자연어 처리 기초

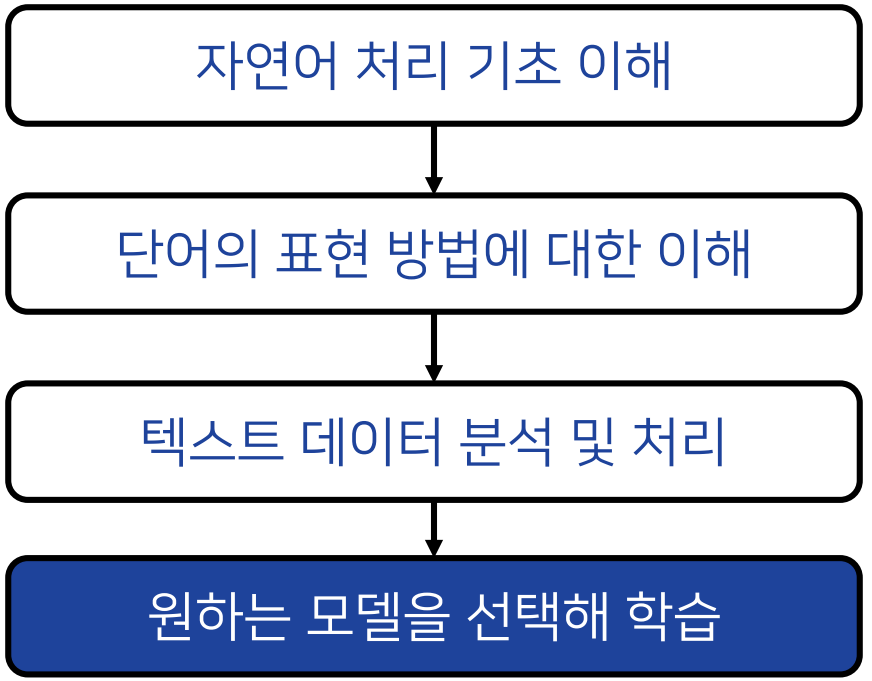
Natural Language Processing Basic

나동빈(dongbinna@postech.ac.kr)

Pohang University of Science and Technology

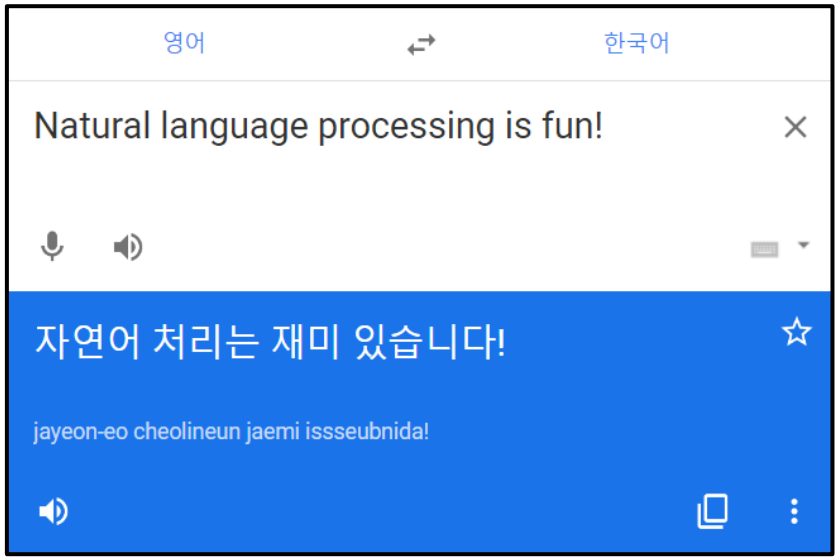
학습자 목표

- 학습자 목표는 다음과 같습니다.



자연어 처리 개요

- **자연어(natural language)**: 일상생활에서 사용하는 언어를 의미합니다.
- **자연어 처리 기술의 활용 분야**: 기계 번역, 음성 인식, 텍스트 분류, 챗봇, 감성 분석 등



기계 번역
(machine translation)

선착순 할인! 자연어 처리 강의 소개	**스팸(spam)**
이전에 연락 드렸던 담당자입니다.	**햄(ham)**

스팸 분류기
(spam detection)

단어의 표현과 유사도 분석

데이터의 표현

- 컴퓨터는 일반적으로 처리할 데이터를 행렬(matrix)이나 벡터(vector)와 같은 형태로 다룹니다.
  - 예) 이미지 처리 분야: 이미지를 텐서, 행렬 혹은 벡터로 표현할 수 있습니다.
  - 예) 자연어 처리 분야: 문장 혹은 단어를 벡터로 표현할 수 있습니다.

이미지(image)

105	202	124	198	125	102	199	205
152	190	197	180	103	202	200	122
188	144	153	165	185	134	111	123
105	202	124	198	125	102	199	205
152	190	197	180	103	202	200	122
188	144	153	165	185	134	111	123
101	203	243	111	170	198	185	203
213	202	154	144	164	195	174	152

행렬(matrix)

105
202
⋮
174
152

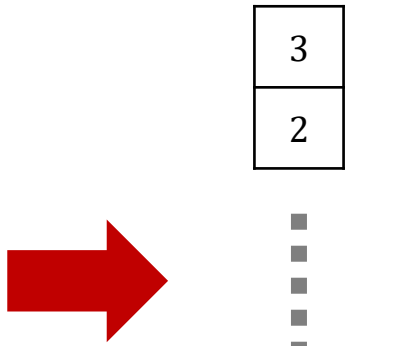
벡터(vector)

데이터의 표현

- 컴퓨터는 일반적으로 처리할 데이터를 행렬(matrix)이나 벡터(vector)와 같은 형태로 다룹니다.
  - 예) 이미지 처리 분야: 이미지를 텐서, 행렬 혹은 벡터로 표현할 수 있습니다.
  - 예) 자연어 처리 분야: 문장 혹은 단어를 벡터로 표현할 수 있습니다.

"컴퓨터 좋아요, 컴퓨터 좋아요, 컴퓨터 재미있어요."

텍스트(text)



벡터(vector)

원-핫 인코딩 (One-hot Encoding)

- 단어를 수치적으로 표현하는 가장 기본적인 방법입니다.
- 전체 단어 목록이 다음과 같다고 해봅시다.

컴퓨터	좋아요	싫어요	재미있어요	라디오	라면	선풍기	스마트폰

- "라면"이라는 단어는 다음과 같은 벡터로 표현할 수 있습니다.

컴퓨터	좋아요	싫어요	재미있어요	라디오	라면	선풍기	스마트폰
0	0	0	0	0	1	0	0

[0, 0, 0, 0, 0, 1, 0, 0]

데이터의 표현

- 데이터를 벡터(vector)로 표현할 때의 장점을 생각해 봅시다.
  - 두 데이터 간의 수치적인 비교가 가능합니다.
  - 머신러닝 모델의 입력으로 넣을 수 있습니다.

“컴퓨터 좋아요”	1	1	0	0	0	0	0	0
“컴퓨터 좋아요, 좋아요.”	1	2	0	0	0	0	0	0
“컴퓨터 싫어요”	1	0	1	0	0	0	0	0

BoW (Bag of Words)

- 순서는 무시하고, 단어의 **출현 빈도(frequency)**를 계산하는 데이터 표현 방법입니다.

"컴퓨터 좋아요, 컴퓨터 좋아요, 컴퓨터 재미있어요."

텍스트(text)

컴퓨터	좋아요	싫어요	재미있어요	라디오	라면	선풍기	스마트폰
3	2	0	1	0	0	0	0

문서 단어 행렬 (Document-Term Matrix, DTM)

- 여러 문서에 대하여 BoW를 적용하여 하나의 행렬(2차원 배열)로 표현한 것을 말합니다.
  - 문서 1: “컴퓨터가 좋아요.“
  - 문서 2: “컴퓨터가 좋아요, 컴퓨터가 좋아요, 컴퓨터가 좋아요.”
  - 문서 3: “컴퓨터가 싫어요.”

	컴퓨터	좋아요	싫어요	재미있어요	라디오	라면	선풍기	스마트폰
문서 1	1	1	0	0	0	0	0	0
문서 2	3	3	3	0	0	0	0	0
문서 3	1	0	1	0	0	0	0	0

BoW (Bag of Words) 단점

- Bag of Words에는 **희소 문제(sparsity problem)**가 있습니다.
  - 우리가 모든 단어를 쓴다면, 벡터의 크기가 수십만 차원이 될 수 있습니다.
  - 실제 유의미한 데이터의 비중에 비해서 많은 메모리가 낭비됩니다.
- 카운트 기반이므로, 처음 본 단어는 처리할 수 없습니다.
- 단어의 순서를 무시합니다.

Q. 데이터의 상당수가 0을 값으로 가지게 됩니다.

컴퓨터	좋아요	싫어요	재미있어요	라디오	라면	선풍기	스마트폰
1	0	0	0	0	0	0	0

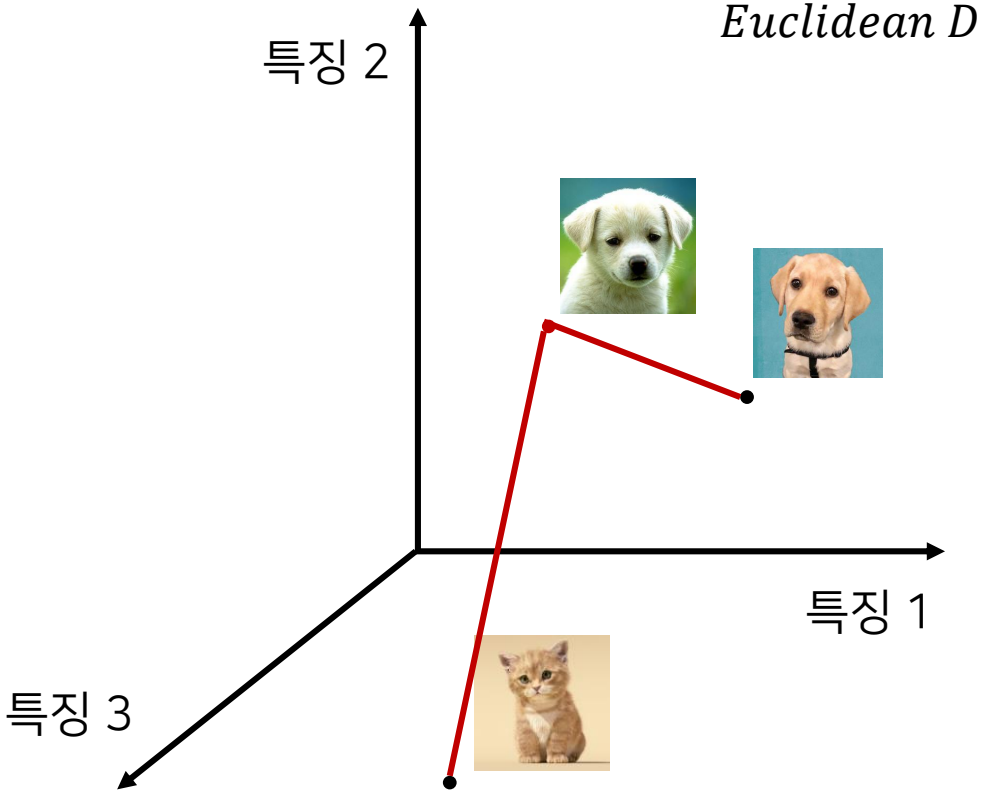
유사도 측정법 (Similarity Measure)

- 머신러닝 분야에서는 두 벡터 (데이터) 간의 유사성을 측정해야 하는 일이 많습니다.
  - 예) 이미지 검색: 두 이미지가 얼마나 유사한지 측정합니다.
  - 예) 유사 문서 검색: 두 문장이 얼마나 유사한지 측정합니다.

유클리드 거리 (Euclidean Distance)

- 유클리드 거리(직선거리)는 피타고라스의 정리를 이용해서 계산할 수 있습니다.

$$Euclidean\ Distance = D(A,B) = \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + \cdots + (A_n - B_n)^2}$$



거리 측정 결과

$D($  ,  $) < D($  ,  $)$

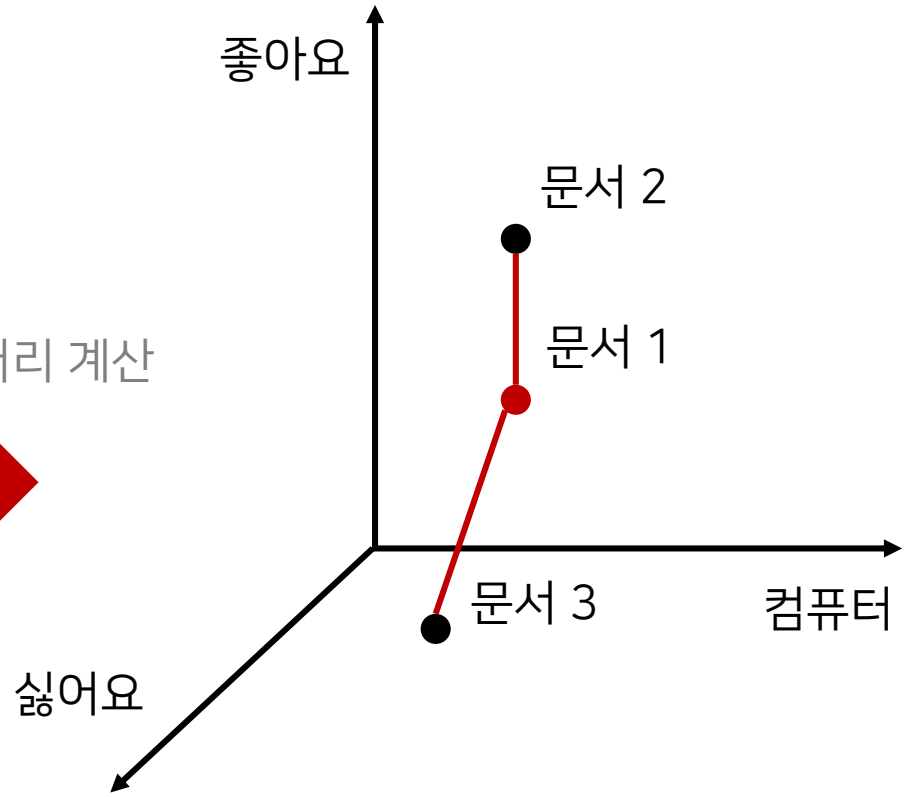
유클리드 거리 (Euclidean Distance)

- 텍스트 또한 벡터(vector)값으로 표현했다면 유클리드 거리 계산이 가능합니다.
  - 문서 1: “컴퓨터가 좋아요.”
  - 문서 2: “컴퓨터가 좋아요, 좋아요.”
  - 문서 3: “컴퓨터가 싫어요.”

Bag of Words (BoW) 표현

	컴퓨터	좋아요	싫어요
문서 1	1	1	0
문서 2	1	2	0
문서 3	1	0	1

유클리드 거리 계산



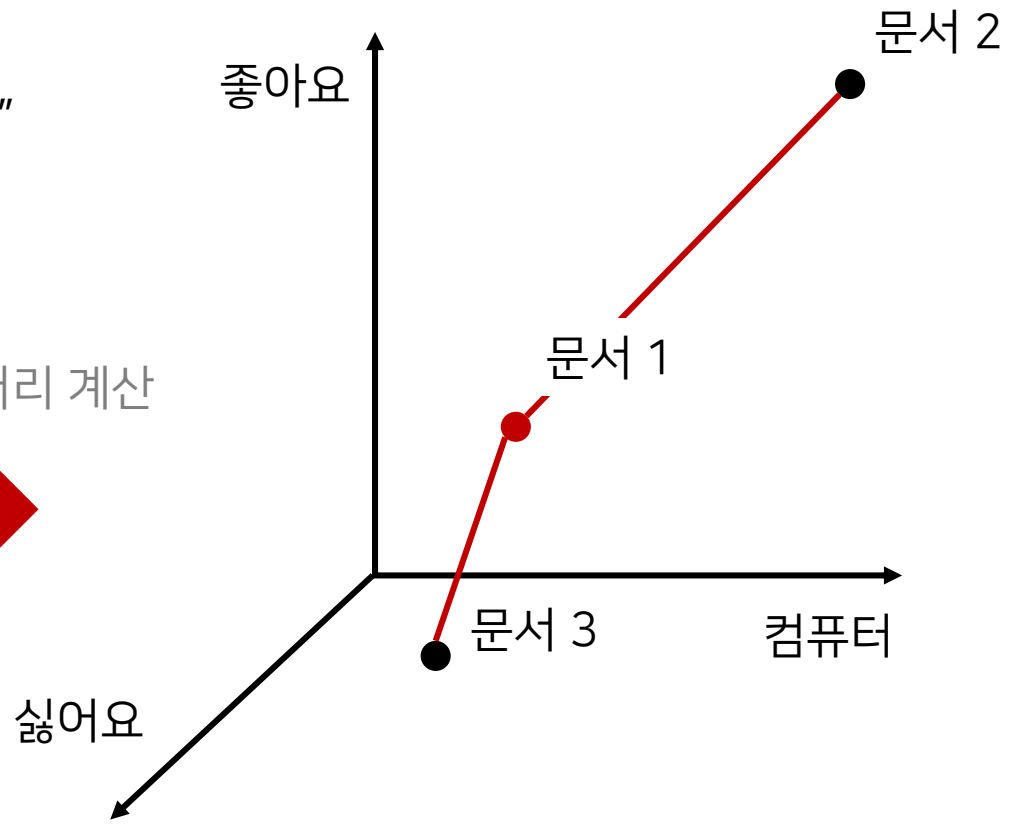
유클리드 거리 (Euclidean Distance)

- 다만, 유클리드 거리 계산이 효과적이지 않은 경우도 있습니다.
  - 문서 1: “컴퓨터가 좋아요.“
  - 문서 2: “컴퓨터가 좋아요, 컴퓨터가 좋아요, 컴퓨터가 좋아요.”
  - 문서 3: “컴퓨터가 싫어요.”

