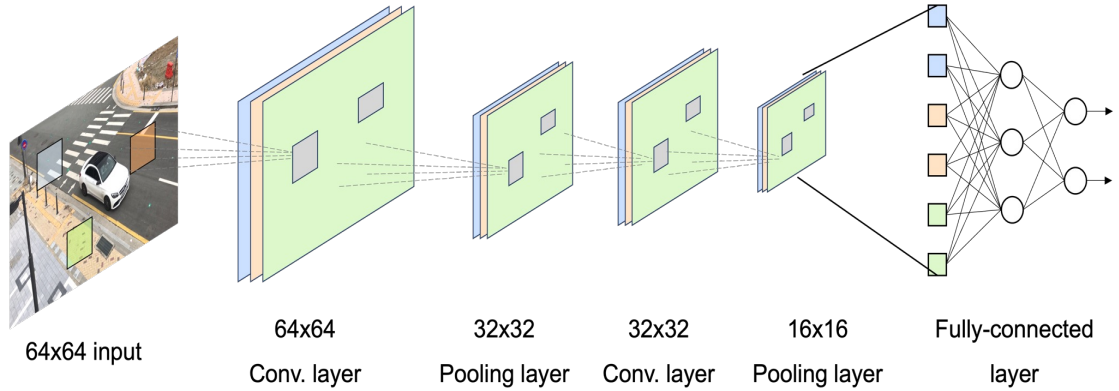


그림 8-10 컨볼루션층과 풀링층을 번갈아 쌓아 만드는 컨볼루션 신경망의 전형적인 구조



- CNN의 핵심 아이디어: "최적의 필터(kernel)을 학습으로 획득"
  - Deep neural network의 weight = CNN의 kernel

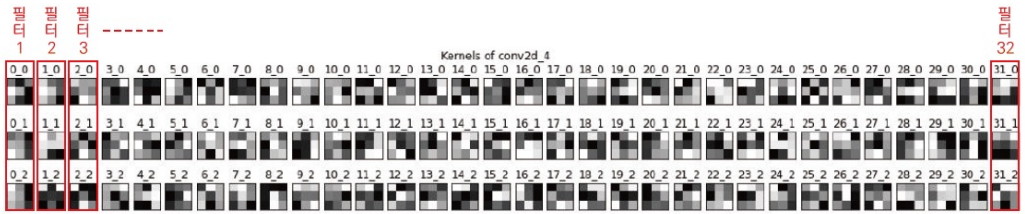
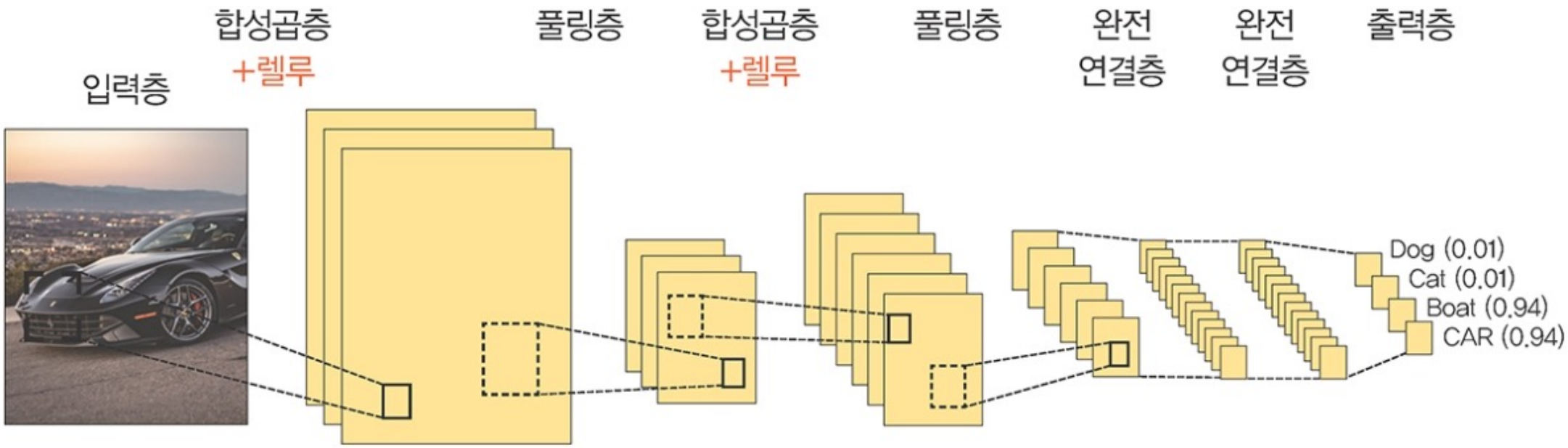


그림 8-14 CIFAR-10 데이터셋으로 학습한 컨볼루션 신경망의 최적 필터

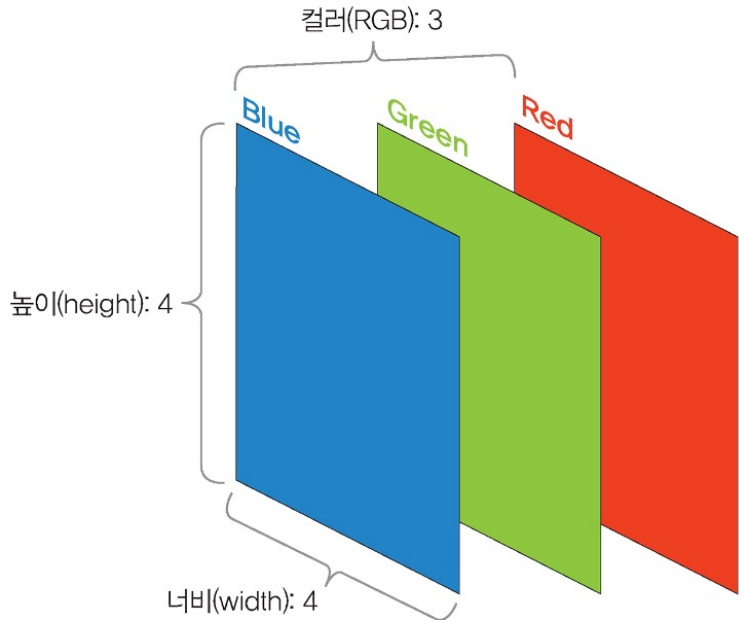
CNN의 기본 구조

- Basic format of CNN



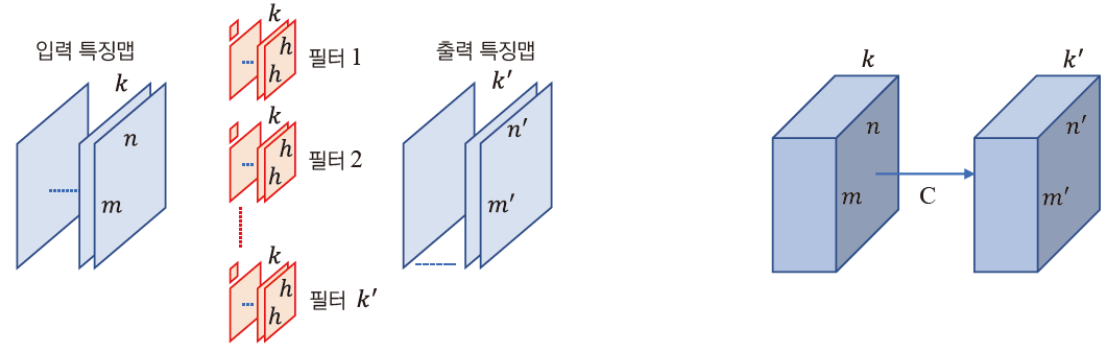
CNN의 기본 구조

- Input layer
  - Image = 3-dimensional data (height, width, channel)
    - Channel = color space (gray = 1, RGB = 3)
    - Height = 4, Width = 4, RGB = 3 ➔ tensor shape = (4, 4, 3)



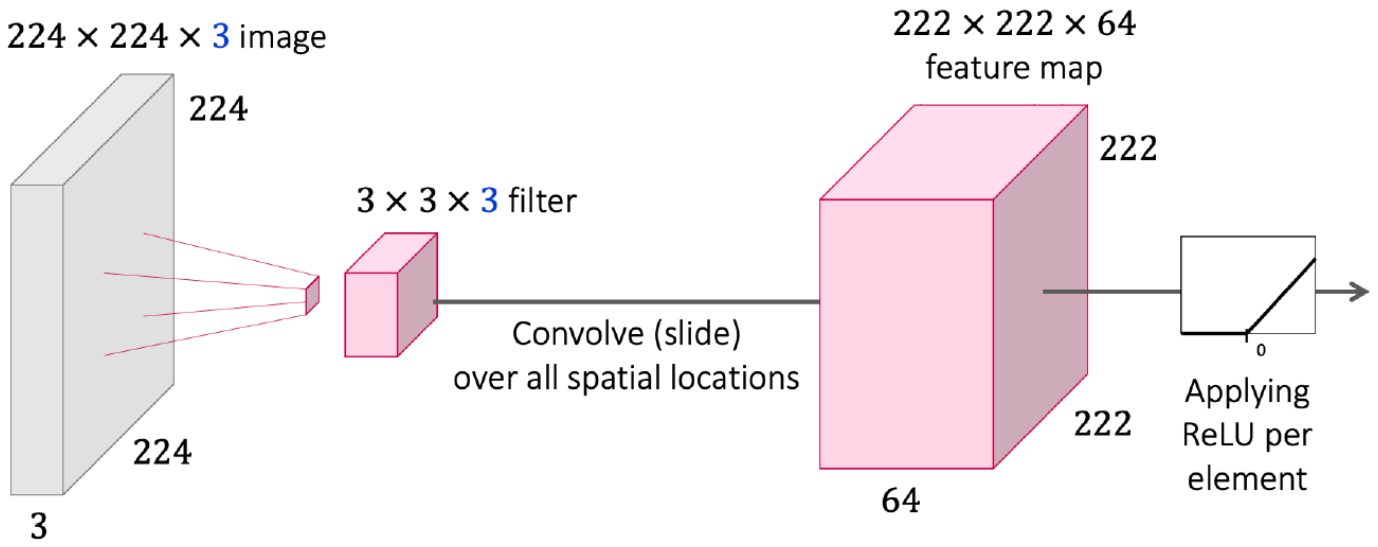
CNN의 기본 구조

- Layer operation 1 – convolutional layers
  - 입력 특징맵(feature map)이 $m \times n \times k$ 크기의 tensor
    ➔ $h \times h \times k$ 크기의 필터(커널) tensor 필요
  - 출력 특징맵은 $m' \times n' \times k'$
    - ➔ padding과 stride로 계산할 수 있음



(a) 세부 내용 (b) 간결한 블록 표현

그림 8-6 컨볼루션층



CNN의 기본 구조

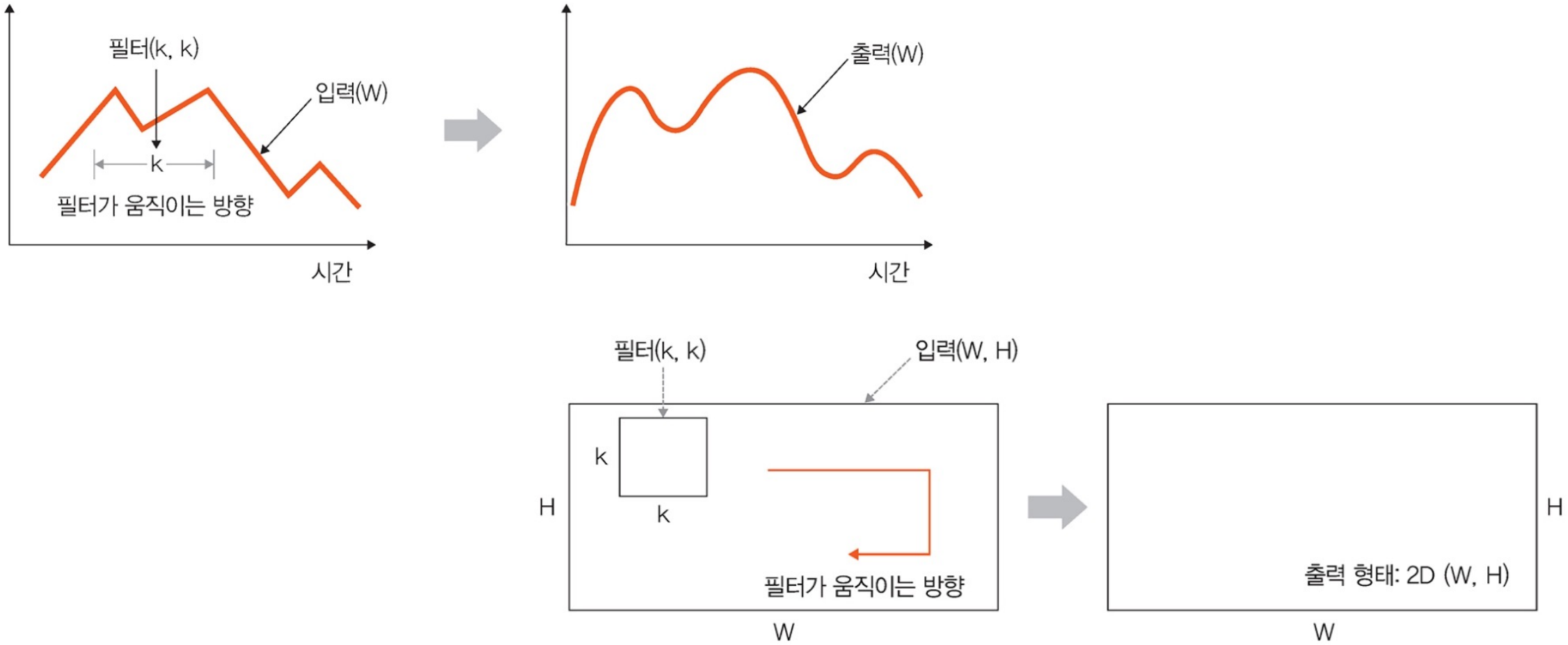
- Layer operation 1 – convolutional layers
  - example of convolution operation for grayscaled-input ($m \times n \times k$ 에서 $k = 1$)

CNN의 기본 구조

- Layer operation 1 – convolutional layers
  - example of convolution operation for grayscaled-input ($m \times n \times k$ 에서 $k = 1$)
    - Shape of the origin image (6, 6, 1) ➔ Shape of output (4, 4, 1)
      - ➔ A new feature map + feature size reduction

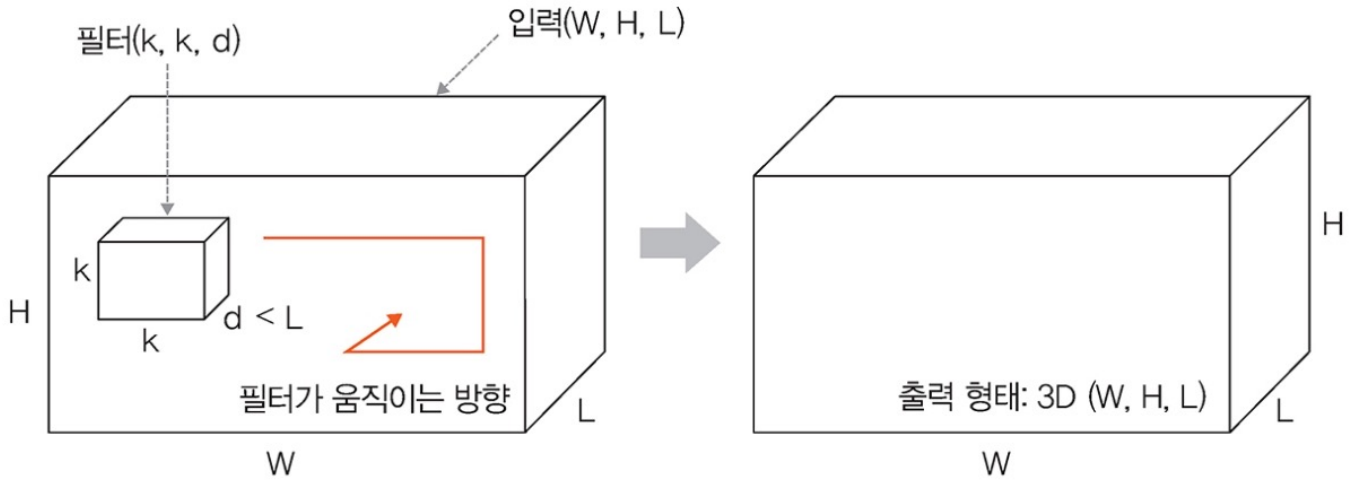
CNN의 기본 구조

- Layer operation 1 – convolutional layers
  - convolution operation for 1D, 2D, and 3D



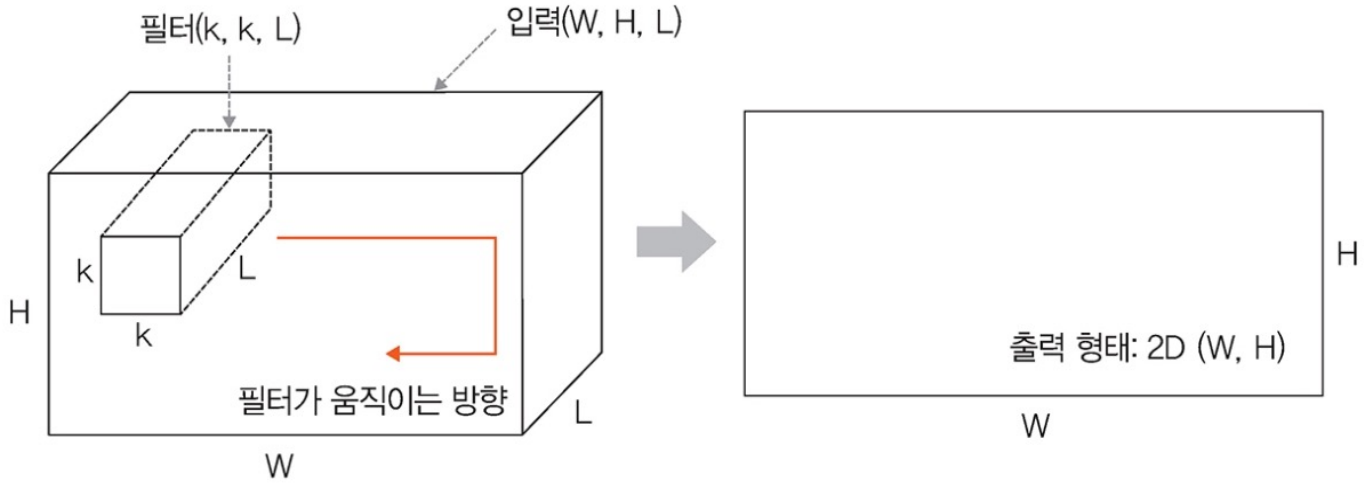
CNN의 기본 구조

- Layer operation 1 – convolutional layers
  - convolution operation for 1D, 2D, and 3D



3D ➔ 2D

LeNet-5, VGG, etc.



CNN의 기본 구조

- Layer operation 1 – convolutional layers
  - Input data
    - $\mathrm{W}_1 \times \mathrm{H}_1 \times \mathrm{D}_1$ (Width x Height x Channel (depth))
  - Hyperparameter

    K: Number of kernels (filters)

    F: Size of kernels

    S: Stride

    P: Padding
  - Output data

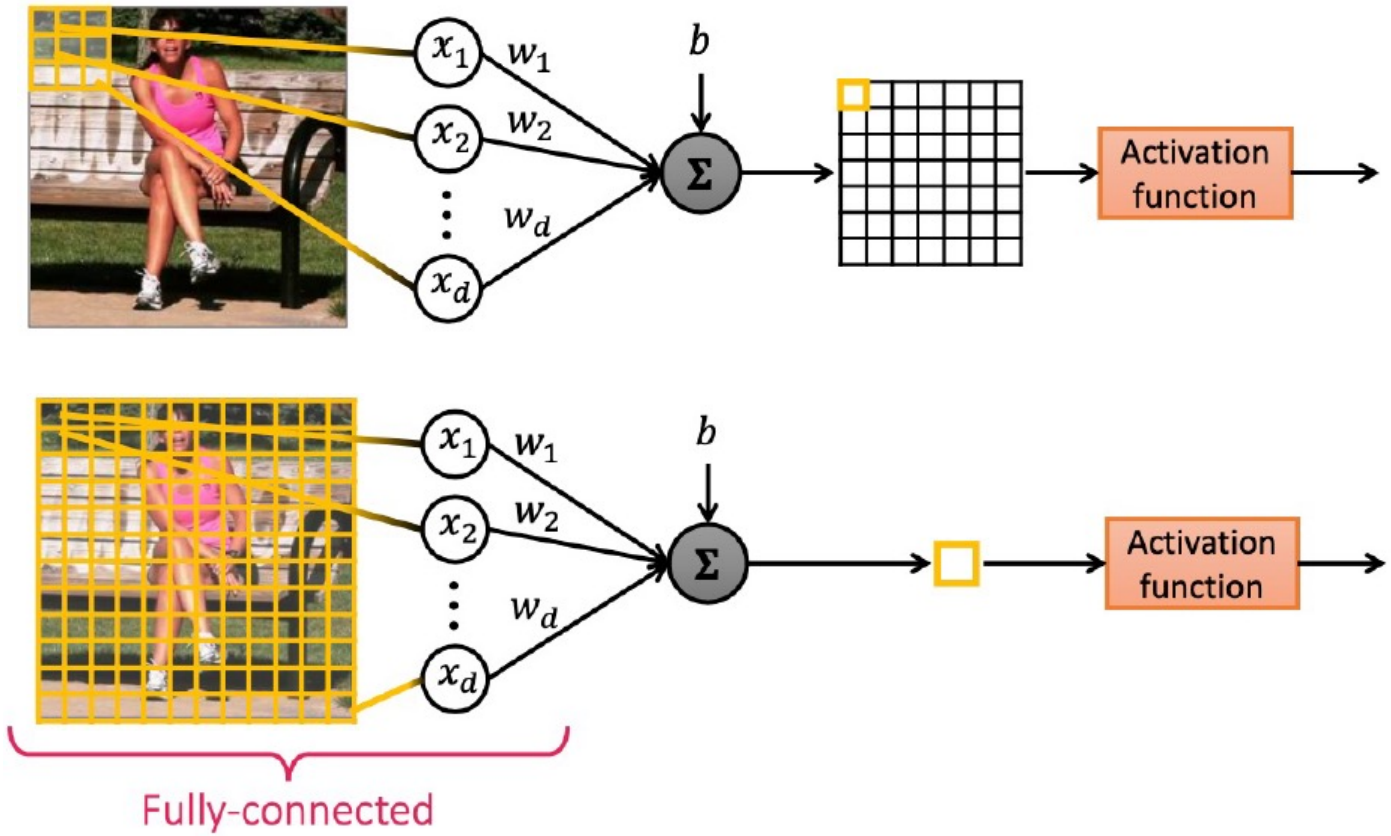
    $W_1 = (W_1 - F + 2P)/S + 1$

    $H_1 = (H_1 - F + 2P)/S + 1$

    $D_2 = K$

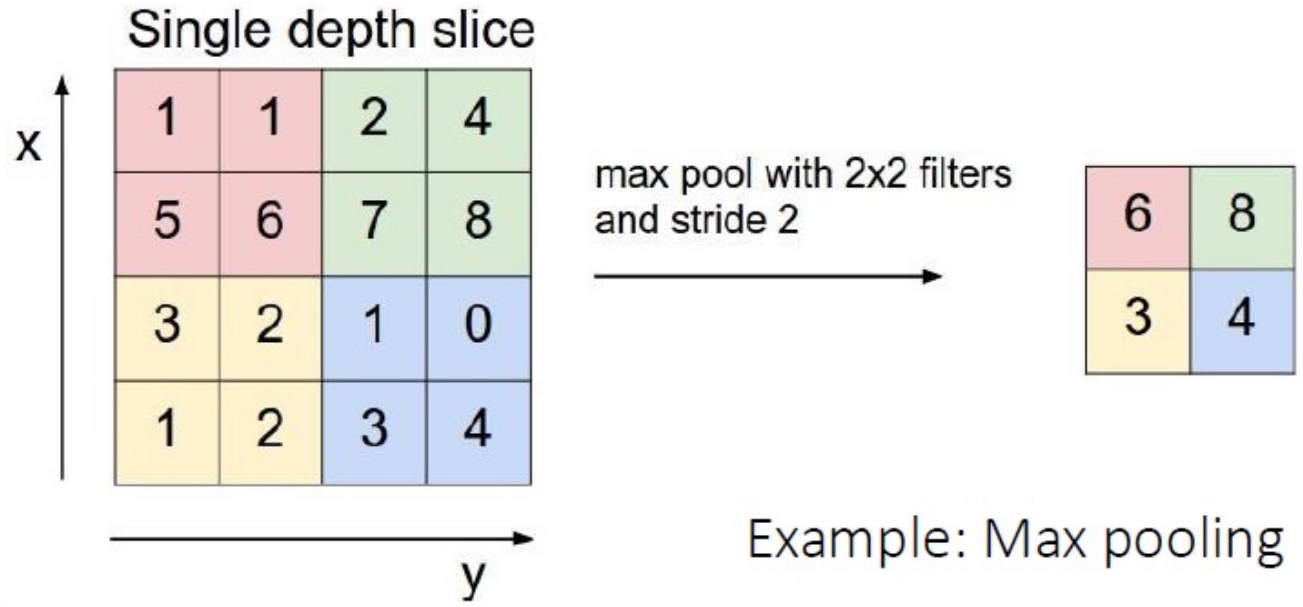
CNN의 기본 구조

- Kernel size의 역할
  - 적당히 작은 크기의 kernel (3x3x1) vs input과 같은 크기의 kernel

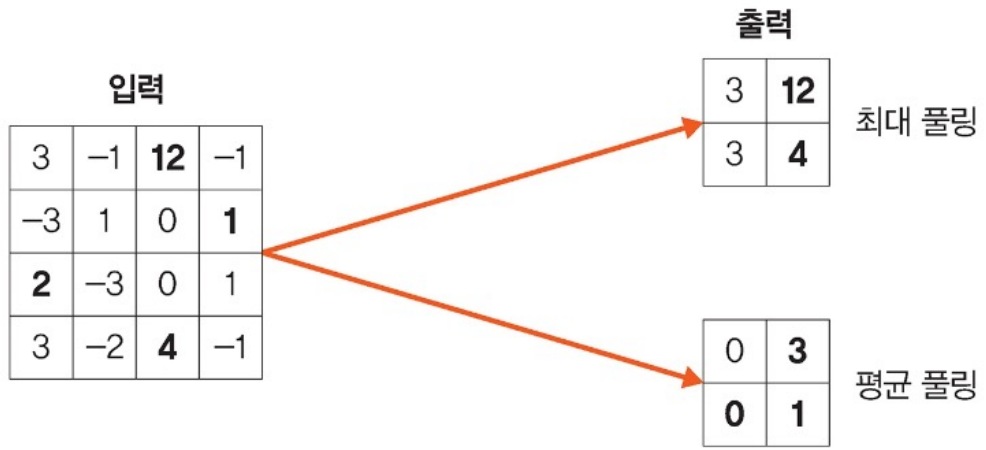


CNN의 기본 구조

- Layer operation 2 – Pooling operations
  - pooling types
    - max pooling
    - average pooling
    - L2-norm pooling, etc.
  - 역할
    - 지역적 불변성 달성
    - 이미지 정보의 요약
    - 차원 축소

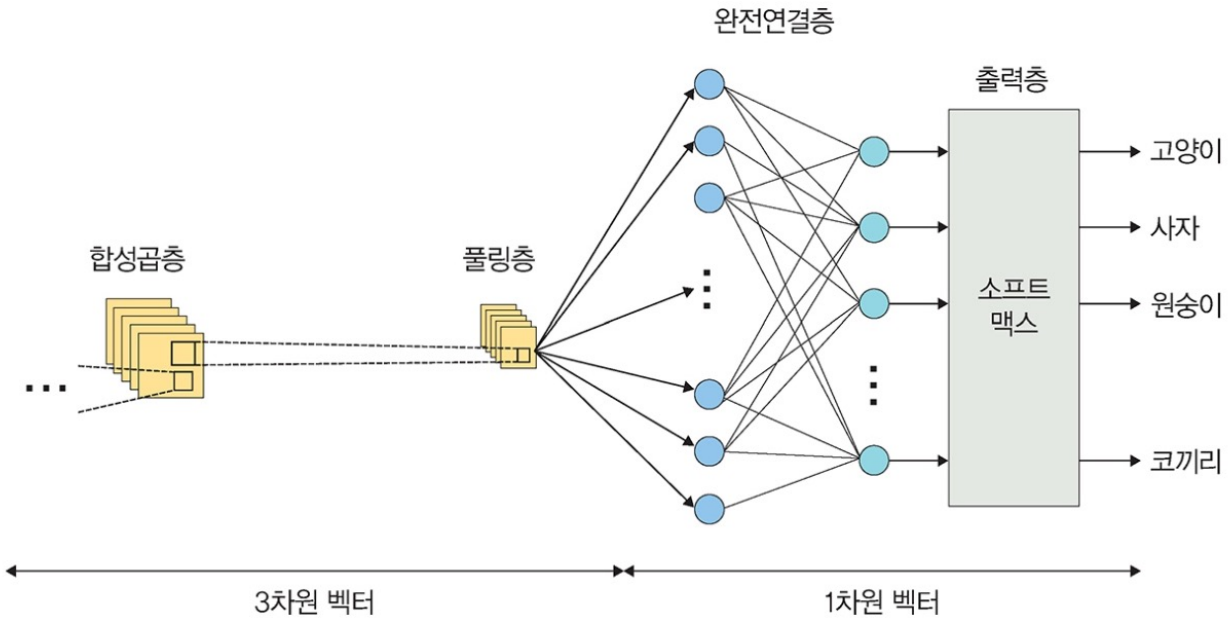


Example: Max pooling



CNN의 기본 구조

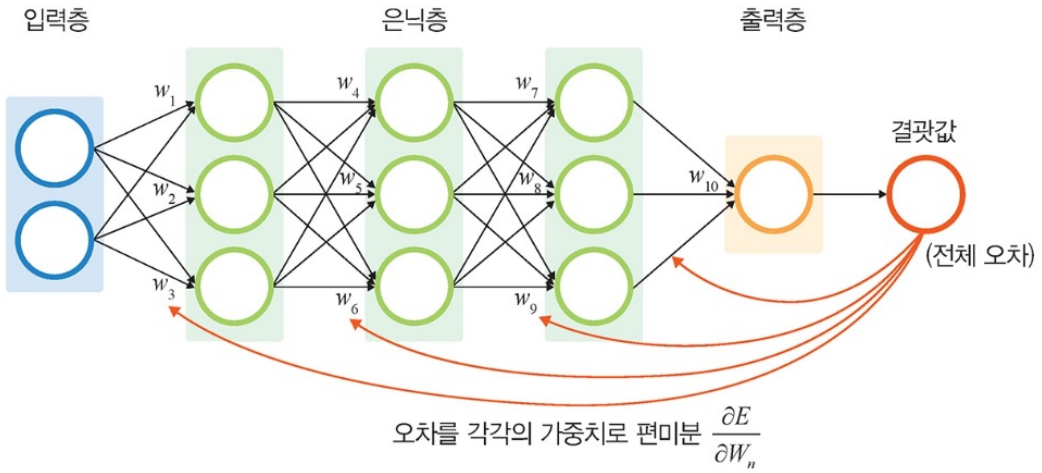
- Layer operation 3 – Fully-connected (output) layers



  - Flattening
    - Multi-dimensional feature maps ➔ flattened into a 1-D vector
    - To convert the spatial information into a format that can be processed by a traditional feedforward neural network

CNN의 오류 역전파

- General DNN



- CNN

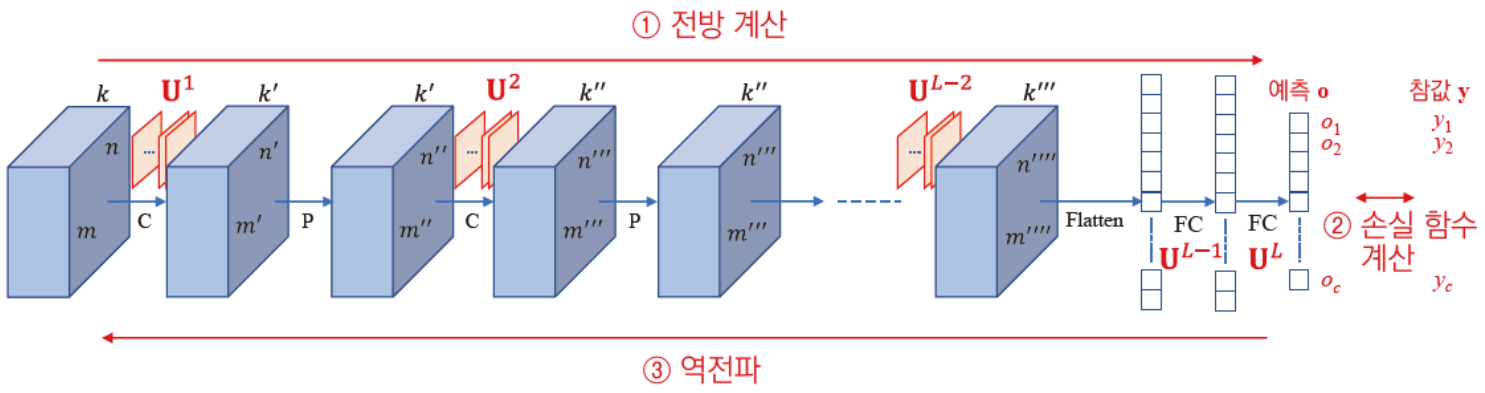


그림 8-13 컨볼루션 신경망을 학습하기 위한 역전파 알고리즘

CNN 모델 구축 실습

- MNIST handwritten digit recognition – classification problem

4. Recurrent Neural Network (RNN)

Getting started

- Time series problems
  - Tasks and challenges that involve analyzing, forecasting, and making predictions based on **data that is collected over time and is ordered chronologically**
    - Data points are typically recorded at regular intervals; daily, monthly, yearly, etc.
    - Goal is to uncover patterns, trends, and relationships within the data to make informed decisions or predictions about future values

Before deep learning era

- **AR (AutoRegressive) model**
  - A time series model where the current value of a variable is predicted based on its own past values

$$Y_t = f\left(Y_{t-1}, Y_{t-2}, \dots Y_{t-p}\right) + \varepsilon = \Phi_0 + \sum_{i=1}^{p} \Phi_i Y_{t-i} + \varepsilon_t$$

  - Y_t: Value at time t (represented as Z in textbook)
  - Φ_i: Parameter representing the influence of the past on the present
  - ε_t: Error term at time t

Before deep learning era

- **MA (Moving Average) model**
  - Focused on the relationship between the observed value and **the residual errors from the past**
    - Assume that the future value of the variable is a linear combination of the past error terms

$$Z_t = \theta_0 + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

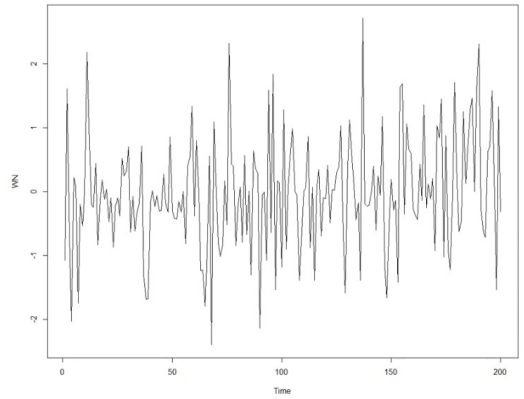
    - Z_t: Value at time t
    - θ_i: Parameter representing the influence of the past on the present
    - ε_t: Error term at time t

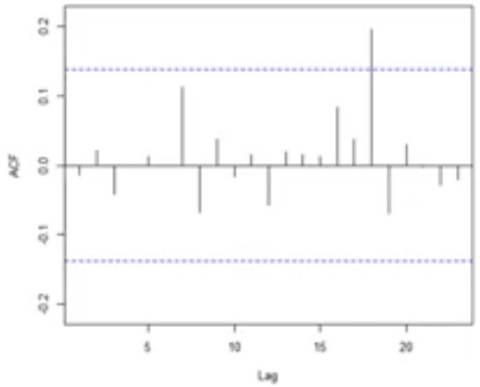
Before deep learning era

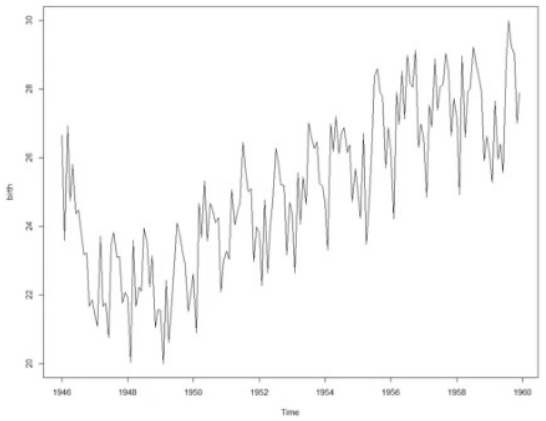
- **ARMA (AutoRegressive Moving Average) model**
  - Combine the features of both the AR and MA models

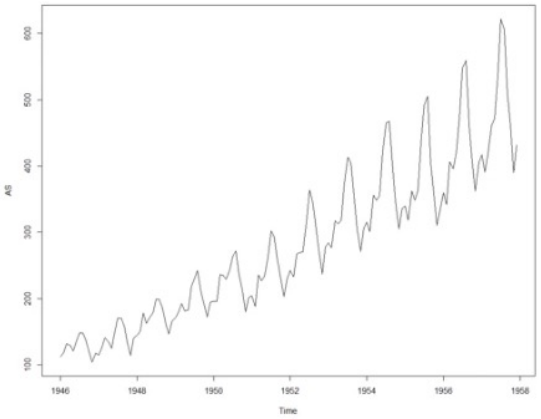
$$Z_t = \Phi_0 + \sum_{i=1}^{p} \Phi_i Z_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

  - AR, MA, ARMA ➔ The given time series data should be “stationary”
    - Stationary: Constant mean, variance over time (Check by using Autocorrelation function (ACF))









Before deep learning era

- **ARIMA (AutoRegressive Integrated Moving Average) model**
  - An extension of the ARMA model that includes an additional **differencing** (차분) step to make the time series **stationary** (정상성)
    - Differencing step: to remove any trends or seasonality present in the data
  - ARIMA model ➔ ARIMA(p, d, q)
    - p: order of the AR part of the model
    - d: order of differencing
    - q: order of the MA part of the model

Before deep learning era

- **Time series data analysis in machine learning**
  - Linear regression/Logistic regression models
  - Support vector machine/regression
  - Random forest
  - Boosting
  - Hidden Markov model (HMM), etc.
  - …

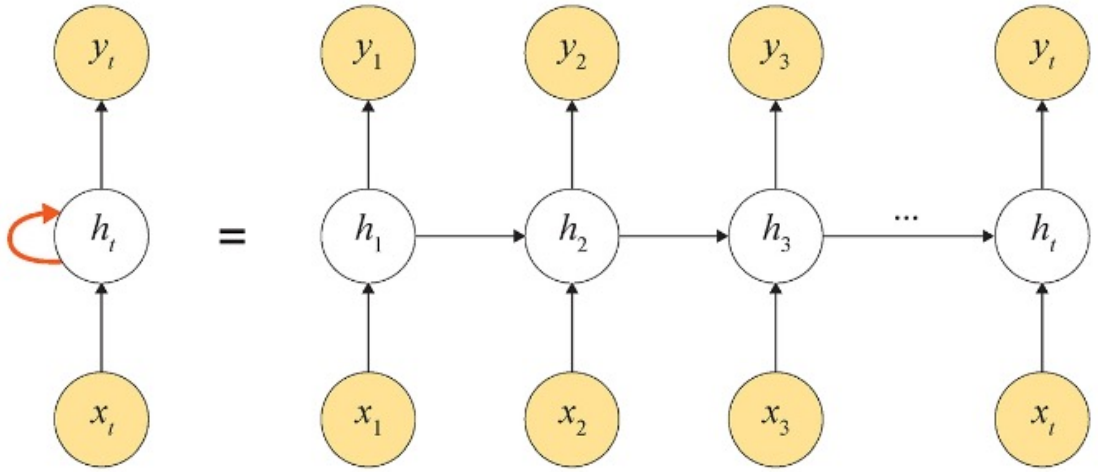
Before deep learning era

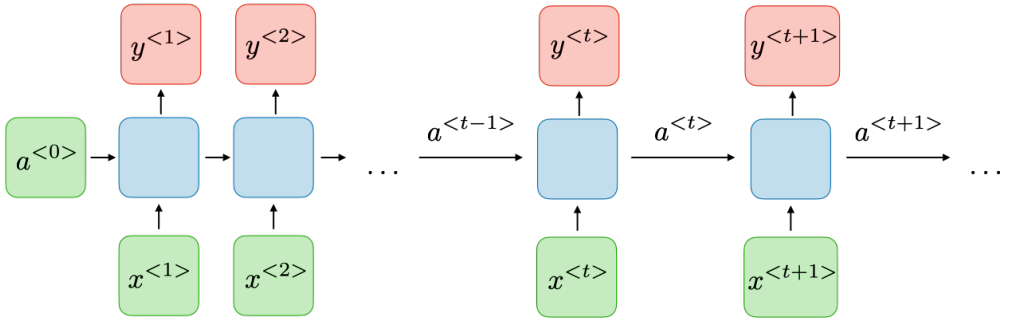
- Time series + Deep learning (neural network) ➔ Concept of "**Recurrent**"
  - Having a repeating cyclical structure
  - RNN (recurrent neural network) (1986)
  - LSTM (1997)
  - GRU (2014)
  - Seq2Seq (2014)
  - Seq2Seq with attention (2015)
  - CNN for time series analysis (2016)
  - Transformer (2017)
  - GPT-1, BERT, GPT-3, GPT-3.5 (2022), ChatGPT (2023)
  - etc.

Concept of RNN (순환신경망)

- Designed to process sequential data (time series, natural language sentence, etc.)
  - By maintaining a hidden state that retains information from previous time steps in sequence

- General structure of RNN
  - A hidden state h_t, which represents the **memory** of the network at that particular time step
    - Allowing the RNN to retain information from previous step

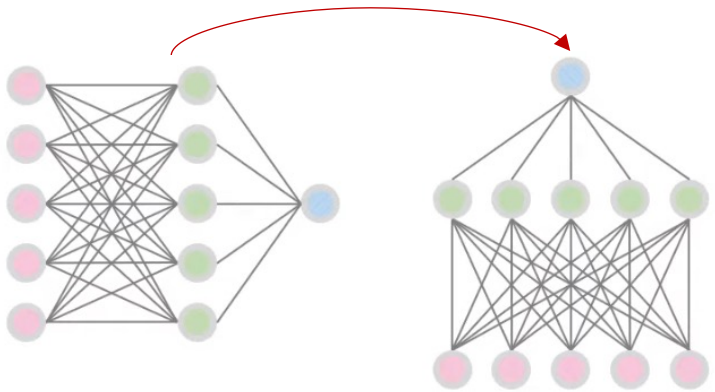


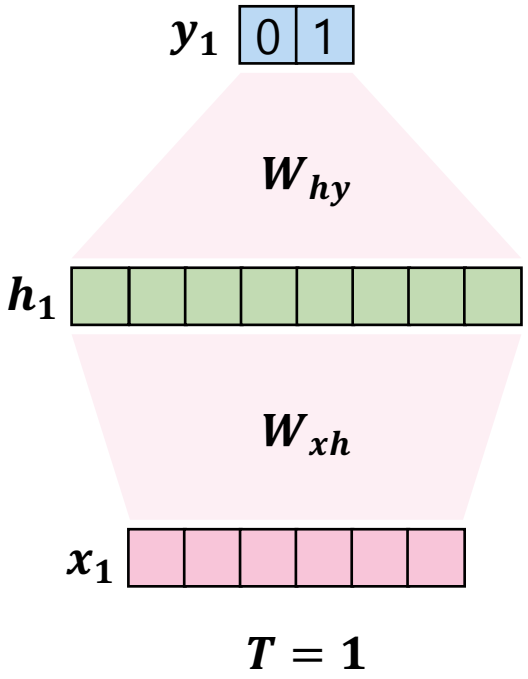


Concept of RNN

- Review (DNN training)

time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Status
12:00	0	98.9	3.9	0.19	0.21	Normal





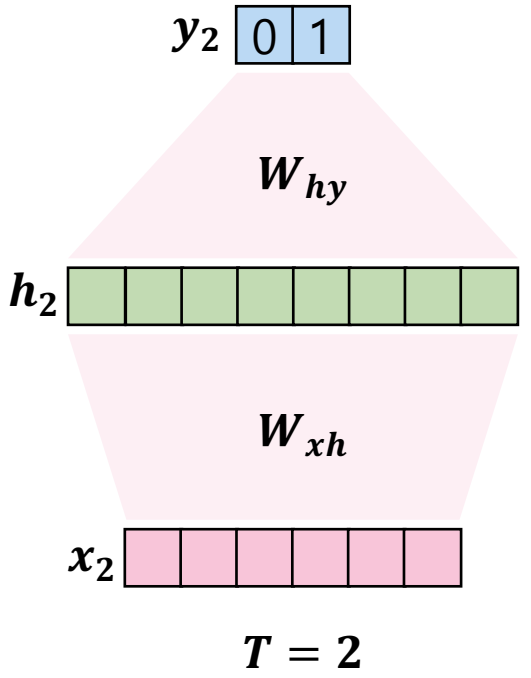
$$y_1 = g(W_{hy}h_1) \quad f(\cdot) = tanh \; g(\cdot) = softmax$$

$$h_1 = f(W_{xh}x_1)$$

Concept of RNN

- Review (DNN training)

time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Status
12:00	0	98.9	3.9	0.19	0.21	Normal
13:00	0	98.9	3.9	0.21	0.23	Normal



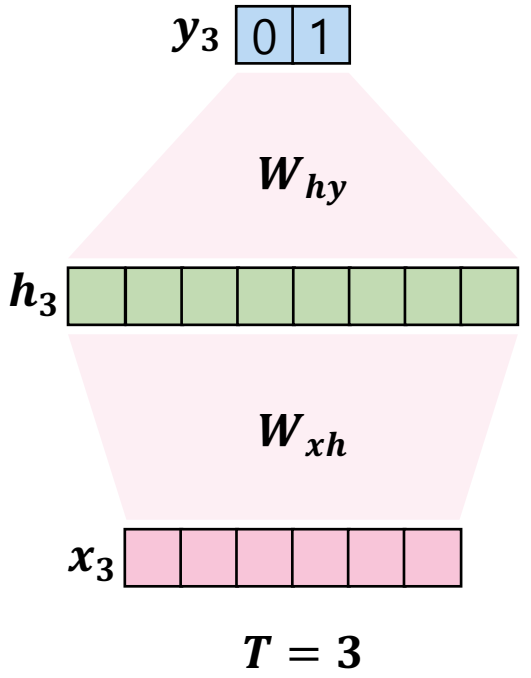
$$y_2 = g(W_{hy}h_2)$$

$$h_2 = f(W_{xh}x_2)$$

Concept of RNN

- Review (DNN training)

time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Status
12:00	0	98.9	3.9	0.19	0.21	Normal
13:00	0	98.9	3.9	0.21	0.23	Normal
14:00	0.3	74.5	6.7	0.23	0.24	Abnormal



$$y_3 = g(W_{hy}h_3)$$

$$h_3 = f(W_{xh}x_3)$$

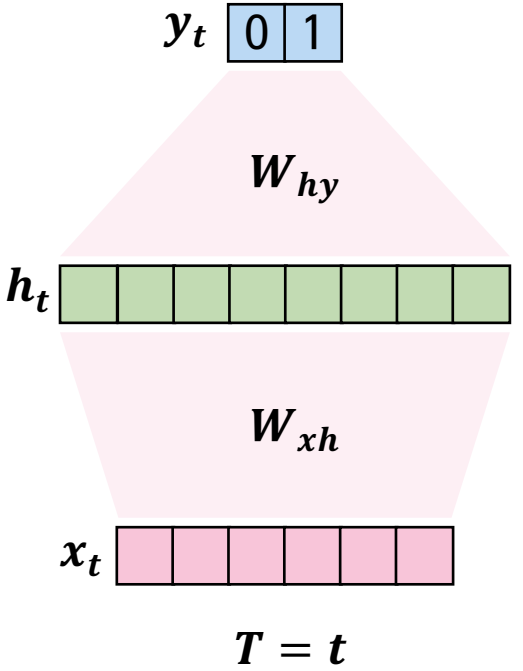
Concept of RNN

- Review (DNN training)

	time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Status
$t-2$	12:00	0	98.9	3.9	0.19	0.21	Normal
$t-1$	13:00	0	98.9	3.9	0.21	0.23	Normal
t	14:00	0.3	74.5	6.7	0.23	0.24	Abnormal

시점 정보를 반영하지 않음

이전 시점 정보를 반영해서 더 나은 예측을 하자



$$y_t = g(W_{hy}h_t)$$

$$h_t = f(W_{xh}x_t)$$

+이전 시점 정보

Concept of RNN

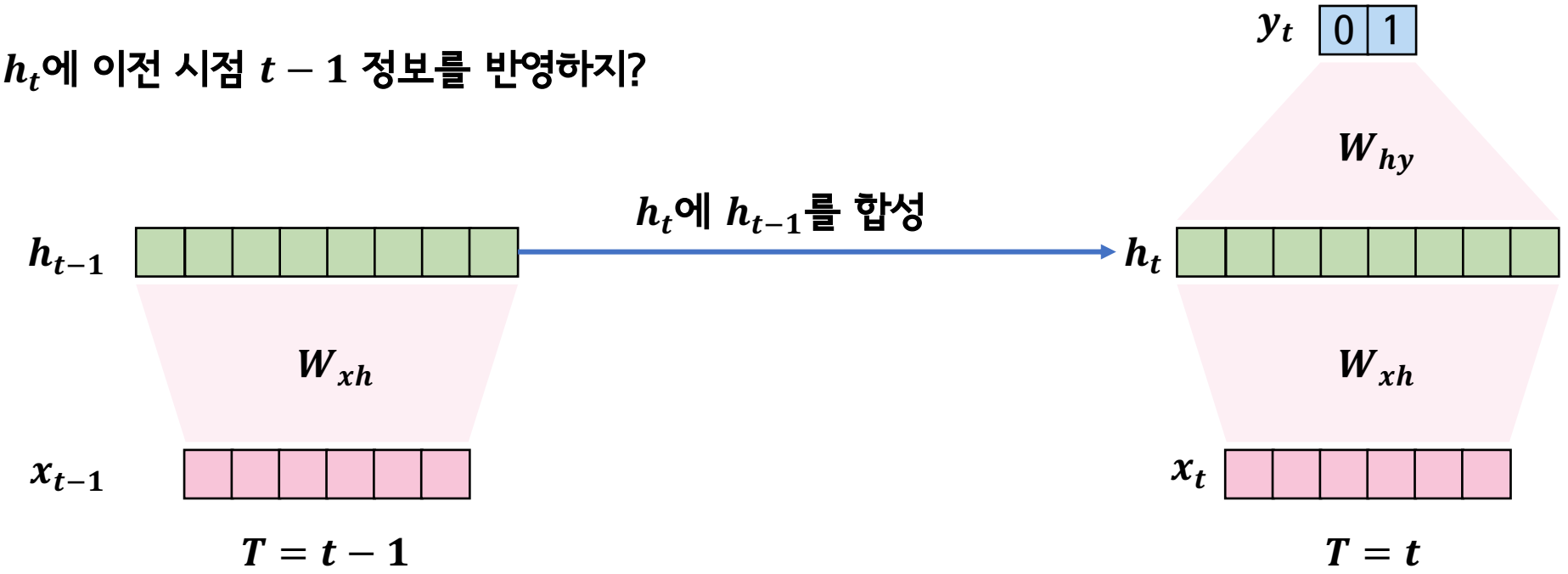
- Review (DNN training)

$$y_t = g(W_{hy} h_t)$$

$$h_t = f(W_{xh} x_t) \quad + t-1 \text{ 정보}$$

	time	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Status
$t-2$	12:00	0	98.9	3.9	0.19	0.21	Normal
$t-1$	13:00	0	98.9	3.9	0.21	0.23	Normal
t	14:00	0.3	74.5	6.7	0.23	0.24	Abnormal

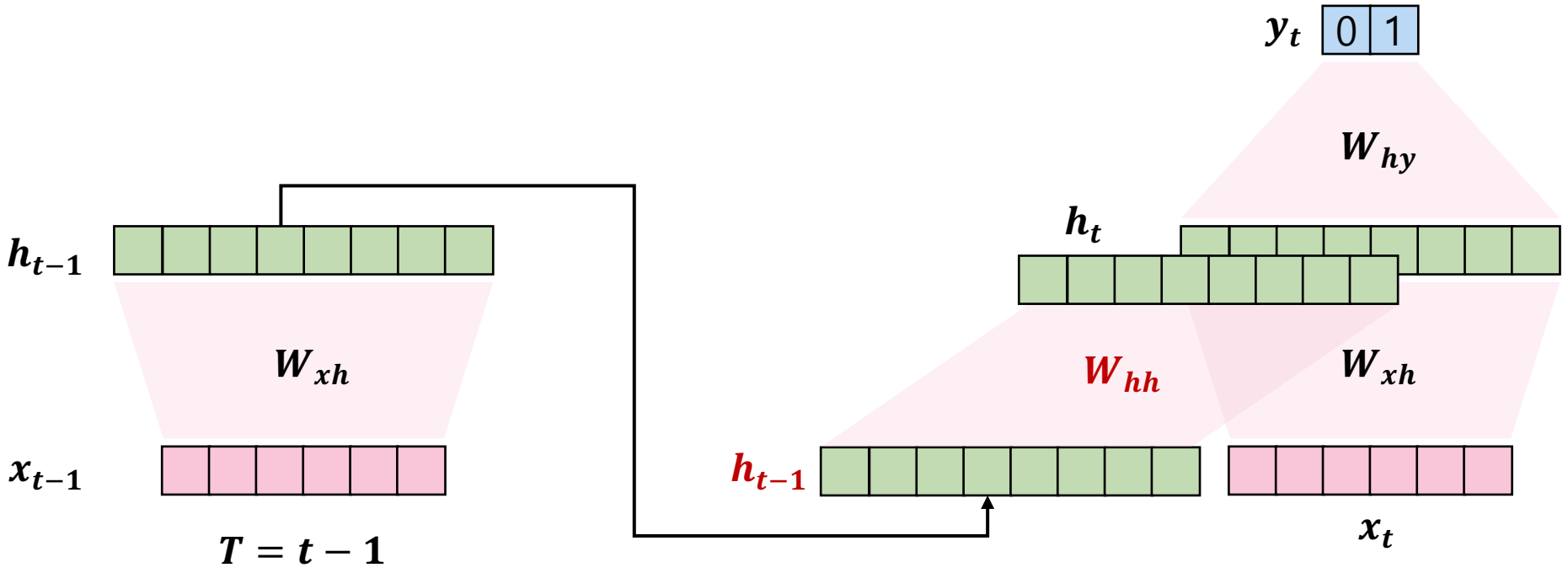
어떻게 h_t에 이전 시점 $t-1$ 정보를 반영하지?



Concept of RNN

$$y_t = g(W_{hy} h_t)$$

$$h_t = f(W_{xh} x_t + W_{hh} h_{t-1})$$

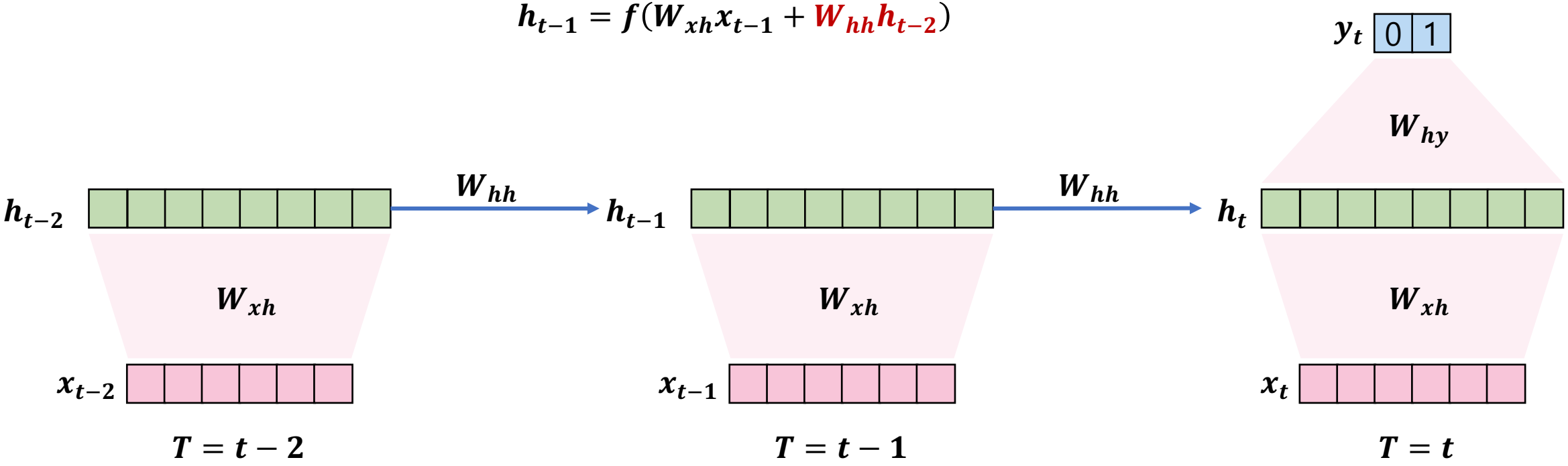


Concept of RNN

$$y_t = g(W_{hy} h_t)$$

$$h_t = f(W_{xh} x_t + W_{hh} h_{t-1})$$

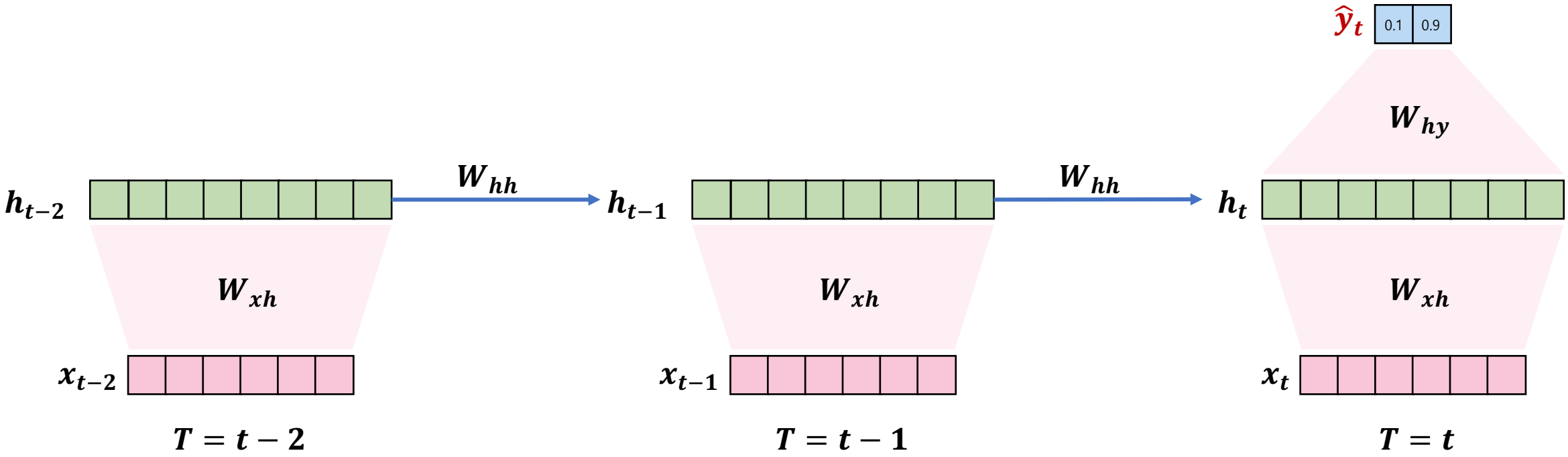
$$h_{t-1} = f(W_{xh} x_{t-1} + W_{hh} h_{t-2})$$



RNN training process

- Parameters (학습 대상)
  - weight: W_{xh}, W_{hh}, W_{hy}
    - W_{hh}: Weight from node at time t to node at time $t+1$
    - Each W_{xh}, W_{hh}, W_{hy} is **the same values at all time points (Parameter sharing)**

- W_{xh}: t 시점 데이터 반영
- W_{hh}: t 시점 이전 정보 반영
- W_{hy}: t 시점의 y를 예측

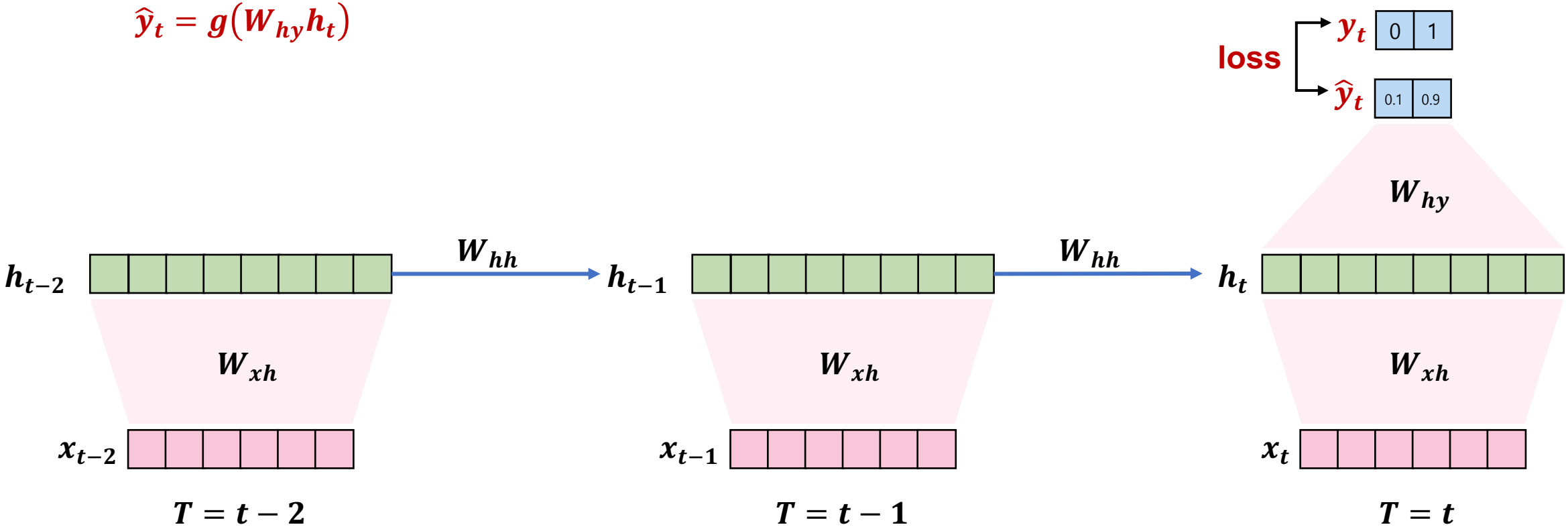


RNN training process

- 최적의 W: W를 매 시점 적용했을 때 **loss**가 최소가 되도록

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$$

$$\hat{y}_t = g(W_{hy}h_t)$$

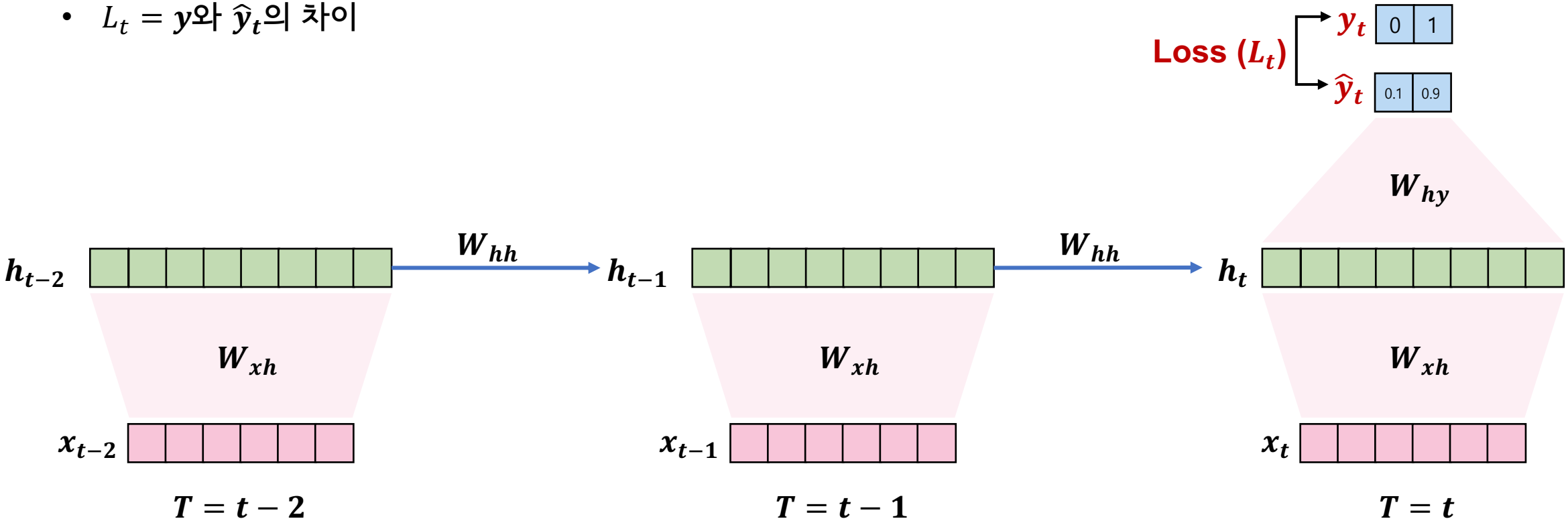


RNN training process

- 최적의 W: W를 매 시점 적용했을 때 **loss**가 최소가 되도록
  - 1) Loss 계산하기: Forward propagation
    - 특정 W_{xh}, W_{hh}, W_{hy}일 때 loss는 얼마?
  - 2) Gradient 계산하기: Backward propagation
    - W_{xh}, W_{hh}, W_{hy}일 때, Loss에 얼마나 영향을 미치는가?
  - 3) Parameter update
    - W_{xh}, W_{hh}, W_{hy}를 어떻게 더 좋은 수치로 업데이트?
  - 방향: Loss를 줄이는 방향
  - 학습 정도: Gradient 만큼 ➔ **Gradient descent method**

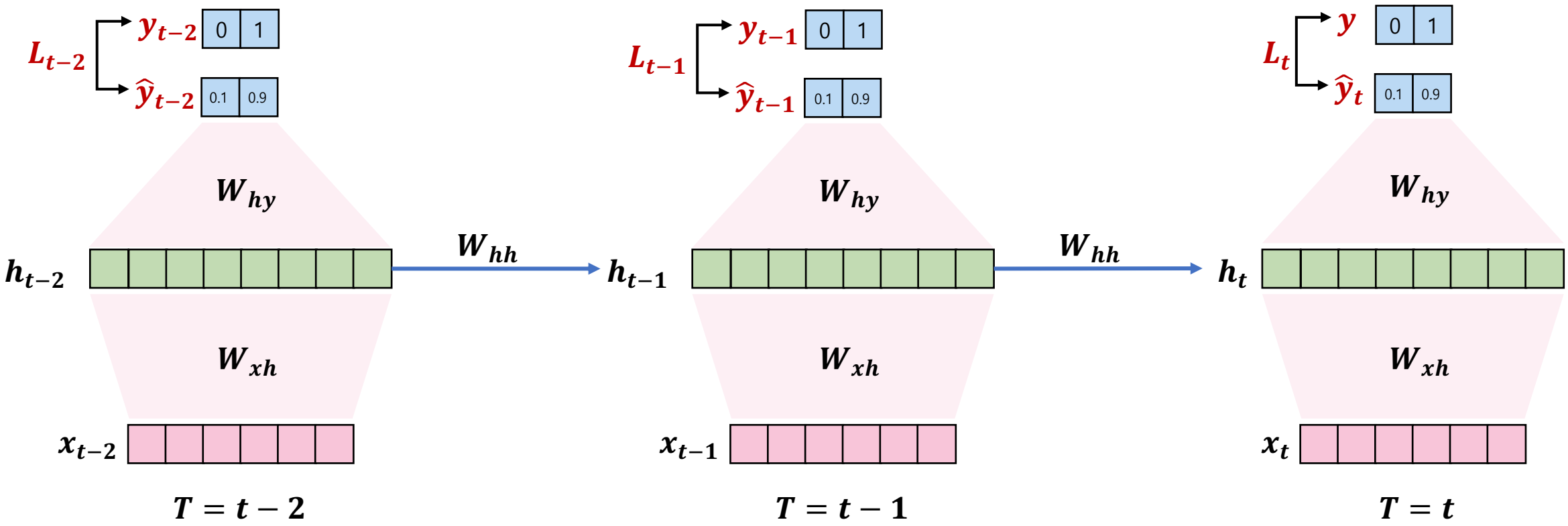
RNN training process

- 1) Forward propagation
  - $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1})$ ➔ Hidden state
  - $\hat{y}_t = \text{softmax}(W_{hy}h_t)$
  - $L_t = y$와 $\hat{y}_t$의 차이



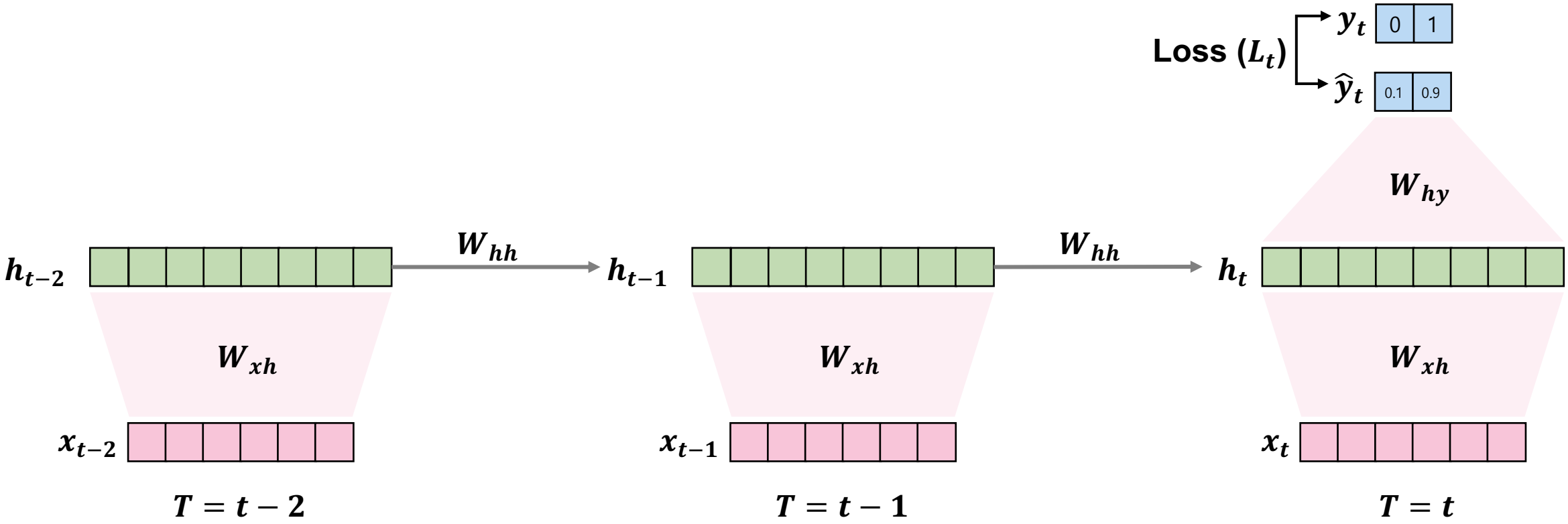
RNN training process

- 1) Forward propagation
  - L_t = y와 $\hat{y}_t$의 차이



RNN training process

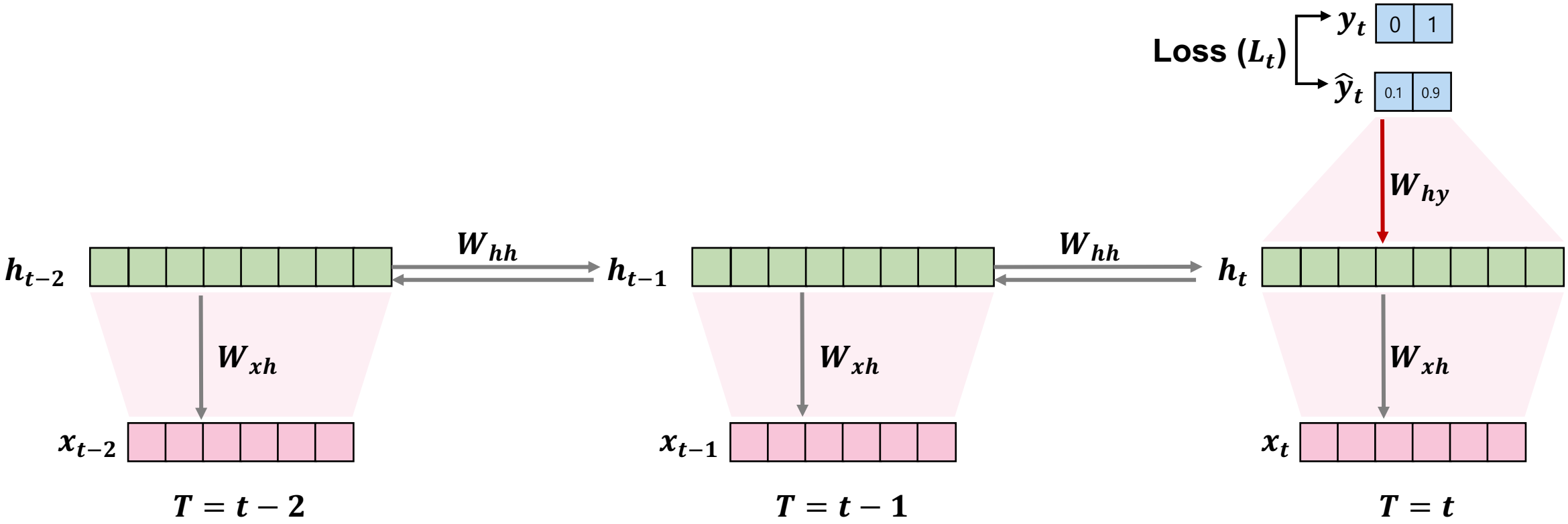
- 2) Backward propagation ➔ Backward Propagation Through Time (BPTT) in RNN
  - W_{xh}, W_{hh}, W_{hy} ➔ $\frac{\partial Loss}{\partial W_{hy}}$, $\frac{\partial Loss}{\partial W_{hh}}$, $\frac{\partial Loss}{\partial W_{xh}}$