Bag of Words (BoW) 표현

	컴퓨터	좋아요	싫어요
문서 1	1	1	0
문서 2	3	3	0
문서 3	1	0	1

유클리드 거리 계산



유클리드 거리 (Euclidean Distance)

- 소스코드

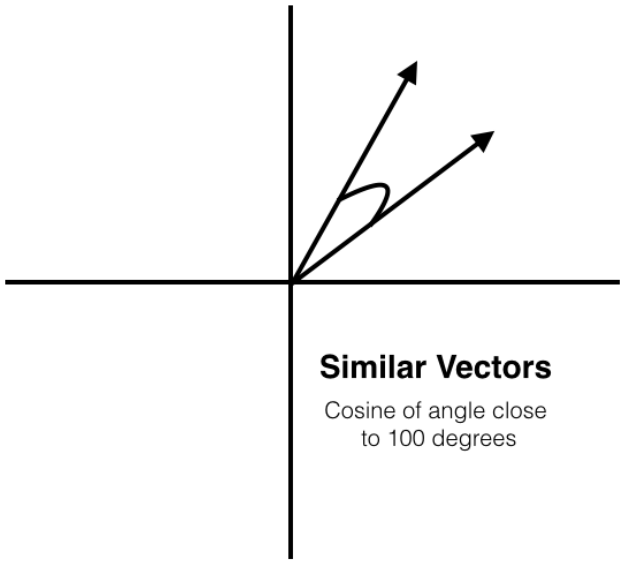
```
from numpy.linalg import norm
import numpy as np

def euclidean_distance(A, B):
    return np.linalg.norm(A - B)

document_1 = np.array([1, 1, 0])
document_2 = np.array([3, 3, 0])
document_3 = np.array([1, 0, 1])

# 문서 1과 문서 2의 유클리드 거리 출력
print(euclidean_distance(document_1, document_2))
# 문서 1과 문서 3의 유클리드 거리 출력
print(euclidean_distance(document_1, document_3))
```

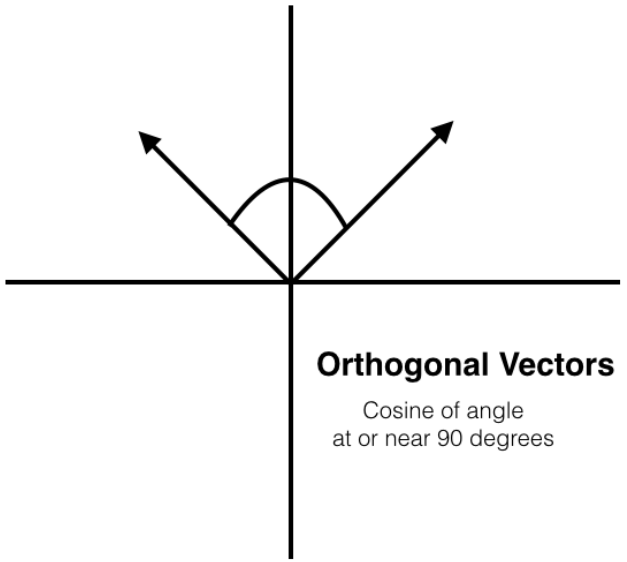
- 실행 결과

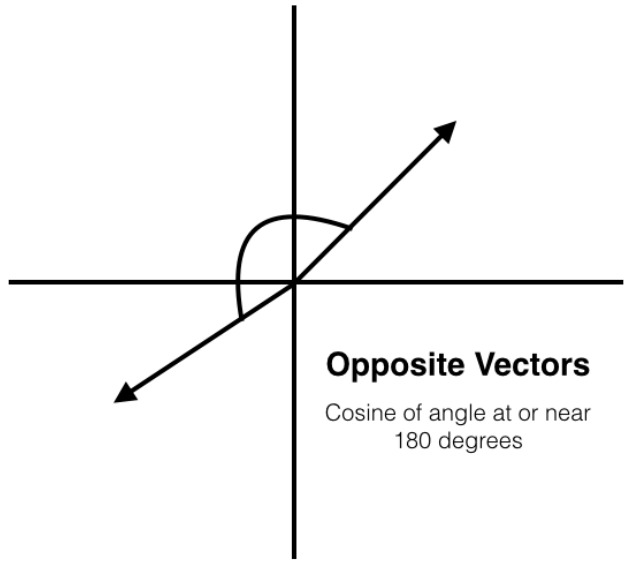
```
2.8284271247461903
1.4142135623730951
```

코사인 유사도 (Cosine Similarity)

- 벡터의 크기는 고려하지 않고, 두 벡터 사이의 각도만 고려하는 측정법입니다.
- 방향이 얼마나 유사한지를 1부터 −1 사이의 값으로 표현합니다. (방향이 동일한 경우: 1)
- $Cosine\ \text{Similarity} = \cos(\theta) = \frac{\mathrm{A} \cdot \mathrm{B}}{\|\mathrm{A}\|\|\mathrm{B}\|}$

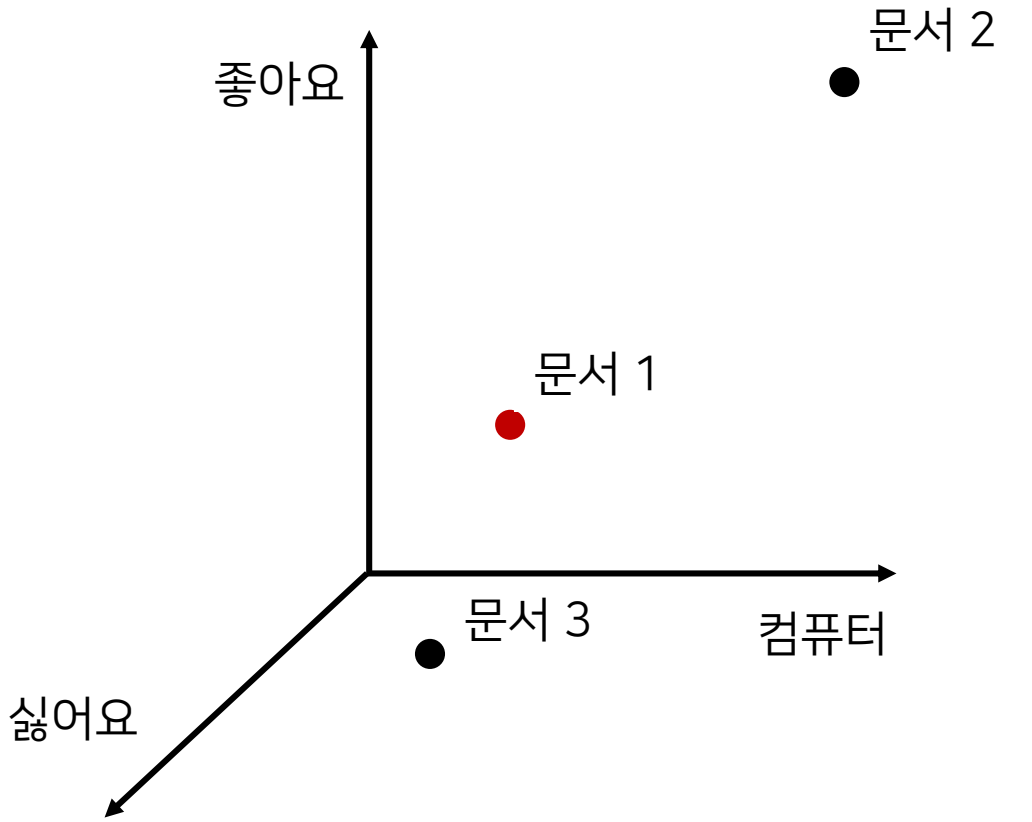






코사인 유사도 (Cosine Similarity)

- 코사인 유사도를 이용하면 문서의 길이가 달라도 유사도 측정이 수월합니다.



코사인 유사도 측정 결과

(1에 가까울 수록 유사)

$$D(\text{문서 } 1, \text{문서} 2) = 1$$

$$D(\text{문서 } 1, \text{문서} 3) = 0.5$$

코사인 유사도 (Cosine Similarity)

- 소스코드

```python
from numpy import dot
from numpy.linalg import norm
import numpy as np

def cosine_similarity(A, B):
    return dot(A, B) / (norm(A) * norm(B))

document_1 = np.array([1, 1, 0])
document_2 = np.array([3, 3, 0])
document_3 = np.array([1, 0, 1])

# 문서 1과 문서 2의 코사인 유사도 출력
print(cosine_similarity(document_1, document_2))
# 문서 1과 문서 3의 코사인 유사도 출력
print(cosine_similarity(document_1, document_3))
```

- 실행 결과

```
1.0
0.5
```

참고사항

- 두 벡터 간의 유사성을 계산하는 측정법들을 알아보았습니다.
  - 유클리드 거리, 코사인 유사도
- 이때 특정한 데이터(이미지, 문장, 오디오 등)를 벡터로 표현하는 방법은 별개의 문제입니다.
  - 데이터에서 특징을 잘 추출하여 **벡터로 표현하는 방법에 대한 연구 분야**가 있습니다.
    - 이미지 처리 분야: Metric Learning
    - 자연어 처리 분야: Word Embedding

TF-IDF (Term Frequency-Inverse Document Frequency)

- BoW는 단순히 단어별 등장 횟수를 기록했습니다.
  - Bow는 문서별로 더 중요한 단어에 대한 정보를 알 수 없습니다.
- TF-IDF는 문서에서 **특정 단어의 중요도**를 고려해야 할 때 사용될 수 있습니다.

TF-IDF (Term Frequency-Inverse Document Frequency)

- 핵심 아이디어
  - "전체 문서에서는 적게 등장하고, 해당 문서에서만 자주 나오는 단어가 해당 문서에서 중요하다."
- Term Frequency
  - $TF(d,t)$ = 특정 문서 d에서의 특정 단어 t의 등장 횟수
- Document Frequency
  - $DF(t)$ = 특정 단어 t가 등장한 문서의 수
- Inverse Document Frequency
  - $IDF(t)$ = $\log$(전체 문서의 수/$DF(t)$ + 1)
- TF-IDF
  - $TF{-}IDF(d,t)$ = $TF(d,t)$ * $IDF(t)$

TF-IDF (Term Frequency-Inverse Document Frequency)

- 문서 1: “자연어 좋아요 자연어 처리 배울래요”
- 문서 2: “이미지 처리 좋아요 나머진 흥미 없어요”
- 문서 3: “저는 쉴래요”

사전(vocabulary)

나머진	배울래요	쉴래요	없어요	이미지	자연어	저는	좋아요	처리	흥미

TF-IDF (Term Frequency-Inverse Document Frequency)

- 문서 1: “자연어 좋아요 자연어 처리 배울래요”
- 문서 2: “이미지 처리 좋아요 나머진 흥미 없어요”
- 문서 3: “저는 쉴래요”
- $\boldsymbol{TF(d,t)}$ = 특정 문서 d에서의 특정 단어 t의 등장 횟수

TF 테이블

	나머진	배울래요	쉴래요	없어요	이미지	자연어	저는	좋아요	처리	흥미
문서 1	0	1	0	0	0	2	0	1	1	0
문서 2	1	0	0	1	1	0	0	1	1	1
문서 3	0	0	1	0	0	0	1	0	0	0

TF-IDF (Term Frequency-Inverse Document Frequency)

- 문서 1: “자연어 좋아요 자연어 처리 배울래요”
- 문서 2: “이미지 처리 좋아요 나머진 흥미 없어요”
- 문서 3: “저는 쉴래요”
- $\boldsymbol{IDF(t)} = \log(\text{전체 문서의 수}/DF(t) + 1)$

IDF 테이블

	나머진	배울래요	쉴래요	없어요	이미지	자연어	저는	좋아요	처리	흥미
IDF	$\log(3/2)$	$\log(3/2)$	$\log(3/2)$	$\log(3/2)$	$\log(3/2)$	$\log(3/2)$	$\log(3/2)$	0	0	$\log(3/2)$

TF-IDF (Term Frequency-Inverse Document Frequency)

- 문서 1: “자연어 좋아요 자연어 처리 배울래요”
- 문서 2: “이미지 처리 좋아요 나머진 흥미 없어요”
- 문서 3: “저는 쉴래요”
- $\boldsymbol{TF-IDF(d,t)} = TF(d,t) * IDF(t)$

TF-IDF 테이블

	나머진	배울래요	쉴래요	없어요	이미지	자연어	저는	좋아요	처리	흥미
문서 1	0	0.405	0	0	0	0.810	0	0	0	0
문서 2	0.405	0	0	0.405	0.405	0	0	0	0	0.405
문서 3	0	0	0.405	0	0	0	0.405	0	0	0

TF-IDF (Term Frequency-Inverse Document Frequency)

- TF-IDF도 여전히 카운트(Count)에 기반한 방법이며, 희소 문제가 존재합니다.
- Bag of Words에 비해서 항상 성능이 좋은 것은 아닙니다.

언어 모델 (Language Model)

조건부 확률 (Conditional Probability)

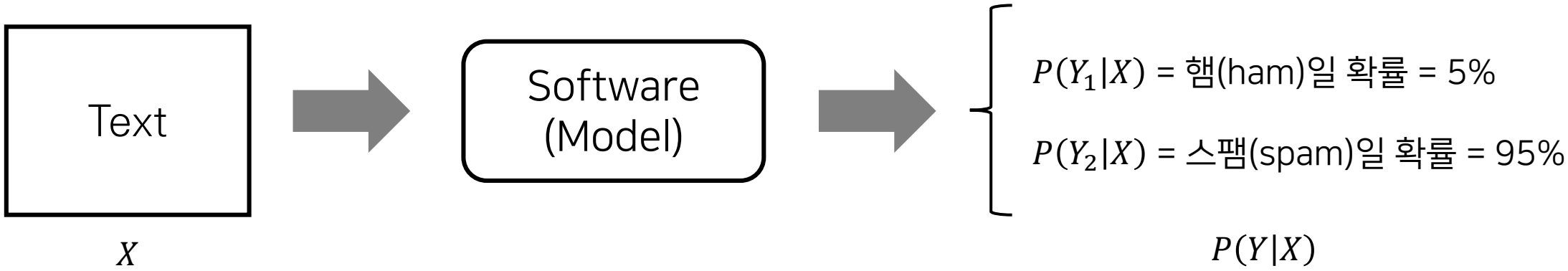
- 특정한 사건이 발생하는 경우에 다른 사건이 발생할 확률

	스팸 메일	일반 메일	합계
학교 계정	40	30	70
회사 계정	50	60	110
합계	90	90	180

- 하나의 메일을 뽑았을 때, 학교 계정으로 온 메일일 확률: $P(\text{학교}) = 70/180$
- 하나의 메일을 뽑았을 때, 학교 계정으로 온 메일이면서 스팸 메일일 확률: $P(\text{학교} \cap \text{스팸}) = 40/180$
- **스팸 메일 중 하나를 뽑았을 때**, 학교 계정으로 온 메일을 확률: $\boldsymbol{P}(\text{학교}|\text{스팸}) = 40/90$

베이즈 정리 (Bayes' Theorem)

- 하나의 소프트웨어를 만들고 싶다고 가정합시다.
  - 입력: 하나의 텍스트
  - 출력: 텍스트가 특정 클래스에 속할 확률
    - 예) 이 텍스트가 스팸(Spam)일 확률
- 이때 텍스트를 X, 클래스를 Y라고 합시다.
  - 클래스는 오직 두 개만 존재한다고 가정합시다. (Y_1 =햄, Y_2 = 스팸)



베이즈 정리 (Bayes' Theorem)

- $P(Y|X)$를 계산할 수 있으면, 우리가 원하는 프로그램을 만들 수 있습니다.
  - 매 텍스트마다 $P(Y = \text{햄}|X)$와 $P(Y = \text{스팸}|X)$의 값을 비교하면 됩니다.
- 하지만 $P(Y|X)$를 바로 계산하는 것은 어려운 경우가 많습니다.
- 반면에 우리가 충분히 데이터를 수집했다면 $P(X|Y)$는 구할 수 있는 경우가 많습니다.
- 만약, 스팸 메일의 95%에 광고성 단어가 존재한다고 가정합시다.
  - $P(\text{광고성 단어}|Y = \text{스팸}) = 95\%$
- 그렇다면 $P(X|Y)$를 이용해서 $P(Y|X)$와 유사한 값을 얻을 수 있을까요?
  - 바로 베이즈 정리를 이용하면 됩니다.

베이즈 정리 (Bayes' Theorem)

- $P(Y|X)$를 계산할 수 있으면, 우리가 원하는 프로그램을 만들 수 있습니다.
  - 매 텍스트마다 $P(Y = \text{햄}|X)$와 $P(Y = \text{스팸}|X)$의 값을 비교하면 됩니다.
- 하지만 $P(Y|X)$를 바로 계산하는 것은 어려우므로, 베이즈 정리에 기반한 방법을 이용합니다.
- 베이즈 정리 공식은 다음과 같습니다.

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

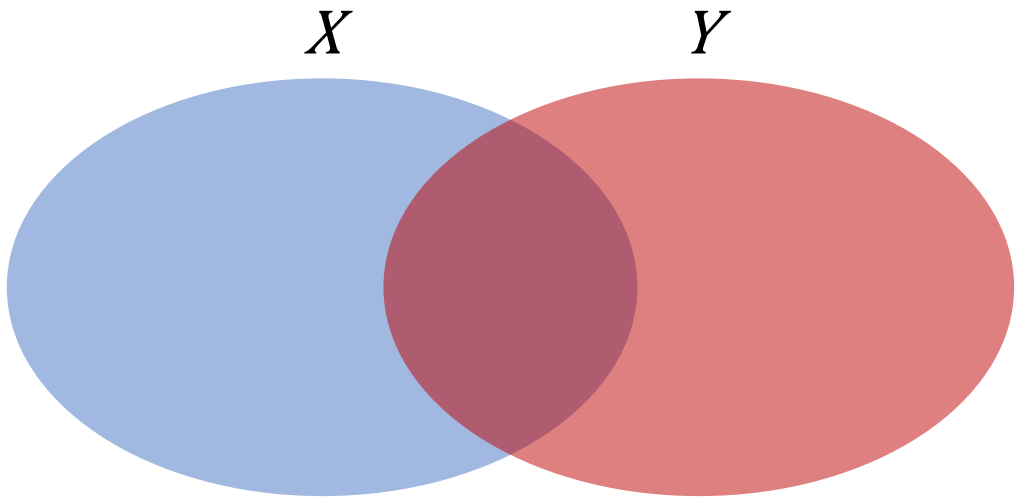
$$posterior \propto likelihood \times prior$$

- P(X): 특정 텍스트가 나올 확률
- P(Y): 특정 클래스가 나올 확률
- P(X|Y): 특정 클래스에서 특정 텍스트가 나올 확률
- P(Y|X): 특정 텍스트가 특정 클래스에서 나올 확률

베이즈 정리 (Bayes' Theorem) 공식 유도

- 기본 공식 1) $P(X,Y) = P(X|Y)P(Y)$
- 기본 공식 2) $P(X,Y) = P(Y,X)$
- **베이즈 정리 유도**
  - $P(X,Y) = P(Y,X)$
  - $P(X|Y)P(Y) = P(Y|X)P(X)$

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

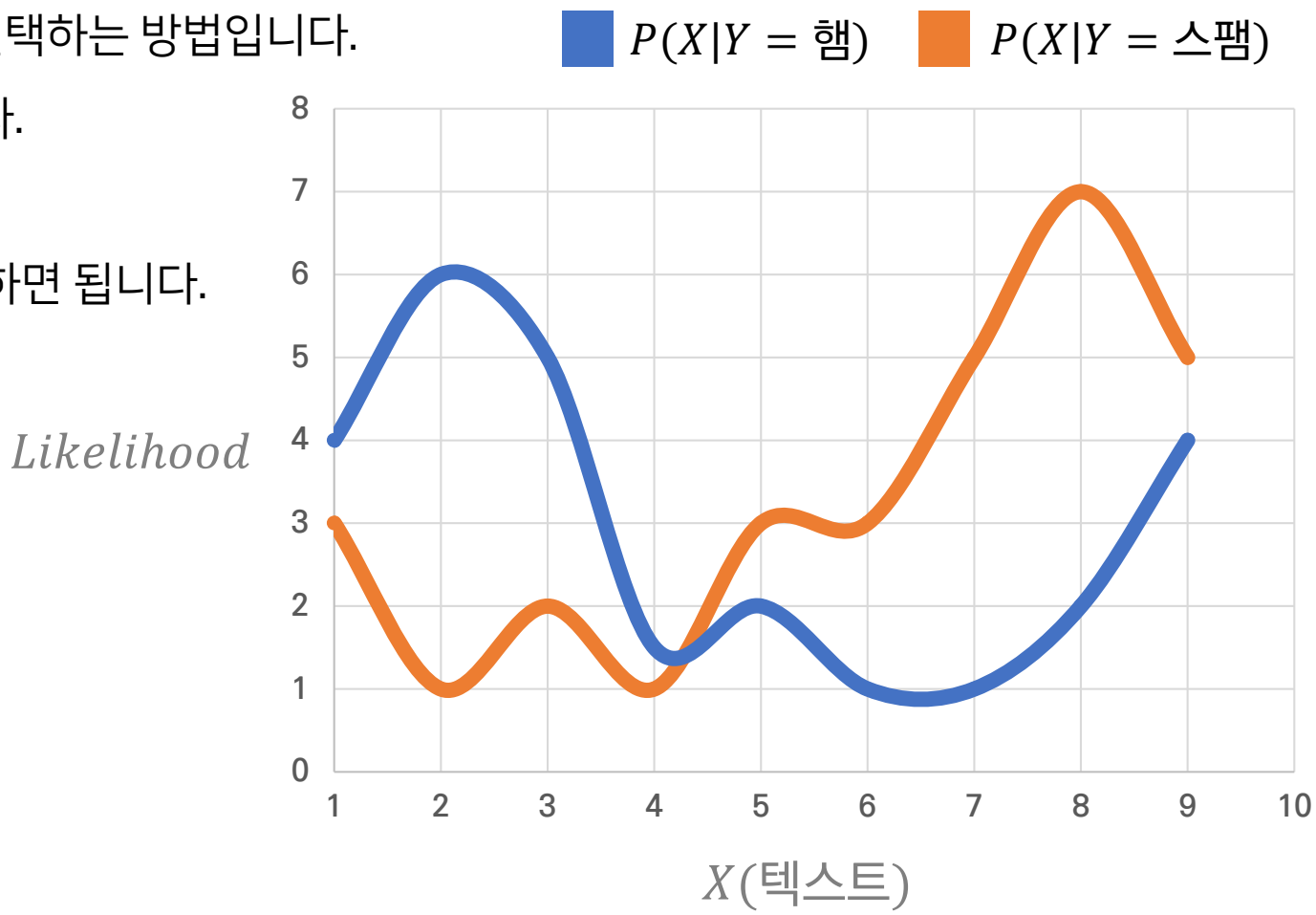


알아 두면 좋은 규칙들 (Rules)

- 황금 법칙 (Golden Rule)
  - 텍스트 B가 주어졌을 때, 이 텍스트는 어떤 클래스 A로 분류될까요?
  - $argmax_A P(A|B) = argmax_A P(B|A)P(A)/P(B) = argmax_A P(B|A)P(A)$
- 연쇄 법칙 (Chain Rule)
  - $P(A_1, A_2, A_3, \dots, A_n) = P(A_1|A_2, A_3, \dots, A_n) * P(A_2|A_3, A_4, \dots, A_n), \dots, P(A_{n-1}|A_n) * P(A_n)$

최대 우도 추정 (Maximum Likelihood Estimation)

- 우도(Likelihood)가 가장 높은 클래스를 선택하는 방법입니다.
- X는 특징(Feature) 혹은 데이터를 말합니다.
- **X = 광고성 단어의 개수**라고 해봅시다.
- 현재 예시에선 $X \geq 5$라면 스팸으로 분류하면 됩니다.



최대 우도 추정 시 유의할 점

- 우리는 사후확률을 계산하기 어렵기 때문에 가능도를 이용합니다.
- 하지만 가능도만으로 사후확률을 완전히 근사할 수 없습니다.

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

$$posterior \propto likelihood \times prior$$

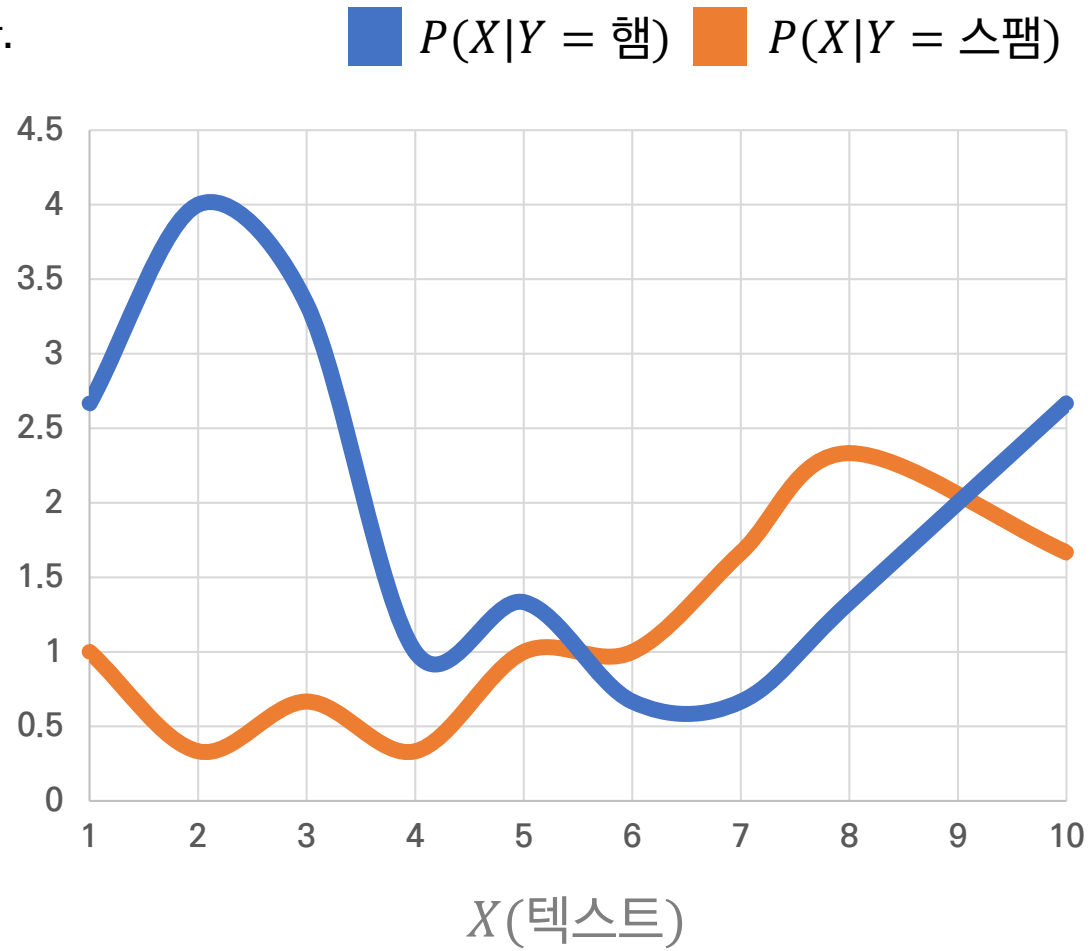
- 만약 $prior$가 $Uniform\ Distribution$을 따르지 않는다면 어떻게 될까요?
  - 전체 메일 중에서 스팸 메일의 수 자체가 적다고 해봅시다.
  - $P(Y = \text{스팸}) = 1/3,\ P(Y = \text{햄}) = 2/3$
  - 분류 기능은 어떻게 바뀔까요?

최대 우도 추정 시 유의할 점

- 전체 메일 중에서 스팸 메일의 수 자체가 적다고 해봅시다.
- $P(Y = \text{스팸}) = 1/3,\ P(Y = \text{햄}) = 2/3$
- 이를 반영하면, 이제 더욱 정교한 분류가 가능합니다.

$$posterior \propto likelihood \times prior$$

$Posterior$



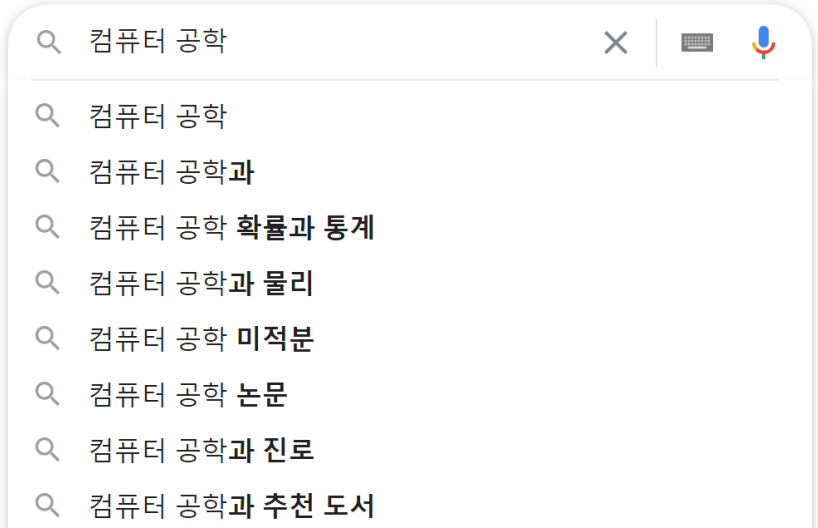
➡ 이제는 $X \geq 6$이라면 스팸으로 분류하면 됩니다.

베이즈 정리 (Bayes' Theorem)

- 정리하면, $prior$를 고려할 때 $posterior$를 더욱 잘 계산할 수 있습니다.
- 베이즈 정리는 머신러닝 분야에서 끊임없이 등장하는 개념이므로 중요합니다.
  - **나이브 베이즈 분류기** (Naïve Bayes Classifier)에서 사용됩니다.
  - 최신 인공지능 기술보다 성능은 많이 뒤떨어지지만 기본적인 모델로 자주 언급됩니다.

언어 모델 (Language Model)

- **언어 모델**이란 문장(시퀀스)에 확률을 부여하는 모델을 의미합니다.
- 언어 모델을 가지고 있으면 특정한 상황에서의 적절한 문장이나 단어를 예측할 수 있습니다.
  - **기계 번역 예시**
    - $P(\text{난 널 사랑해}|I\ love\ you) > P(\text{난 널 싫어해}|I\ love\ you)$
  - **다음 단어 예측 예시**
    - $P(\text{먹었다}|\text{나는 밥을}) > P(\text{싸웠다}|\text{나는 밥을})$



언어 모델 (Language Model)

- 하나의 **문장**은(W)은 여러 개의 **단어**(w)로 구성됩니다.
  - $P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots\ w_n)$
  - $P(\text{친구와 친하게 지낸다}) = P(\text{친구와}, \text{친하게}, \text{지낸다})$
- 연쇄 법칙 (Chain Rule)
  - $P(w_1, w_2, w_3, \dots, w_n) = P(w_1) * P(w_2|w_1) * P(w_3|w_1, w_2), \dots, P(w_n|w_1, w_2, \dots, w_{n-1})$

$$= \prod_{i=1}^{n} P(w_i|w_1, \dots, w_{i-1})$$

  - $P(\text{친구와 친하게 지낸다}) = P(\text{친구와}) * P(\text{친하게}|\text{친구와}) * P(\text{지낸다}|\text{친구와 친하게})$

언어 모델의 희소 문제 (Sparsity Problem)

- **전통적인 통계적 언어 모델**은 카운트 기반의 접근을 사용합니다.
  - $P(\text{지낸다}|\text{친구와 친하게}) = \frac{count(\text{친구와 친하게 지낸다})}{count(\text{친구와 친하게})}$
- 현실 세계에서 모든 문장에 대한 확률을 가지고 있으려면 매우 방대한 양의 데이터가 필요합니다.
- 예를 들어 "친구와 친하게"라는 시퀀스 자체가 학습 데이터에 존재하지 않으면 어떻게 될까요?
- 긴 문장은 처리하기가 매우 어렵습니다.
  - $P(\text{나는 공부를 마치고 집에서 밥을 먹었다}) = P(\text{나는}) * P(\text{공부를}|\text{나는}) * P(\text{마치고}|\text{나는 공부를}) *$
    $P(\text{집에서}|\text{나는 공부를 마치고}) * P(\text{밥을}|\text{나는 공부를 마치고 집에서}) * P(\text{먹었다}|\text{나는 공부를 마치고 집에서 밥을})$
- 현실적인 해결책으로 **N-gram 언어 모델**이 사용됩니다.
  - 인접한 일부 단어만 고려하는 아이디어입니다.

N-gram 언어 모델

- 전통적인 언어 모델은 등장한 '모든 단어'를 고려해야 합니다.
- 모든 단어를 고려하지는 않고, 일부 단어만 고려하는 접근이 N-gram입니다.

$$P(W) = \prod_{i=1}^{n} P(w_i | w_{i-n+1}, w_{i-n+2} \cdots, w_{i-1})$$

  - 앞의 두 글자만 보면 어떨까요?
  - P(먹었다|나는 공부를 마치고 집에서 밥을) $=$ P(먹었다|집에서 밥을)

N-gram 언어 모델

- N-gram: 길이가 N인 단어 시퀀스
- 문장: “나는 공부를 마치고 집에서 밥을 먹었다”
  - unigrams: 나는, 공부를, 마치고, 집에서, 밥을, 먹었다
  - bigrams: 나는 공부를, 공부를 마치고, 마치고 집에서, 집에서 밥을, 밥을 먹었다
  - trigrams: 나는 공부를 마치고, 공부를 마치고 집에서, 마치고 집에서 밥을, 집에서 밥을 먹었다

N-gram 언어 모델

- 기본적인 언어 모델은 등장한 '모든 단어'를 고려해야 합니다.
- 모든 단어를 고려하지는 말고, 일부 단어만 고려하는 접근이 N-gram입니다.
- trigrams 예시
  - P(먹었다|나는 공부를 마치고 집에서 밥을) $=$ P(먹었다|집에서 밥을)

    $=$ $count$(집에서 밥을 먹었다) / $count$(집에서 밥을)
- 만약에 전체 문서 데이터베이스가 "나는 집에서 밥을 했다, 그리고 나는 집에서 밥을 먹었다"라면,
  - "집에서 밥을"은 2번 등장했고
  - "집에서 밥을 했다"는 1번, "집에서 밥을 먹었다"는 1번 등장했습니다.
    - 따라서 P(먹었다|나는 공부를 마치고 집에서 밥을) $=$ $1/2$

N-gram 언어 모델에서의 유의사항

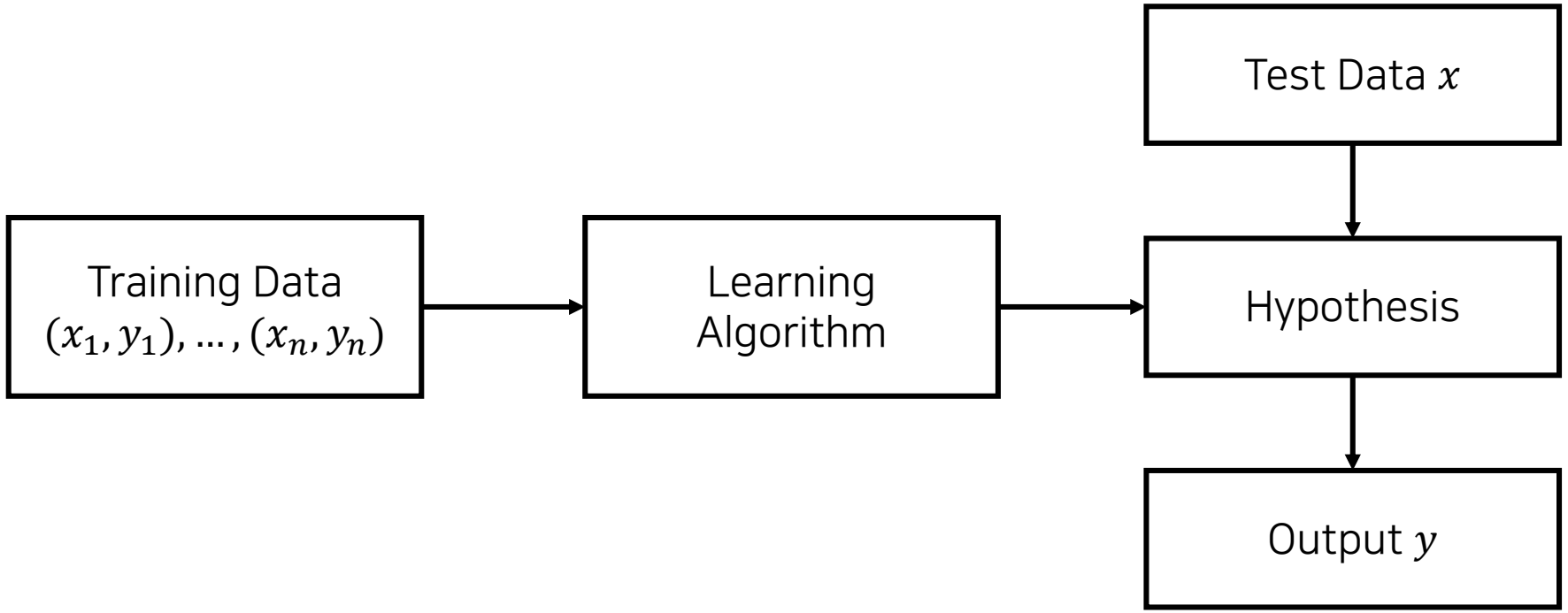
- 일반적으로 n을 어느 정도 크기로 설정할까요?
- 이론적으로 n이 클수록 성능은 개선됩니다.
  - 하지만 n이 커질수록 필요한 파라미터의 수가 기하급수적으로 많아집니다.
- 예를 들어 $|V| = 60{,}000$일 때, 파라미터의 수는 다음과 같습니다.
  - $1-gram\ LM\colon\ 6 \times 10^{4}\ parameters$
  - $2-gram\ LM\colon\ 3.6 \times 10^{9}\ parameters$
  - $3-gram\ LM\colon 2.16 \times 10^{14}\ parameters$
  - $4-gram\ LM\colon 12.96 \times 10^{19}\ parameters$
- 일반적으로 $n = 3$ 정도로 설정합니다.

N-gram 언어 모델의 한계점

- N-gram은 고전적인 언어 모델입니다.
  - 일반적으로 특정한 시퀀스가 등장한 카운트를 계산합니다.
- 희소 문제가 존재하며 성능이 좋지 못해 최근에는 인공 신경망을 활용한 접근이 많이 이용되고 있습니다.