RNN training process

- 2) Backward propagation

$$\frac{\partial Loss}{\partial W_{hy}} = \frac{\partial L_t}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial W_{hy} h_t} \times \frac{\partial W_{hy} h_t}{\partial W_{hy}}$$

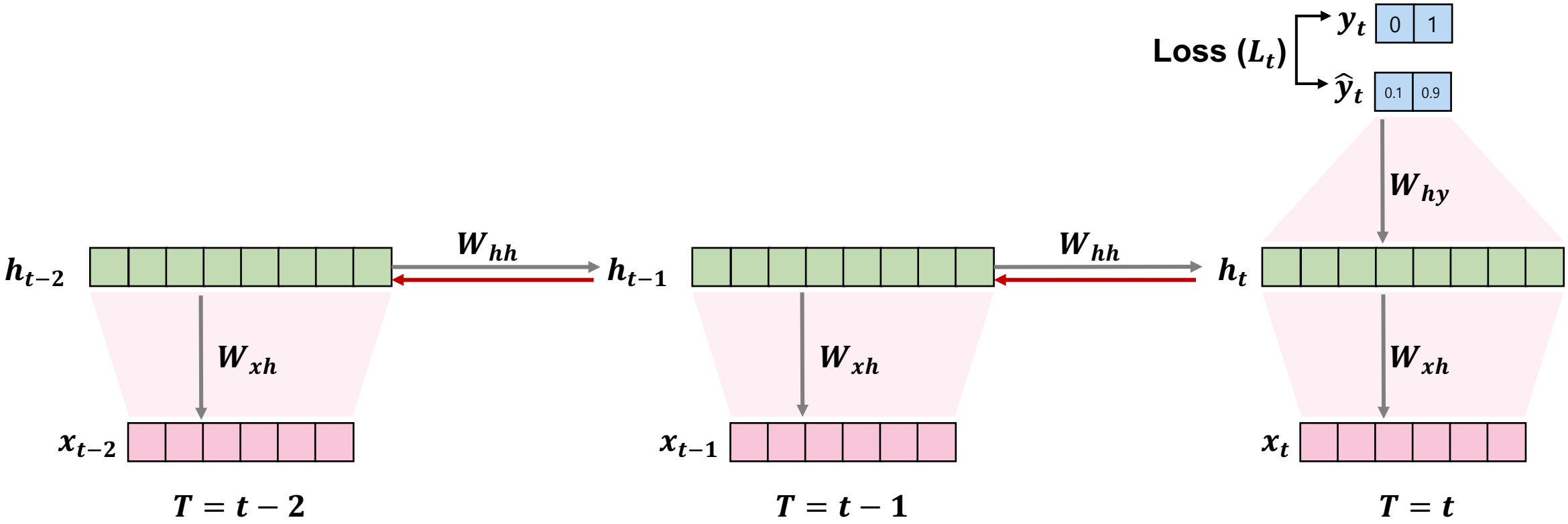


RNN training process

- 2) Backward propagation

$$\frac{\partial Loss}{\partial W_{hh}} = \left(\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial W_{hh}}\right) + \left(\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial W_{hh}}\right) + \left(\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial h_{t-2}} \times \frac{\partial h_{t-2}}{\partial W_{hh}}\right) + \cdots$$

시점 t 에서의 영향 | 시점 t 로부터 전해진 영향 | 시점 $t-1$ 로부터 전해진 영향

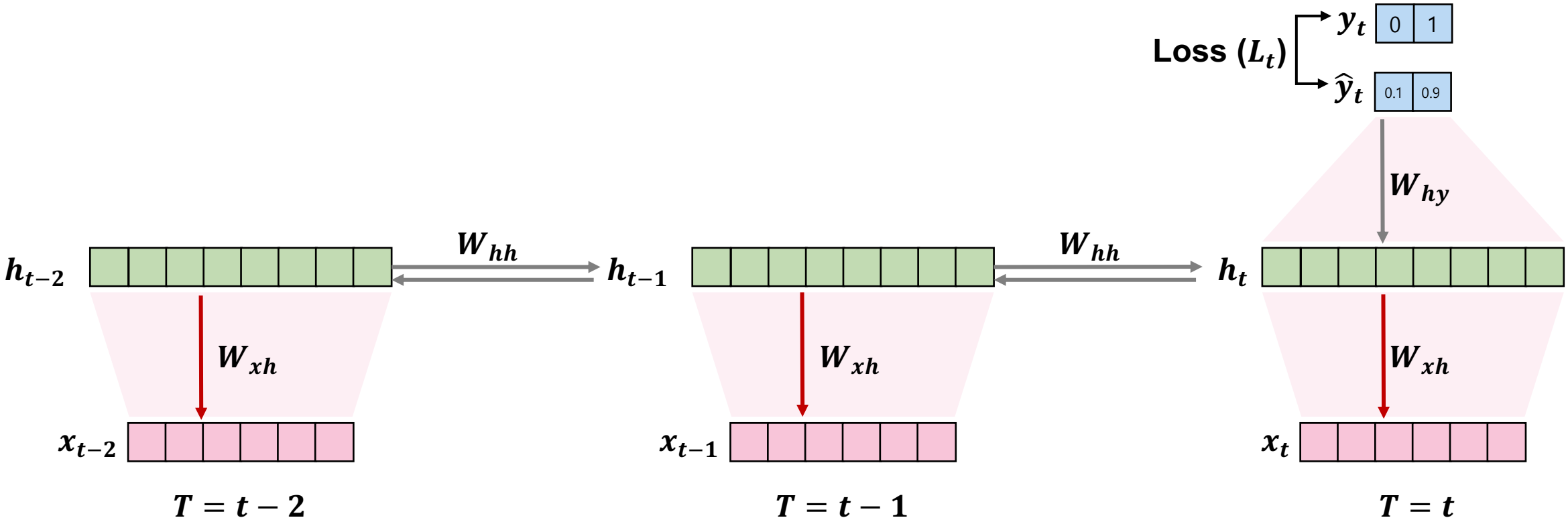


RNN training process

- 2) Backward propagation

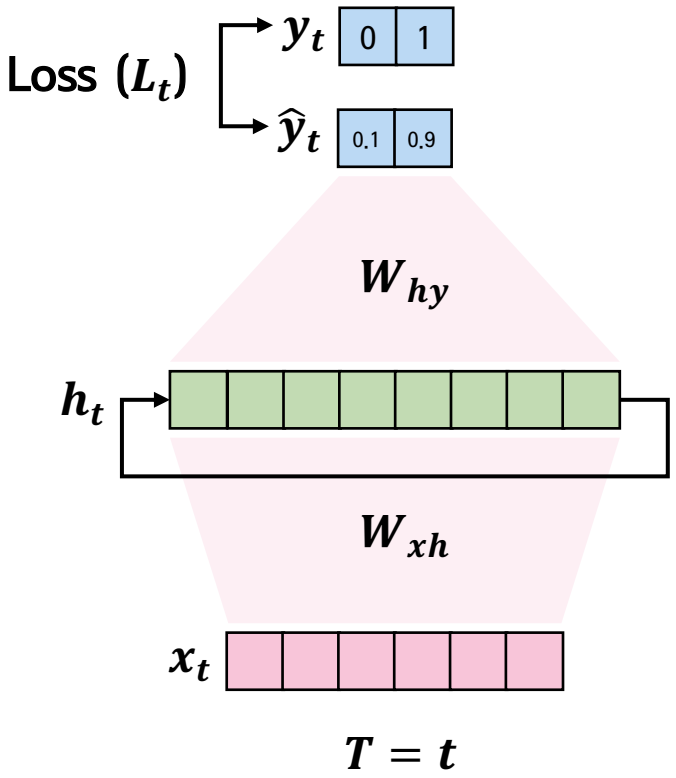
$$\frac{\partial Loss}{\partial W_{xh}} = \left(\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial W_{xh}}\right) + \left(\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial W_{xh}}\right) + \left(\frac{\partial Loss}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_{t-1}} \times \frac{\partial h_{t-1}}{\partial h_{t-2}} \times \frac{\partial h_{t-2}}{\partial W_{xh}}\right) + \cdots$$

시점 t 에서의 영향 | 시점 t 로부터 전해진 영향 | 시점 $t-1$ 로부터 전해진 영향



RNN training process

- 2) Backward propagation
  - W_{xh}, W_{hh}, W_{hy} ➔ $\frac{\partial Loss}{\partial W_{hy}}$, $\frac{\partial Loss}{\partial W_{hh}}$, $\frac{\partial Loss}{\partial W_{xh}}$



$$\frac{\partial Loss}{\partial W_{hy}} = \frac{\partial L_t}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial W_{hy} h_t} \times \frac{\partial W_{hy} h_t}{\partial W_{hy}}$$

$$\frac{\partial Loss}{\partial W_{hh}} = \sum_{k=0}^{t} \left(\frac{\partial L}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_k} \times \frac{\partial h_k}{\partial W_{hh}} \right)$$

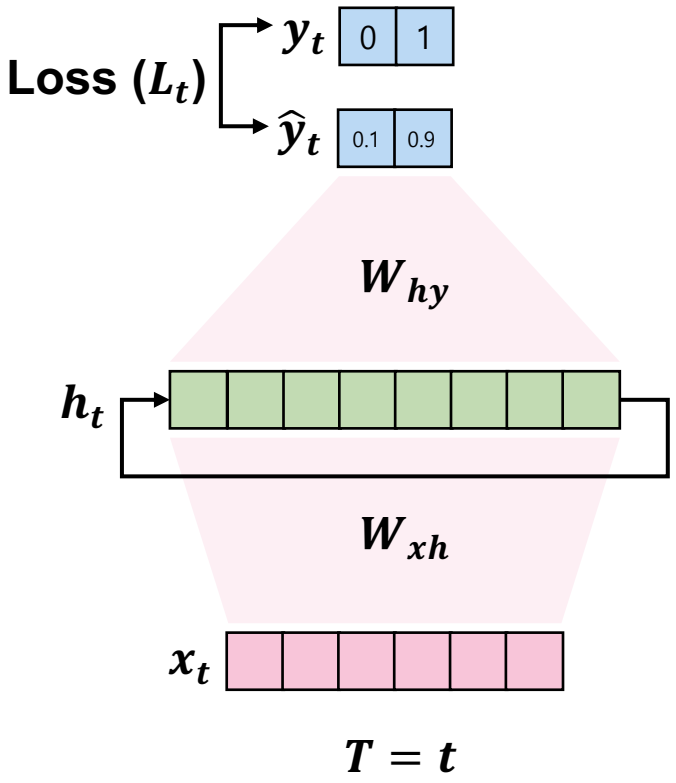
$$\frac{\partial Loss}{\partial W_{xh}} = \sum_{k=0}^{t} \left(\frac{\partial L}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_k} \times \frac{\partial h_k}{\partial W_{xh}} \right)$$

Loss function

- Regression problem ➔ MSE = $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
- Classification problem ➔ Cross Entropy = $\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$

RNN training process

- **3) Parameter update**
  - Parameter ➔ BPTT를 통해 gradient (기여도) 계산
  - Learning rate (η): 기여도를 얼마나 반영할 것인가



$$W_{hy}\text{의 기여도} = \frac{\partial Loss}{\partial W_{hy}} \rightarrow W_{hy}^{new} = W_{hy}^{old} - \eta \times \frac{\partial Loss}{\partial W_{hy}}$$

$$W_{hh}\text{의 기여도} = \frac{\partial Loss}{\partial W_{hh}} \rightarrow W_{hh}^{new} = W_{hh}^{old} - \eta \times \frac{\partial Loss}{\partial W_{hh}}$$

$$W_{xh}\text{의 기여도} = \frac{\partial Loss}{\partial W_{xh}} \rightarrow W_{xh}^{new} = W_{xh}^{old} - \eta \times \frac{\partial Loss}{\partial W_{xh}}$$

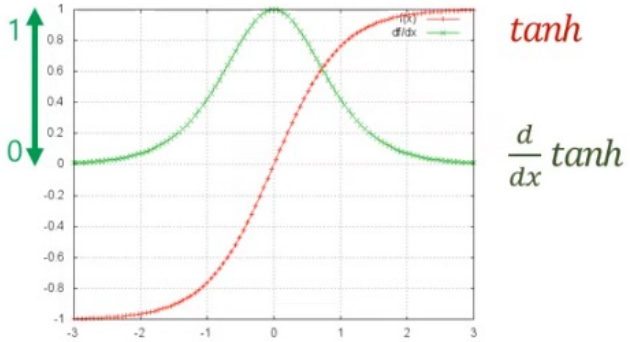
Limitations of RNN

- **Long-term dependency problem** (장기의존성 문제)
  - Sequence의 길이가 길어질수록, 과거 정보 학습에 어려움이 발생
  - ➔ By gradient vanishing ➔ Parameter가 업데이트가 안됨

$$\frac{\partial Loss}{\partial W_{hh}} \rightarrow W_{hh}^{new} = W_{hh}^{old} - \eta \times \frac{\partial Loss}{\partial W_{hh}}$$

  - Example) $T = 100$
  - $$\frac{\partial Loss}{\partial W_{hh}} = \left(\frac{\partial Loss}{\partial \hat{y}_{100}} \times \frac{\partial \hat{y}_{100}}{\partial h_{100}} \times \frac{\partial h_{100}}{\partial W_{hh}}\right) + \left(\frac{\partial Loss}{\partial \hat{y}_{100}} \times \frac{\partial \hat{y}_{100}}{\partial h_{100}} \times \frac{\partial h_{100}}{\partial h_{99}} \times \frac{\partial h_{99}}{\partial W_{hh}}\right) + \left(\frac{\partial Loss}{\partial \hat{y}_{100}} \times \frac{\partial \hat{y}_{100}}{\partial h_{100}} \times \frac{\partial h_{100}}{\partial h_{99}} \times \frac{\partial h_{99}}{\partial h_{98}} \times \frac{\partial h_{98}}{\partial W_{hh}}\right) + \cdots$$

$$+ \left(\frac{\partial Loss}{\partial \hat{y}_{100}} \times \frac{\partial \hat{y}_{100}}{\partial h_{100}} \times \frac{\partial h_{100}}{\partial h_{99}} \times \frac{\partial h_{99}}{\partial h_{98}} \times \cdots \times \frac{\partial h_6}{\partial h_5} \times \frac{\partial 5}{\partial h_4} \times \frac{\partial h_4}{\partial h_3} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial W_{hh}}\right)$$

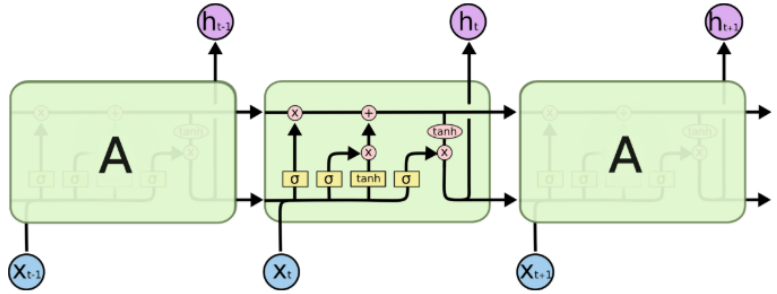


$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1})$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{xh}x_t + W_{hh}h_{t-1})\, W_{hh}$$

LSTM and GRU의 등장

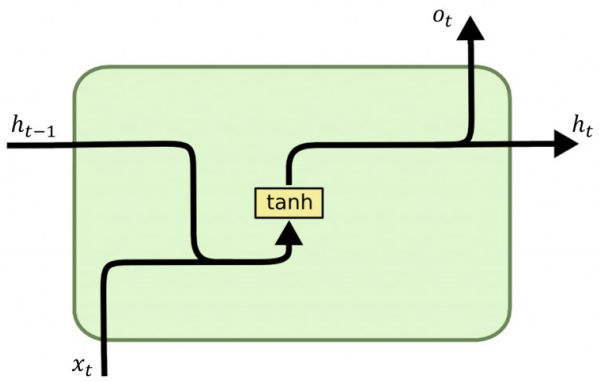
- **Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)**
  - RNN의 long-term dependency problem를 완화한 모델



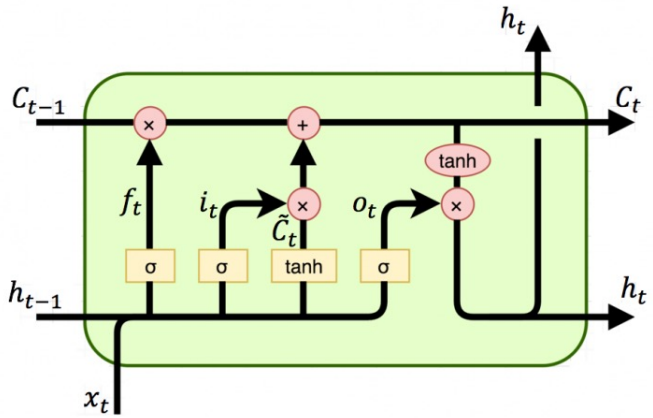
The repeating module in an LSTM contains four interacting layers.

  - Structure
    - Three gates: Forget gate (f_t), Input gate (i_t), Output gate (o_t)

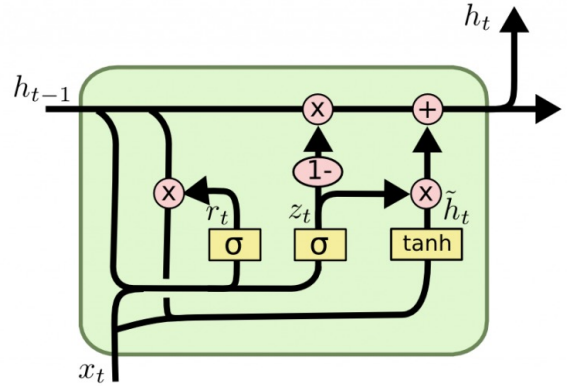
RNN



LSTM

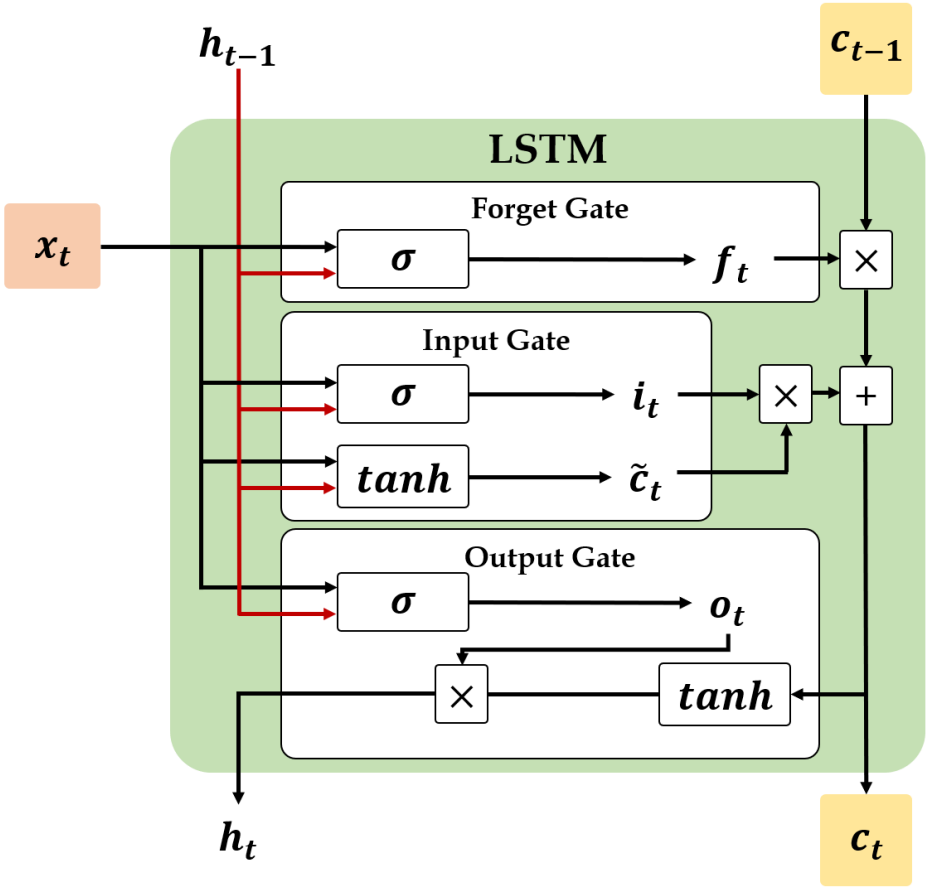


GRU



LSTM

- Hidden state h_t: 단기적인 정보를 제공 (Short term)
- Cell state c_t : 장기적으로 정보를 유지 (Long term)



$$f_t = \sigma\left(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f}\right)$$

$$i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}\right)$$

$$o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}\right)$$

$$\tilde{c}_t = \tanh\left(W_{xh_g}x_t + W_{hh_g}h_{t-1} + b_{h_g}\right)$$

$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$$

$$h_t = o_t \otimes \tanh(c_t)$$

LSTM training process

1) Gate 계산: Forget gate (f_t), Input gate (i_t), Output gate (o_t)

- $\boldsymbol{f_t = \sigma\left(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f}\right)}$
- $\boldsymbol{i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}\right)}$
- $\boldsymbol{o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}\right)}$

2) Cell state (c_t) 업데이트

- $\tilde{c}_t = \tanh\left(W_{xh_g}x_t + W_{hh_g}h_{t-1} + b_{h_g}\right)$
- $c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$

3) Hidden state (h_t) 업데이트

- $h_t = o_t \otimes \tanh(c_t)$

LSTM training process

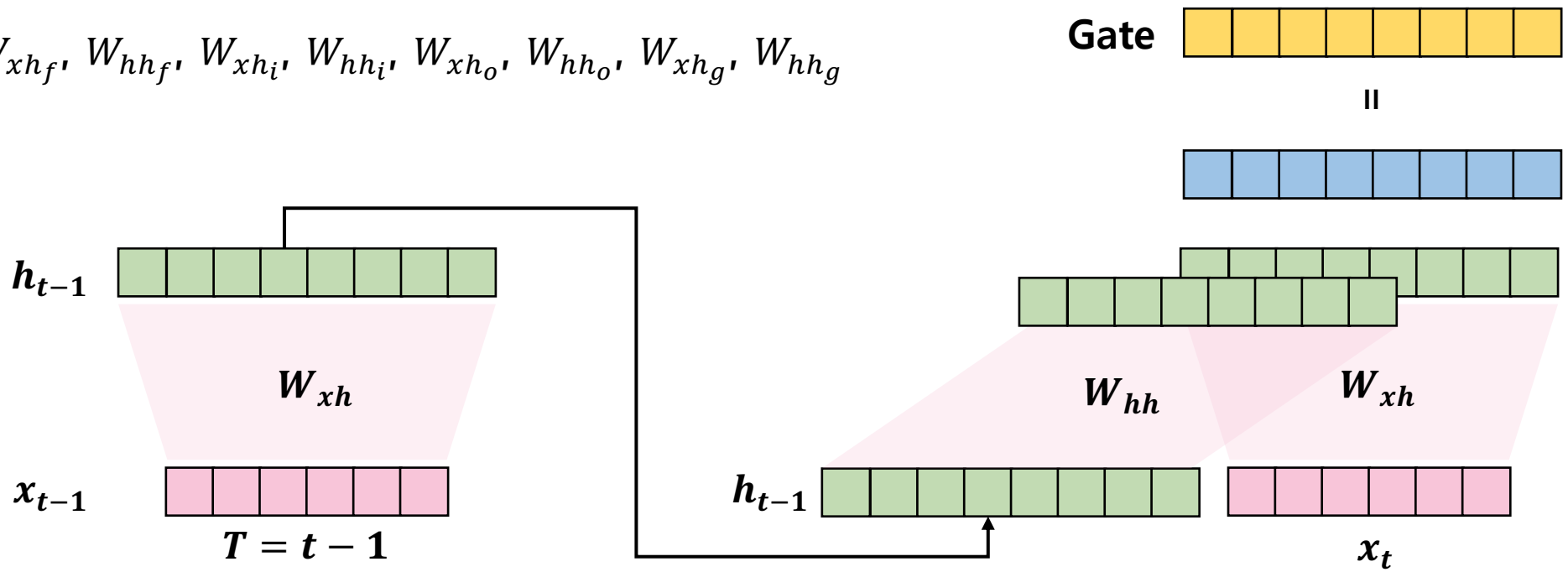
1) Gate 계산

- Gate: 해당 정보를 얼마나 적용(기억)할 것인지에 대한 가중치
- **Gate = $\sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$**
  - 0~1 사이의 값을 갖는 벡터
- Gate마다 서로 다른 파라미터 (W_{xh}, W_{hh})를 사용하기 때문에 서로 다른 값
  - W_{xh_f}, W_{hh_f}, W_{xh_i}, W_{hh_i}, W_{xh_o}, W_{hh_o}, W_{xh_g}, W_{hh_g}

$$f_t = \sigma\left(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f}\right)$$

$$i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}\right)$$

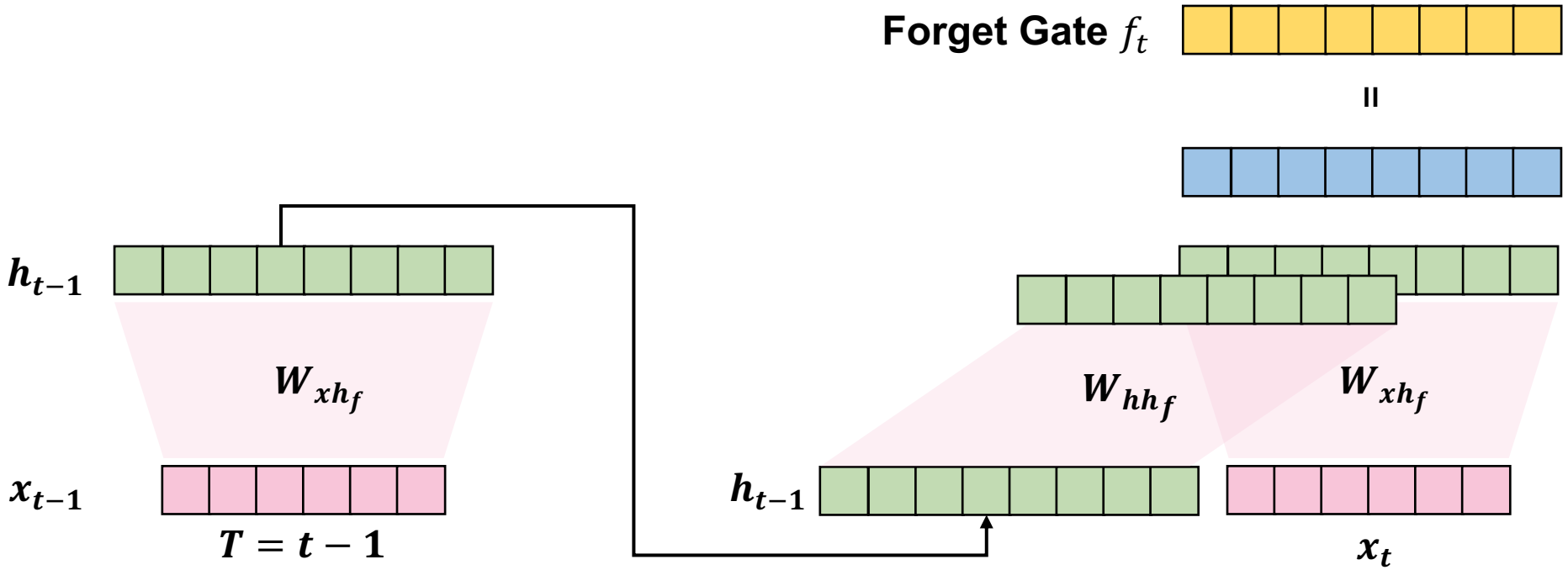
$$o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}\right)$$



LSTM training process

1) Gate 계산 – Forget gate

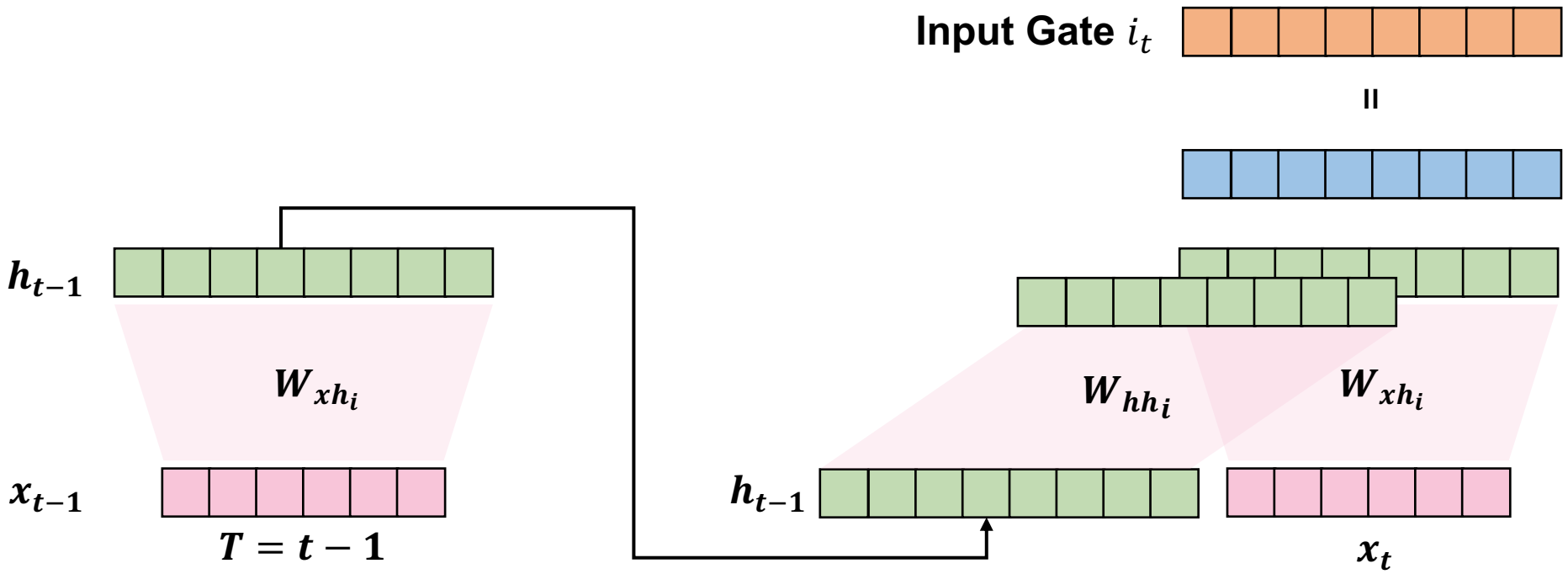
- 과거 cell state에서 사용하지 않을 데이터에 대한 가중치
- **Forget Gate** $f_t = \left(W_{xh_f} x_t + W_{hh_f} h_{t-1} + b_{h_f}\right)$



LSTM training process

1) Gate 계산 – Input gate

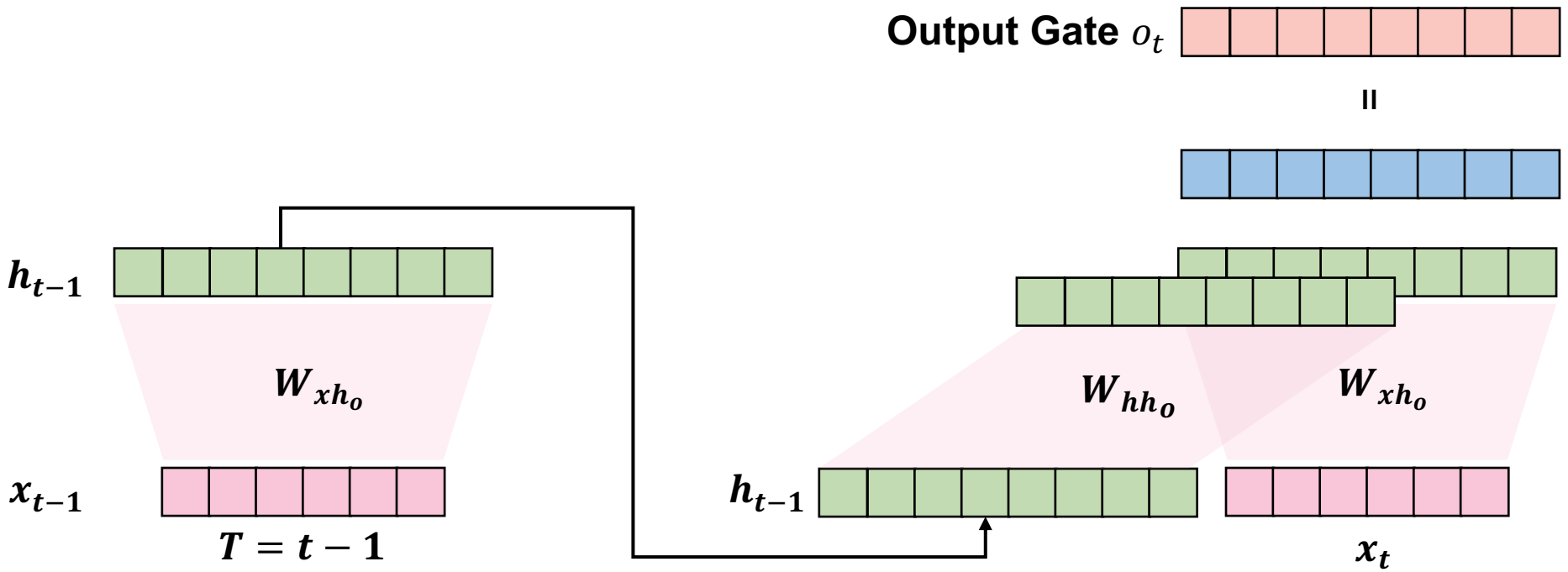
- Cell state에서 사용할 데이터를 저장하기 위한 가중치
- **Input Gate** $i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}\right)$