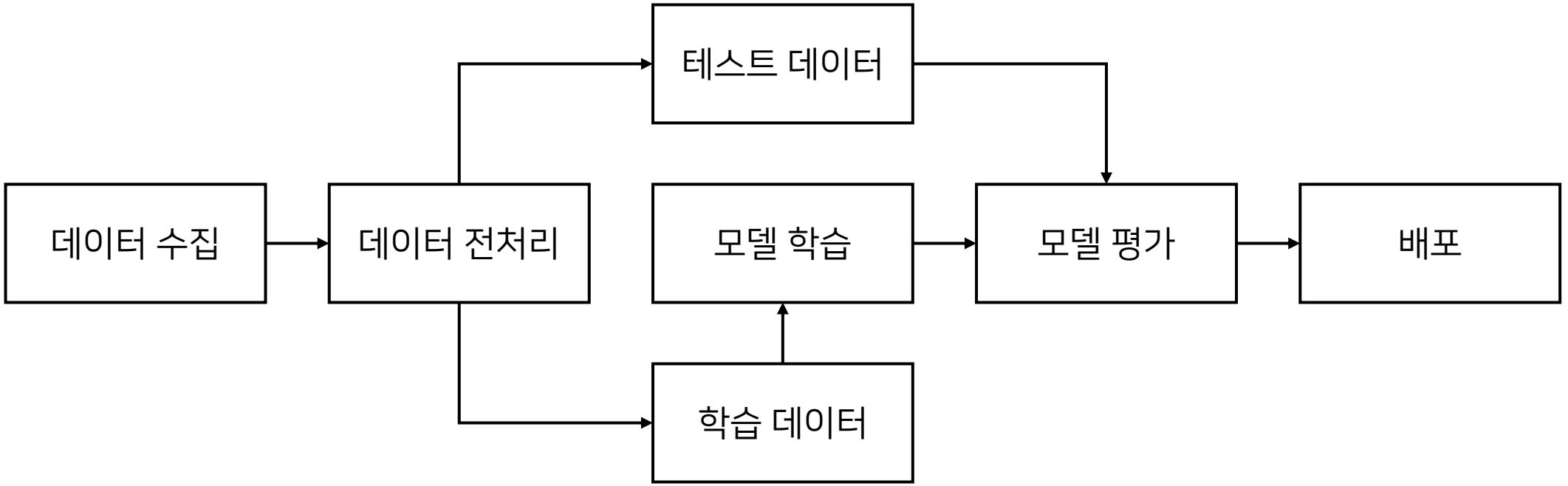
기계 학습 (Machine Learning)

- 학습을 통해 자동으로 기능을 개선하는 컴퓨터 알고리즘에 대한 분야입니다.



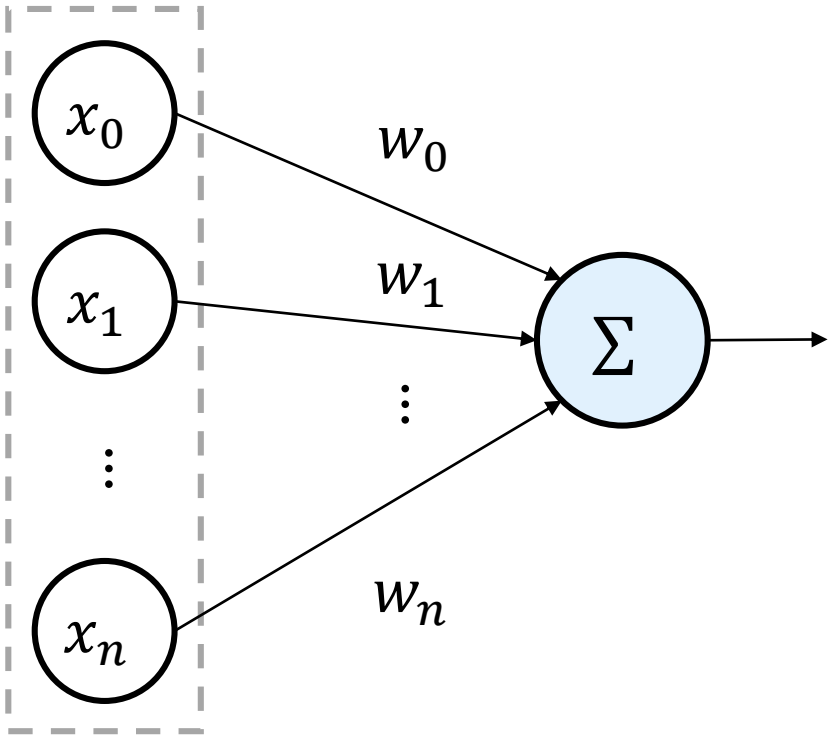
머신 러닝의 수행 단계

- 머신 러닝 분야의 작업 과정을 단계별로 제시하면 다음과 같습니다.

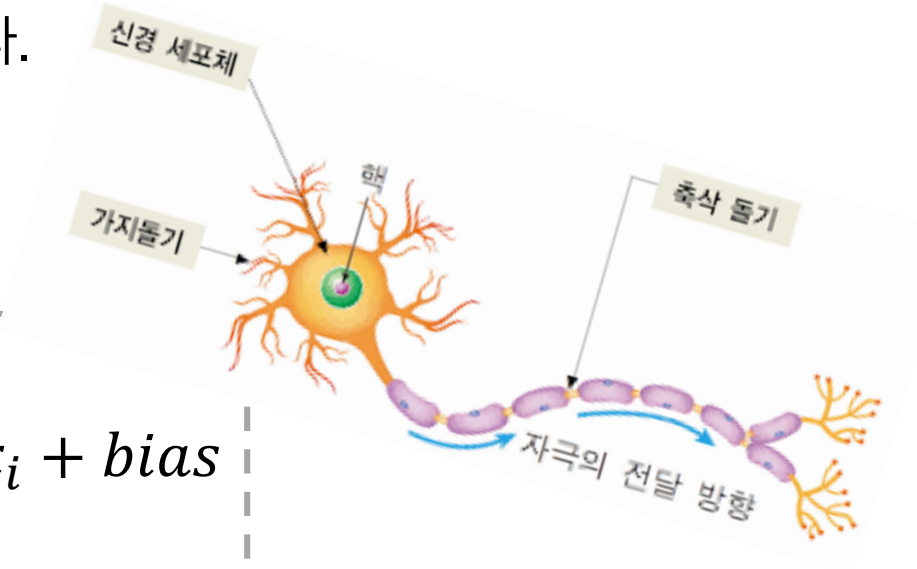


퍼셉트론 (Perceptron)

- 퍼셉트론은 다수의 입력(input)을 받아서 하나의 출력(output)을 내보냅니다.
- 기계 학습은 모델의 가중치(weight)를 학습하는 과정을 말합니다.



$$Output = \sum_{i=0}^{n} w_i \, x_i + bias$$



희소 표현과 밀집 표현

- 희소 표현 (Sparse Representation)
  - 원-핫 인코딩은 단어 집합의 길이가 벡터의 차원이 되며 대부분의 값이 0이 됩니다.
  - 이는 메모리 낭비를 불러일으킵니다.
  - “컴퓨터” →

1	0	0	0	0	0	0	0

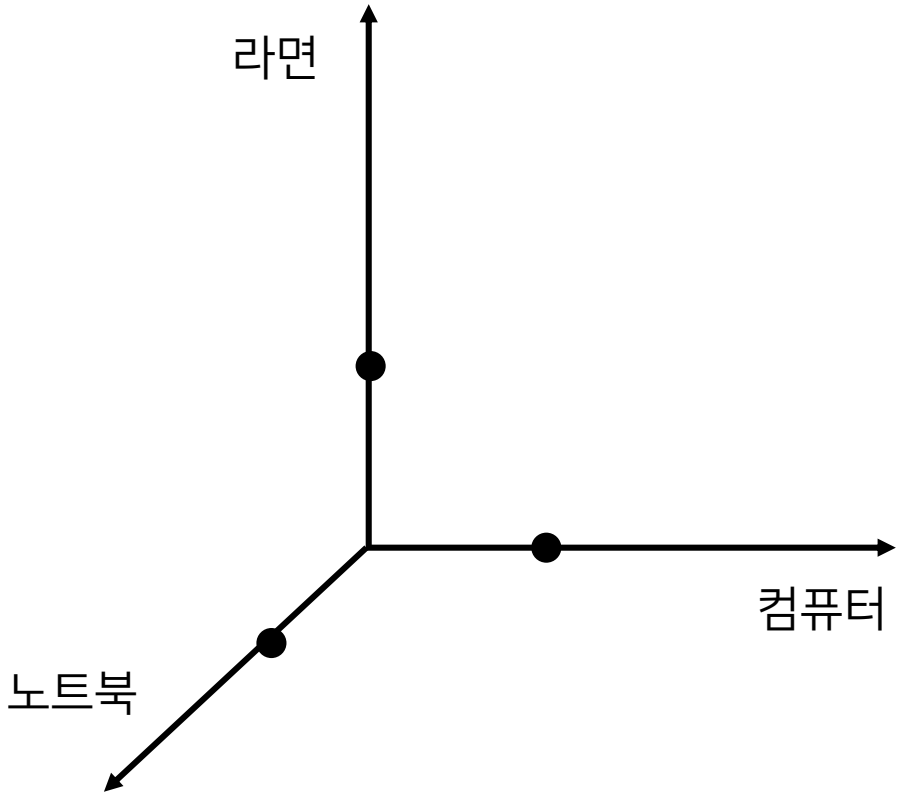
- 밀집 표현 (Dense Representation)
  - 사용자가 설정한 크기로 벡터 표현의 차원을 맞춥니다.
  - “컴퓨터 좋아요, 좋아요.” →

0.8	2.3	1.5

	표현 종류	차원의 크기	값의 범위	표현 과정
원-핫 벡터	희소 표현	고차원	0 혹은 1	수동적으로 설정
임베딩 벡터	밀집 표현	저차원	실수형	훈련 데이터로부터 학습

유사도 측정을 위한 단어 표현

- 희소 표현 방법을 이용하면 단어간 유사도를 측정하기 어렵습니다.

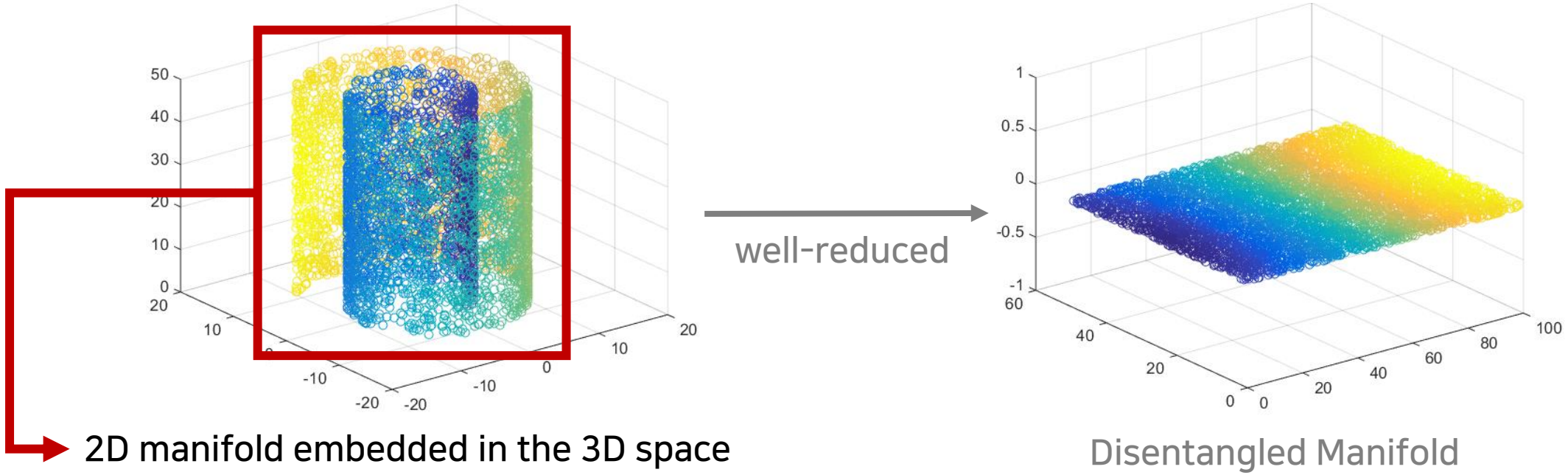


	컴퓨터	노트북	라면
인코딩	[1, 0, 0]	[0, 1, 0]	[0, 0, 1]

- 모든 벡터들이 서로 수직 관계입니다.
- 코사인 유사도 (Cosine Similarity) = 0

차원 축소 (Dimension Reduction)

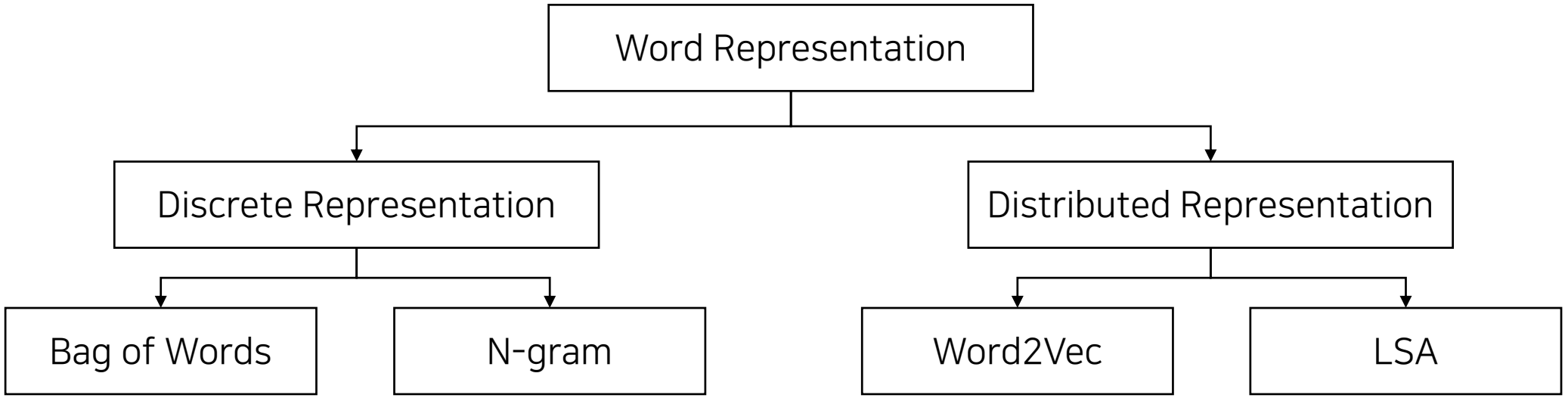
- 차원 축소는 높은 차원상의 데이터를 낮은 차원상의 데이터로 차원을 줄이는 작업입니다.
  - 예시: 데이터 시각화, 데이터 압축을 통한 복잡도 개선



기계 학습과 단어 임베딩

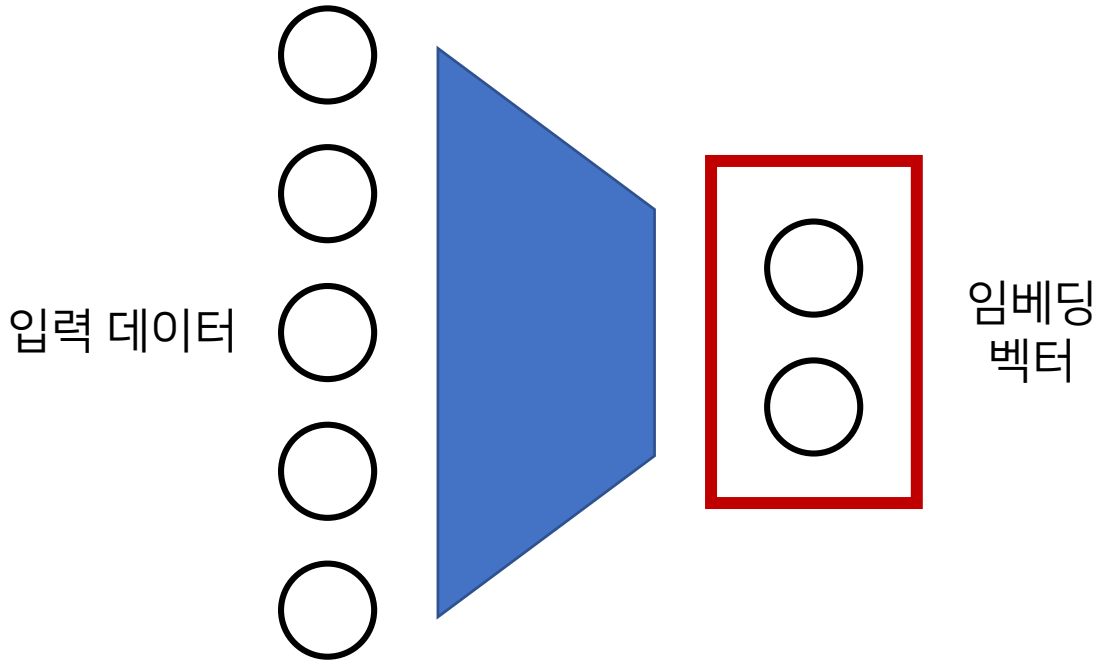
단어 표현의 분류

- 이산 표현 (Discrete Representation): 해당 단어에 직접 특정한 값을 매핑하여 표현합니다.
- 분산 표현 (Distributed Representation): 해당 단어를 표현하기 위해 주변 단어를 참고하여 표현합니다.
  - 예로 '강아지'와 '귀여운'이라는 단어가 함께 나타나는 경우가 많다는 점을 더욱 잘 고려할 수 있습니다.



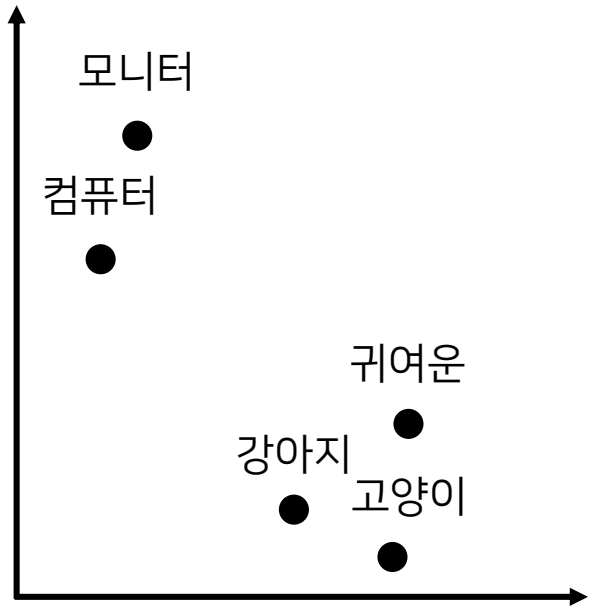
Word2Vec

- Word2Vec을 이용하면 단어 간의 유사도를 효과적으로 측정할 수 있습니다.
  - 이름에서부터 알 수 있듯이, 단어를 벡터(vector)로 만드는 방법입니다.
  - 일반적으로 특정한 데이터를 축소된 차원의 벡터(vector)로 임베딩(Embedding)합니다.



Word2Vec

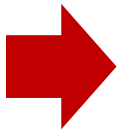
	고양이	강아지	귀여운	컴퓨터	모니터
인코딩	[1, 0, 0, 0, 0]	[0, 1, 0, 0, 0]	[0, 0, 1, 0, 0]	[0, 0, 0, 1, 0]	[0, 0, 0, 0, 1]
임베딩	[5.4, 0.3]	[4.1, 0.9]	[5.6, 1.4]	[0.7, 4.5]	[1.6, 6.1]



Word2Vec 데이터 생성

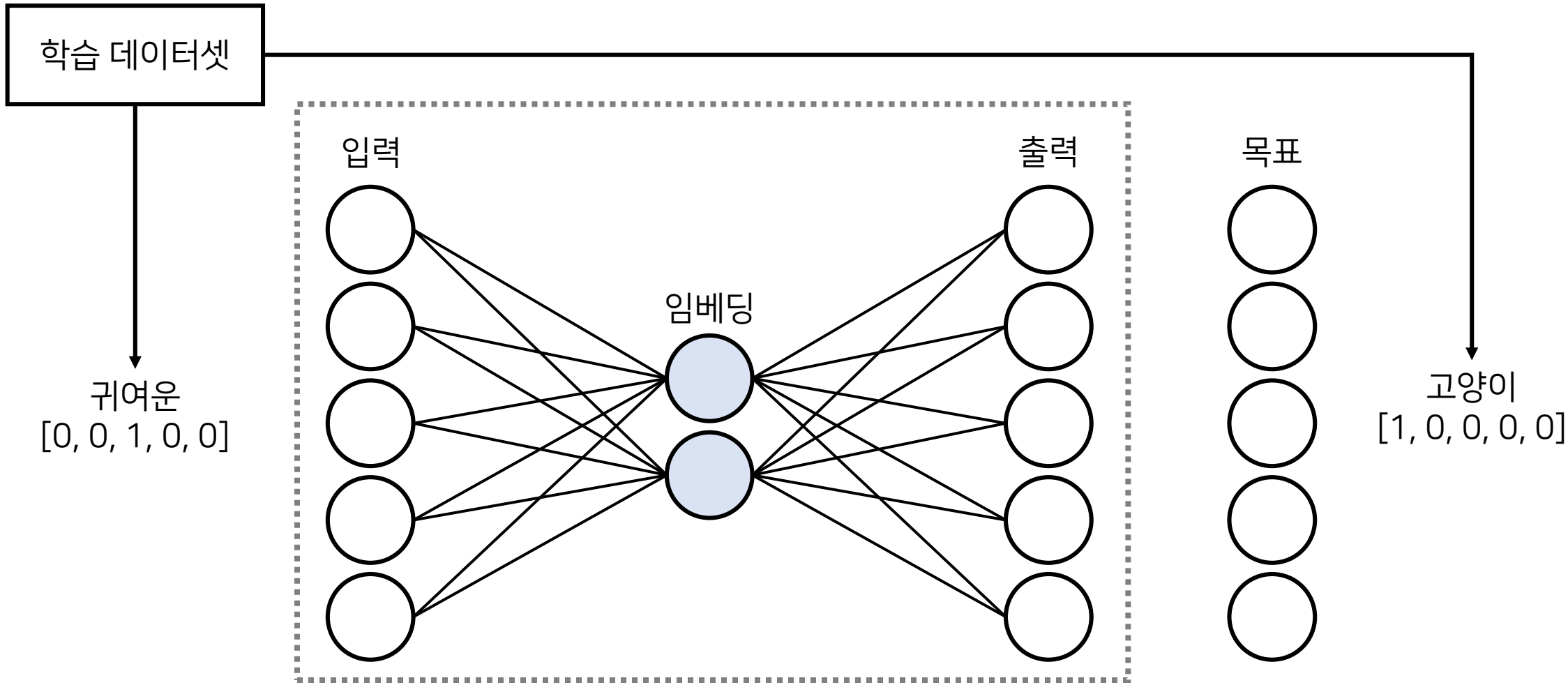
- 학습 데이터셋 (window size: 1)
  - 문장 1: “고양이, 귀여운 강아지”
  - 문장 2: “컴퓨터와 모니터”

중심 단어	주변 단어
고양이	귀여운
귀여운	고양이, 강아지
강아지	귀여운
컴퓨터	모니터
모니터	컴퓨터

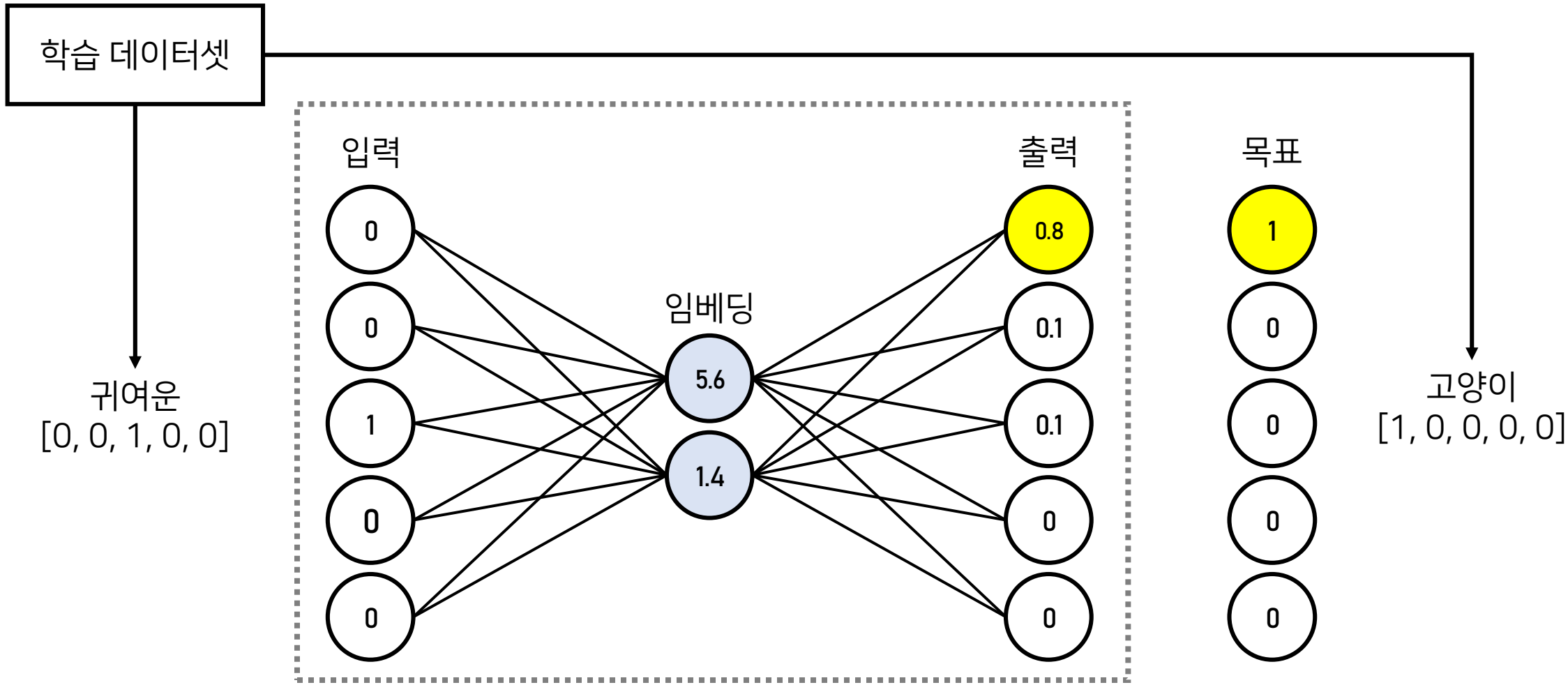


중심 단어	주변 단어
[1, 0, 0, 0, 0]	[0, 0, 1, 0, 0]
[0, 0, 1, 0, 0]	[1, 0, 0, 0, 0], [0, 1, 0, 0, 0]
[0, 1, 0, 0, 0]	[0, 0, 1, 0, 0]
[0, 0, 0, 1, 0]	[0, 0, 0, 0, 1]
[0, 0, 0, 0, 1]	[0, 0, 0, 1, 0]

Word2Vec

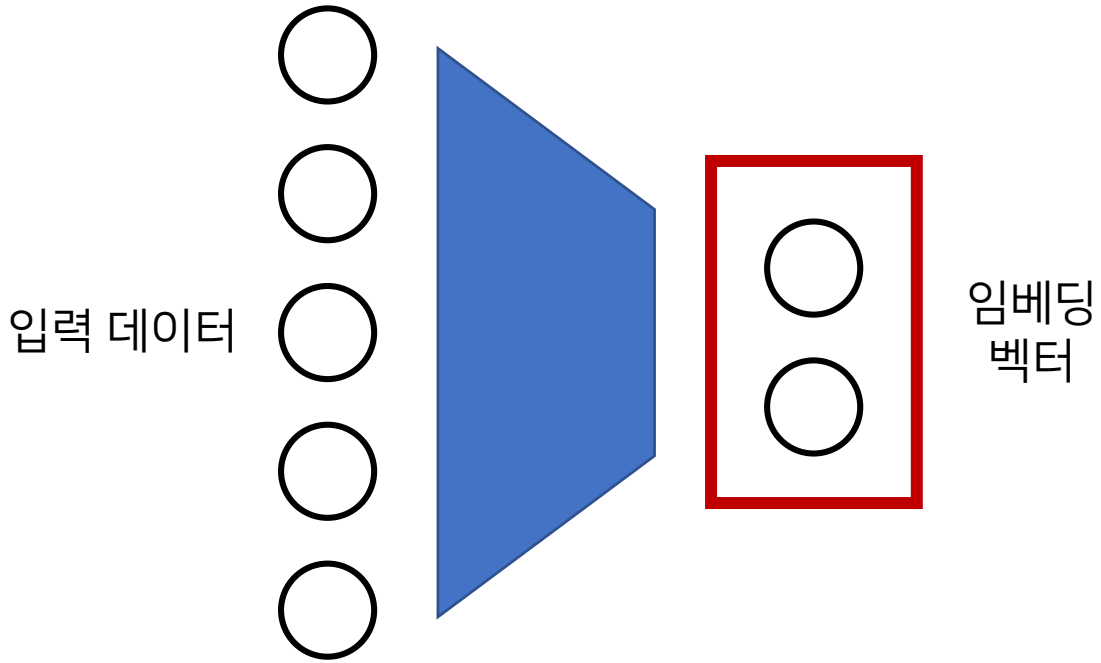


Word2Vec (학습 경과)



Word2Vec

- Word2Vec을 이용하면 단어 간의 유사도를 효과적으로 측정할 수 있습니다.
  - 이름에서부터 알 수 있듯이, 단어를 벡터(vector)로 만드는 방법입니다.
  - 일반적으로 특정한 데이터를 축소된 차원의 벡터(vector)로 임베딩(Embedding)합니다.



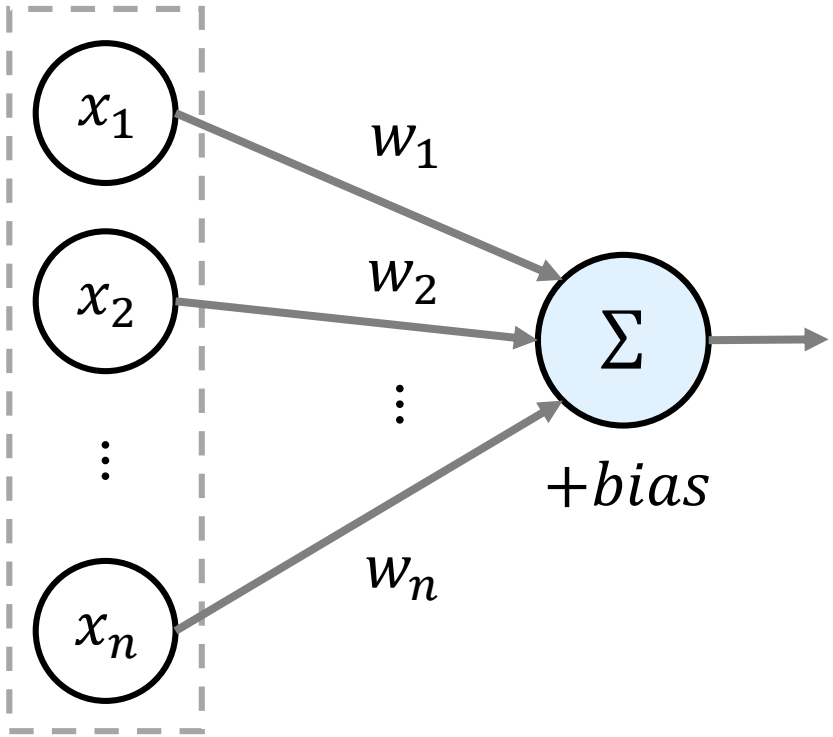
[실습] 한국어 Word2Vec

- 실습을 진행합니다.
  - Dataset: 네이버 영화 리뷰 문장 데이터셋

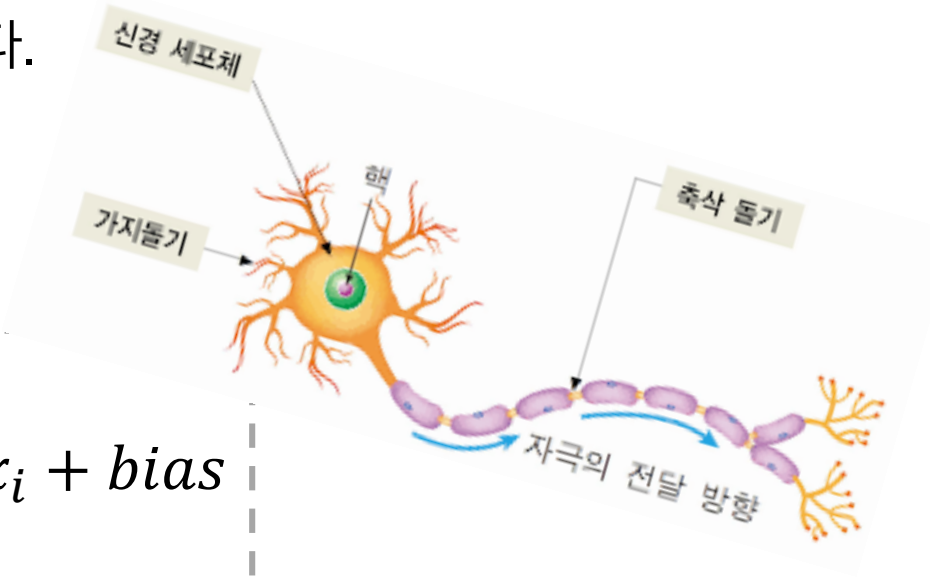
자연어 처리를 위한 딥러닝

퍼셉트론 (Perceptron)

- 퍼셉트론은 다수의 입력(input)을 받아서 하나의 출력(output)을 내보냅니다.
- 기계 학습은 모델의 가중치(weight)를 학습하는 과정을 말합니다.

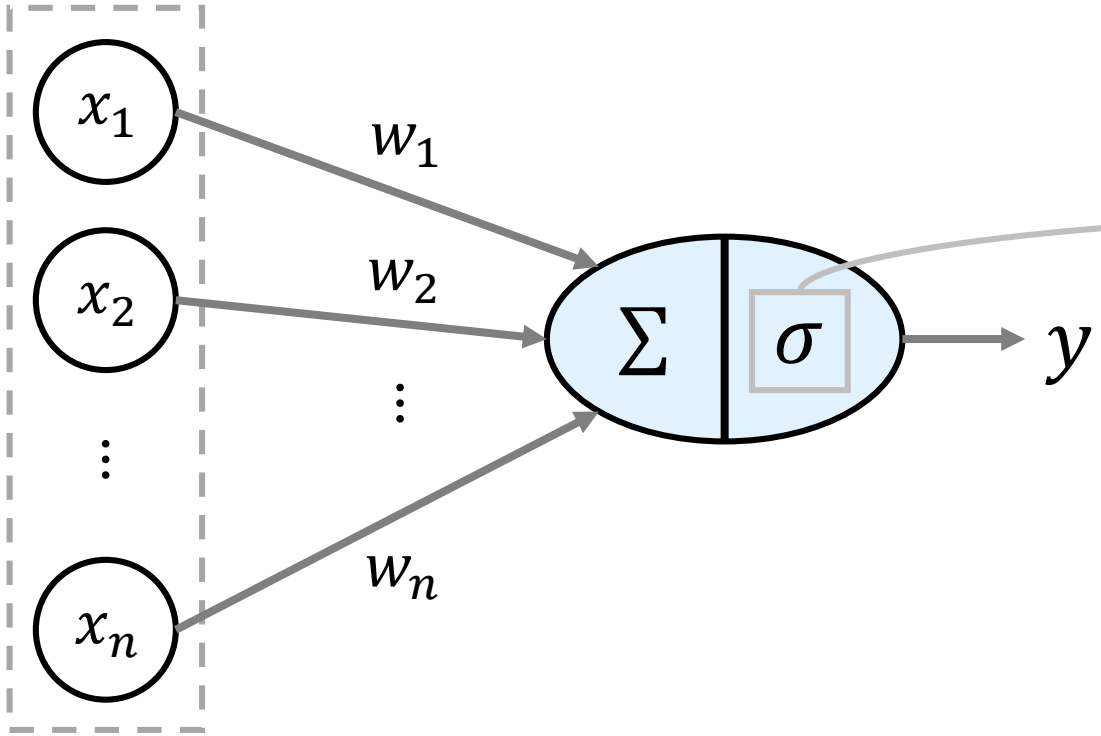


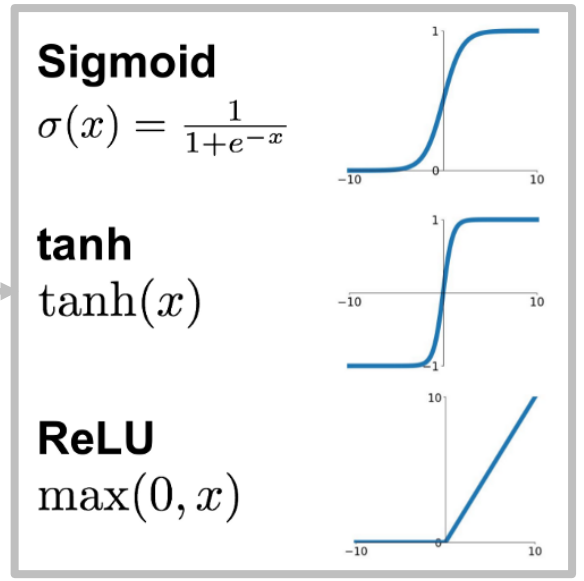
$$Output = \sum_{i=1}^{n} w_i \, x_i + bias$$



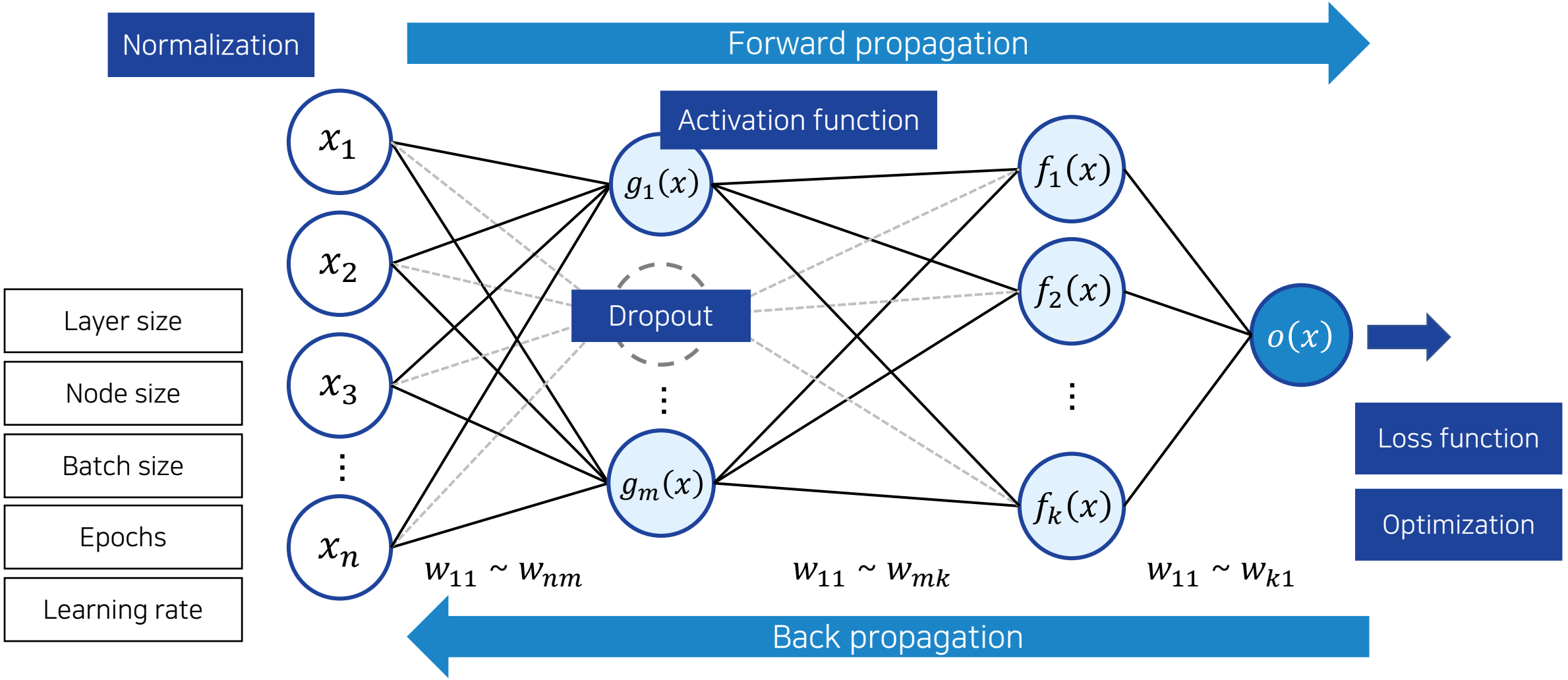
활성화 함수 (Activation Function)

- 활성화 함수는 입력 신호를 얼마나 출력할지 결정하고 비선형성을 추가해주는 역할을 수행합니다.