LSTM training process

1) Gate 계산 – Output gate

- Hidden state에 cell state를 얼마나 반영할 것인지에 대한 가중치
- **Output Gate** $o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}\right)$



LSTM training process

1) Gate 계산: Forget gate (f_t), Input gate (i_t), Output gate (o_t)

- $\boldsymbol{f_t = \sigma\left(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f}\right)}$
- $\boldsymbol{i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}\right)}$
- $\boldsymbol{o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}\right)}$

2) Cell state (c_t) 업데이트

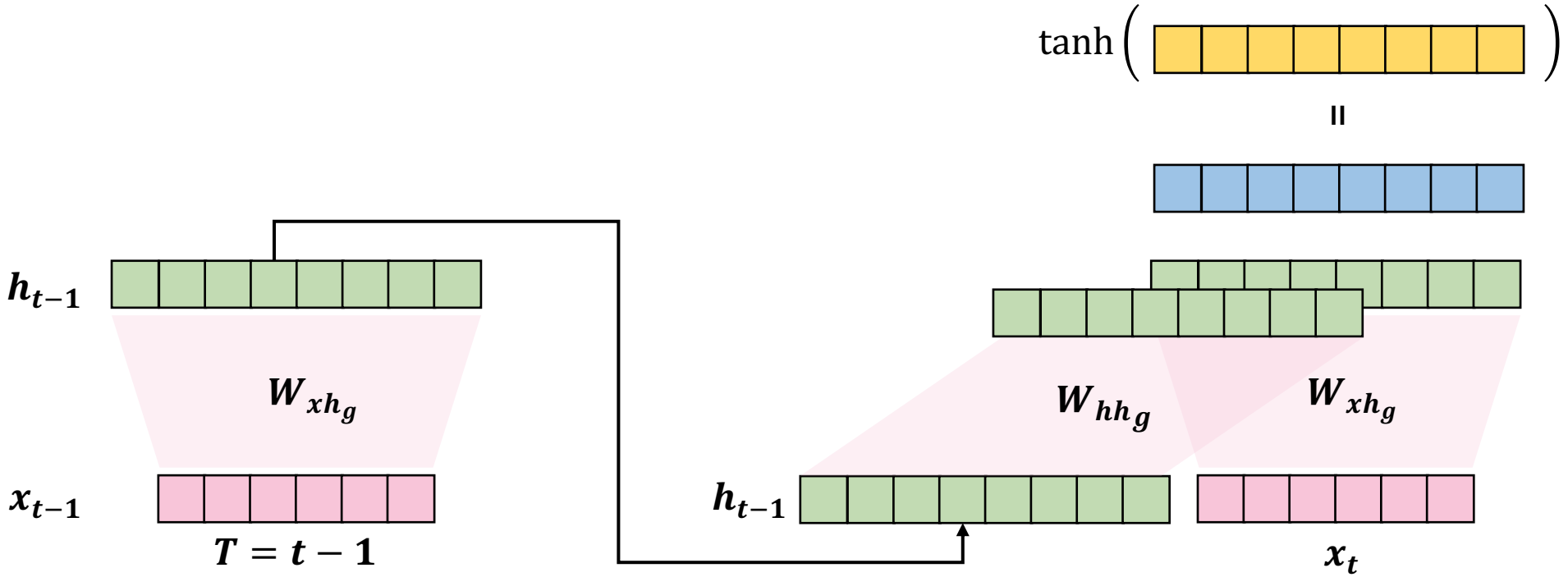
- $\tilde{c}_t = \tanh\left(W_{xh_g}x_t + W_{hh_g}h_{t-1} + b_{h_g}\right)$
- $c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$

3) Hidden state (h_t) 업데이트

- $h_t = o_t \otimes \tanh(c_t)$

LSTM training process

- 2) Cell state 업데이트
  - 임시 cell state ($\tilde{c}_t$) 생성
    - 임시 cell state: 현재 입력값과 과거 hidden state 정보에 대한 요약
    - $\tilde{c}_t = \tanh\left(W_{xh_g} x_t + W_{hh_g} h_{t-1} + b_{h_g}\right)$



LSTM training process

- 2) Cell state 업데이트
  - 현 시점에 대한 cell state (c_t)를 업데이트 ➔ forget gate, input gate활용
  - $c_t = \boldsymbol{f_t} \otimes \boldsymbol{c_{t-1}} \oplus i_t \otimes \tilde{c}_t$
    - ⊗: element-wise product

- Forget gate: 불필요한 과거 정보를 잊기 위한 gate

$\boldsymbol{f_t}$	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{c_{t-1}}$	-0.2	0.1	0.5	0.7	0.9	-0.3	-1.1	2.0	0.6	0.7
$\boldsymbol{f_t \otimes c_{t-1}}$	-0.2	0.1	0.5	0.7	0.9	-0.3	-1.1	2.0	0.6	0.7

다 기억하는 경우

LSTM training process

- 2) Cell state 업데이트
  - 현 시점에 대한 cell state (c_t)를 업데이트 ➔ forget gate, input gate활용
  - $c_t = \boldsymbol{f_t} \otimes \boldsymbol{c_{t-1}} \oplus i_t \otimes \tilde{c}_t$
    - ⊗: element-wise product

  - Forget gate: 불필요한 과거 정보를 잊기 위한 gate

$\boldsymbol{f_t}$	0	0	0	0	0	0	0	0	0	0
$\boldsymbol{c_{t-1}}$	-0.2	0.1	0.5	0.7	0.9	-0.3	-1.1	2.0	0.6	0.7
$\boldsymbol{f_t \otimes c_{t-1}}$	0	0	0	0	0	0	0	0	0	0

다 잊는 경우

LSTM training process

- 2) Cell state 업데이트
  - 현 시점에 대한 cell state (c_t)를 업데이트 ➔ forget gate, input gate활용
  - $c_t = \boldsymbol{f_t} \otimes \boldsymbol{c_{t-1}} \oplus i_t \otimes \tilde{c}_t$
    - ⊗: element-wise product

- Forget gate: 불필요한 과거 정보를 잊기 위한 gate

$\boldsymbol{f_t}$	0	0.2	0.9	0.1	1	0.2	0.1	0.9	0.4	0.1
$\boldsymbol{c_{t-1}}$	-0.2	0.1	0.5	0.7	0.9	-0.3	-1.1	2.0	0.6	0.7
$\boldsymbol{f_t \otimes c_{t-1}}$	0	0.02	0.45	0.07	0.9	-0.06	-0.11	1.8	0.24	0.07

LSTM training process

- 2) Cell state 업데이트
  - 현 시점에 대한 cell state (c_t)를 업데이트 ➔ forget gate, input gate활용
  - $c_t = \boldsymbol{f_t} \otimes \boldsymbol{c_{t-1}} \oplus i_t \otimes \tilde{c}_t$
    - ⊗: element-wise product

- Input gate: 현재 정보를 기억하기 위한 gate

$\boldsymbol{i_t}$	0.1	0	0.8	0.2	0.8	0.7	1	0.9	0.2	0.4
$\tilde{\boldsymbol{c}}_t$	-0.1	0.3	6	0.2	0.9	0.1	0.4	2.1	0.6	0.9
$\boldsymbol{i_t} \otimes \tilde{\boldsymbol{c}}_t$	-0.01	0	0.48	0.04	0.72	0.07	0.4	1.89	0.12	0.36

LSTM training process

- 2) Cell state 업데이트
  - 현 시점에 대한 cell state (c_t)를 업데이트 ➔ forget gate, input gate활용
  - $c_t = \boldsymbol{f_t} \otimes \boldsymbol{c_{t-1}} \oplus i_t \otimes \tilde{c}_t$
    - ⊗: element-wise product

- 불필요한 정보는 잊고, 추가할 정보는 추가해서 cell state 구성

$\boldsymbol{f_t} \otimes \boldsymbol{c_{t-1}}$	0	0.02	0.45	0.07	0.9	-0.06	-0.11	1.8	0.24	0.07
$\boldsymbol{i_t} \otimes \boldsymbol{\tilde{c}_t}$	-0.01	0	0.48	0.04	0.72	0.07	0.4	1.89	0.12	0.36
$\boldsymbol{c_t}$	-0.01	0.02	0.93	0.11	1.62	0.01	0.29	3.69	0.36	0.43

LSTM training process

1) Gate 계산: Forget gate (f_t), Input gate (i_t), Output gate (o_t)

- $\boldsymbol{f_t = \sigma\left(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f}\right)}$
- $\boldsymbol{i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}\right)}$
- $\boldsymbol{o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}\right)}$

2) Cell state (c_t) 업데이트

- $\tilde{c}_t = \tanh\left(W_{xh_g}x_t + W_{hh_g}h_{t-1} + b_{h_g}\right)$
- $c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$

3) Hidden state (h_t) 업데이트

- $h_t = o_t \otimes \tanh(c_t)$

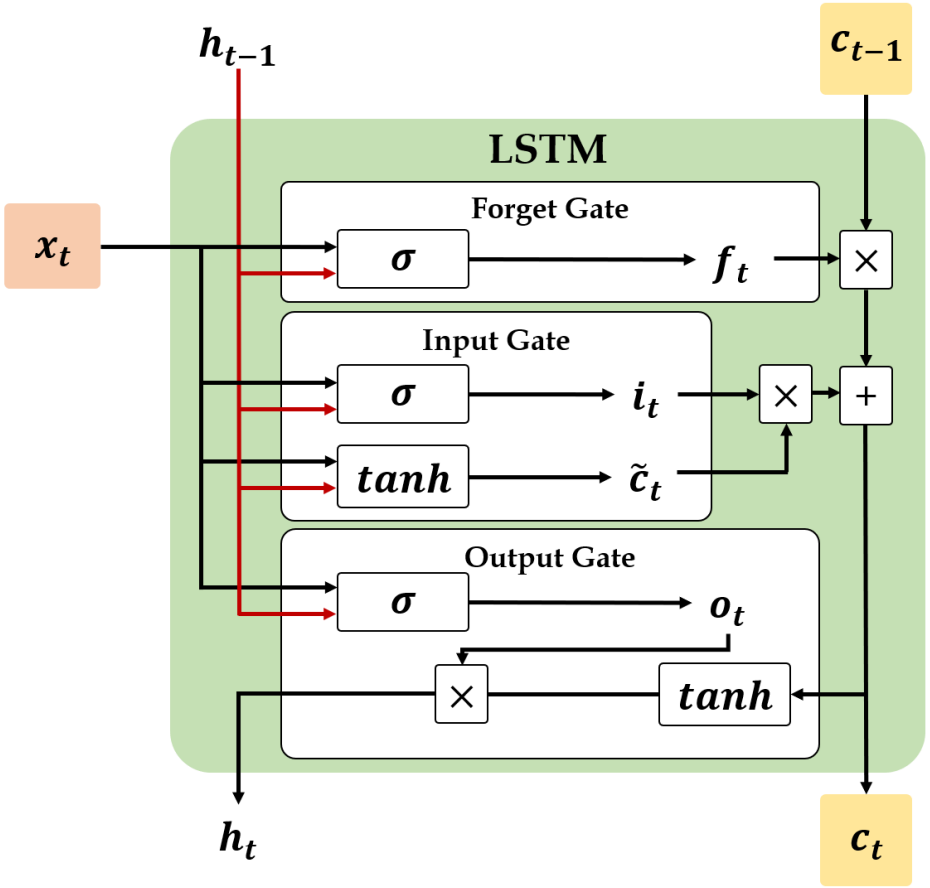
LSTM training process

- **3) Hidden state 업데이트**
  - 최종적으로 단기적 정보 h_t 업데이트
  - Output gate = 어떤 정보를 output으로 보낼지 결정하기 위한 gate
    - ➔ Hidden state에 cell state를 얼마나 반영할 것인지에 대한 가중치
  - $\boldsymbol{h_t = o_t \otimes \tanh(c_t)}$

$\boldsymbol{o_t}$	0.5	0.4	0.1	0.9	0.2	0.3	0.8	0.7	0.1	0.9
$\boldsymbol{c_t}$	-0.01	0.02	0.93	0.11	1.62	0.01	0.29	3.69	0.36	0.43
$\boldsymbol{\tanh(c_t)}$	-0.01	0.02	0.73	0.11	0.92	0.01	0.28	1.00	0.35	0.41
$\boldsymbol{h_t}$	0	0.08	0.073	0.099	0.184	0.003	0.224	0.7	0.035	0.369

LSTM summary

- Hidden state h_t: 단기적인 정보를 제공 (Short term)
- Cell state c_t : 장기적으로 정보를 유지 (Long term)



$$f_t = \sigma\left(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f}\right)$$

$$i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i}\right)$$

$$o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o}\right)$$

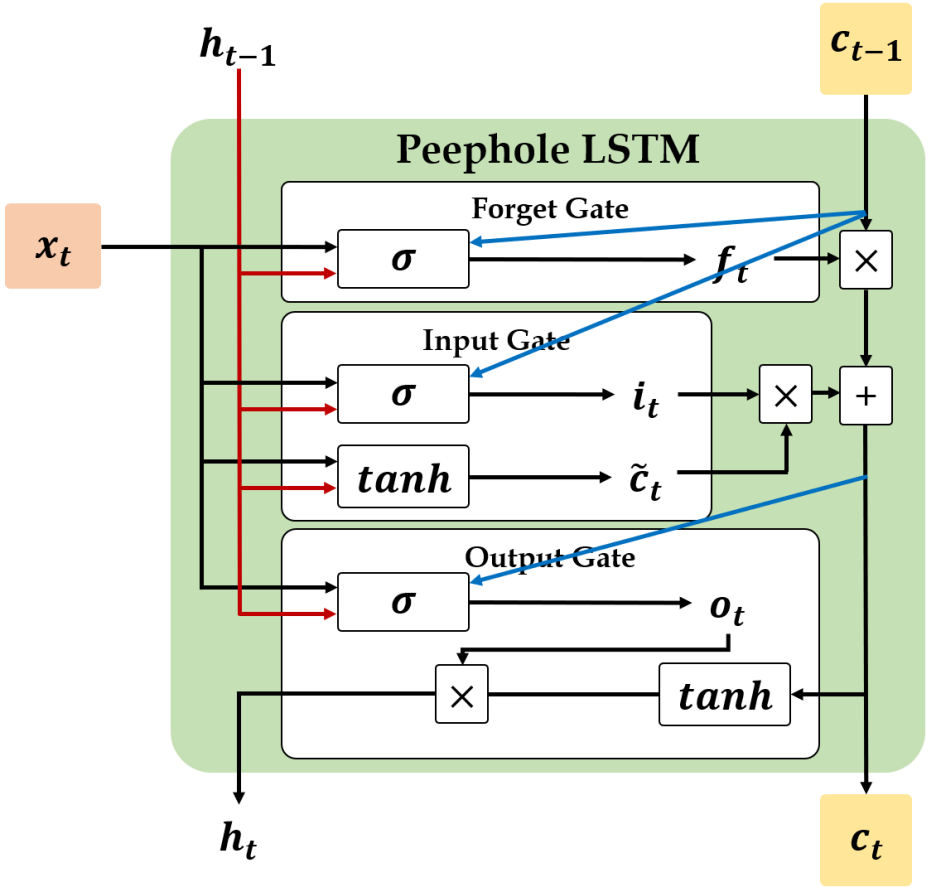
$$\tilde{c}_t = \tanh\left(W_{xh_g}x_t + W_{hh_g}h_{t-1} + b_{h_g}\right)$$

$$c_t = f_t \otimes c_{t-1} \oplus i_t \otimes \tilde{c}_t$$

$$h_t = o_t \otimes \tanh(c_t)$$

LSTM: Peephole connection

- General LSTM ➔ gate는 x_t와 단기정보 (hidden state) h_{t-1}만을 고려
- **Peephole connection**
  - Gate에 장기 정보 (cell state) c_t도 활용하여 더 많은 정보가 수용되도록 개선



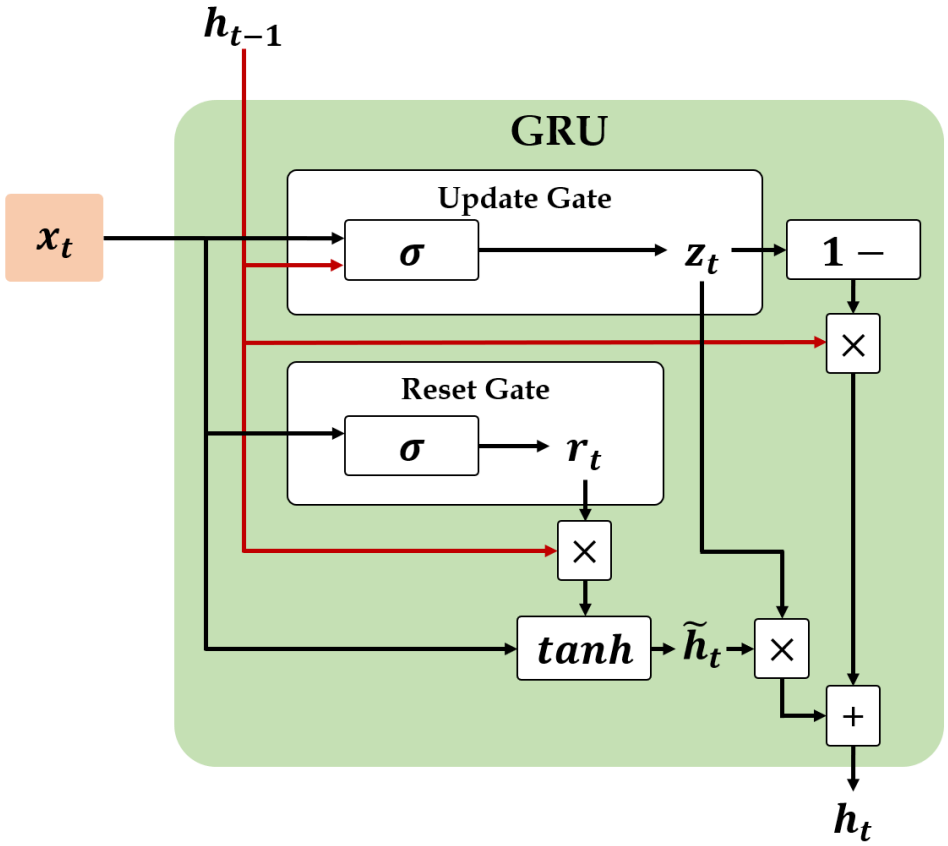
$$f_t = \sigma\left(W_{xh_f}x_t + W_{hh_f}h_{t-1} + W_{ch_f}c_{t-1} + b_{h_f}\right)$$

$$i_t = \sigma\left(W_{xh_i}x_t + W_{hh_i}h_{t-1} + W_{ch_i}c_{t-1} + b_{h_i}\right)$$

$$o_t = \sigma\left(W_{xh_o}x_t + W_{hh_o}h_{t-1} + W_{ch_o}c_t + b_{h_o}\right)$$

GRU

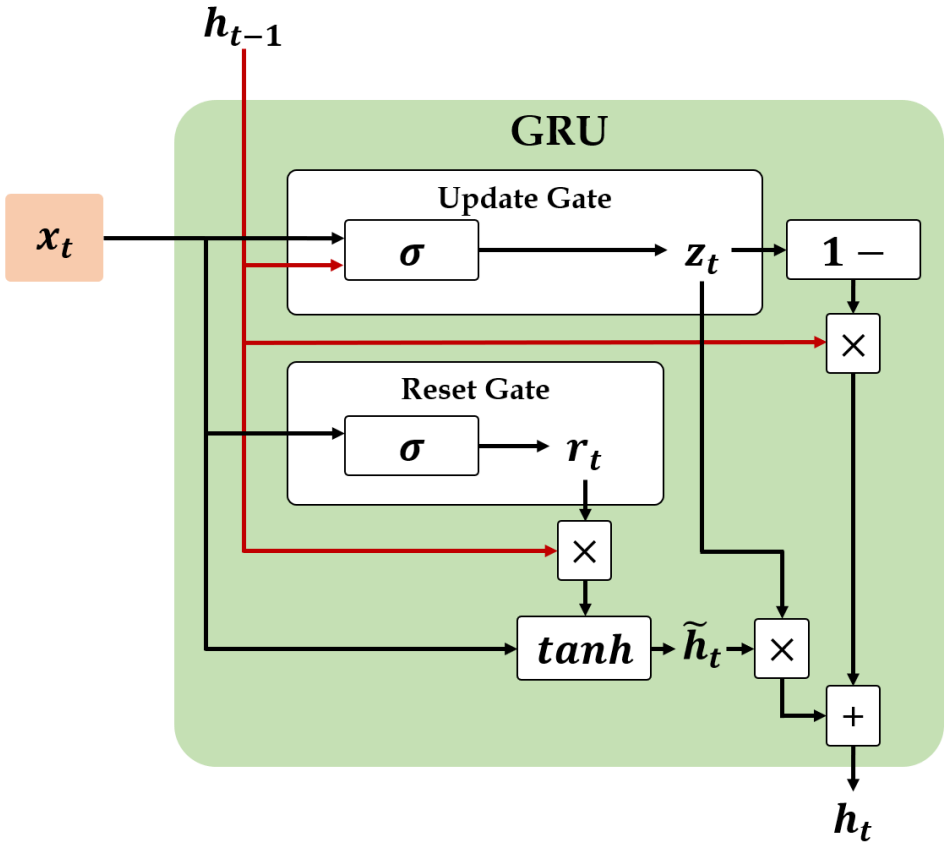
- LSTM의 구조를 간단하게 개선 ➔ parameter 개수를 줄임
  - Forget gate, input gate ➔ update gate (z_t), Output gate ➔ reset gate (r_t)
  - Cell state, hidden state ➔ hidden state



GRU

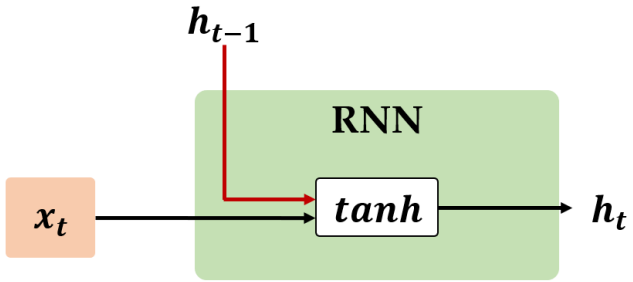
- Update gate $z_t = \sigma\left(W_{xh_z}x_t + W_{hh_z}h_{t-1} + b_{h_z}\right)$
- Reset gate $r_t = \sigma\left(W_{xh_r}x_t + W_{hh_r}h_{t-1} + b_{h_r}\right)$
- Cell state, hidden state ➔ hidden state $h_t = \left((1 - z_t)\otimes h_{t-1}\right)\oplus\left(z_t\otimes\tilde{h}_t\right)$

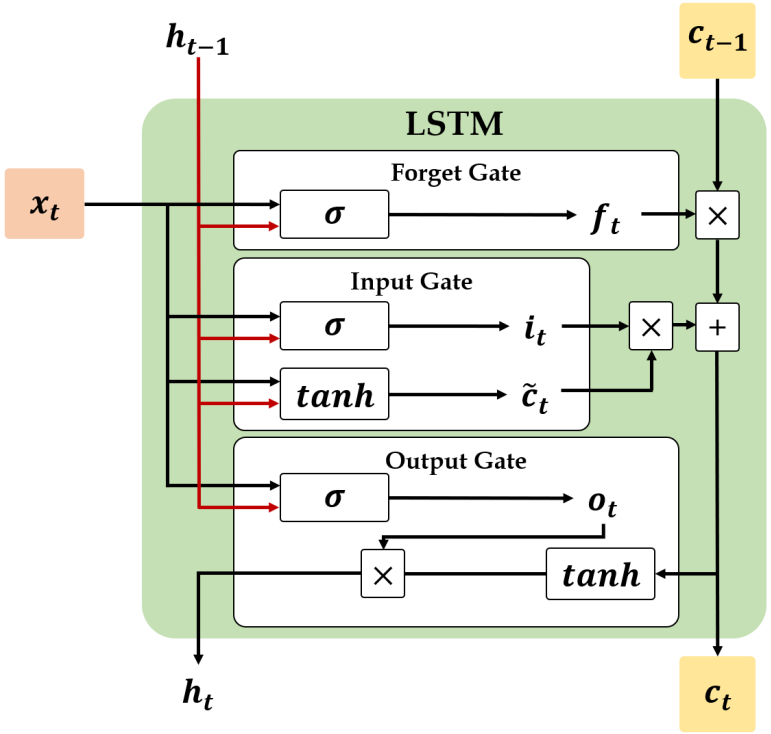
Forget gate 역할 Input gate 역할

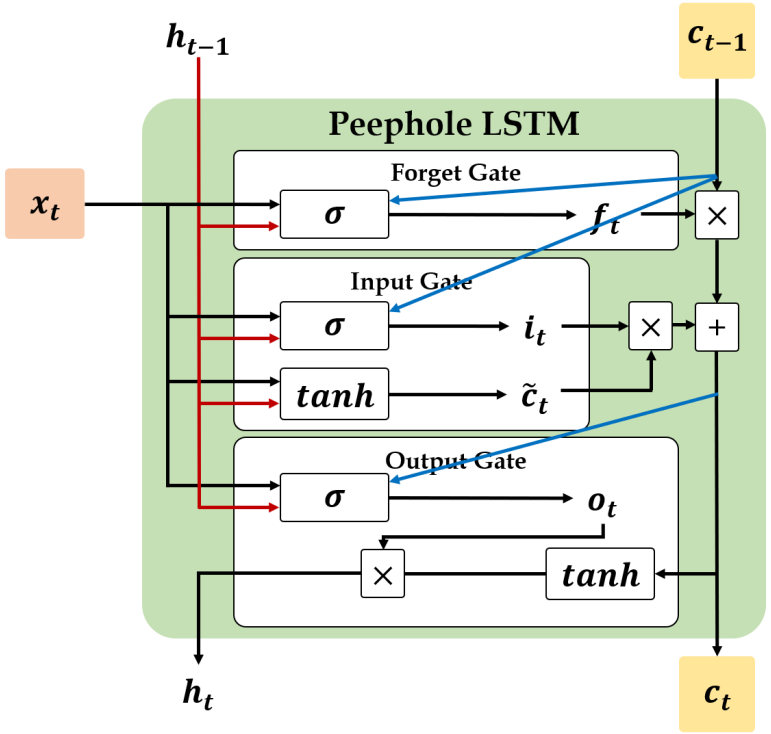


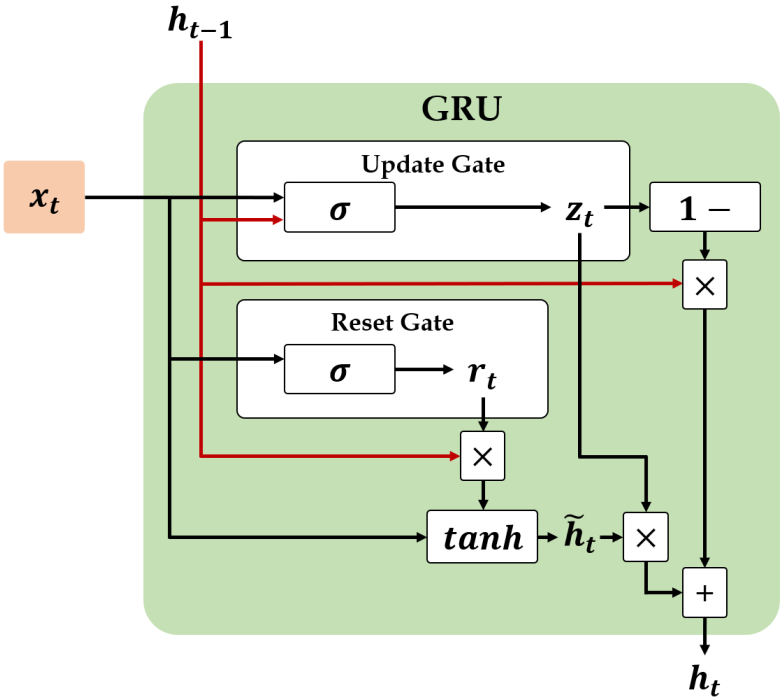
임시 hidden state $\tilde{h}_t = \tanh\left(W_{xh_g}x_t + W_{hh_g}(r_t\otimes h_{t-1}) + b_{h_g}\right)$

RNN Summary


h_{t-1}
RNN
x_t
$tanh$
h_t


h_{t-1}
c_{t-1}
LSTM
x_t
Forget Gate
σ
f_t
Input Gate
σ
i_t
$tanh$
$\tilde{c}_t$
Output Gate
σ
o_t
$tanh$
h_t
c_t


h_{t-1}
c_{t-1}
Peephole LSTM
x_t
Forget Gate
σ
f_t
Input Gate
σ
i_t
$tanh$
$\tilde{c}_t$
Output Gate
σ
o_t
$tanh$
h_t
c_t

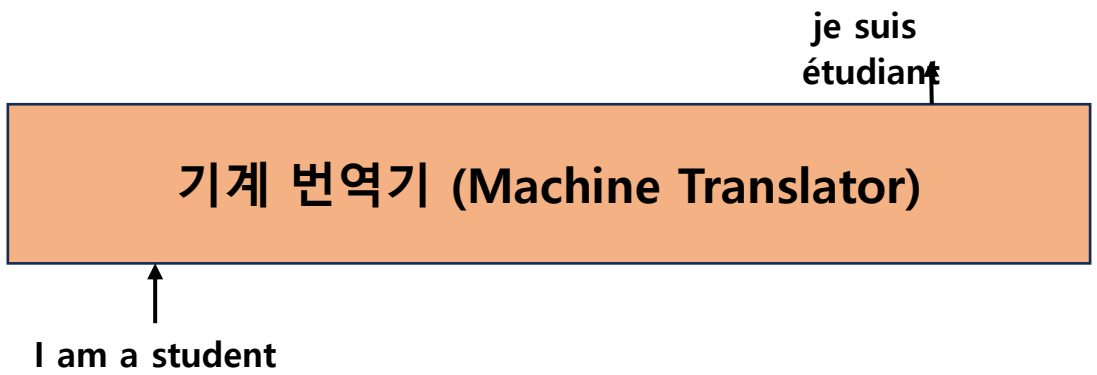

h_{t-1}
GRU
x_t
Update Gate
σ
z_t
1 −
Reset Gate
σ
r_t
$tanh$
$\tilde{h}_t$
h_t

5. Transformer

Background

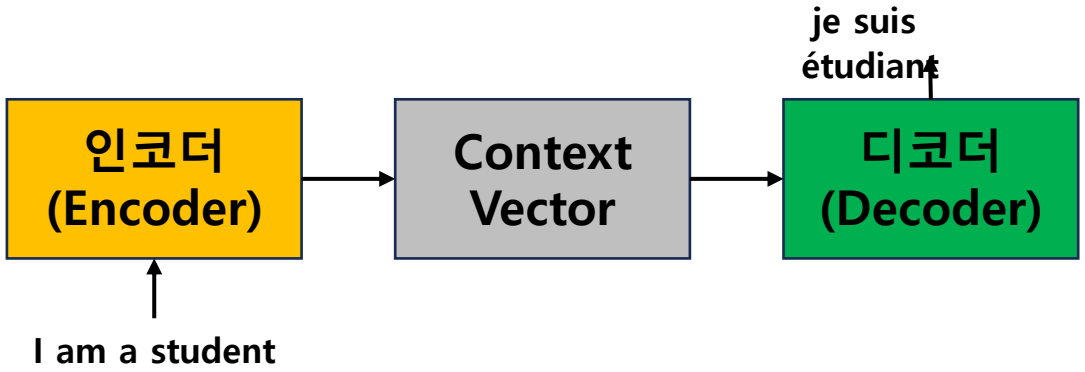
- DNN 기반 모델이 시계열 데이터 처리 분야에서도 이 준수한 성과를 보임
  - Time series (시계열) 데이터: 소리 데이터, 자연어 등의 언어 정보 등
  - RNN (Recurrent Neural Network)
  - LSTM (Long-Short Term Memory)
  - GRU (Gated Recurrent Unit)

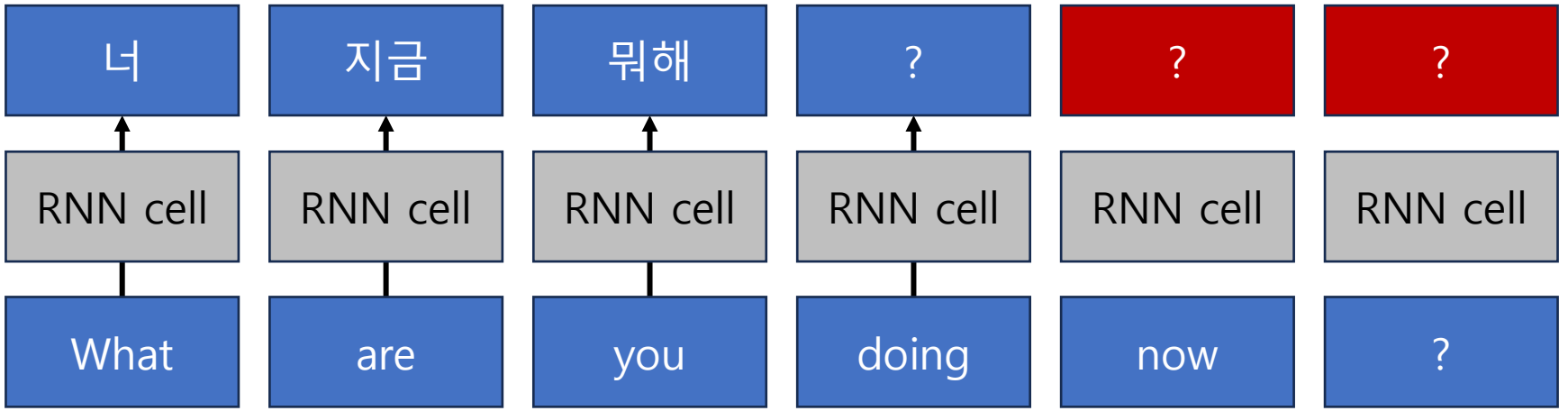
  - 그러나, 모델의 입력/출력 크기가 고정된다는 명확한 한계가 존재
    - 자연어, 소리 등은 가변적인 길이의 입출력을 가짐



Seq2Seq 구조

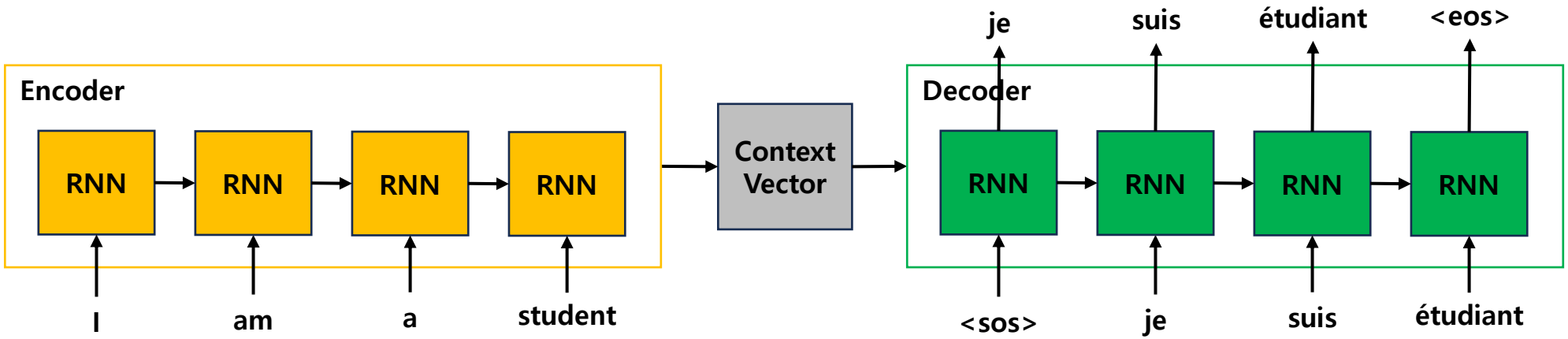
- Seq2Seq (Sequence To Sequence)의 등장
  - By Google 2014
    - Paper: https://arxiv.org/abs/1409.3215
  - Limitations of RNN ➔ Not work for variable length
  - Encoder/decoder for the variable length input/output by using two LSTM networks
    - Input: Word sequence of fixed length with LSTM/GRU-based structure
    - Output: Sequence of length appropriate for the input sequence





Seq2Seq 구조

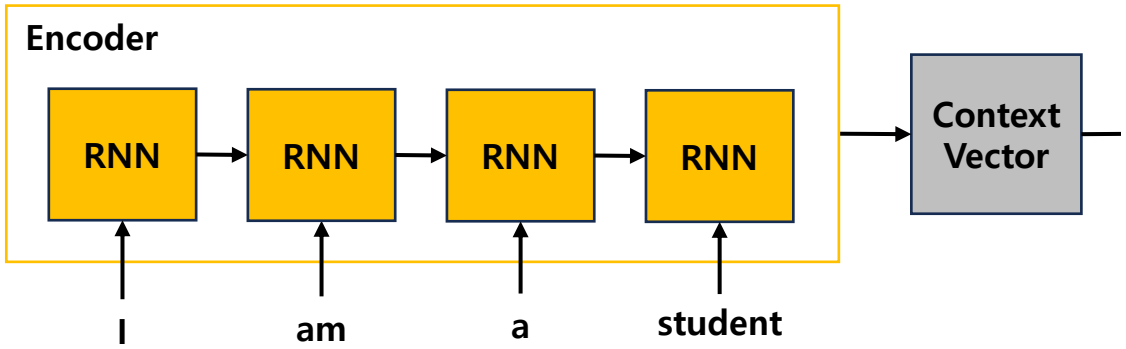
- Encoder
  - 입력 문장의 모든 단어들을 순차적으로 입력받은 후
  - 마지막에 모든 단어 정보를 압축해서 하나의 벡터로 만듦 (encoding)
    - ➔ Context vector
- Decoder
  - Context vector를 받아서 번역된 단어를 한 개 씩 순차적으로 출력
  - ➔ Decoder가 출력 문장을 생성(decoding)하는 방식으로 서로 다른 입/출력 시퀀스에 대응



Seq2Seq 구조

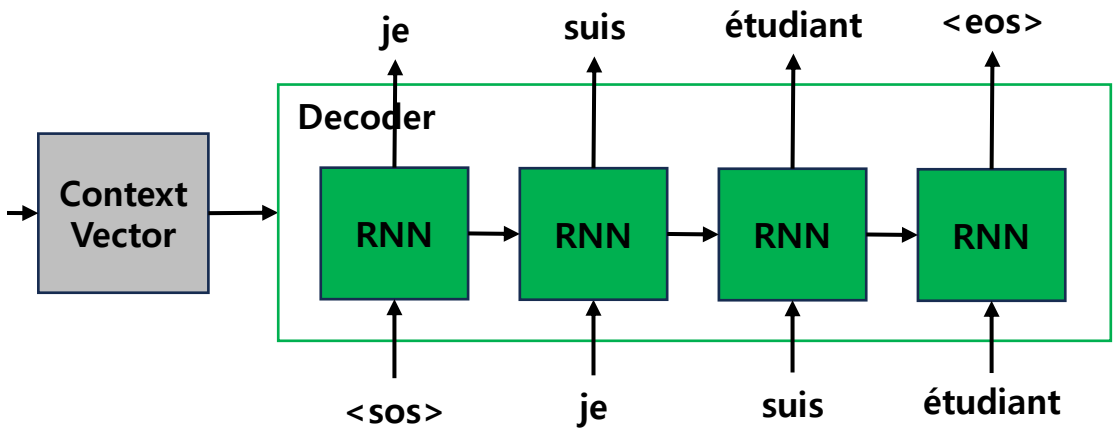
- **Encoder**
  - 입력 문장을 context vector에 encoding (압축)하는 역할
  - Encoder의 LSTM은 입력 문장을 단어 순서대로 처리 ➔ 고정된 크기의 context vector 생성

- **Context vector**
  - Encoder의 마지막 스텝에서 출력된 hidden state와 같음
  - 입력 문장의 정보를 함축하는 벡터 ➔ 문장 수준의 벡터로 활용 가능



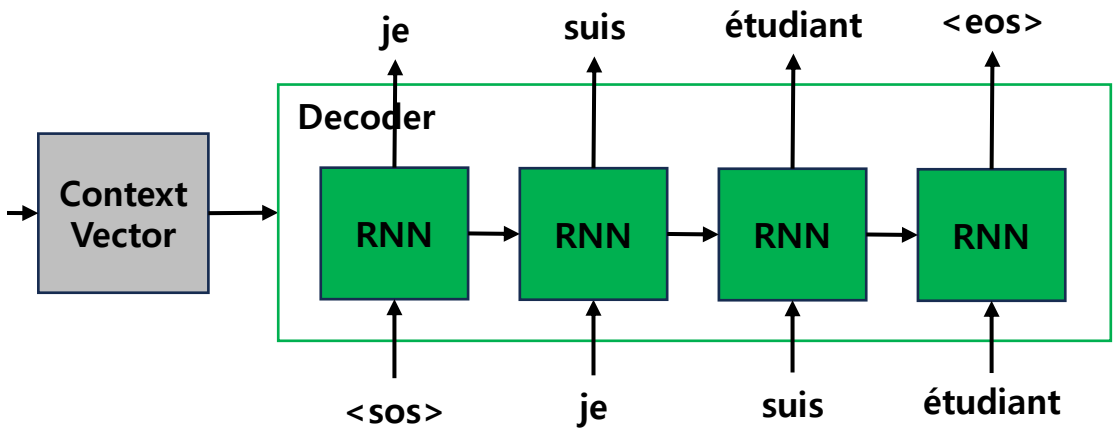
Seq2Seq 구조

- **Decoder**
  - Context vector를 사용하여 출력 문장을 디코딩 하는 역할
  - <sos> 토큰을 입력으로 받아서 <eos> 토큰이 나올 때까지 문장 생성
    - <sos>: Start of sequence
    - <eos> End of sequence



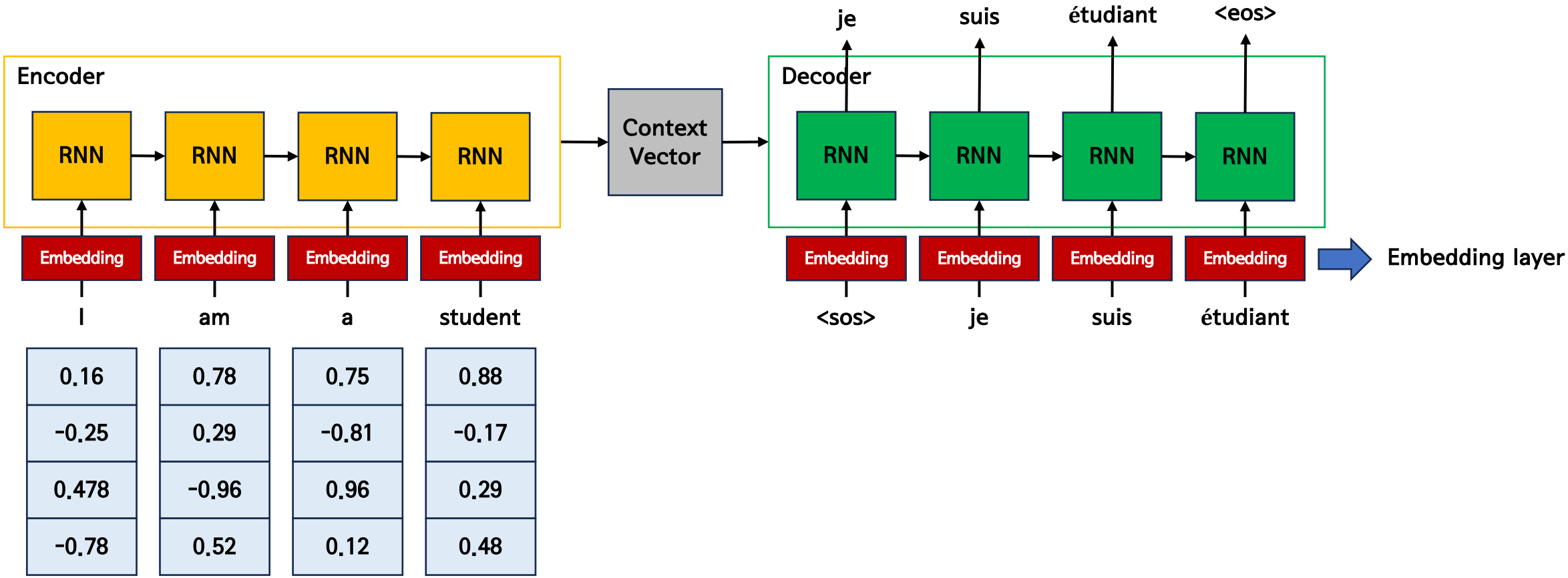
Seq2Seq 구조

- Decoder
  - 디코딩 과정
    - 첫번째 LSTM cell의 입력
      - <sos> 토큰 + context vector ➔ 그 다음에 가장 등장할 확률이 높은 단어 예측
    - 그 다음 cell의 입력
      - **이전 cell에서 예측한 단어** ➔ 그 다음에 등장할 확률이 가장 높은 단어 예측
    - 위 과정을 재귀적으로 반복



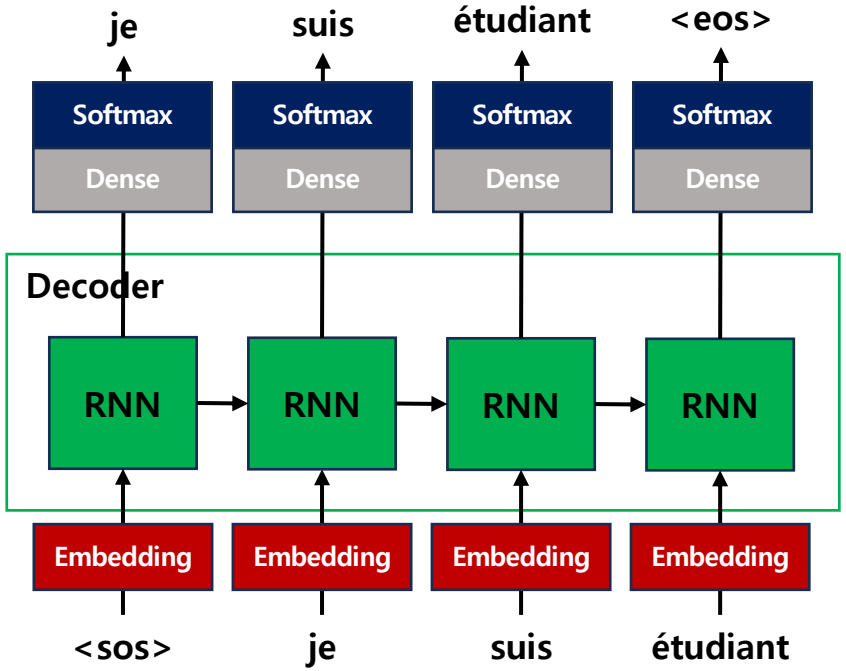
Seq2Seq 구조

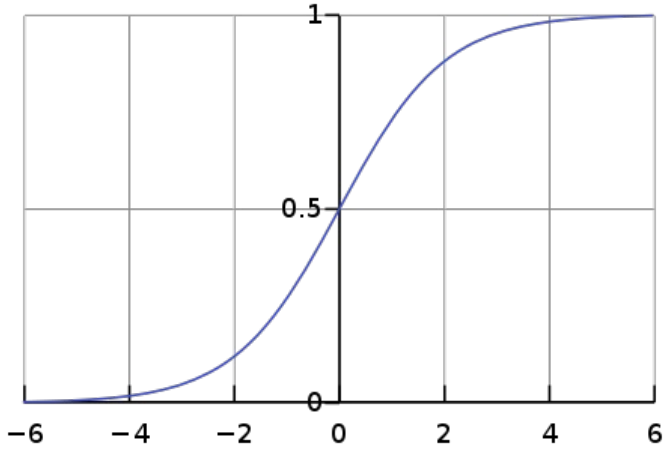
- 단어를 입력으로 넣는 방법
  - 워드 임베딩 사용 (Text ➔ vector)
  - 입력으로 들어가는 모든 단어들은 워드 임베딩을 통해 임베딩 벡터로 변환 후 입력으로 사용됨



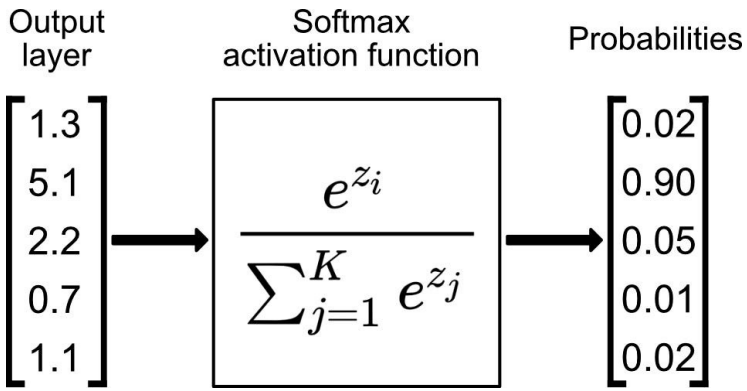
Seq2Seq 구조

- **출력 단어를 선택하는 방법**
  - 어떻게 출력 단어로 나올 확률이 가장 높은 단어를 예측할까?
  - Softmax함수를 activation function으로 활용함
    - Softmax ➔ 출력값을 확률로 바꿈





$$softmax\ \sigma(\vec{z})_i = \frac{e^{zj}}{\sum_{j=1}^{K} e^{zj}}$$



Seq2Seq 모델의 한계

- **Seq2Seq 모델**
  - 가변 길이의 입력/출력을 처리하는데 효과적인 모델 구조
  - 실제로 기계 번역에서 성능의 비약적 향상
- **한계점**
  - 고정된 크기의 벡터에 모든 정보를 압축 ➔ **정보 손실 발생**
    - Encoder가 출력하는 벡터 사이즈가 고정됨
      - 입력 단어의 수가 매우 많아지면 성능 저하
    - 고정된 사이즈의 벡터에 많은 정보 함축할 수 없음
  - **Vanishing Gradient Problem**
    - RNN의 구조적 한계 (문장 길이가 길어지면 초기 cell에 전달하는 정보가 점차 0에 수렴함)
    - Encoder/Decoder에 LSTM, GRU 같은 모델을 활용 ➔ 이전 정보 압축에 한계

Attention의 등장

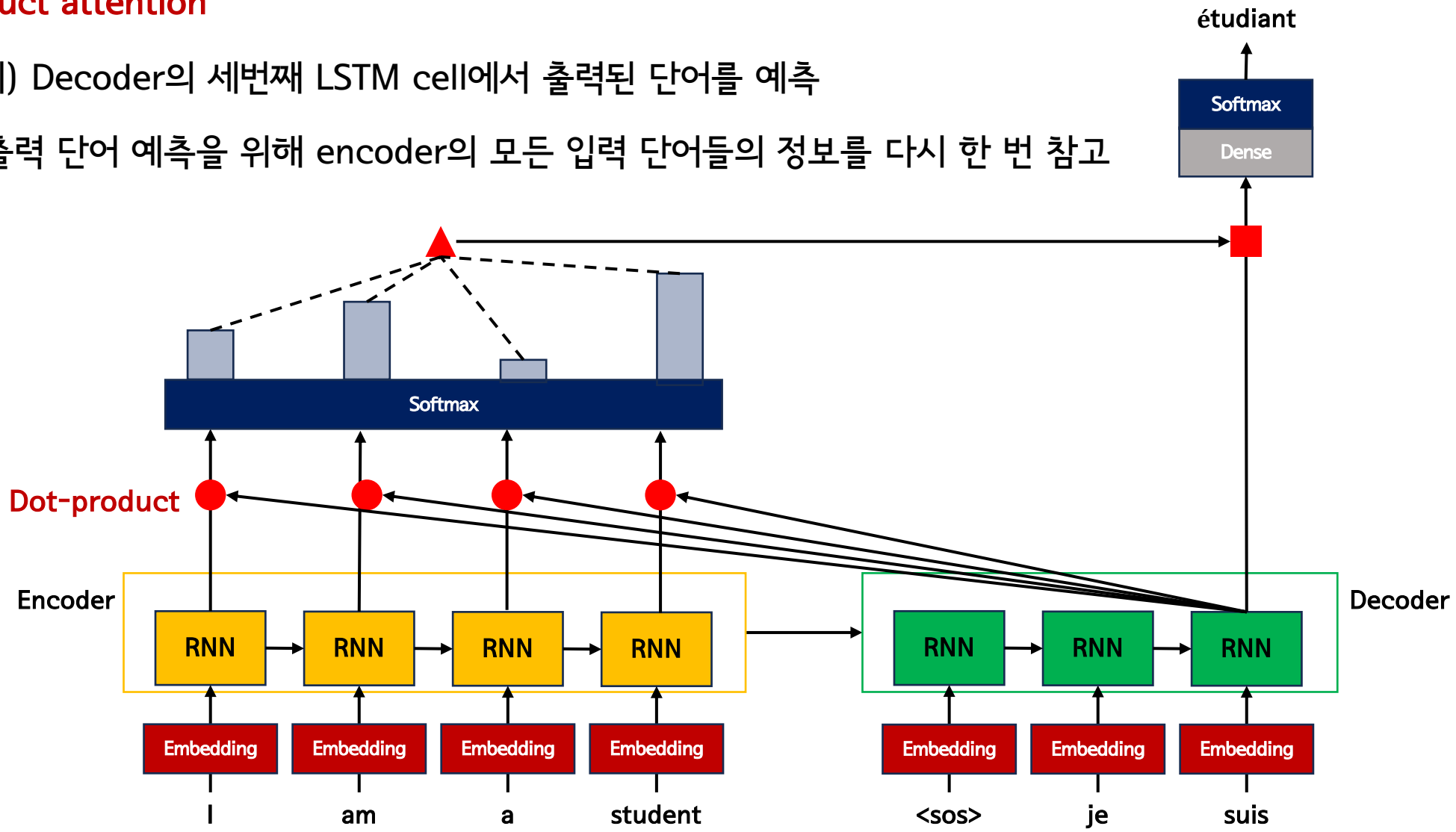
- Seq2Seq 모델의 한계를 극복하기 위해 제안됨
  - https://arxiv.org/abs/1409.0473 (2014)

  - 입력 시퀀스가 길어지면 출력 시퀀스의 정확도가 떨어지는 문제를 해결
    - Seq2Seq 모델
      - 입력 시퀀스 ➔ Context vector라는 고정된 크기의 벡터로 압축 ➔ 출력 시퀀스
      - Machine translation에서 입력 문장이 길면 번역 품질 저하

- **Attention의 아이디어**
  - Decoder에서 출력 단어를 예측하는 매 시점 (time step)마다 encoder의 전체 입력 문장을 참고함
    - 해당 시점에서 예측해야 할 단어와 연관이 있는 입력 단어 부분을 보다 **집중(attention)**해서 살펴봄

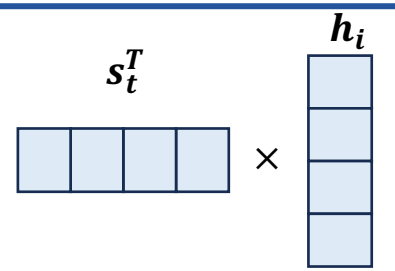
Attention mechanism

- **Dot-product attention**
  - (예시) Decoder의 세번째 LSTM cell에서 출력된 단어를 예측
  - ➔ 출력 단어 예측을 위해 encoder의 모든 입력 단어들의 정보를 다시 한 번 참고

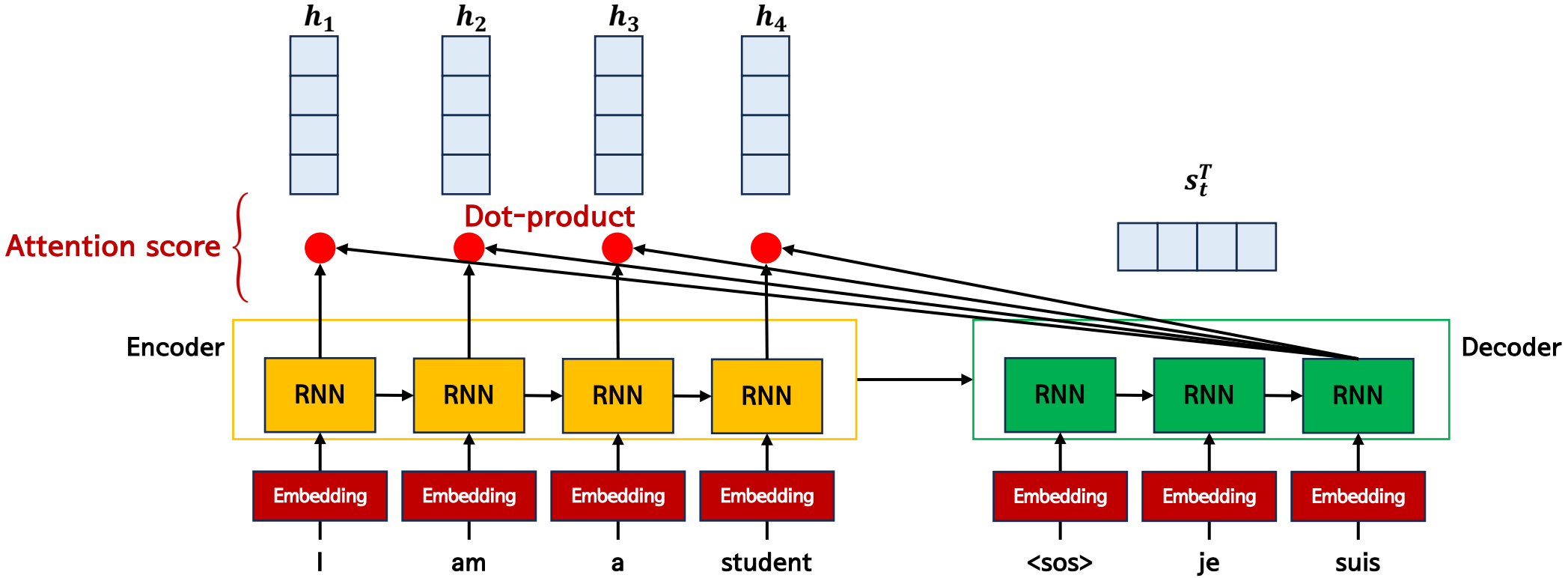


Attention mechanism

- 1) Attention score calculation
  - Encoder의 시점 (time step)을 각각 $1, 2, \ldots N$
    - h_1~ h_n : Encoder의 각 cell에 대한 hidden state
    - s_t^T : Decoder의 현재 시점 (time step) t에서의 해당 cell의 hidden state

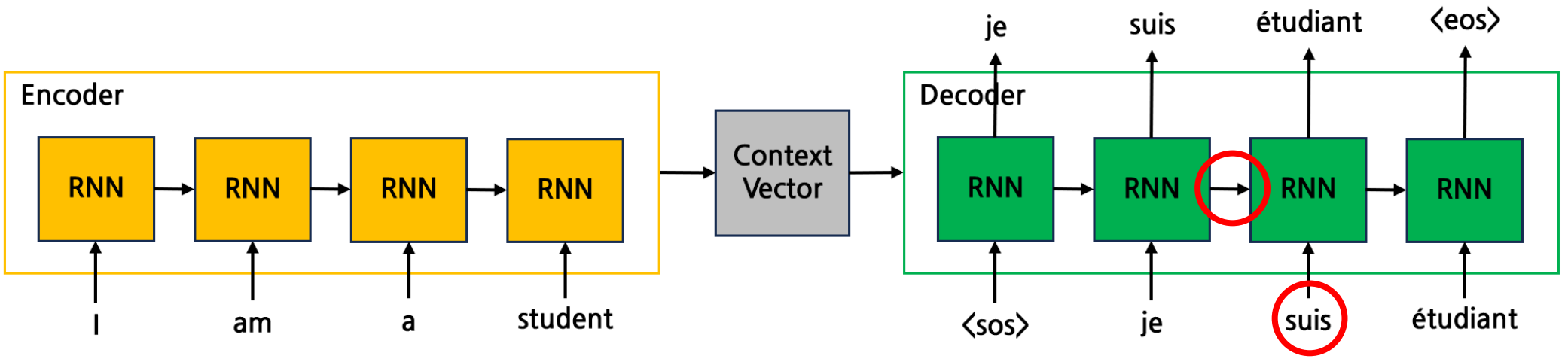


$$score(s_t, h_i) = s_t^T h_i$$



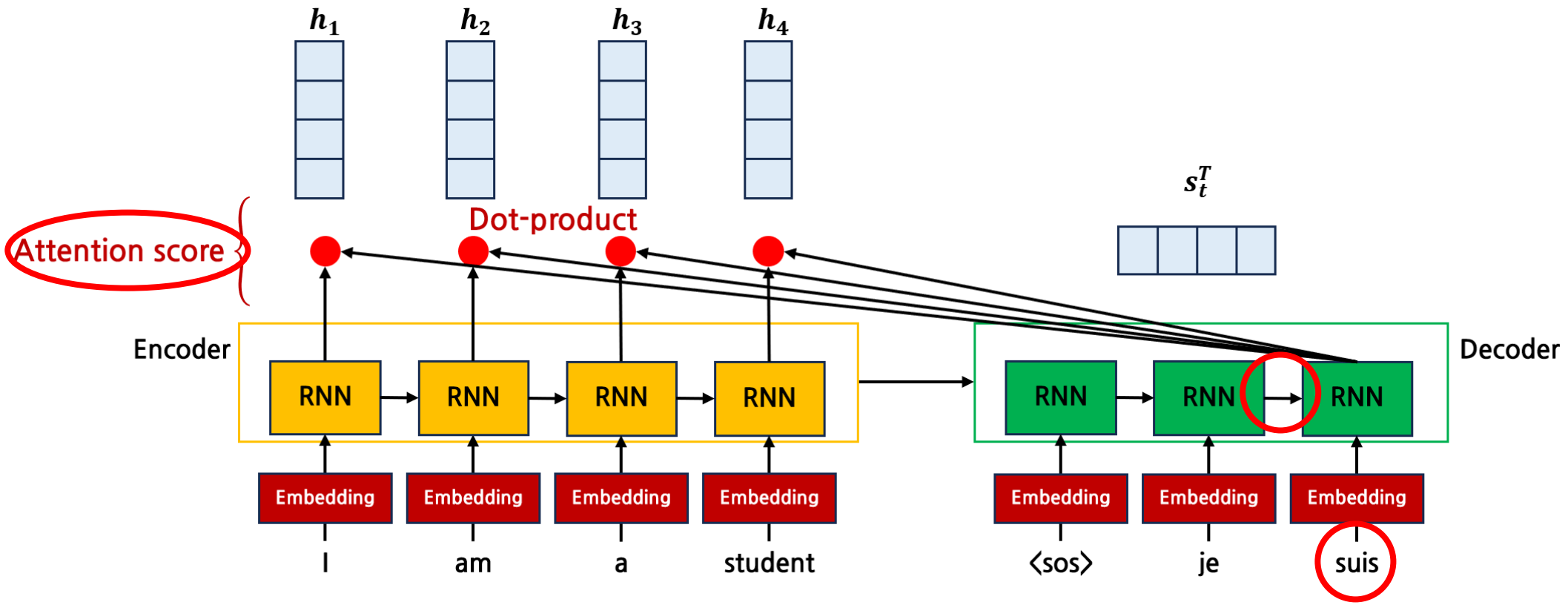
Attention mechanism

- 1) Attention score calculation
  - Seq2Seq model
    - 특정 시점 t에서 출력 단어 예측을 위해 decode의 cell이 필요로 하는 입력
      - ➔ **이전 시점 $t-1$의 hidden state와 이전 시점 $t-1$에 나온 출력 단어**
      - (예시) 'étudiant' 예측을 위해서 'suis'와 이전 cell에서의 출력값 필요



Attention mechanism

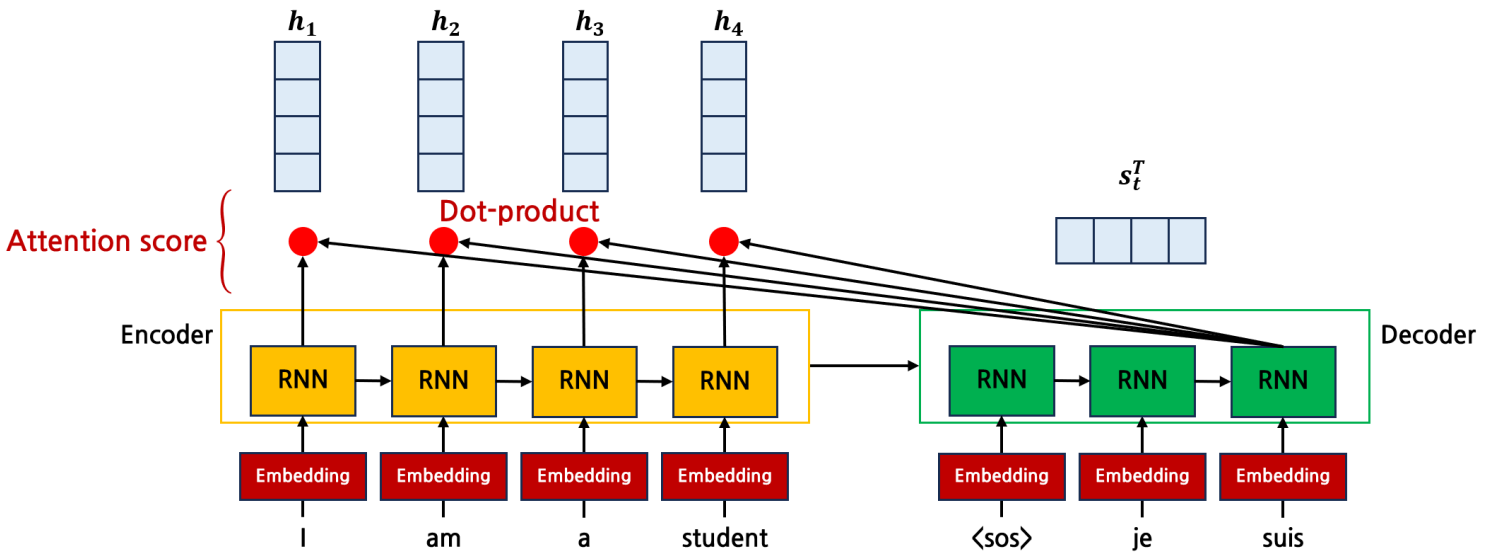
- 1) Attention score calculation
  - Attention
    - 이전 시점 $t-1$의 hidden state + 이전 시점 $t-1$에 나온 출력 단어 + Attention value 필요



Attention mechanism

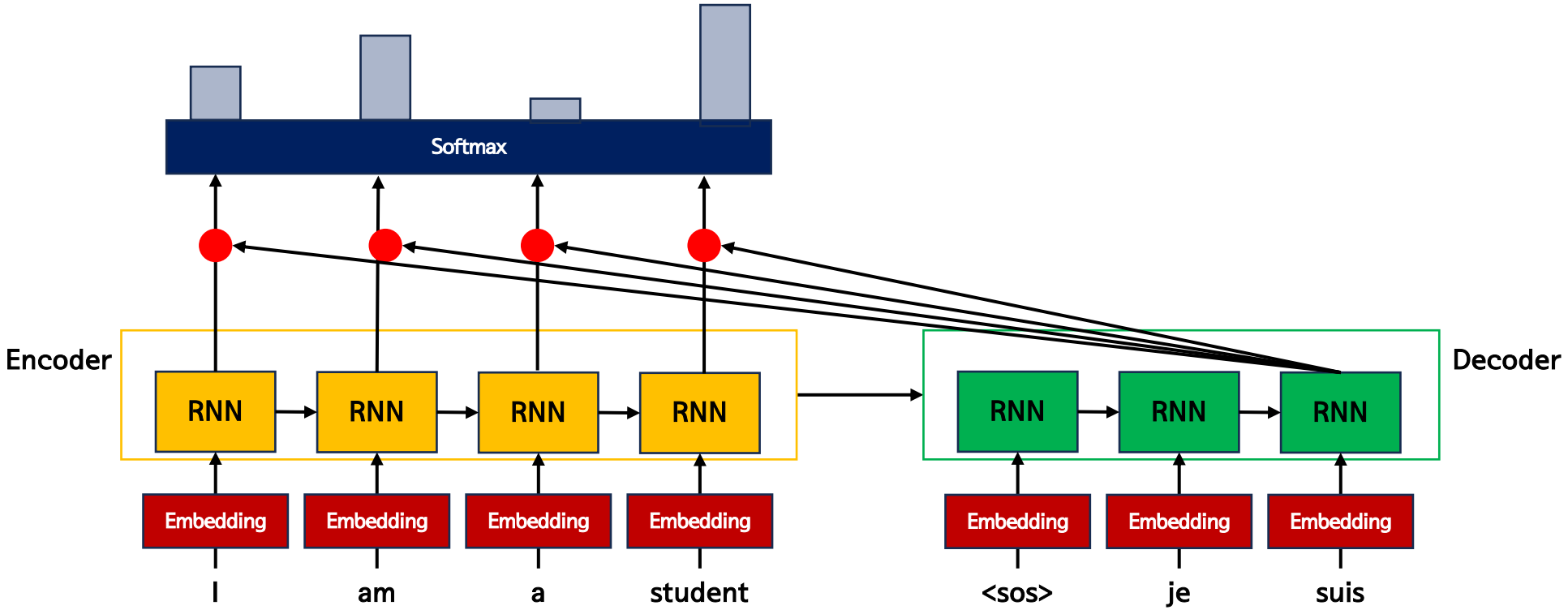
- 1) Attention score calculation
  - Attention score의 의미
    - Decoder가 현 시점에서 추론할 때 **어떤 단어에 집중해야 하는지 알려주는 지표**
      - 현재 decoder의 시점 t에서 단어를 예측하기 위해 Encoder의 모든 hidden state 각각이 decoder의 현 시점 hidden state와 얼마나 유사한지를 판단하는 점수

  - 계산 결과 ➔ 스칼라 값
    - 실제 유사도처럼 간주 가능



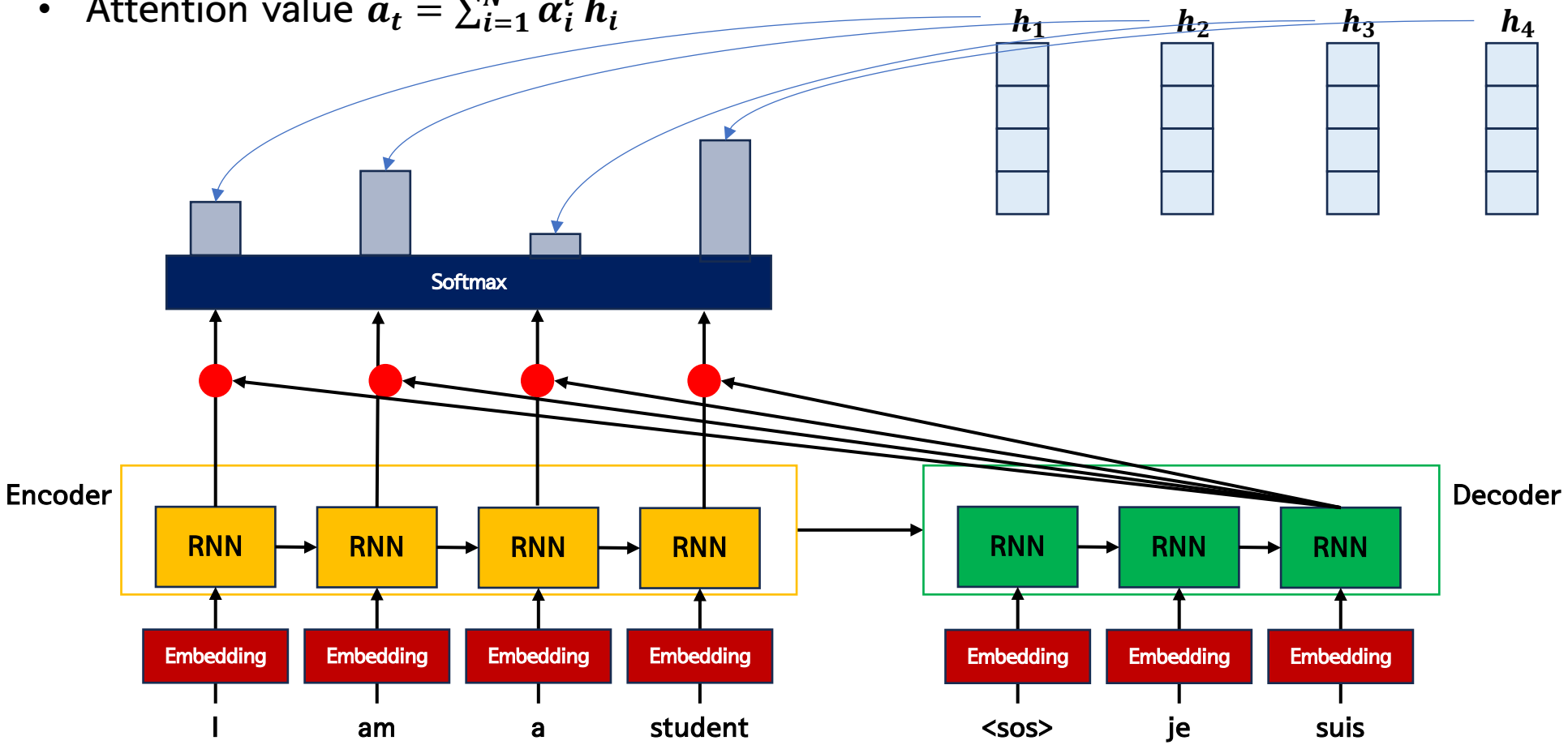
Attention mechanism

- 2) Attention distribution using Softmax function
  - e^t: s_t와 encoder의 모든 hidden state에 대한 attention score의 집합; $e^t = [s_t^T h_1, s_t^T h_2, \ldots, s_t^T h_N]$
  - e^t에 softmax함수 적용하여 **attention 분포** (α^t) 계산; $\alpha^t = softmax(e^t)$
  - 분포 내 각각의 값은 해당 입력 단어에 대한 **attention weight** (α_i^t) 라고 함