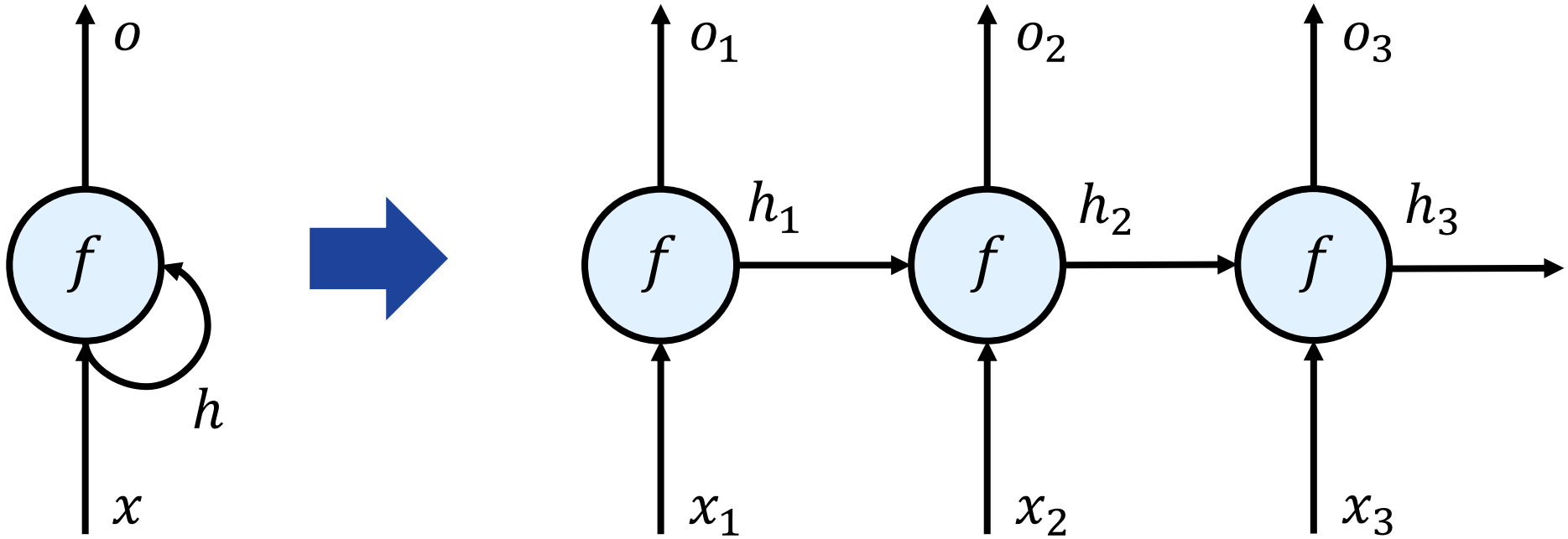


딥러닝 (Deep Learning)



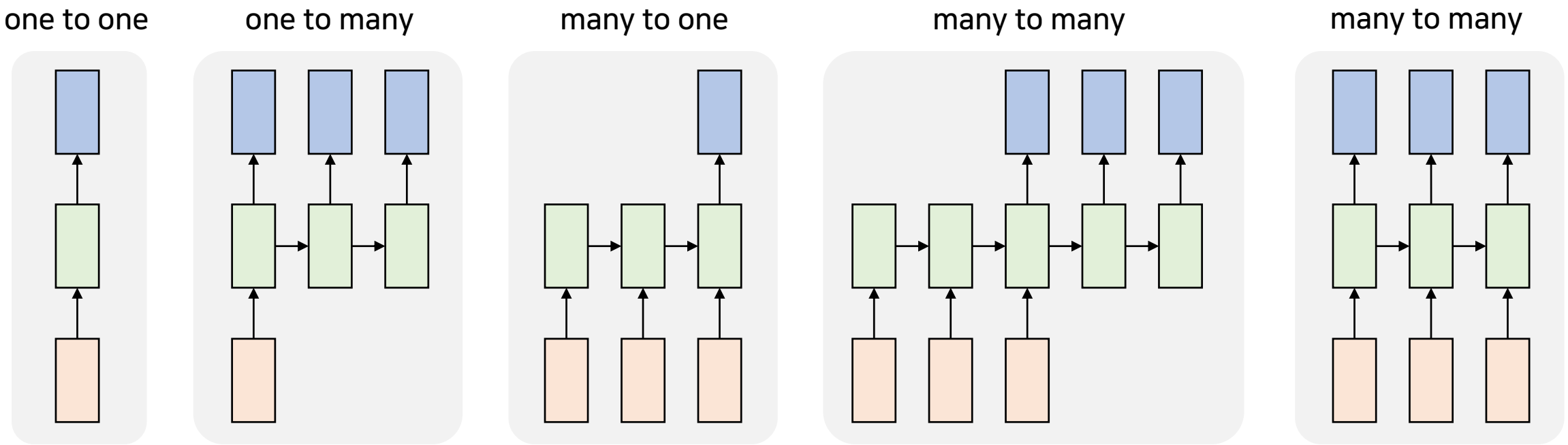
RNN (Recurrent Neural Network)

- RNN은 시퀀스의 길이와 관계없이 입력을 받아들일 수 있는 네트워크 구조를 가집니다.
  - 이전 상태를 저장하기 위해 **순환적인 구조**로 동작합니다.
  - 실제로는 하나의 RNN 파라미터를 반복적으로 호출합니다.



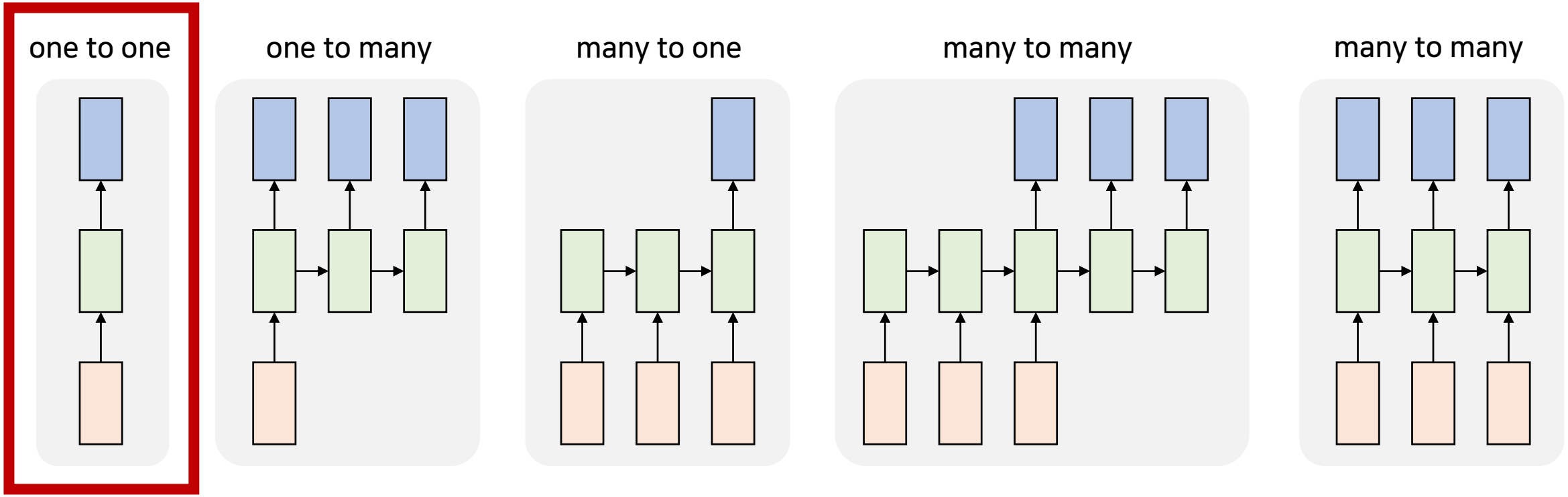
RNN의 다양한 아키텍처

- RNN은 다양한 아키텍처를 가질 수 있습니다.
  - 다음과 같이 다양한 형태의 아키텍처를 비교해 봅시다.



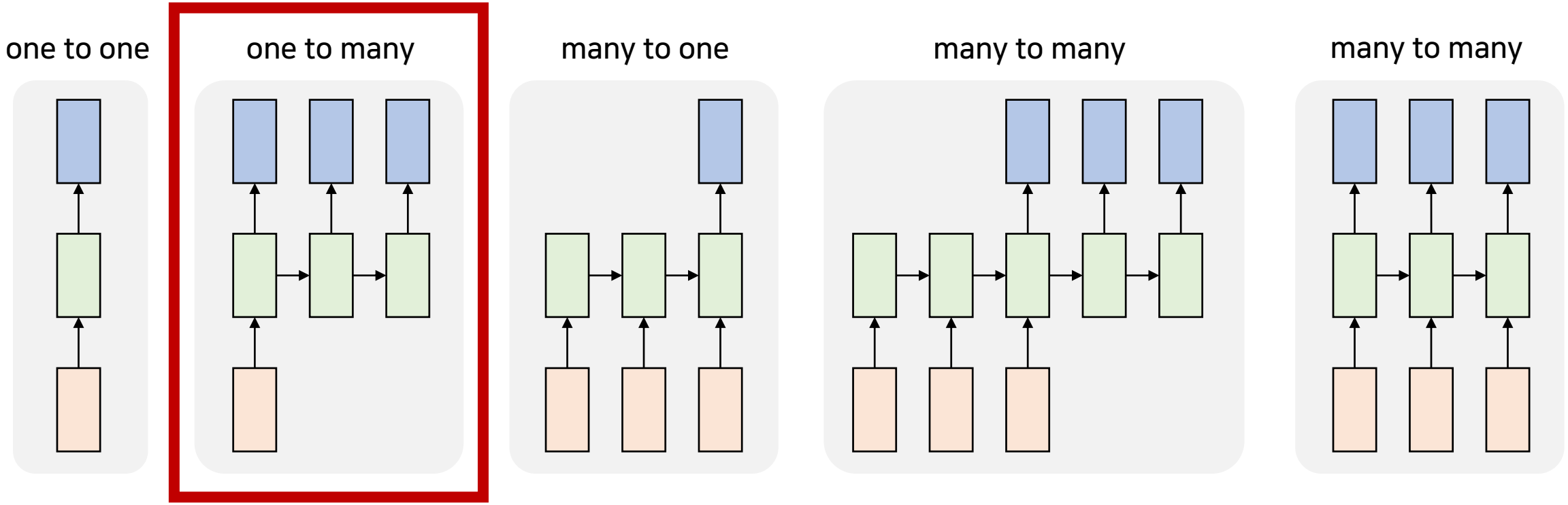
RNN의 다양한 아키텍처

- One-to-One: 하나의 입력을 받아 하나의 출력을 내보내는 네트워크
  - **대표적인 예시**) 이미지 분류(classification)



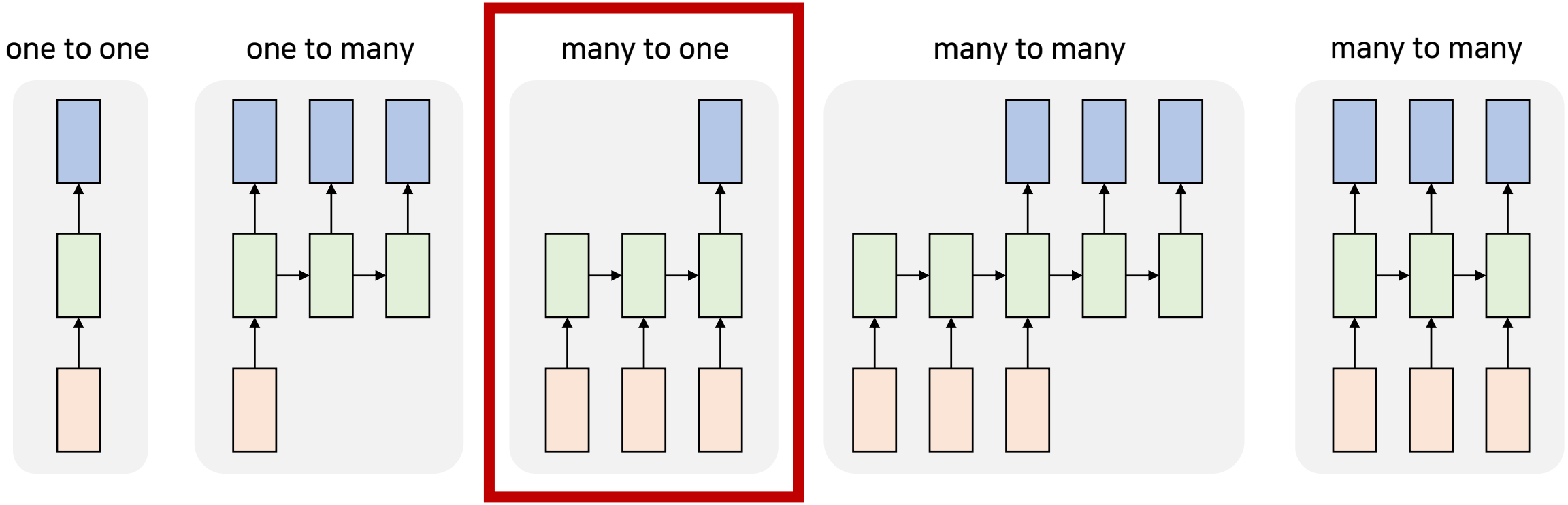
RNN의 다양한 아키텍처

- One-to-Many: 하나의 입력을 받아 다수의 출력을 내보내는 네트워크
  - **대표적인 예시**) 이미지 캡션(caption) 생성



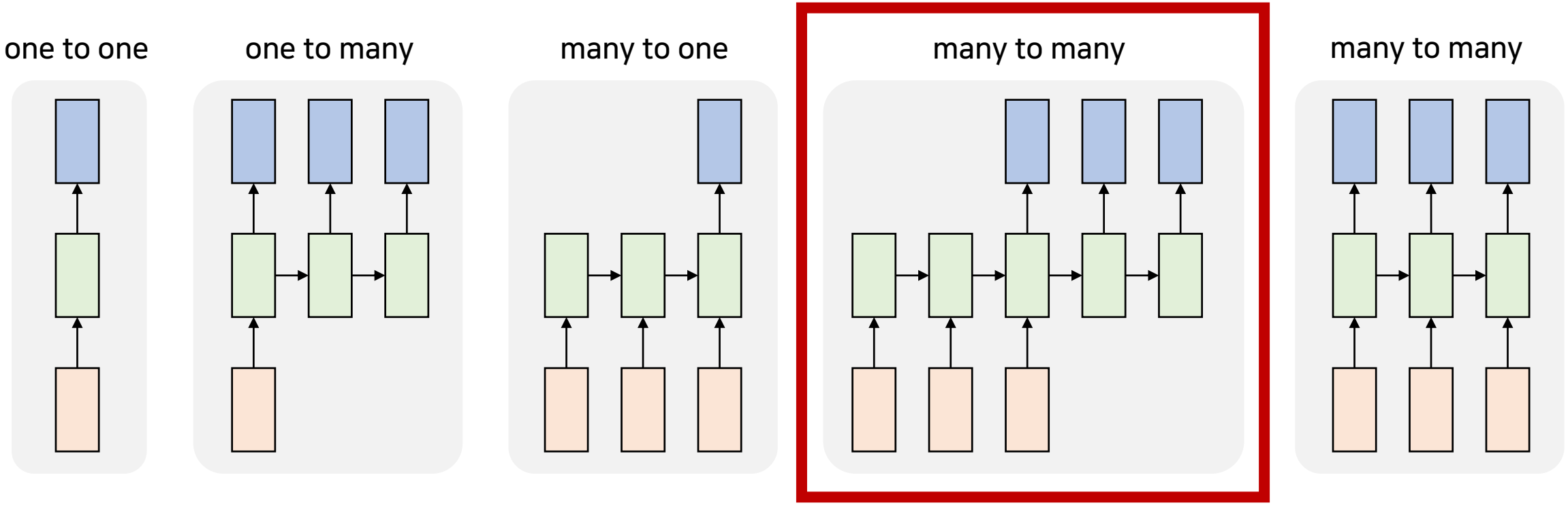
RNN의 다양한 아키텍처

- Many-to-One: 다수의 입력을 받아 하나의 출력을 내보내는 네트워크
  - **대표적인 예시**) 스팸 메시지 분류(classification)