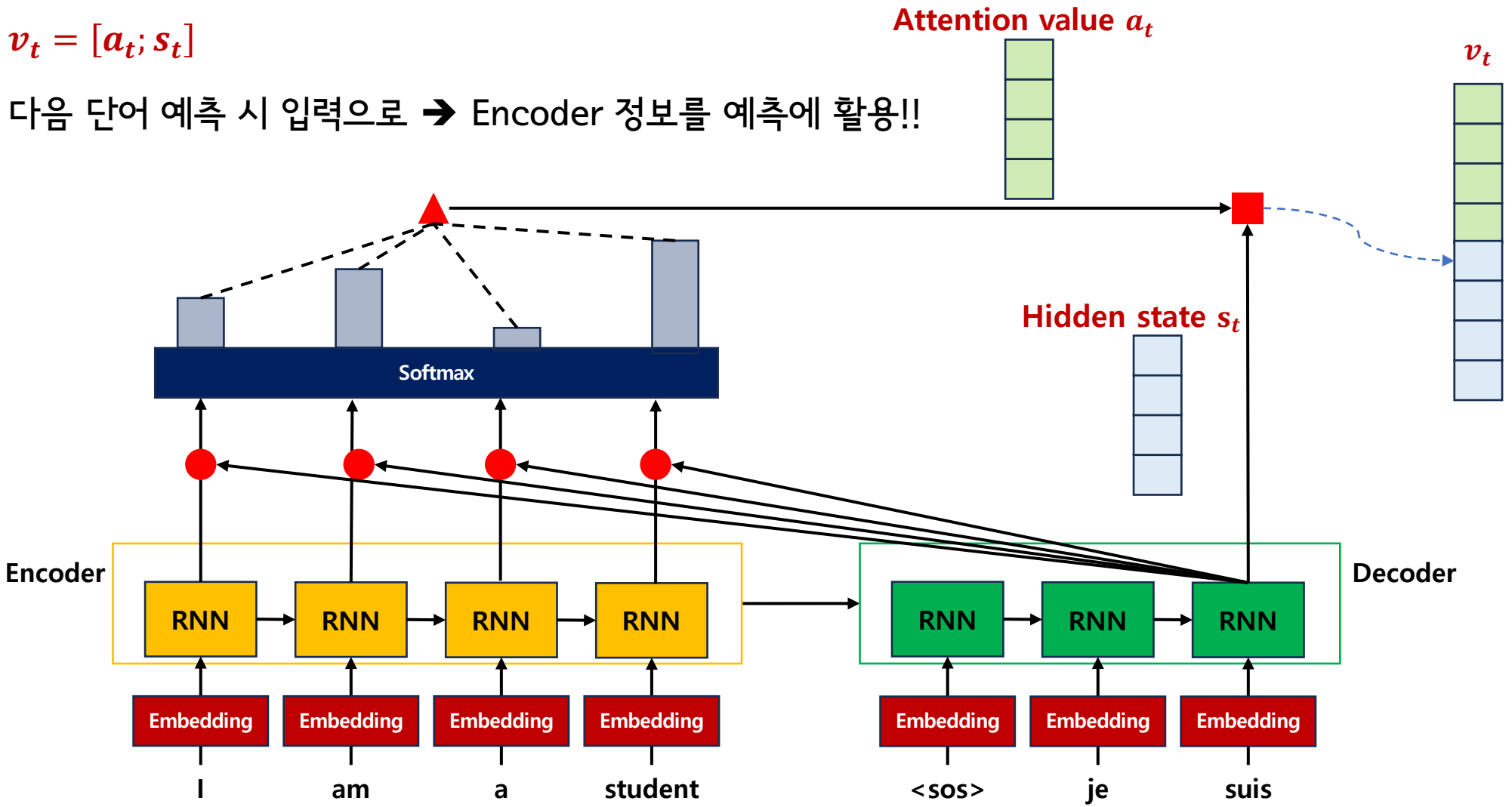
Attention mechanism

- 3) Attention value calculation ➔ Context vector
  - 각 attention weight (α_i^t)와 hidden state ($h_1 \dots h_N$)를 가중합하여 계산
    - Attention value $\boldsymbol{a_t} = \sum_{i=1}^{N} \boldsymbol{\alpha_i^t}\, \boldsymbol{h_i}$



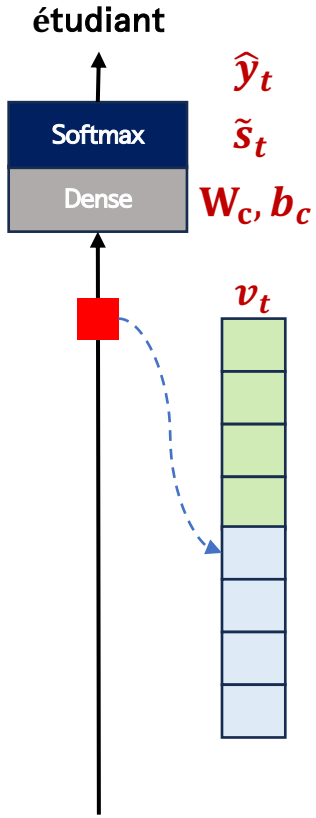
Attention mechanism

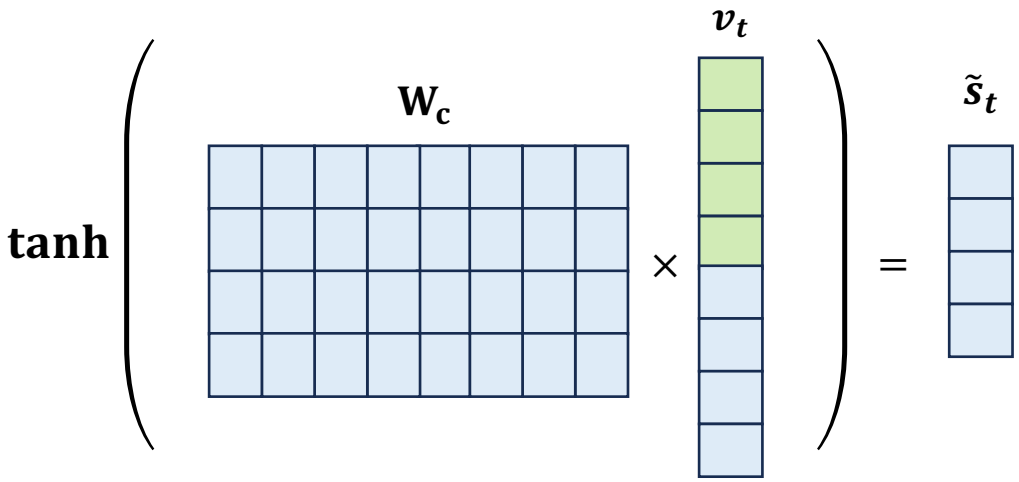
- 4) Concatenate a_t and s_t ➔ one vector v_t
  - $v_t = [a_t; s_t]$
  - 다음 단어 예측 시 입력으로 ➔ Encoder 정보를 예측에 활용!!



Attention mechanism

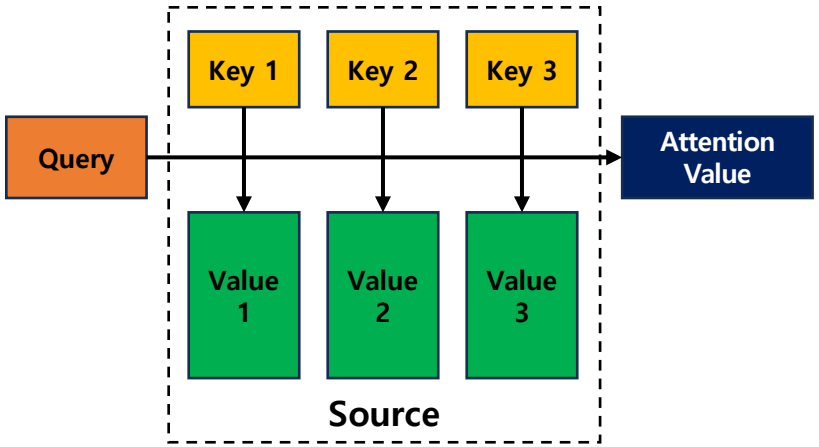
- 5) Predict the next work after obtaining $\tilde{s}_t$
  - $\tilde{s}_t = \tanh(\mathbf{W_c}[a_t; s_t] + b_c)$
  - $\hat{y}_t = softmax(\tilde{s}_t + b_y)$

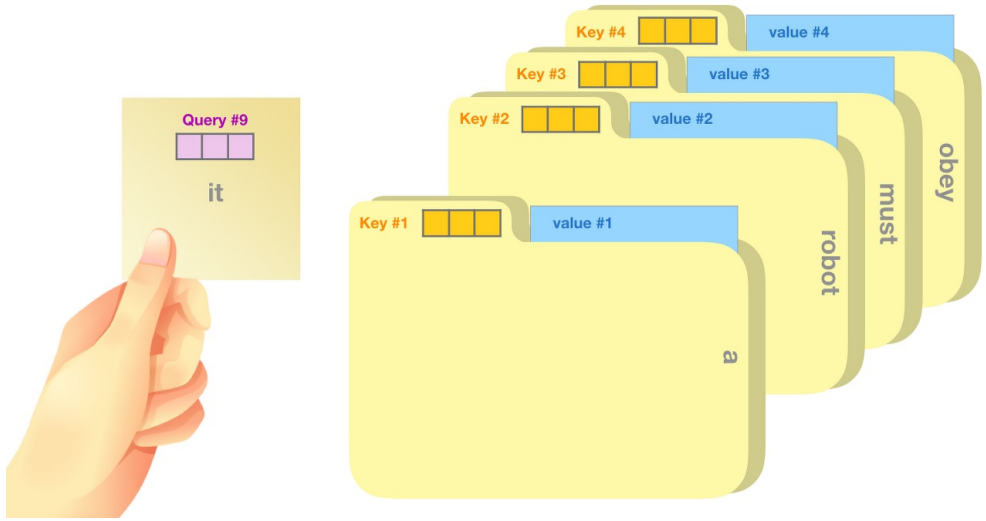




Attention function using Q, K, V

- **Query, Key, Value**
  - Q = Query
    - t 시점의 decoder cell에서의 hidden state
  - K = Keys
    - 모든 시점의 encoder cell 각각에 대한 hidden state들
  - V = Values
    - 모든 시점의 encoder cell 각각에 대한 hidden state들





Attention function using Q, K, V

- **Attention (Q, K, V) = Attention value**
  - 유사도 (Similarity) 계산 ➔ Score function
    - 주어진 query에 대해서 모든 key들 간의 attention score 계산
  - 정규화 (Normalization) ➔ Softmax function
    - Query와 각 key들 간의 attention score를 attention distribution으로 변환
  - 가중합 (Weighted sum) 계산 ➔ Attention distribution
    - 각 value들의 attention value 계산
  - $Attention(Q, K, V) = softmax(QK^T)V$

Various types of Attention algorithms

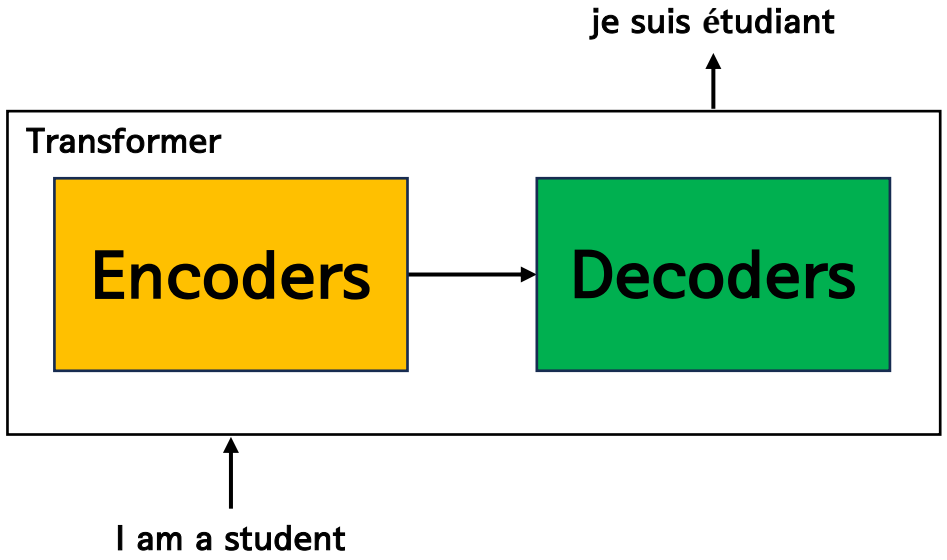
이름	스코어 함수	**Defined by**
dot	$score(s_t,\ h_i) = s_t^T h_i$	Luong et al. (2015)
$scaled\ dot$	$score(s_t,\ h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	Vaswani et al. (2017)
$general$	$score(s_t,\ h_i) = s_t^T W_a h_i$ // 단, W_a는 학습 가능한 가중치 행렬	Luong et al. (2015)
$concat$	$score(s_t,\ h_i) = W_a^T\ tanh(W_b[s_t; h_i]), score(s_t,\ h_i) = W_a^T\ tanh(W_b s_t + W_c h_i)$	Bahdanau et al. (2015)
$location - base$	$\alpha_t = softmax(W_a s_t)$ // α_t 산출 시에 s_t만 사용하는 방법.	Luong et al. (2015)

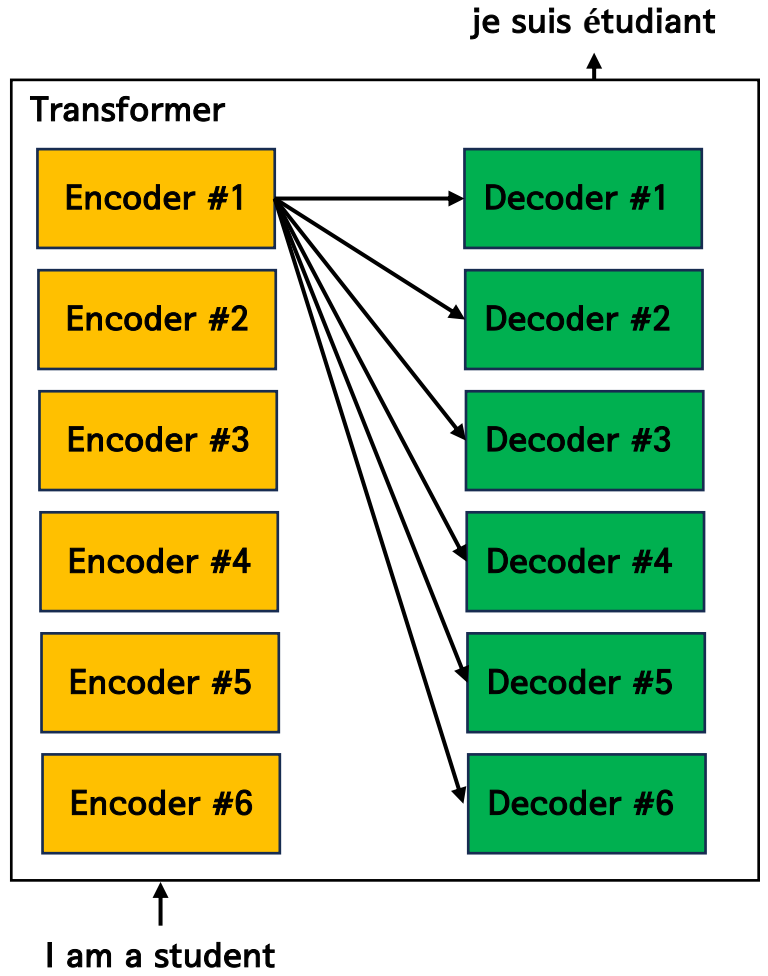
Transformer의 등장

- **Attention Is All You Need:** https://arxiv.org/abs/1706.03762 (2017)
- **Seq2Seq (Encoder-Decoder)를 따르면서도 Attention만으로 구현된 모델**
  - 기존 Seq2Seq 모델, RNN 계열 모델
    - 시간에 의존적인 입력 sequence에 따라 hidden state를 생성하여 입력을 처리하는 구조
  - RNN을 사용하지 않고 Encoder-Decoder 구조로만 설계 ➔ RNN보다 우수한 성능
  - 아직까지도 다양한 분야에서 사용되는 범용적 모델
- **Seq2Seq / Attention의 고질적 한계를 개선**
  - 고정된 크기의 벡터에 모든 정보 압축 ➔ 정보 손실 발생
  - Vanishing Gradient Problem
  - 입력의 순차적 처리 ➔ 병렬화 불가능

Transformer 모델 구조

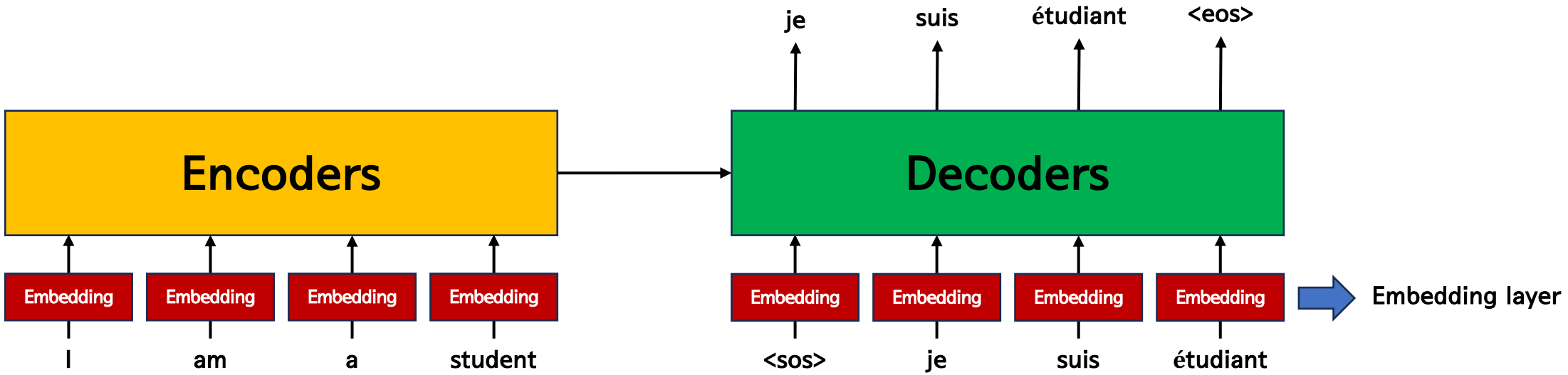
- **Seq2Seq 모델 구조처럼 Encoder-Decoder 구조를 사용**
  - (Seq2Seq 모델) Encoder/Decoder 각각이 하나의 RNN 모델 처럼 동작
  - **(Transformer) Encoder/Decoder가 각각 N개로 확장**되는 구조
    - Encoder/Decoder 각각 6개인 예시





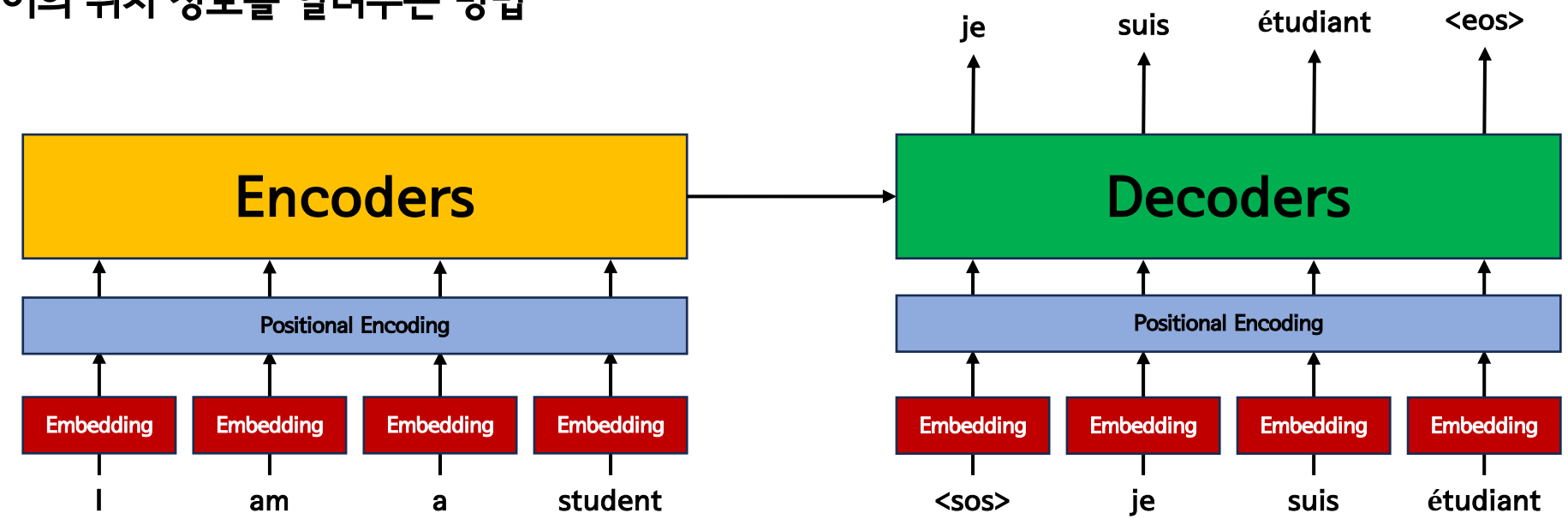
Transformer 모델 구조

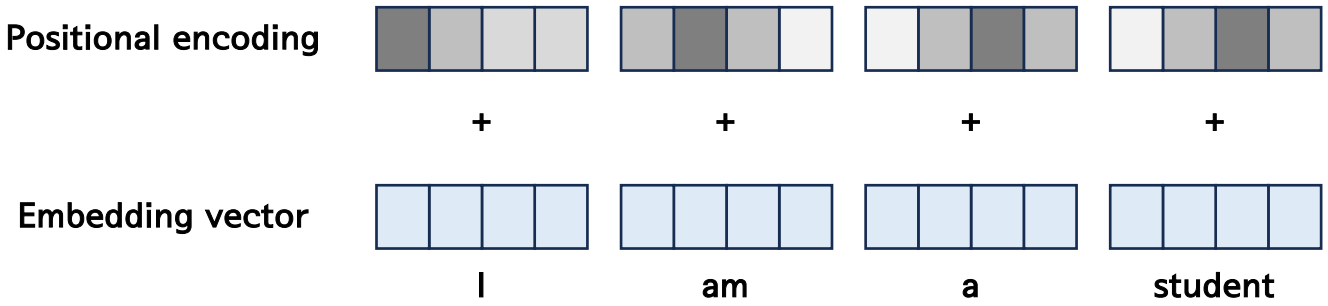
- **Seq2Seq 모델 구조처럼 Encoder-Decoder 구조를 사용**
  - <sos> 토큰을 입력으로 받아 <eos> 토큰 나올 때까지 연산 수행
  - RNN은 사용되지 않았지만 encoder-decoder의 구조를 유지
- **Transformer의 입력**
  - **Embedding vector에서 조정된 값을 입력으로 활용**
    - 단순히 각 단어의 embedding vector를 입력으로 받지 않음 (RNN의 고유한 특성)



Transformer 모델 구조

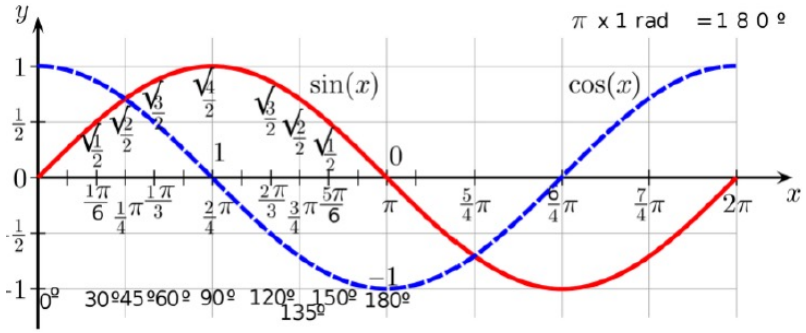
- Positional encoding
  - 각 단어의 위치 정보를 알려주는 방법

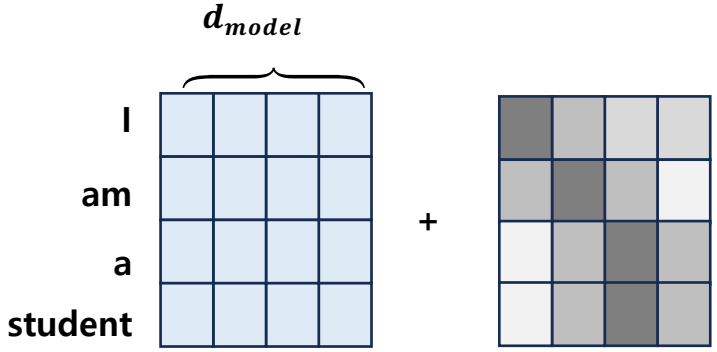




Transformer 모델 구조

- Positional encoding: 위치 정보를 가진 값을 만들기 위해 두 개의 함수를 사용
  - $PE(pos, 2i) = \sin\left(pos/10000^{2i/d_{model}}\right)$
  - $PE(pos, 2i+1) = cos\left(pos/10000^{2i/d_{model}}\right)$
  - d_{model} (Hyperparameter)
    - Transformer의 encoder/decoder에서 정해진 입력과 출력의 크기 (= Embedding vector의 차원)
    - 모든 층의 출력 차원을 의미 (논문에서는 512)
  - pos : 입력 문장에서의 embedding vector의 위치
  - i : Embedding vector 내의 차원의 index





Embedding vector + Position encoding

➔ 순서 정보 보존

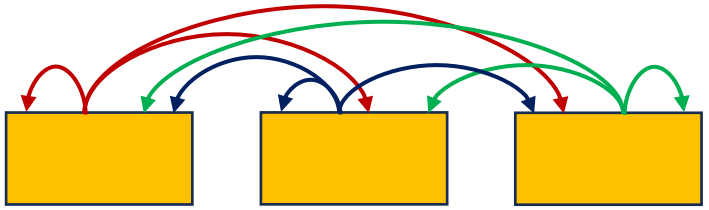
➔ 같은 단어라도 문장 내의 위치에 따라 입력 vector값이 달라짐

Transformer 모델 구조

- Positional encoding: 위치 정보를 가진 값을 만들기 위해 두 개의 함수를 사용
  - $PE(pos, 2i) = \sin\left(pos/10000^{2i/d_{model}}\right)$
  - $PE(pos, 2i+1) = cos\left(pos/10000^{2i/d_{model}}\right)$
  - d_{model} (Hyperparameter)
    - Transformer의 encoder/decoder에서 정해진 입력과 출력의 크기 (= Embedding vector의 차원)
    - 모든 층의 출력 차원을 의미 (논문에서는 512)
  - pos : 입력 문장에서의 embedding vector의 위치
  - i : Embedding vector 내의 차원의 index

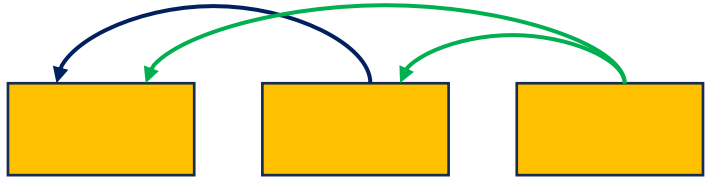
Transformer 모델 구조

- Transformer에서 사용되는 attention 종류



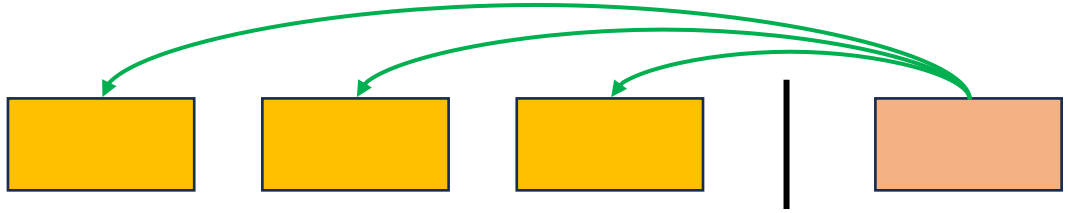
Encoder Self-Attention

- Encoder에서 사용되는 attention



Masked Decoder Self-Attention

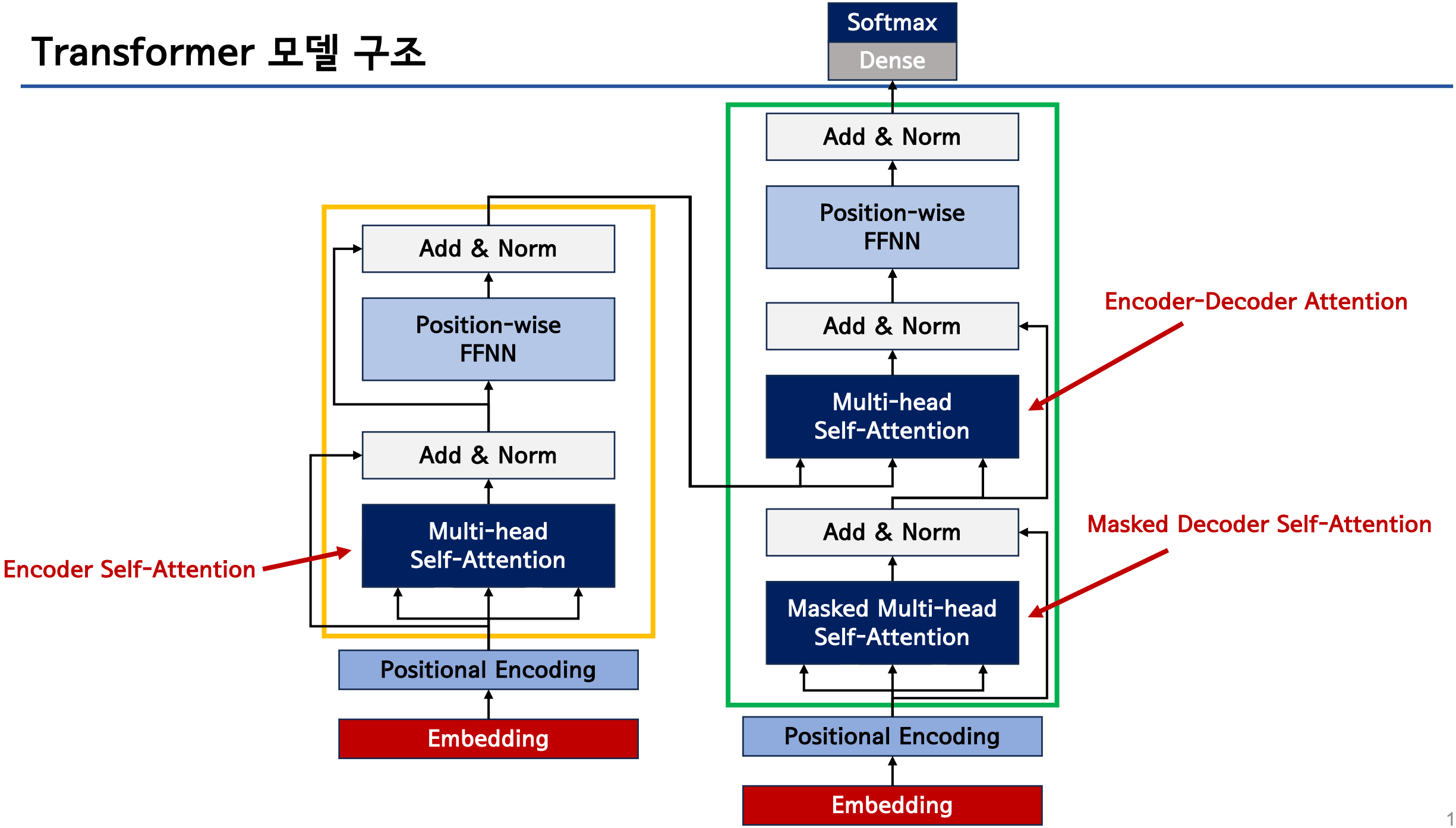
- Decoder에서 사용되는 attention



Encoder-Decoder Attention

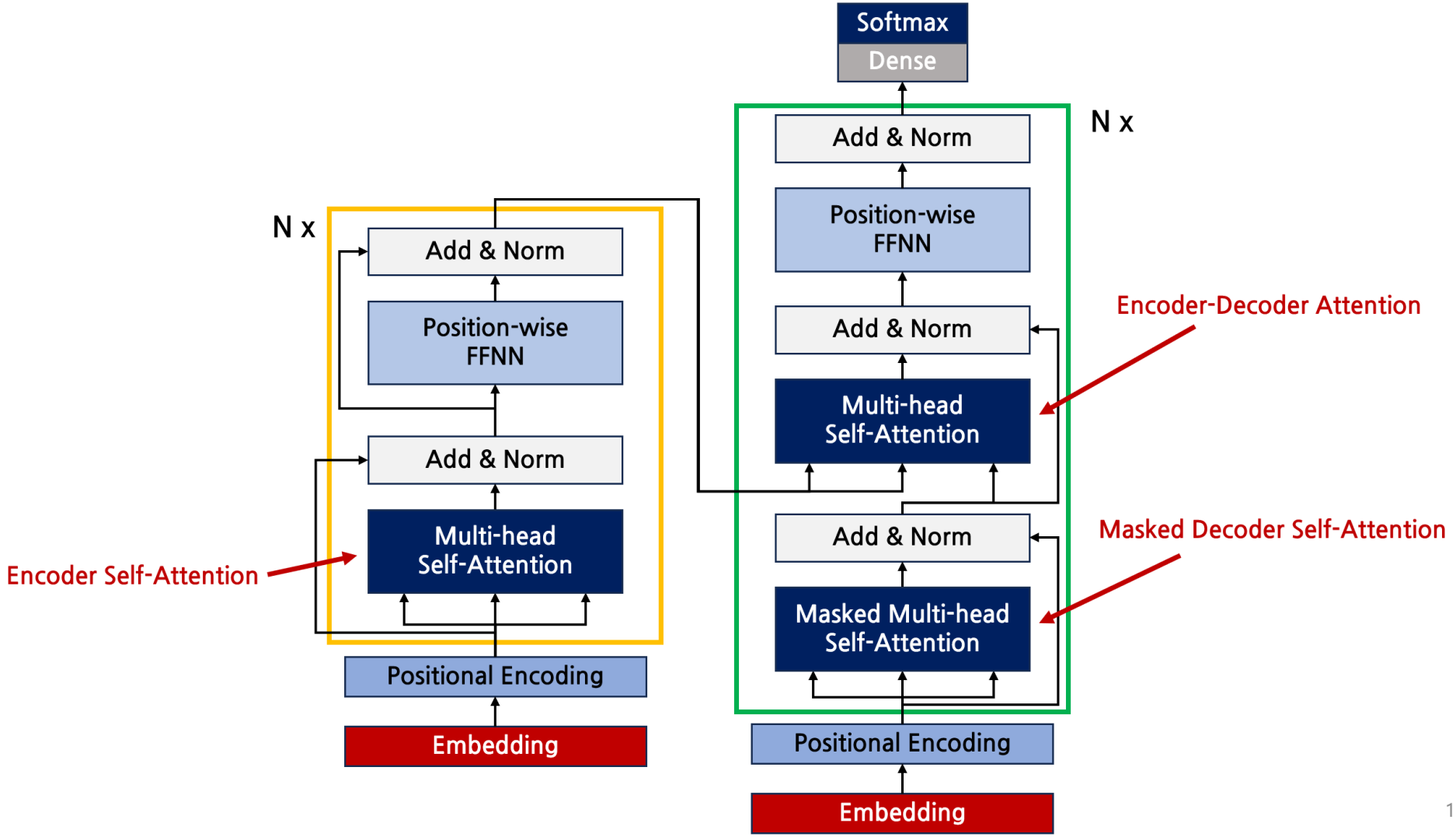
- Decoder에서 사용되는 attention

Transformer 모델 구조



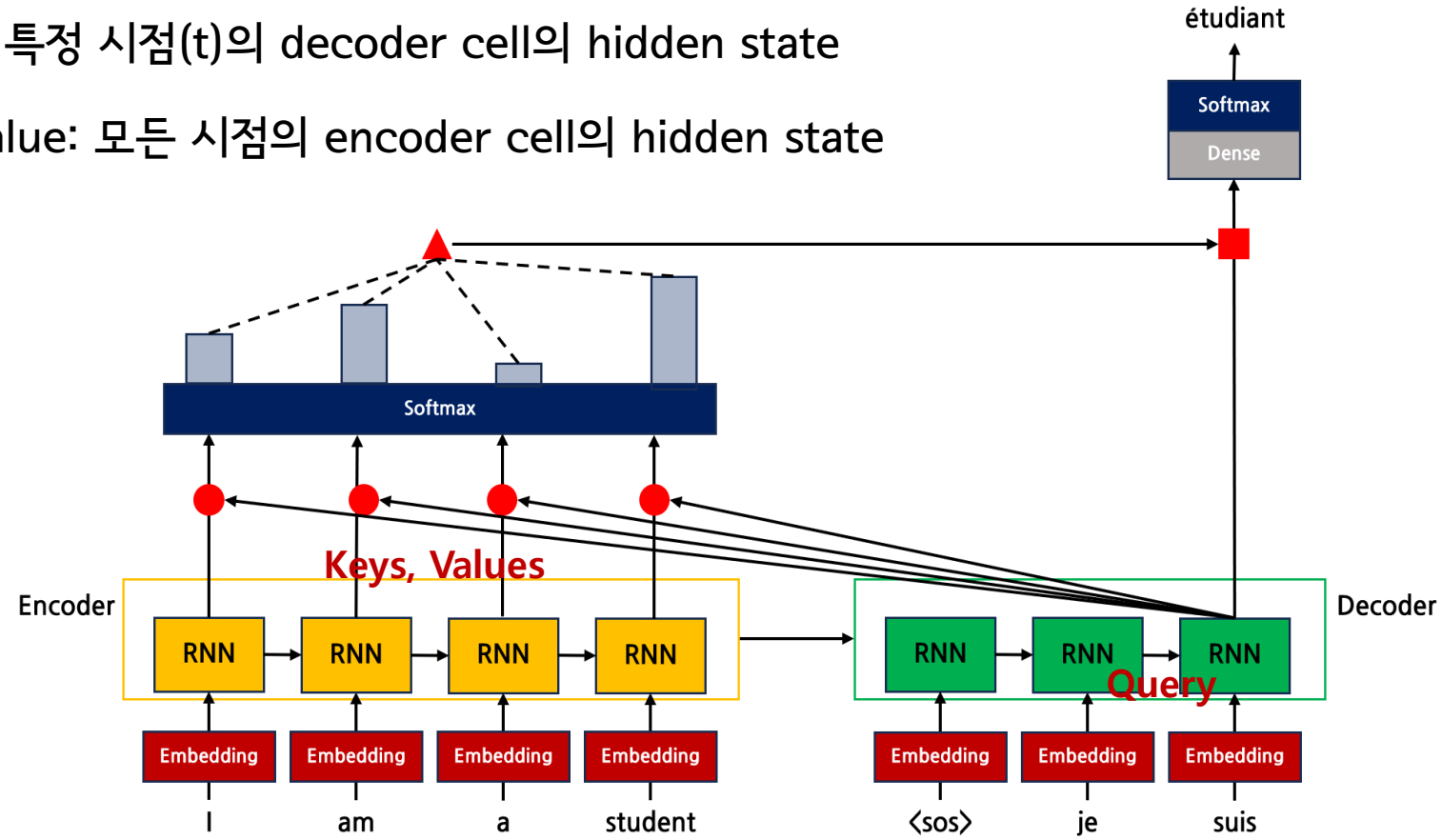
Transformer 모델 구조

- **개념 1)** Multi-head: Transformer가 attention을 병렬적으로 수행하는 방법



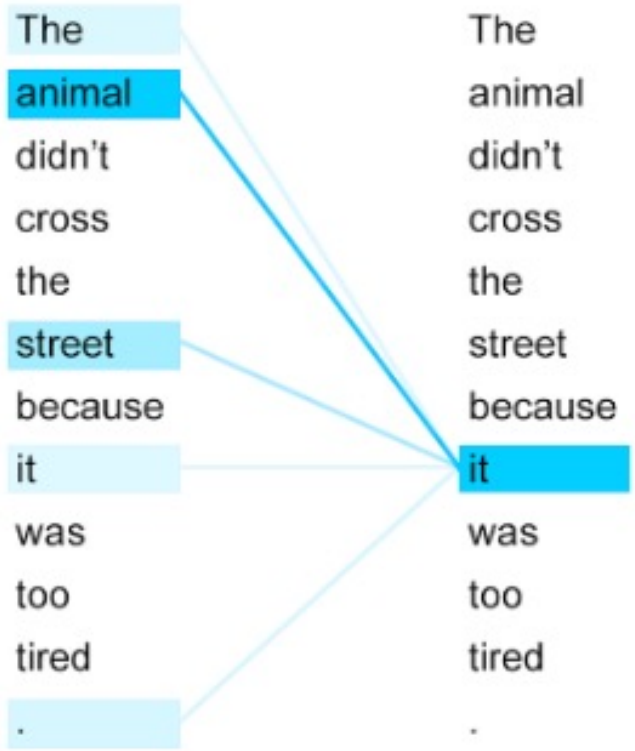
Transformer 모델 구조

- 개념 2) Self-Attention: Attention을 스스로에게 수행한다는 의미
  - Seq2Seq 모델에서 사용하는 Attention (function)
    - Decoder가 생성하는 특정 시점의 hidden state와 입력 문장들 간의 관계를 계산하기 위해 사용
    - Query: 특정 시점(t)의 decoder cell의 hidden state
    - Key, Value: 모든 시점의 encoder cell의 hidden state



Transformer 모델 구조

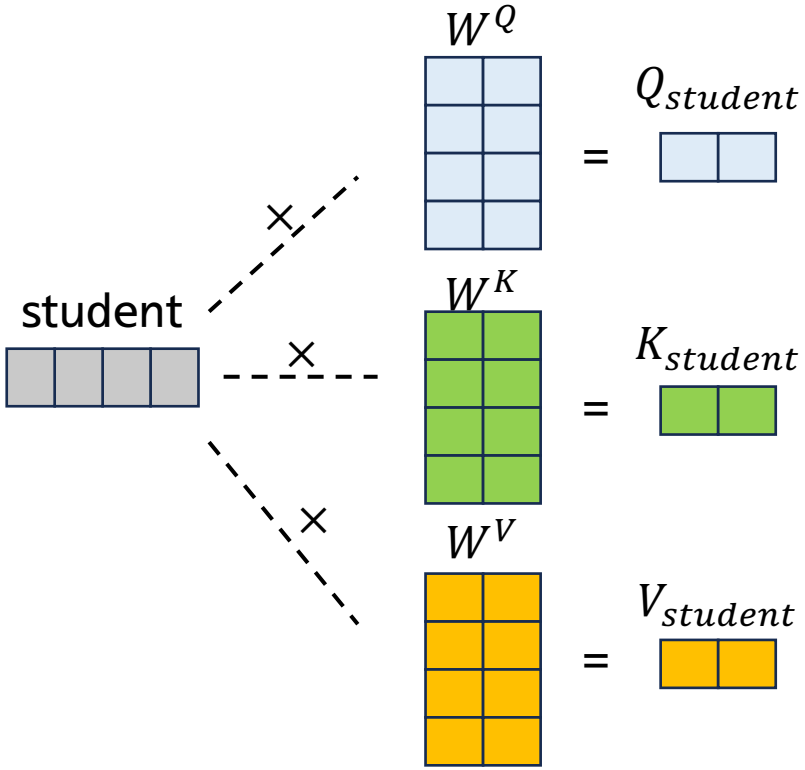
- **개념 2)** Self-Attention: Attention을 스스로에게 수행한다는 의미
  - Self-Attention의 의미와 이점
    - Self-Attention은 입력 문장 내 단어들 간의 유사도를 계산
      - (예시) 'it'이 의미하는 것은? ➔ 'animal'?, 'street'?, …
        - ➔ 'it' 이 'animal'과 연관되었을 확률이 높다는 것을 찾을 수 있음



Self-attention mechanism

1) Q, K, V 벡터 얻기

- 각 단어 벡터들로부터 Q, K, V 벡터를 추상화하는 작업 수행
  - 입력 크기 (d_{model}) 차원을 갖는 벡터를 바로 Self-Attention 수행하는 것이 아님
  - ➔ Q, K, V 벡터는 초기 입력보다 훨씬 더 작은 차원을 가짐



논문에서 각 가중치 행렬의 크기는 $\boldsymbol{d_{model} \times (d_{model}/num\_heads)}$

- Example
  - $d_{model} = 512, \boldsymbol{num\_heads = 8}$
  - ➔ Q, K, V 벡터의 크기 (d_q, d_k, d_v) 각각 64차원

Self-attention mechanism

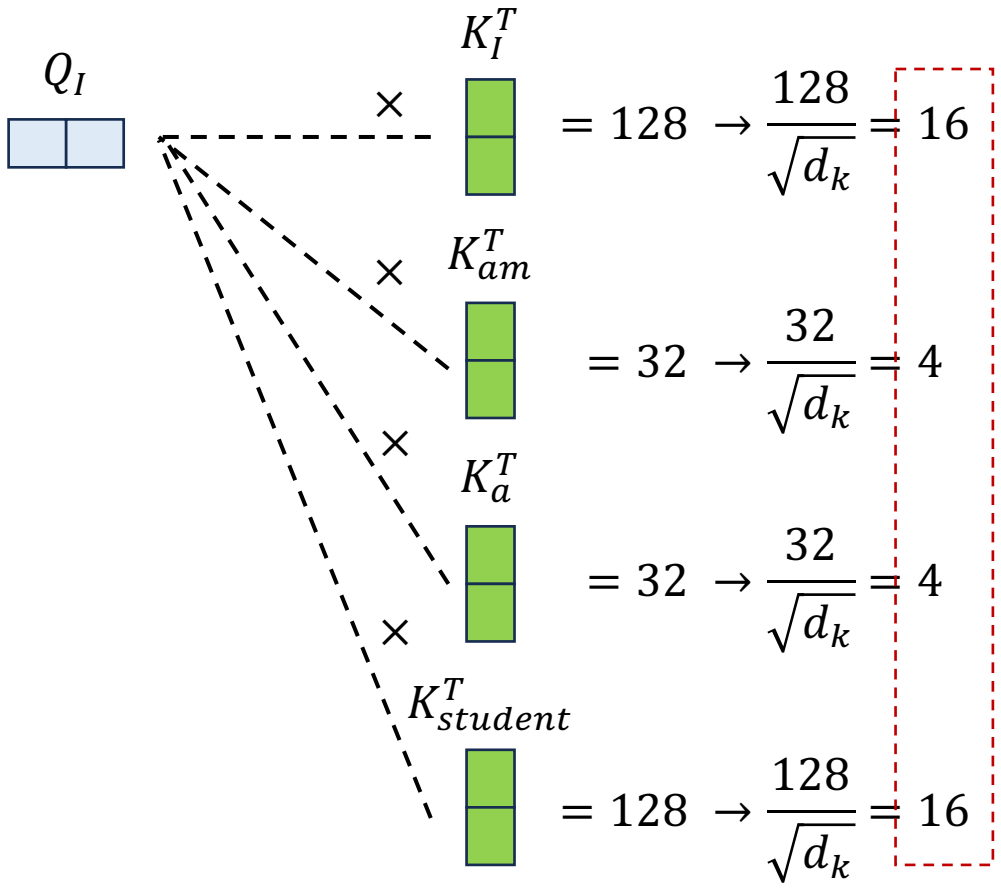
- 2) Scaled dot-product Attention
  - 기존의 Attention mechanism과 동일
  - 단어 I에 대한 Q벡터와 모든 K벡터에 대해서 attention score를 계산

$$score(s_t, h_i) = s_t^T h_i$$

Attention score

$$score(q, k) = \frac{q \cdot k}{\sqrt{n}}$$

Self-Attention score



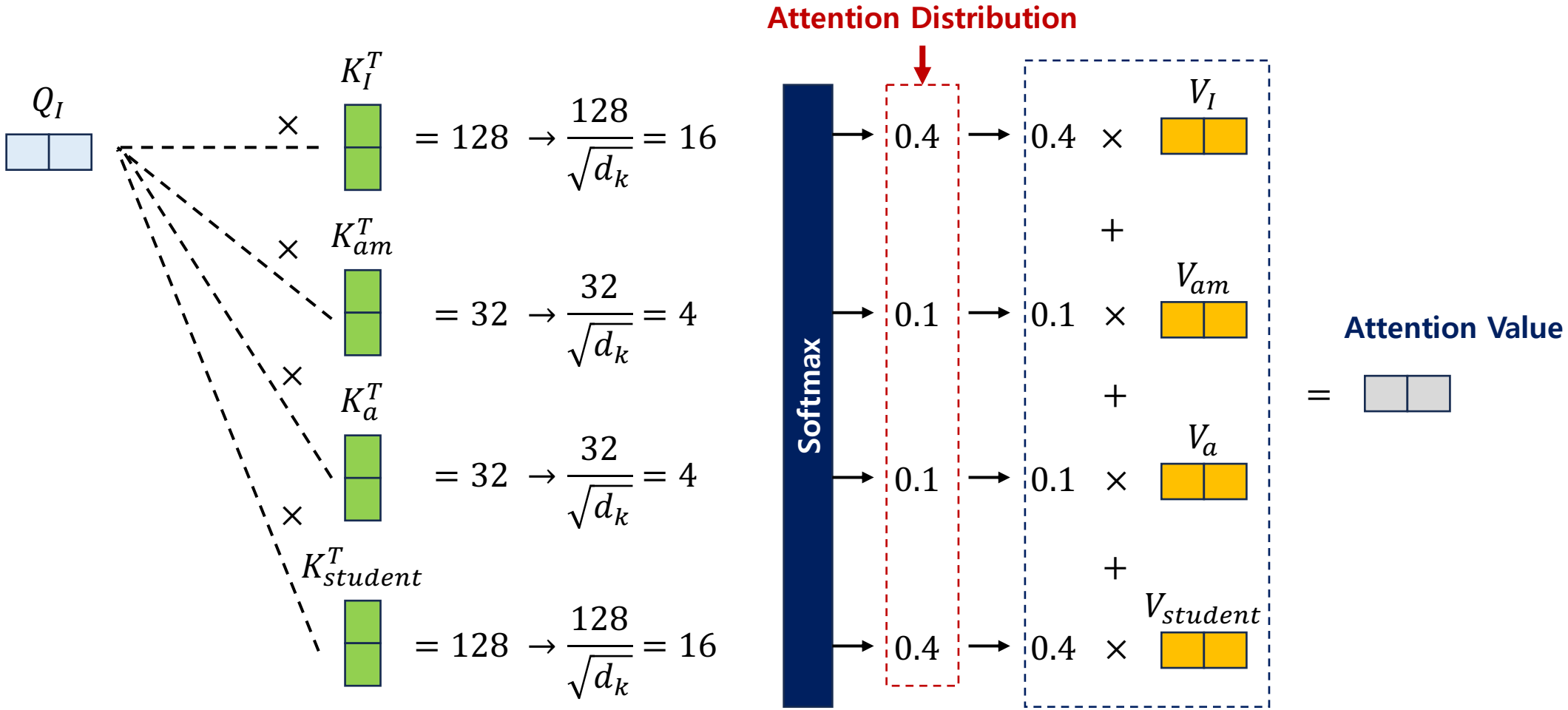
Self-Attention score

논문에서는 스케일링을 위해 $\sqrt{d_k}$를 활용

➔ 두 벡터의 내적값을 스케일링하는 값

Self-attention mechanism

- 2) Scaled dot-product Attention
  - Softmax ➔ Attention distribution 계산 ➔ Weighted sum ➔ 단어 I에 대한 Attention Value 획득

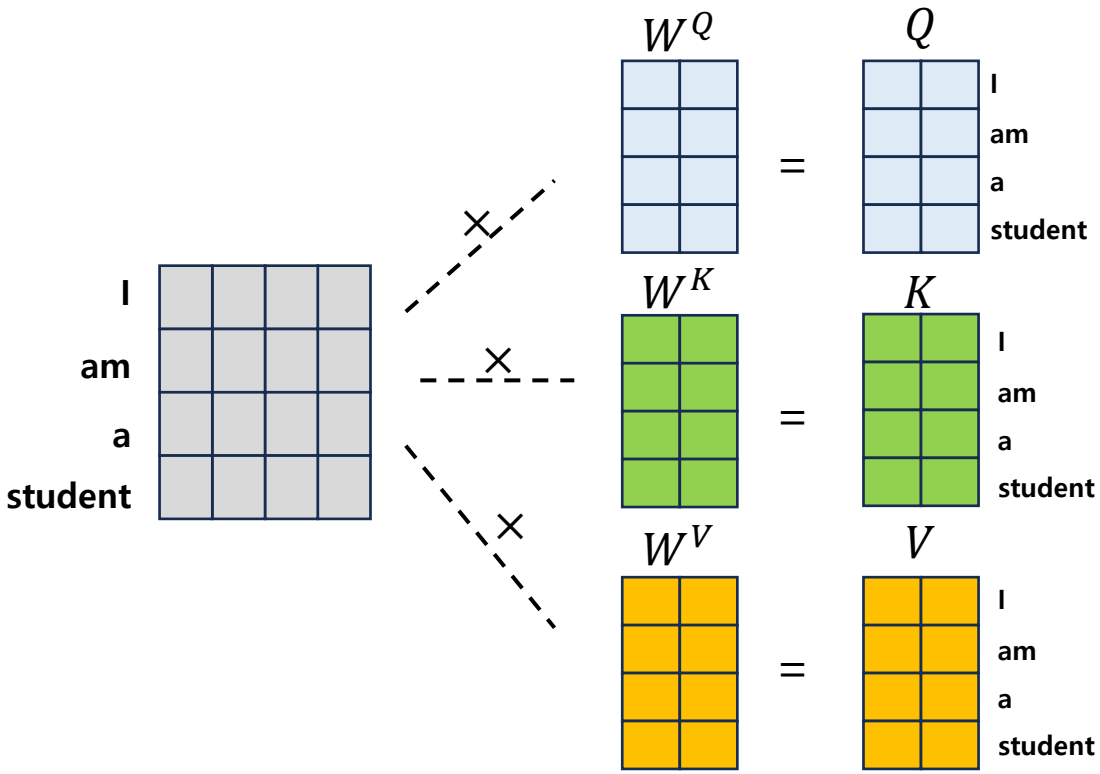


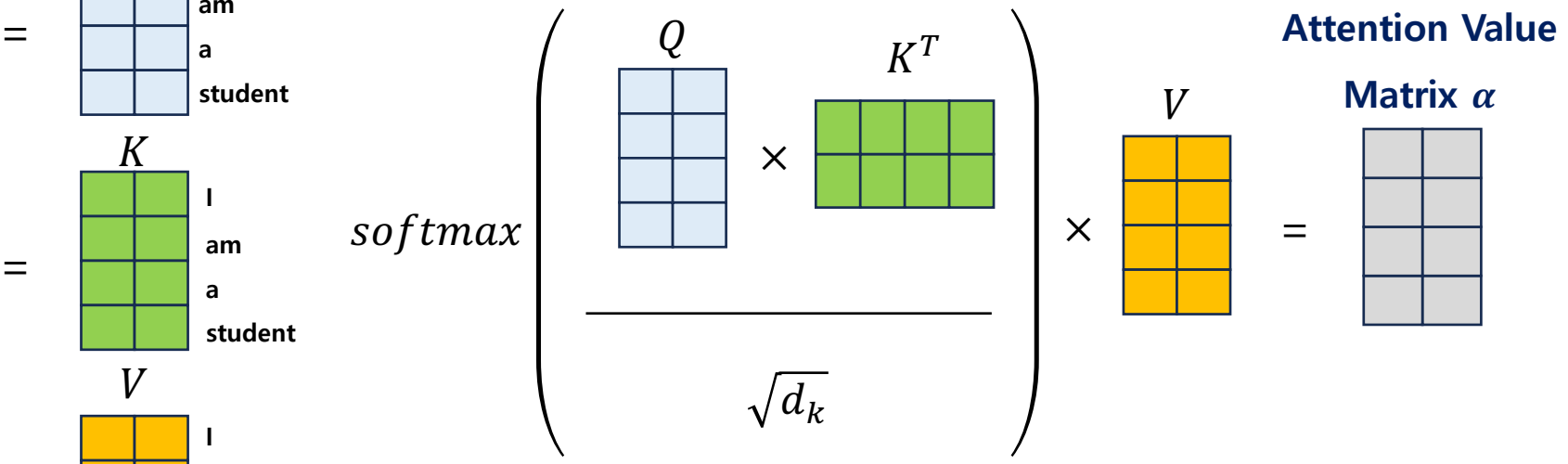
Self-attention mechanism

- **Batch with matrix operations (No vector operations)**
  - 실제로는 Q, K, V 벡터 연산이 아닌 행렬 연산으로 구현됨

$$Attention(Q,K,V) = softmax(QK^T)V$$

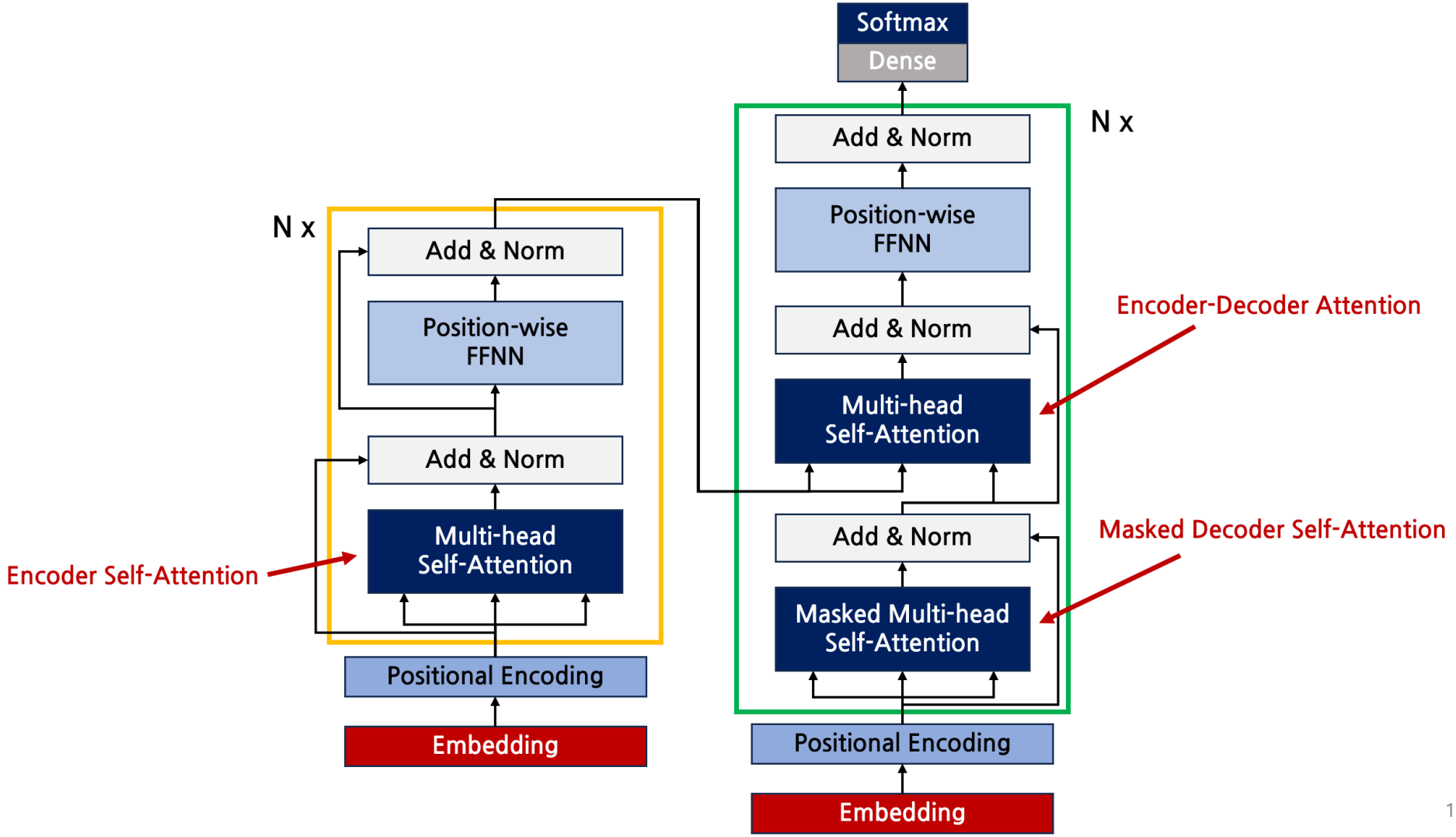
$$Self-Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$





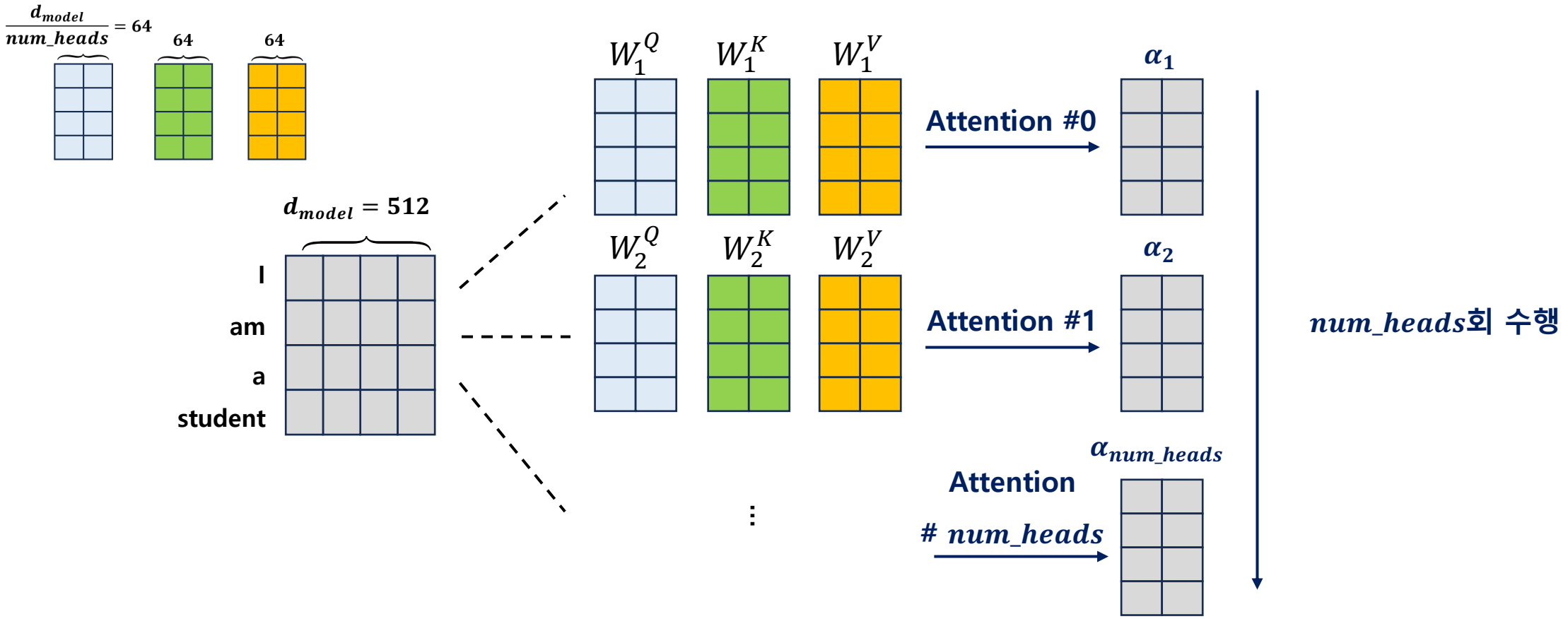
Multi-head Attention

- Transformer가 attention을 병렬적으로 수행하는 방법



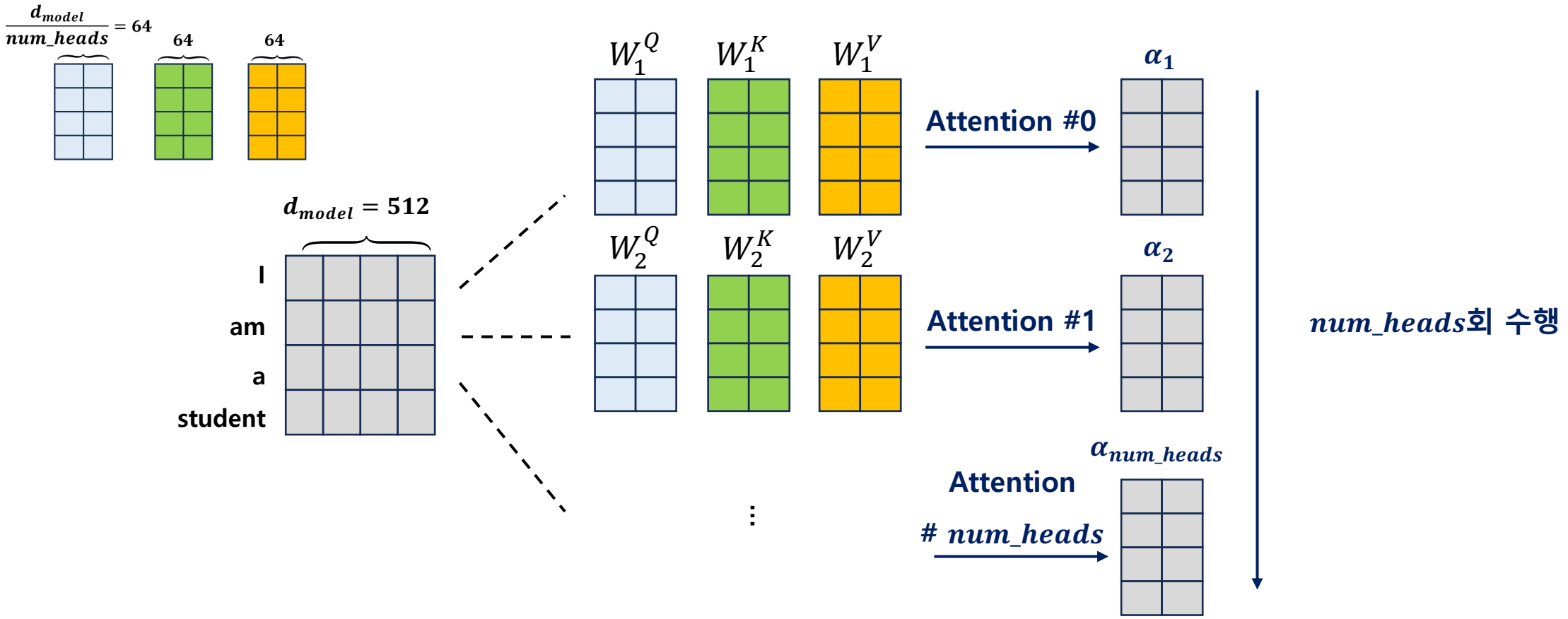
Multi-head Attention

- 한 번의 Attention 보다 여러 번의 Attention을 병렬로 사용하는 것이 더 효과적임
  - d_{model}의 차원을 갖는 단어 vector를 num_heads로 나눈 차원으로 Attention 수행
    - 512 차원의 각 단어 vector를 8로 나누어 64차원의 Q, K, V벡터로 Attention 수행



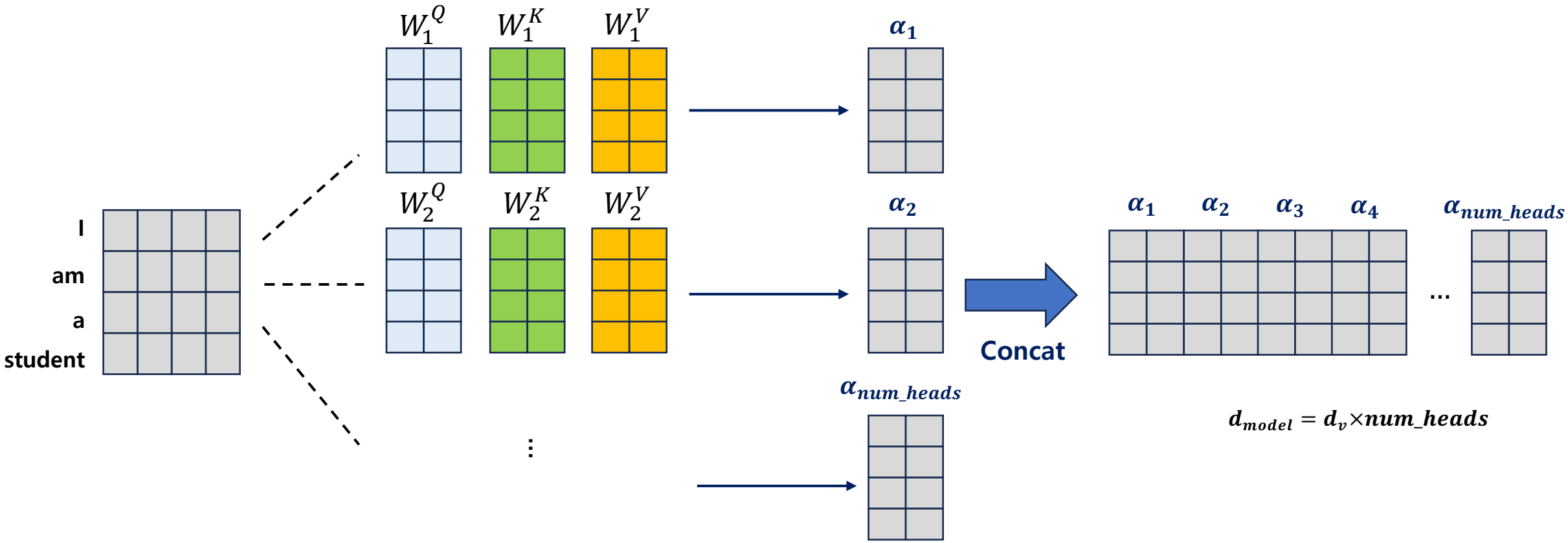
Multi-head Attention

- 한 번의 Attention 보다 여러 번의 Attention을 병렬로 사용하는 것이 더 효과적임
  - d_{model}/num_heads 차원을 갖는 Q, K, V에 대해서 num_heads회 Attention 수행
    - 각 Attention mechanism이 병렬로 수행 ➔ **각각의 Attention value matrix를 "Attention Head"**