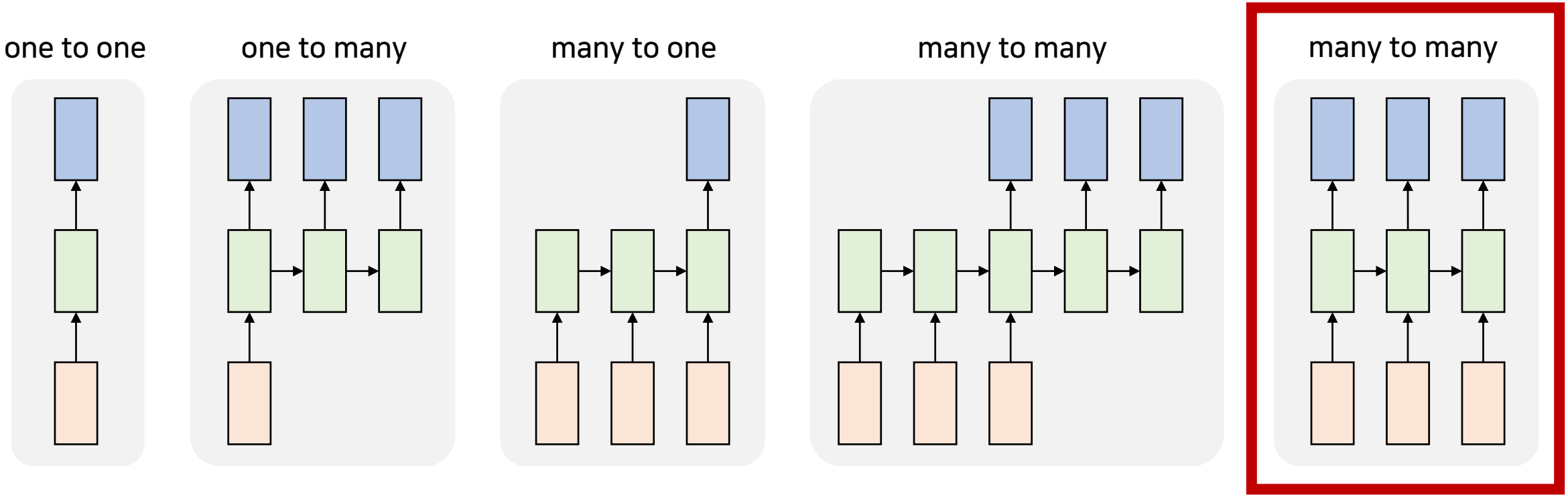
RNN의 다양한 아키텍처

- Many-to-Many: 다수의 입력을 받아 다수의 출력을 내보내는 네트워크
  - **대표적인 예시**) 기계 번역(machine translation)



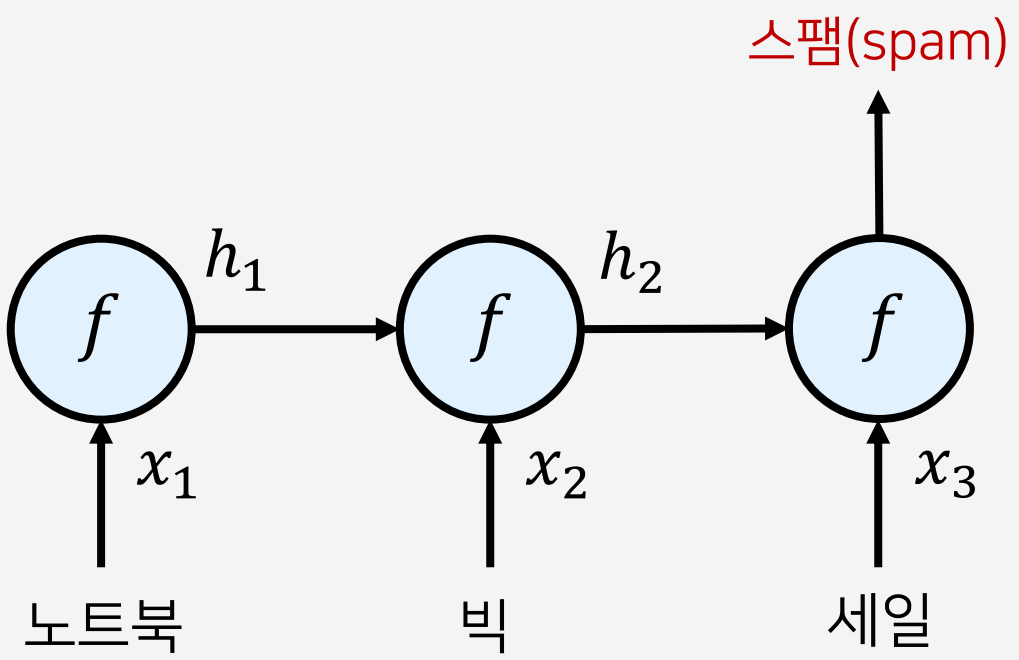
RNN의 다양한 아키텍처

- Many-to-Many: 다수의 입력을 받아 다수의 출력을 내보내는 네트워크
  - **대표적인 예시**) 품사 태깅(POS tagging)

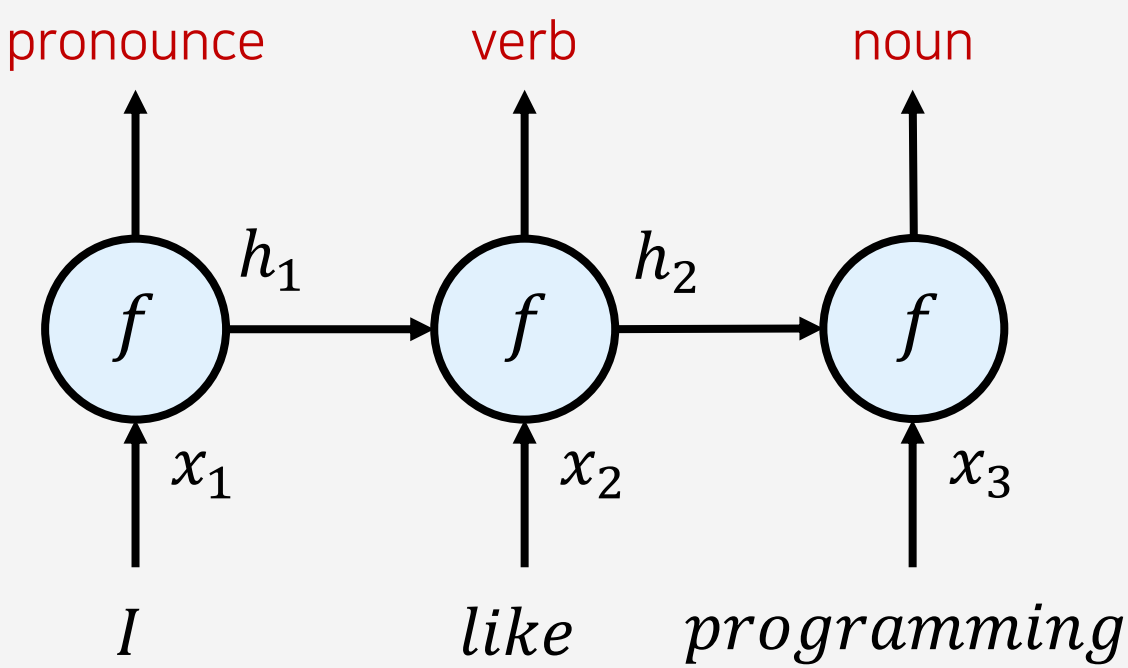


RNN의 다양한 아키텍처

스팸 분류 모델(many to one)

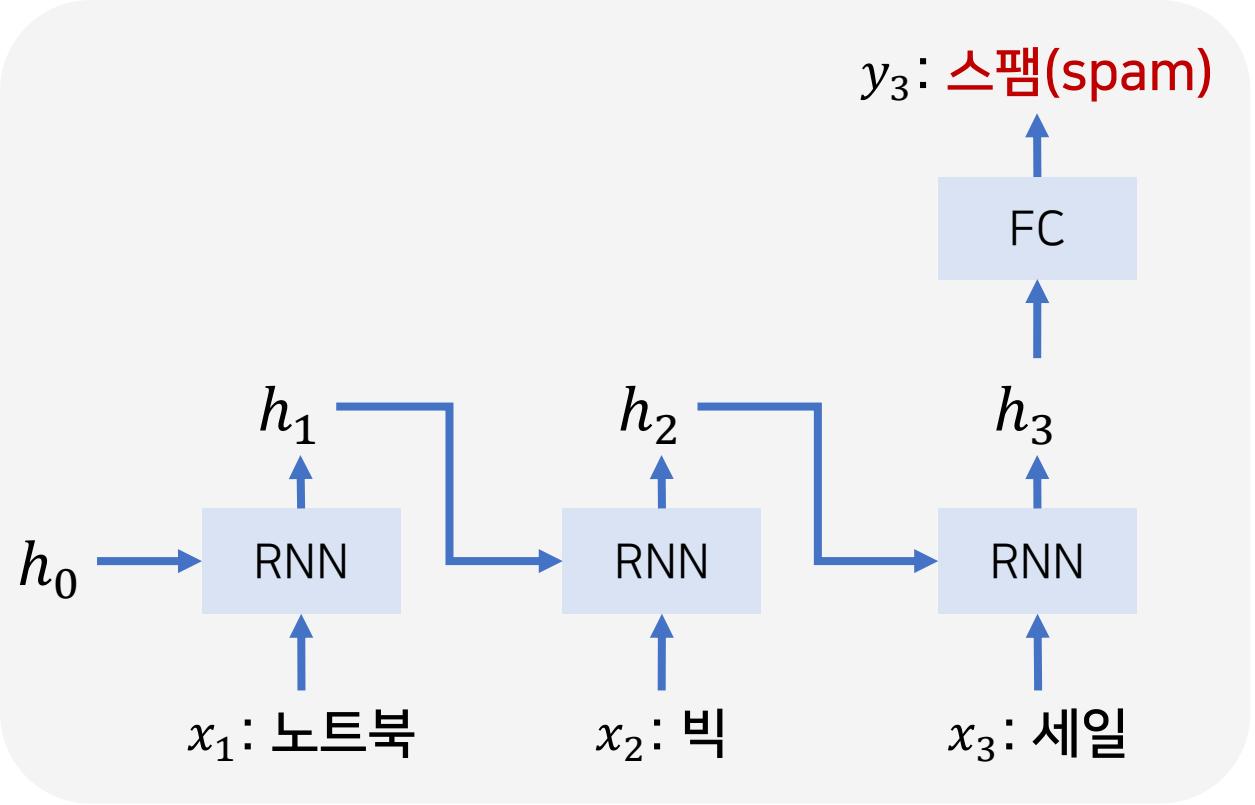


품사 태깅(many to many)



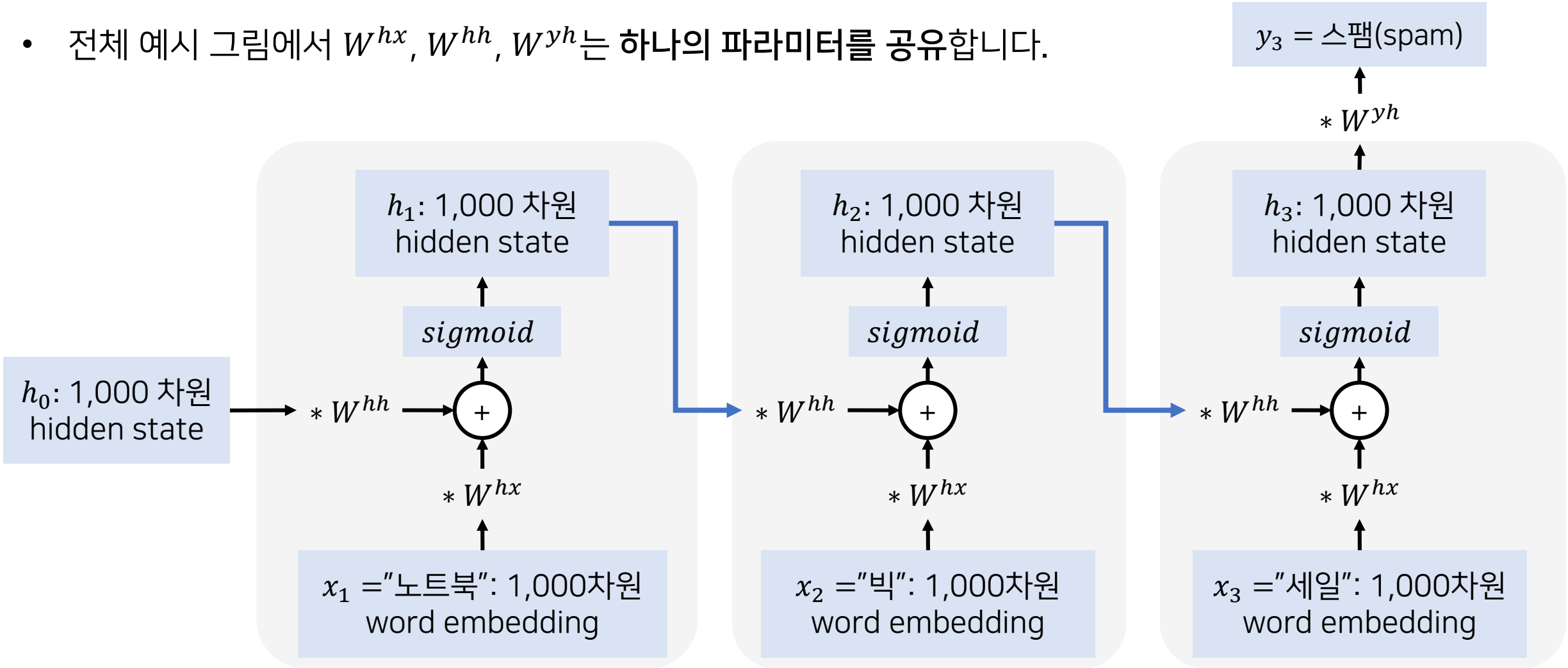
RNN 자세히 알아보기

- 입력(input)
  - x_t = 각각의 입력 단어
- 히든 상태(hidden state)
  - $h_t = sigmoid\left(W^{hx}x_t + W^{hh}h_{t-1}\right)$
- 출력(output)
  - $y_t = W^{yh}h_t$



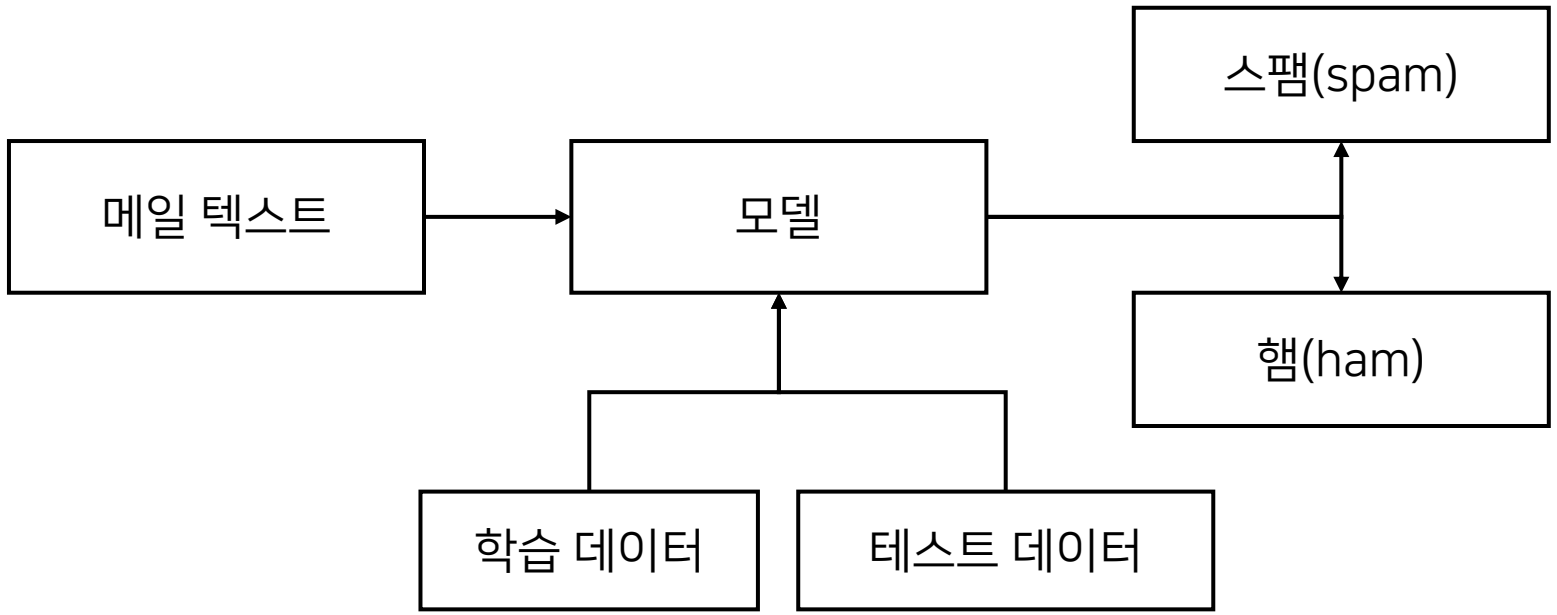
RNN 자세히 알아보기

- 전체 예시 그림에서 W^{hx}, W^{hh}, W^{yh}는 **하나의 파라미터를 공유**합니다.



훈련 데이터와 테스트 데이터

- 텍스트 분류는 주어진 텍스트를 **특정한 클래스(Class)로 분류**하는 작업을 의미합니다.