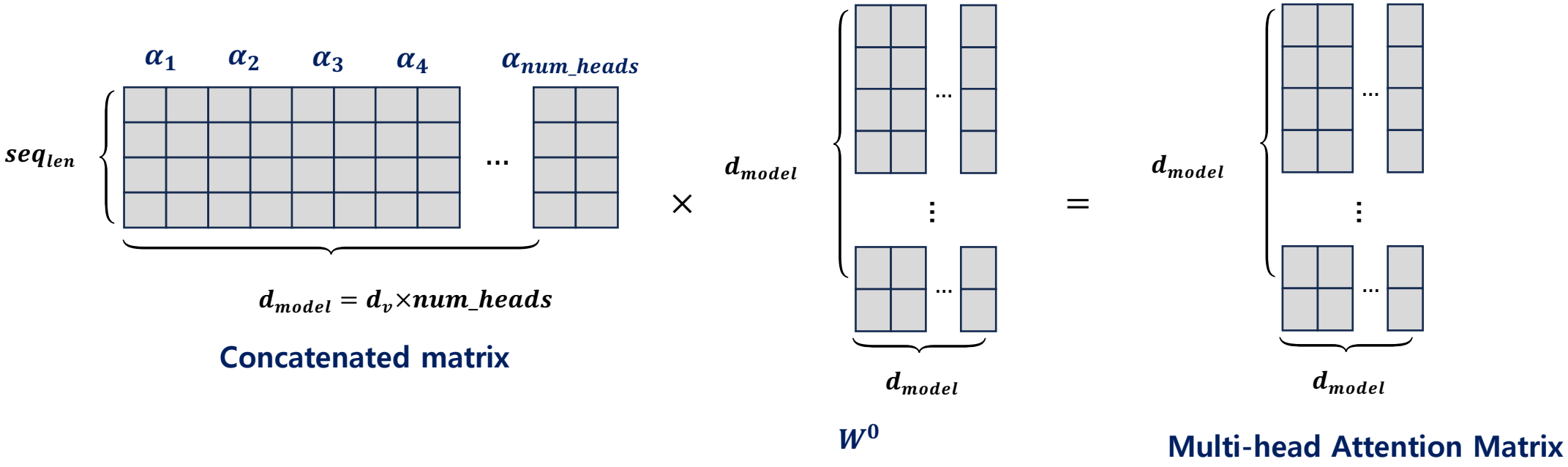
Multi-head Attention

- 모든 Attention Head를 concatenation ➔ Concatenated matrix
  - 연결된 Attention head matrix (Concatenated matrix)의 크기 = (seq_{len}, d_{model})
    - seq_{len} : 문장 길이



Multi-head Attention

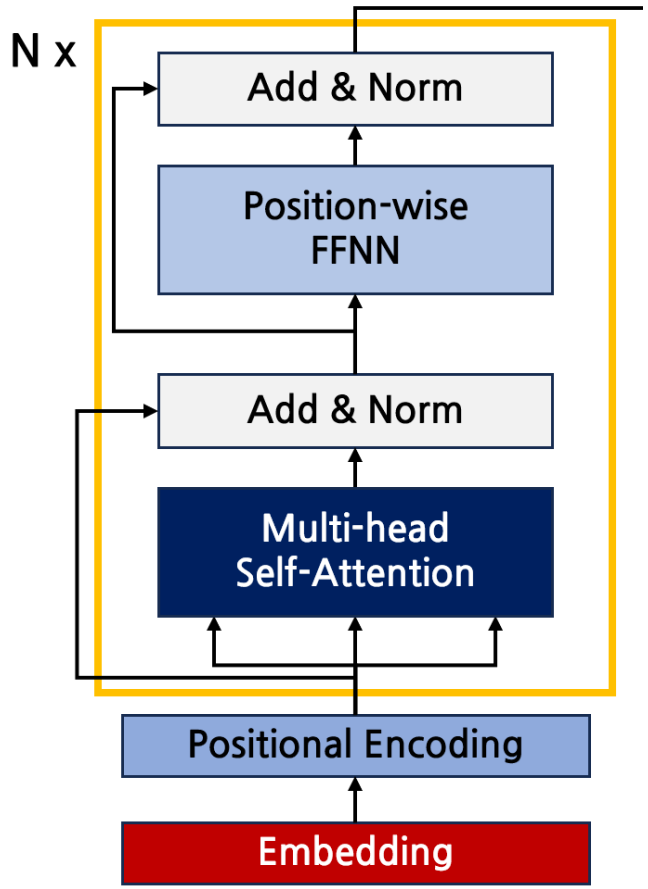
- Multi-head Attention matrix : Concatenated matrix와 가중치 W^0의 곱
  - 크기: $(seq_{len}, d_{model}) \times (d_{model}, d_{model}) = (seq_{len}, d_{model})$
    - Encoder의 입력 문장 행렬 크기와 동일
      - ➔ Multi-head Attention 단계를 끝 마쳐도 Encoder의 입력으로 들어왔던 행렬의 크기 유지
      - 그래야 다음 Encoder의 입력으로 활용 가능 (Note. Transformer = Encoders/Decoders)



Encoder in Transformer

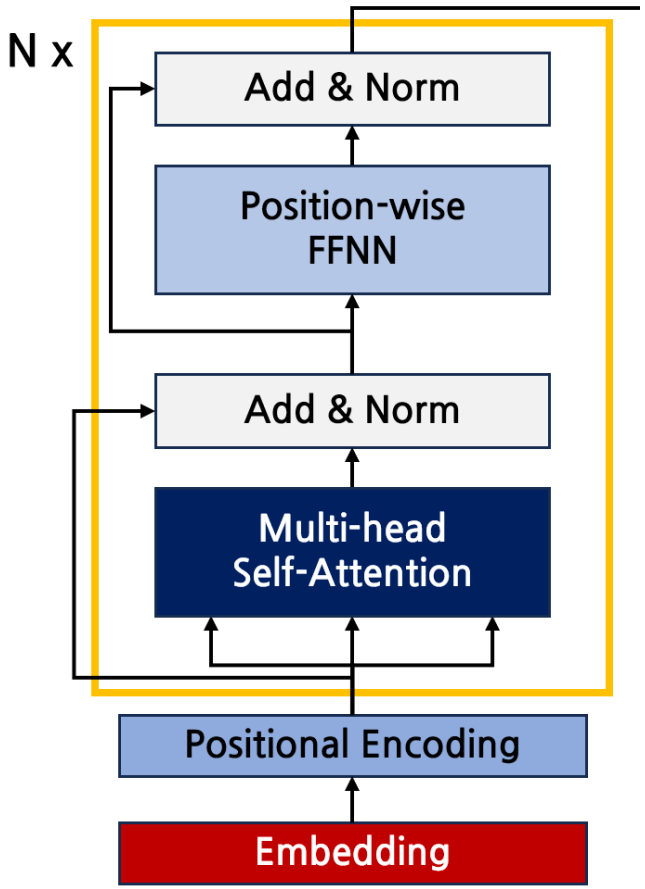
- Note. Transformer는 여러 개의 Encoder와 Decoder로 구성
  - N (num_layers)개의 encoder 층을 쌓음
    - 논문에서는 총 6개의 encoder 층

- Transformer Encoder의 핵심 sublayers
  - Multi-head Self-Attention Layer
  - Position-wise FFNN (Feed Forward Neural Network)



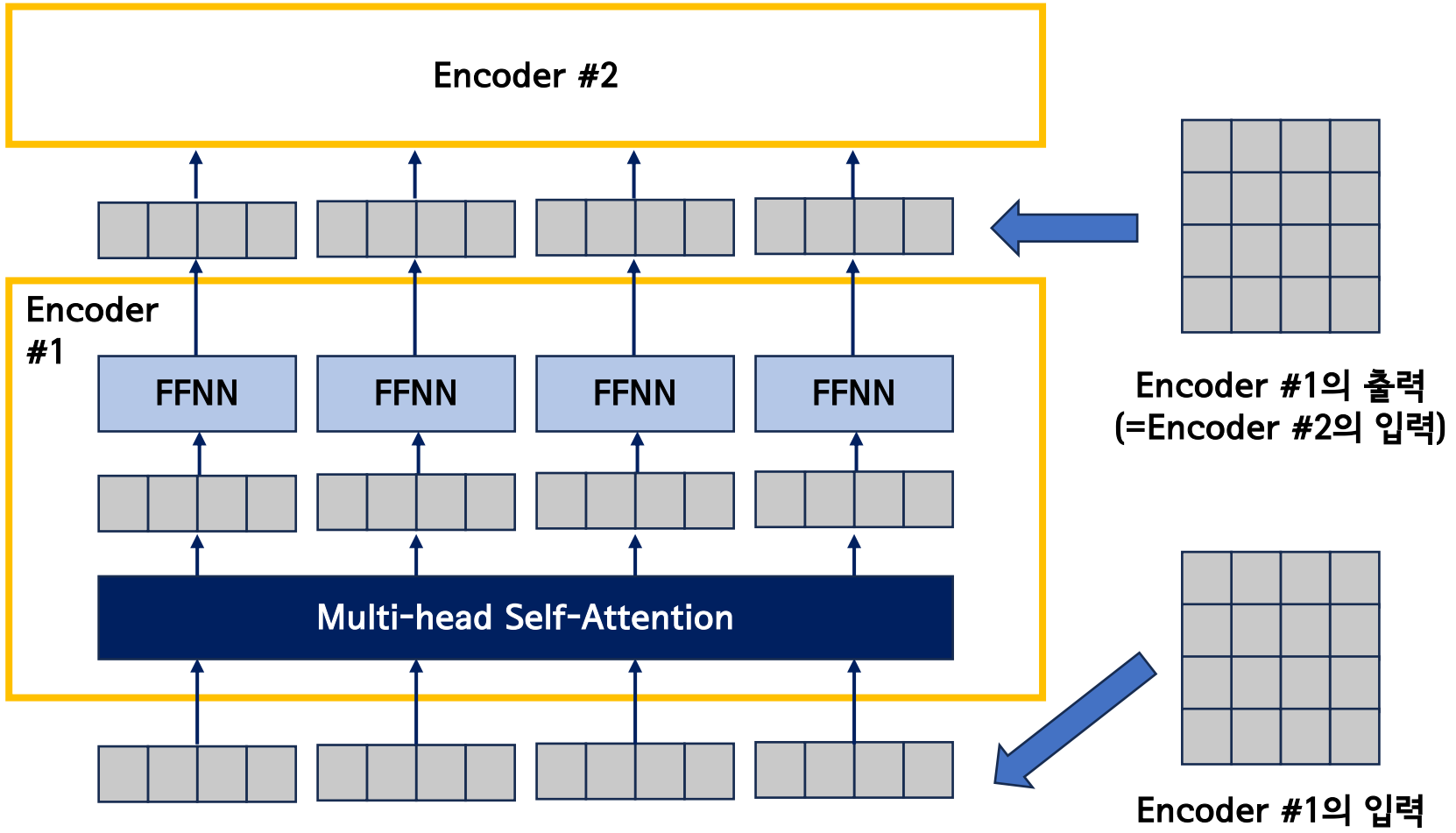
Encoder in Transformer

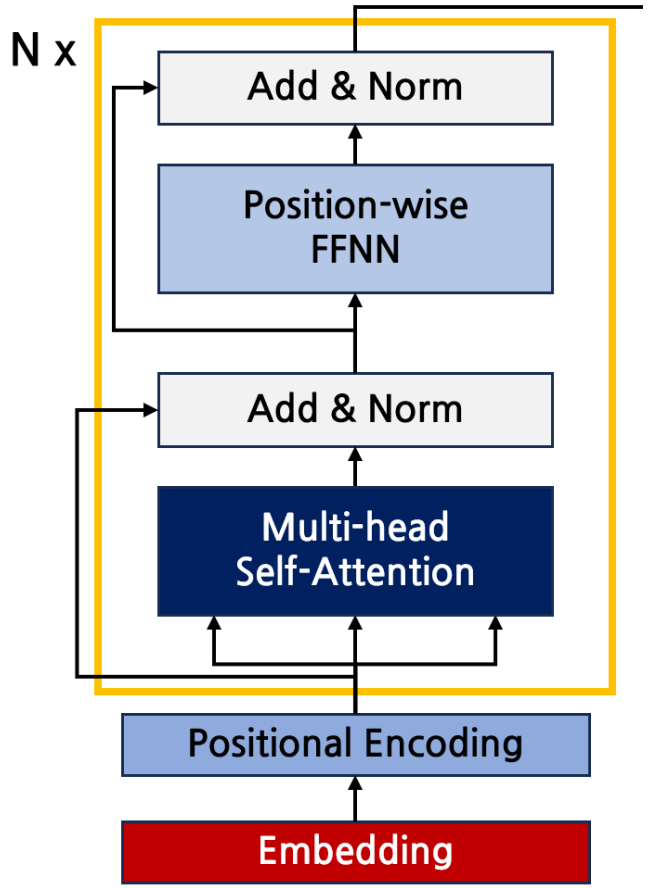
- Position-wise FFNN
  - 일반적인 deep neural network의 feed forward 신경망과 같음
    - 각각의 학습 노드가 서로 완전하게 연결 (Fully-connected)
    - $FFNN(x) = \alpha(xW_1 + b_1)W_2 + b_2$
      - α : activation function
  - Encoder와 decoder에서 공통으로 갖고 있는 sublayer



Encoder in Transformer

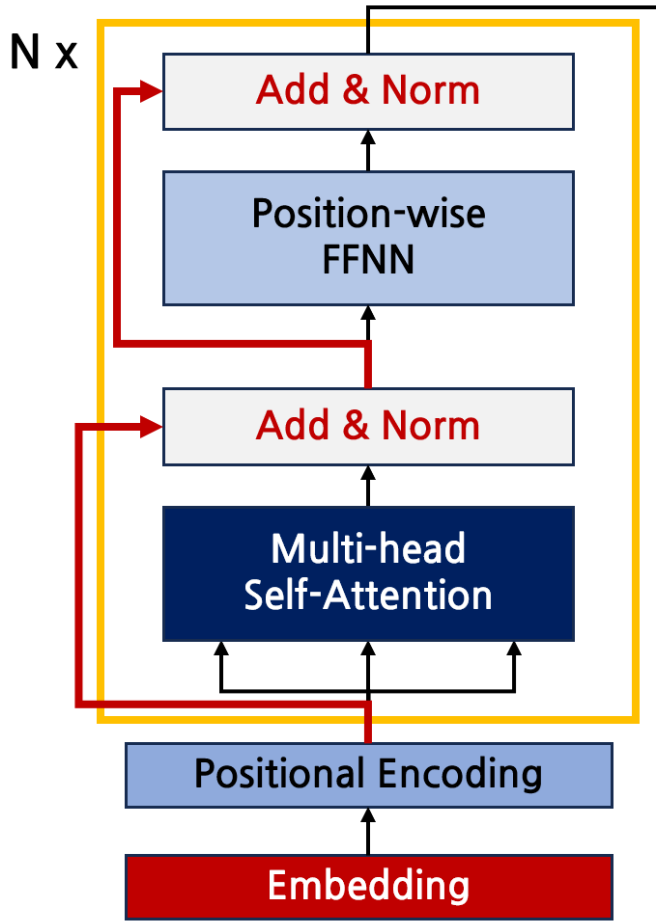
- Position-wise FFNN
  - 각 vector들이 multi-head attention sublayer ➔ FFNN sublayer 통과하는 과정





Encoder in Transformer

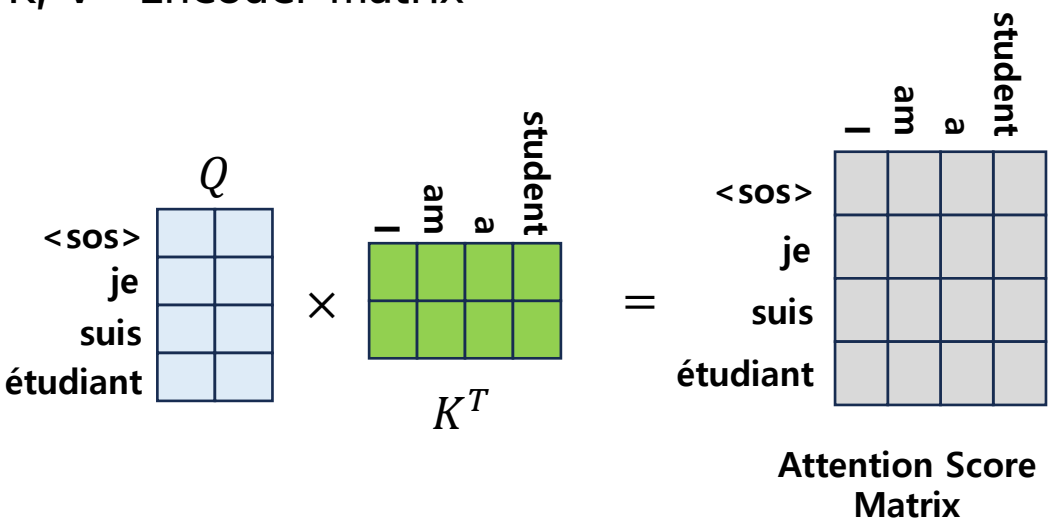
- 추가적인 기법들
  - Residual Connection
    - Sublayer의 출력에 입력 값을 더해주는 기법
      - $H(x) = x + F(x)$
    - 입력과 출력은 동일한 차원 ➔ 덧셈 가능
    - 모델 학습에 효과가 있음 (https://arxiv.org/pdf/1512.03385.pdf)
  - Layer Normalization
    - Matrix의 scale을 통일해주는 기법 (평균, 분산 사용)

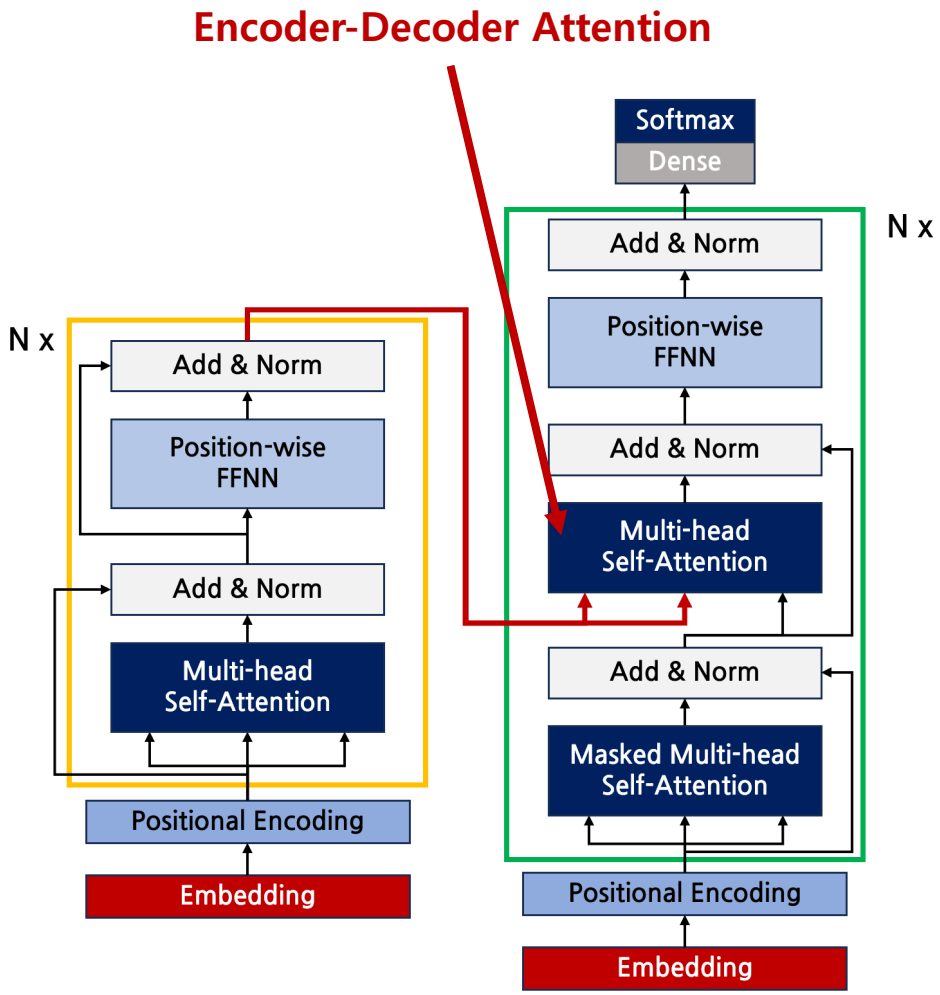


Decoder in Transformer

- N (num_layers)개의 encoder 연산 수행 후 Decoder로 입력 전달
  - Multi-head Attention 수행, but Self-Attention은 아님
    - Note. Self-Attention ➔ Q == K == V인 경우

- Encoder-Decoder Attention
  - Q : Decoder matrix
  - K, V : Encoder matrix



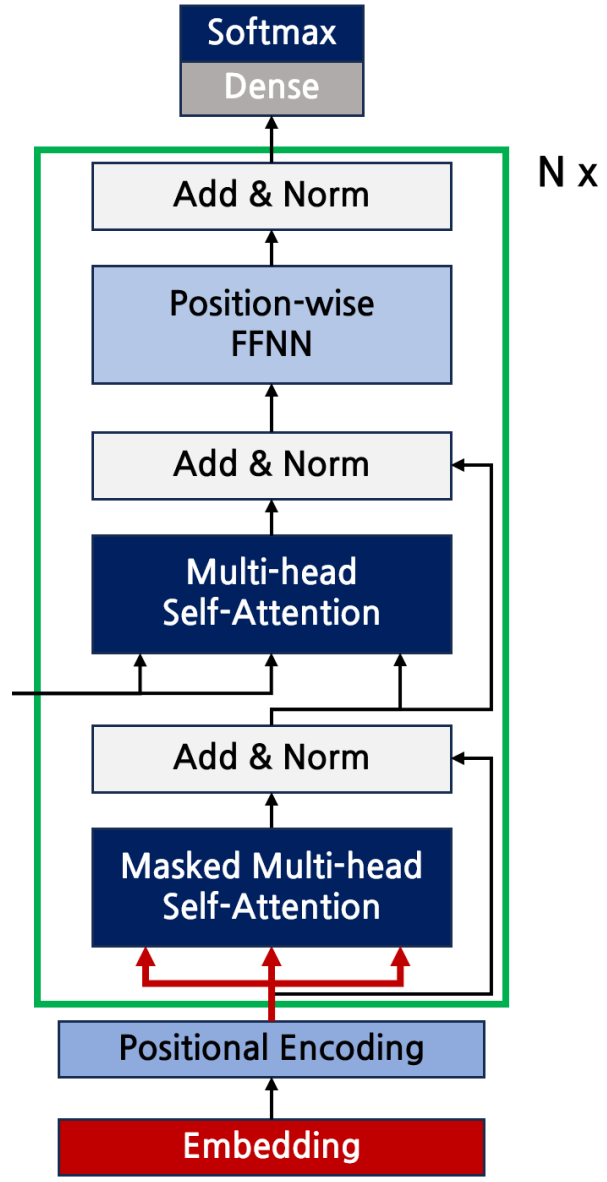


Decoder in Transformer

- **Decoder의 입력** (Encoder와 동일)
  - Embedding layer/Positional encoding 거친 후 문장이 입력

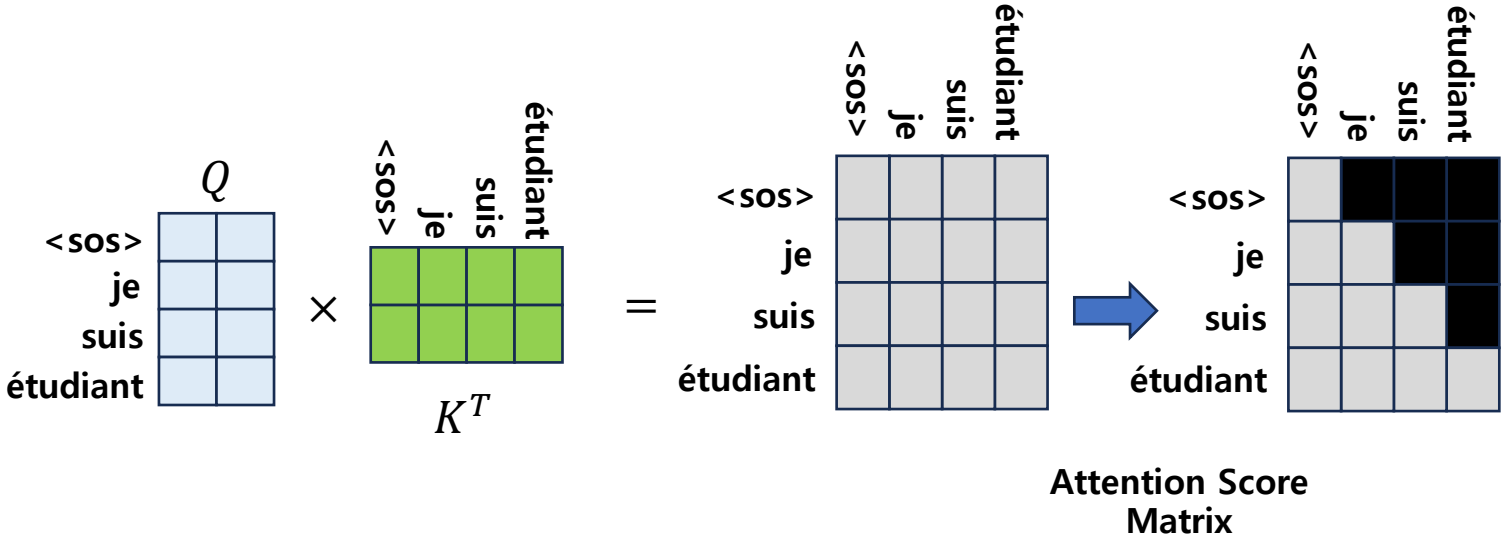
- **Teacher forcing (in training)**
  - (Seq2Seq model) 학습 과정에서 정답 문장에 해당되는 값을 입력
  - **(Transformer) 문장을 통째로 입력으로**
    - **➔ 현 시점의 정답 + 이후의 정답까지 참조!**

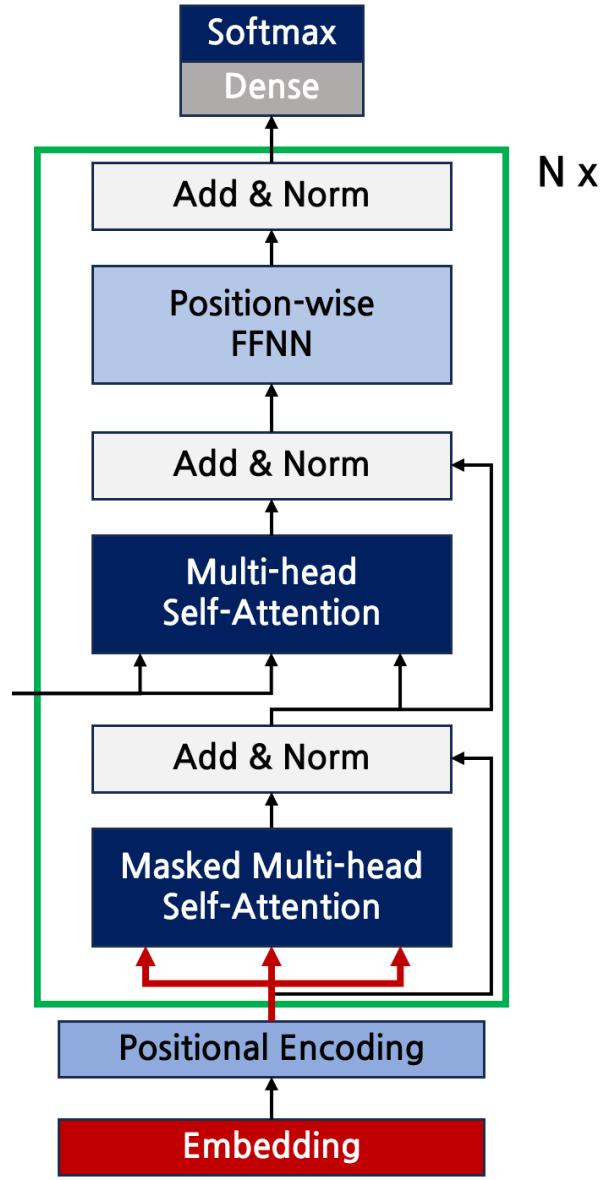
  - 'I am a student' ➔ 'je suis étudiant' / 'suis' 예측 시점의 decoder의 입력
    - (Seq2Seq model) <sos>, 'je'
    - (Transformer) <sos>, 'je', 'étudiant'



Decoder in Transformer

- Self-Attention & Look-ahead mask의 도입
  - 현재 시점 보다 미래에 있는 단어들을 참고하지 못하도록 함
    - Mask의 개념 도입
      - 값을 0으로 설정

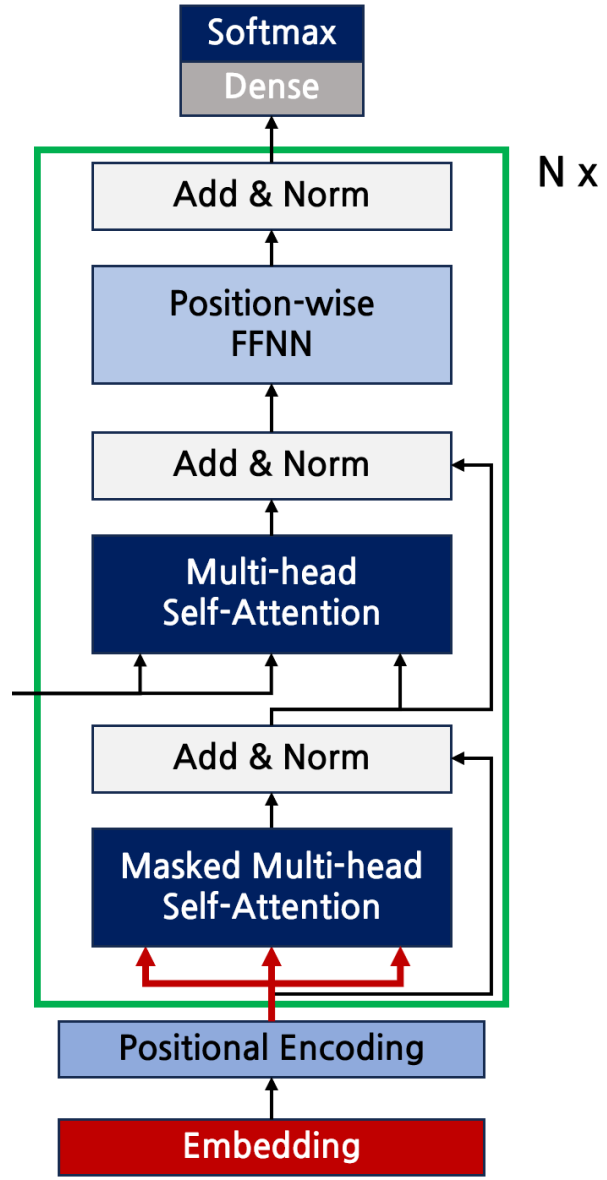




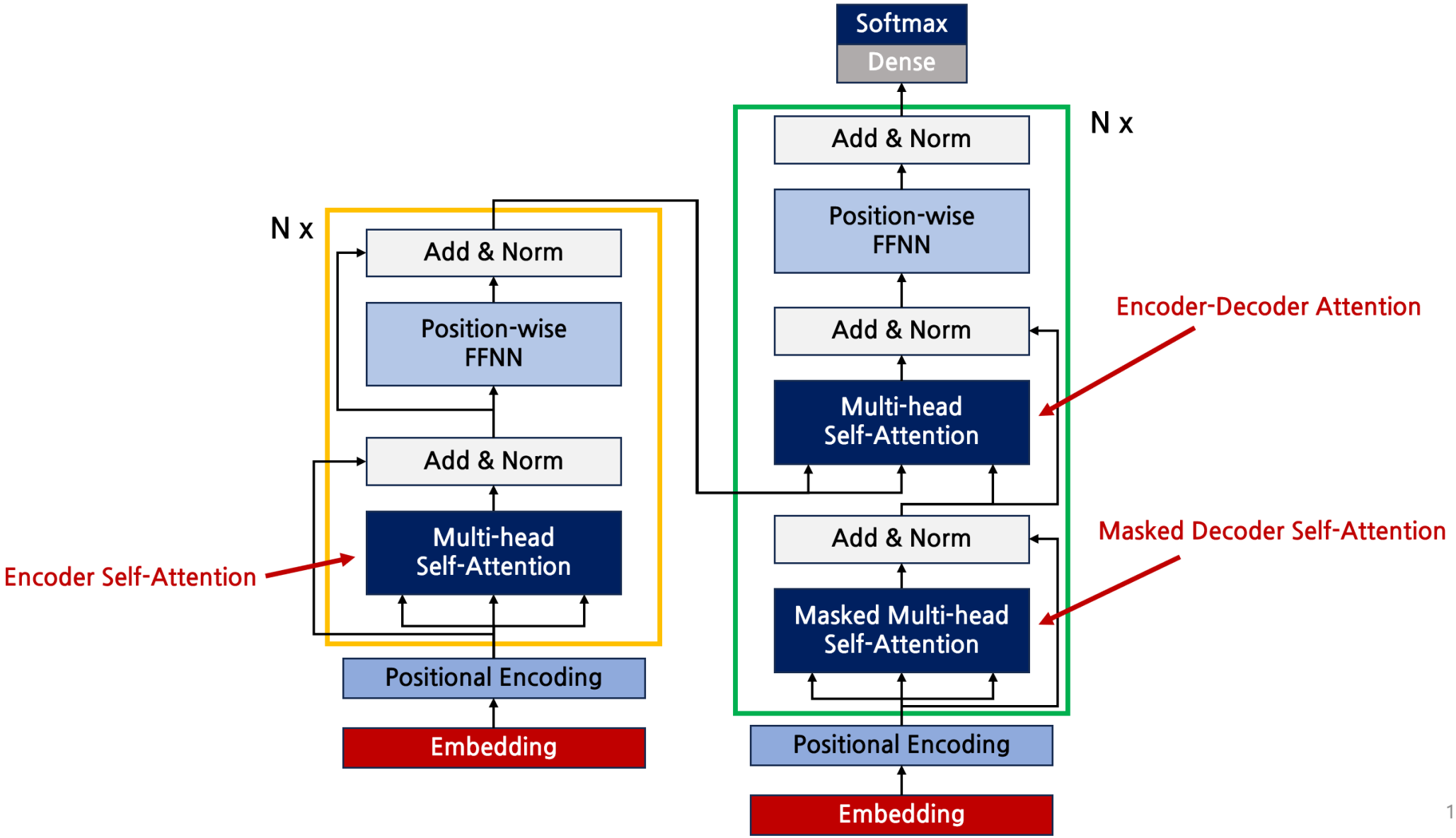
Decoder in Transformer

- Note. Transformer는 여러 개의 Encoder와 Decoder로 구성
  - N (num_layers)개의 decoder 층을 쌓아서 연산을 반복
    - 논문에서는 총 6개의 decoder 층

- Dense와 softmax는 Seq2Seq 모델과 동일하게 동작
  - $\tilde{s}_t = tanh(FFNN(x) + b_c)$
  - $\hat{y}_t = softmax(\tilde{s}_t + b_y)$

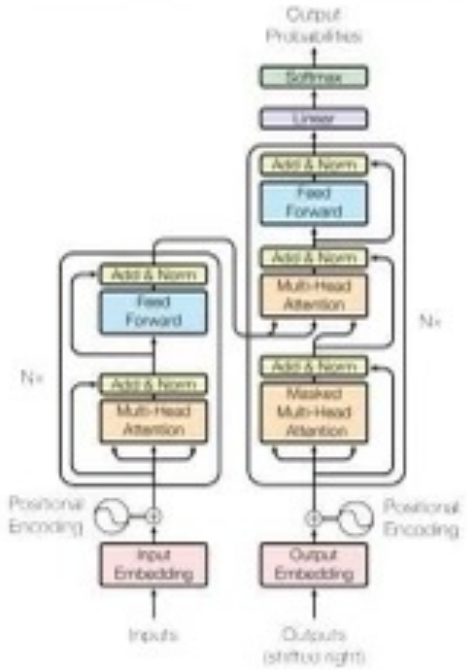


Transformer summary



현실

AI Engineers

The Interview	The Job
	`import transformers`

End of slide