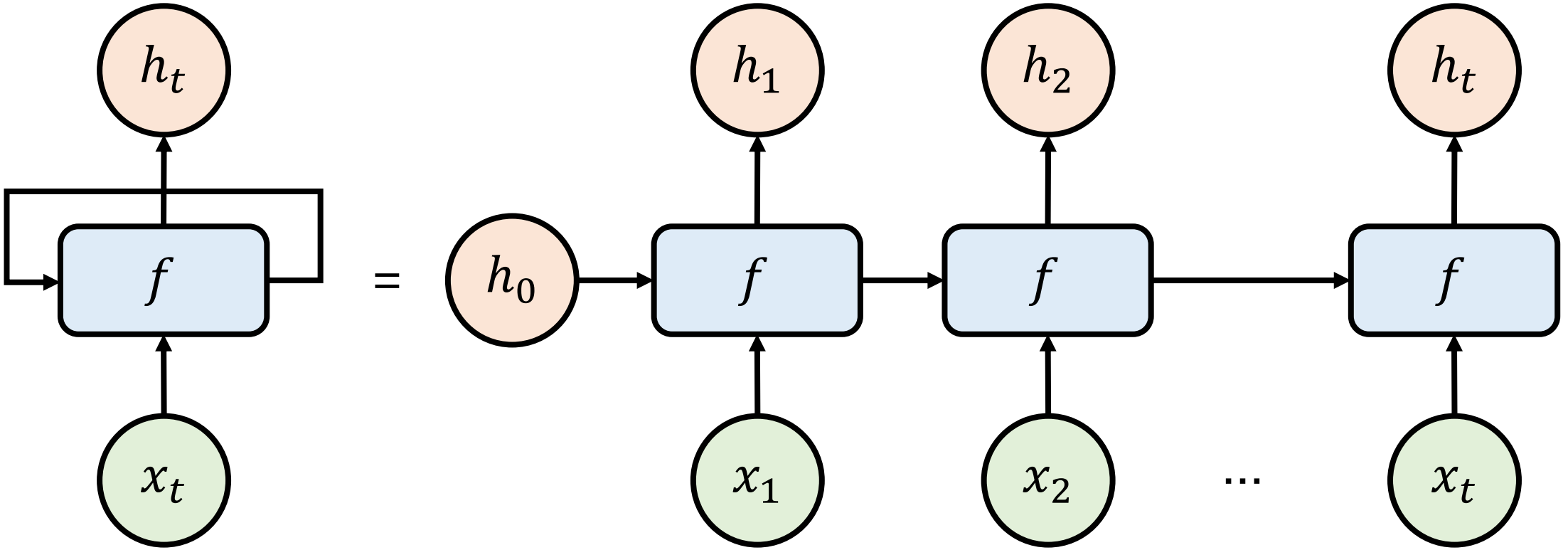


[실습] RNN 공부하기

- **RNN을 직접 구현하여 주기 함수 학습하기**
  - RNN은 연속적인 형태의 데이터를 효과적으로 학습할 수 있습니다.
- **스팸 메일 분류 (Spam Detection)**
  - 스팸 메일 분류는 전형적인 텍스트 분류 예시입니다.
  - Dataset: Kaggle SMS Spam Collection

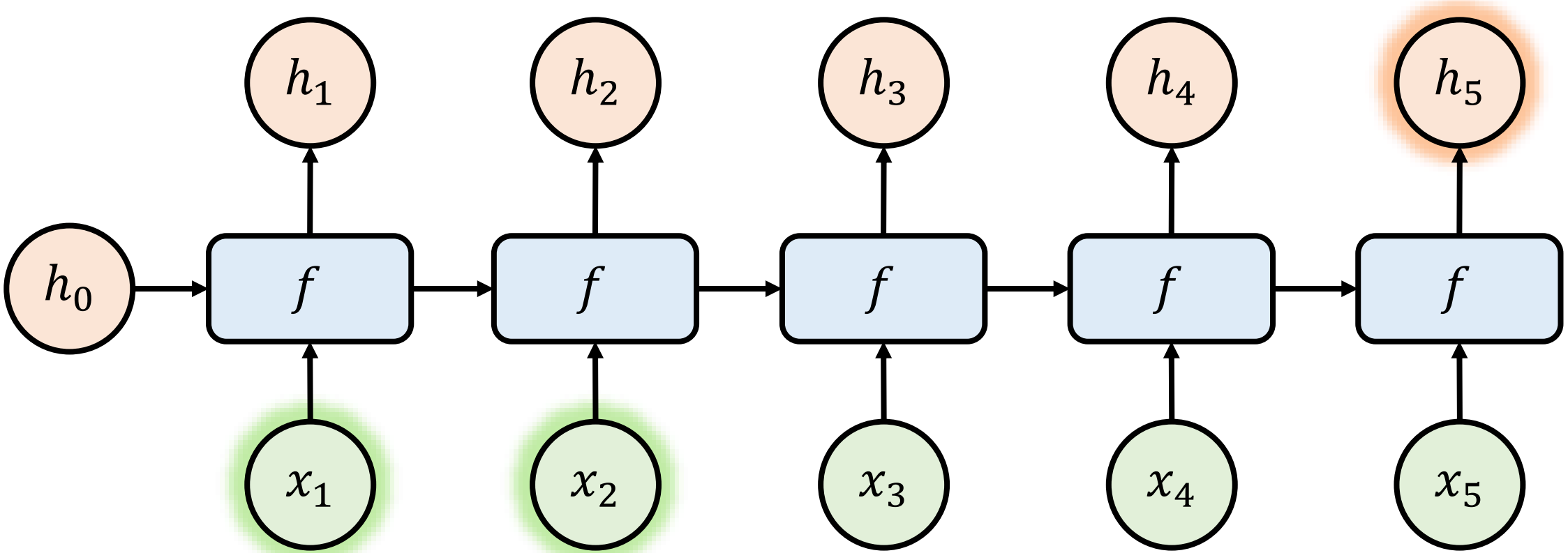
RNN의 한계점

- 이론적으로는 RNN을 이용하여 긴 길이의 순차적인 데이터를 효과적으로 처리할 수 있습니다.

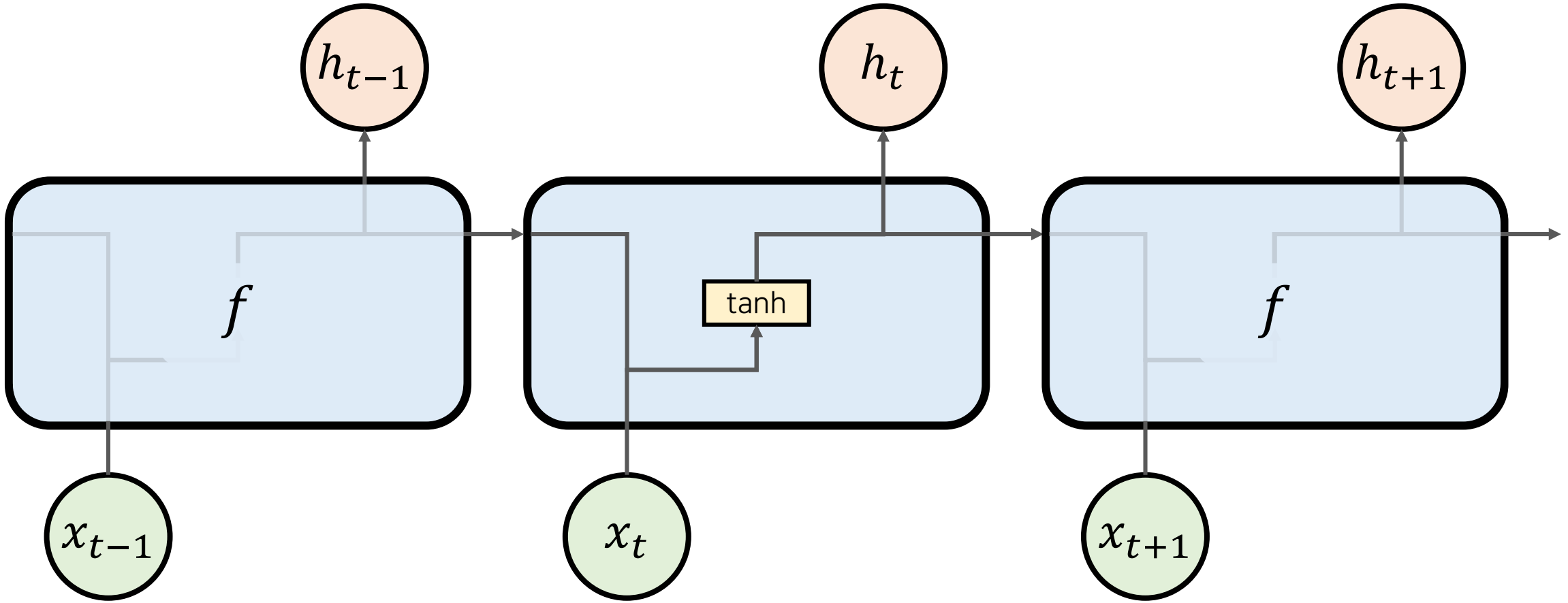


RNN의 한계점

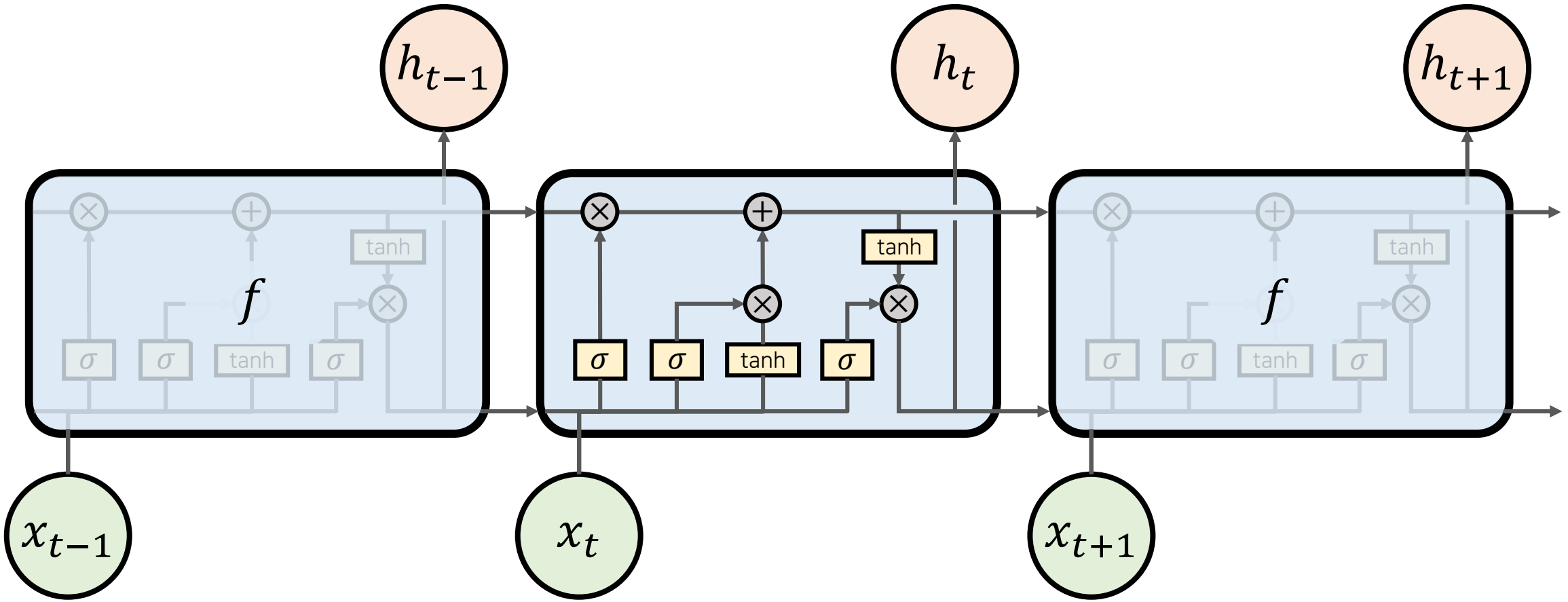
- 실제로는 토큰(token) 사이의 거리가 먼 경우 연속적인 정보가 잘 전달되지 않을 수 있습니다.



RNN(Recurrent Neural Network) 아키텍처

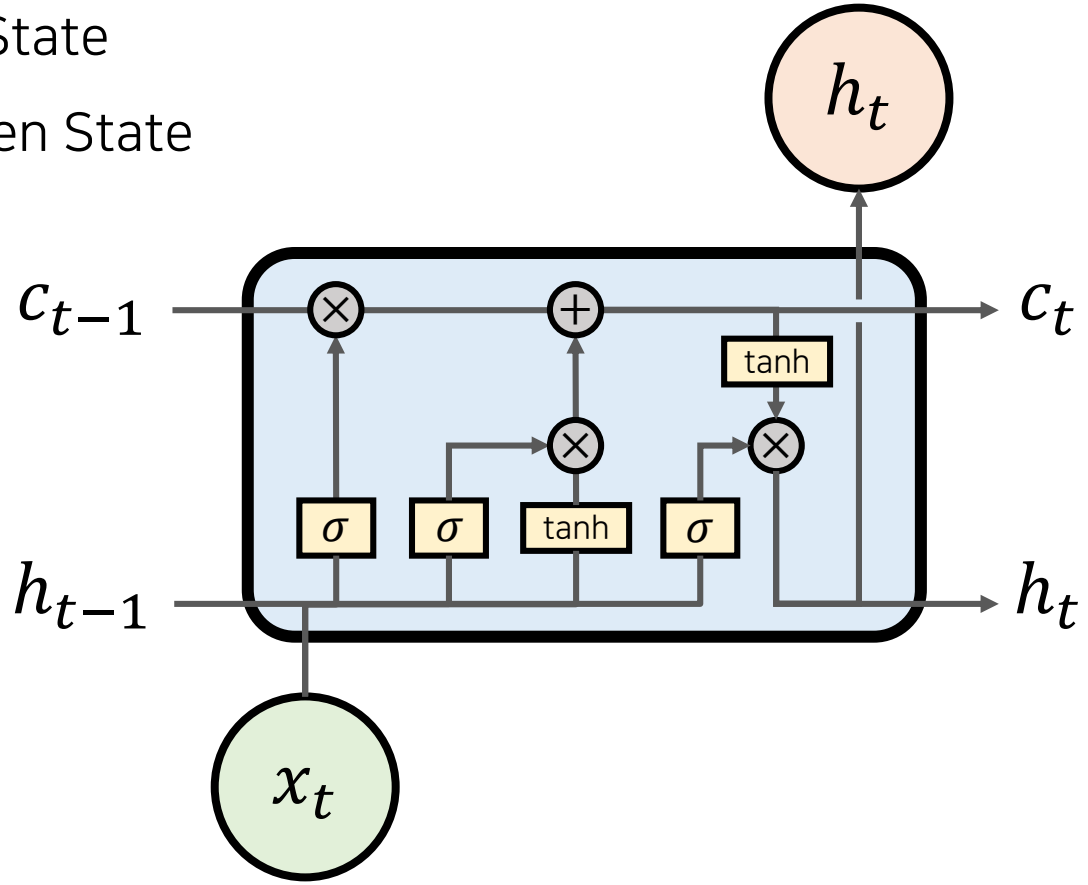


LSTM(Long Short-Term Memory) 아키텍처



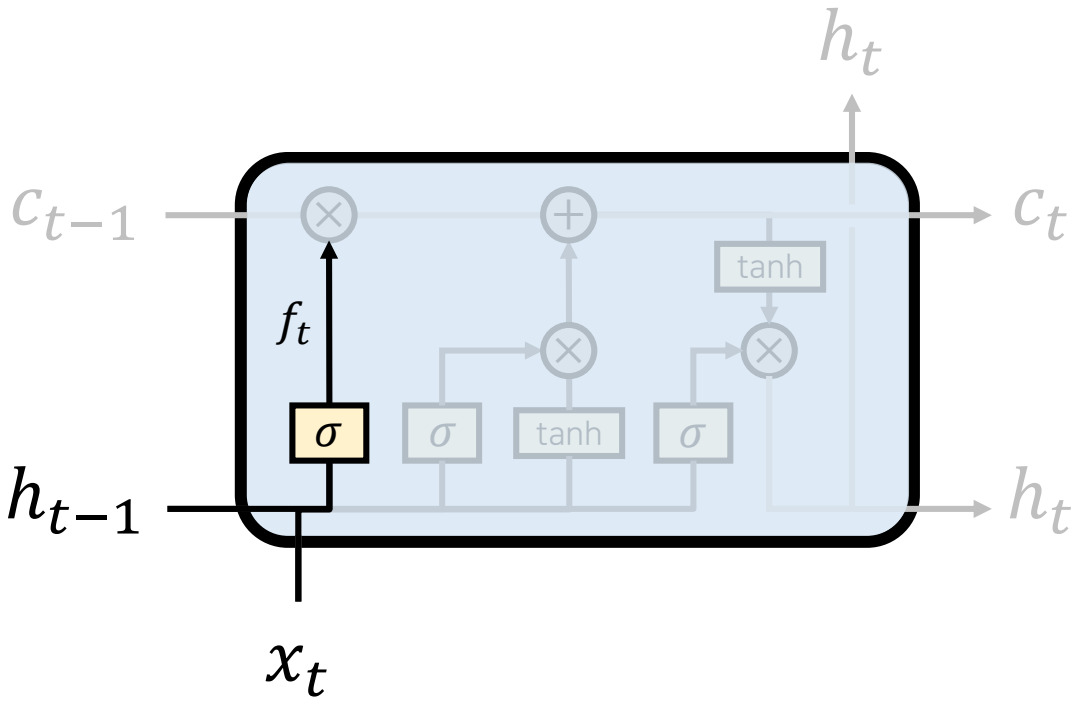
LSTM 핵심 아이디어: 두 개의 상태 정보

- LSTM은 RNN과는 다르게 **두 개의 상태 정보**를 저장하고 처리합니다.
  - **장기 기억**: Cell State
  - **단기 기억**: Hidden State



LSTM 핵심 아이디어: 게이트(Gates)

- Forget Gate는 어떠한 정보를 잊게 만들지 결정하는 레이어입니다.
  - 오래된 정보 중에서 필요 없는 정보는 잊게 됩니다.

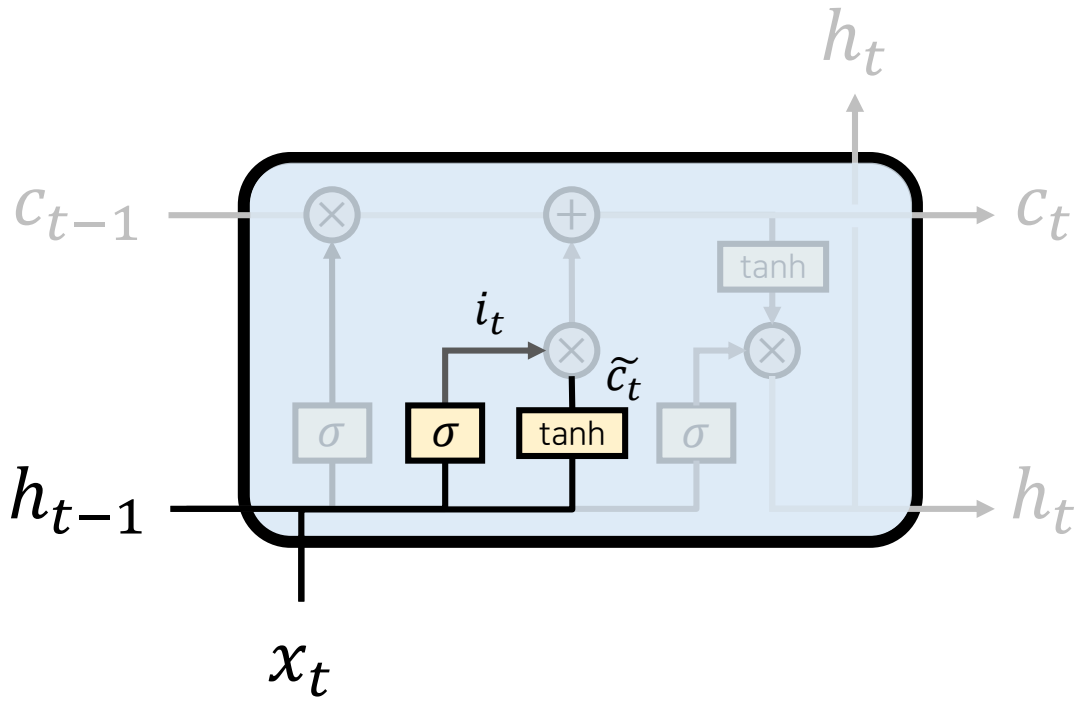


Formulation

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1})$$

LSTM 핵심 아이디어: 게이트(Gates)

- Input Gate는 새로운 정보를 장기 기억(Cell State)에 반영하는 역할을 수행합니다.
  - 새롭게 특정한 정보를 기억하도록 만듭니다.



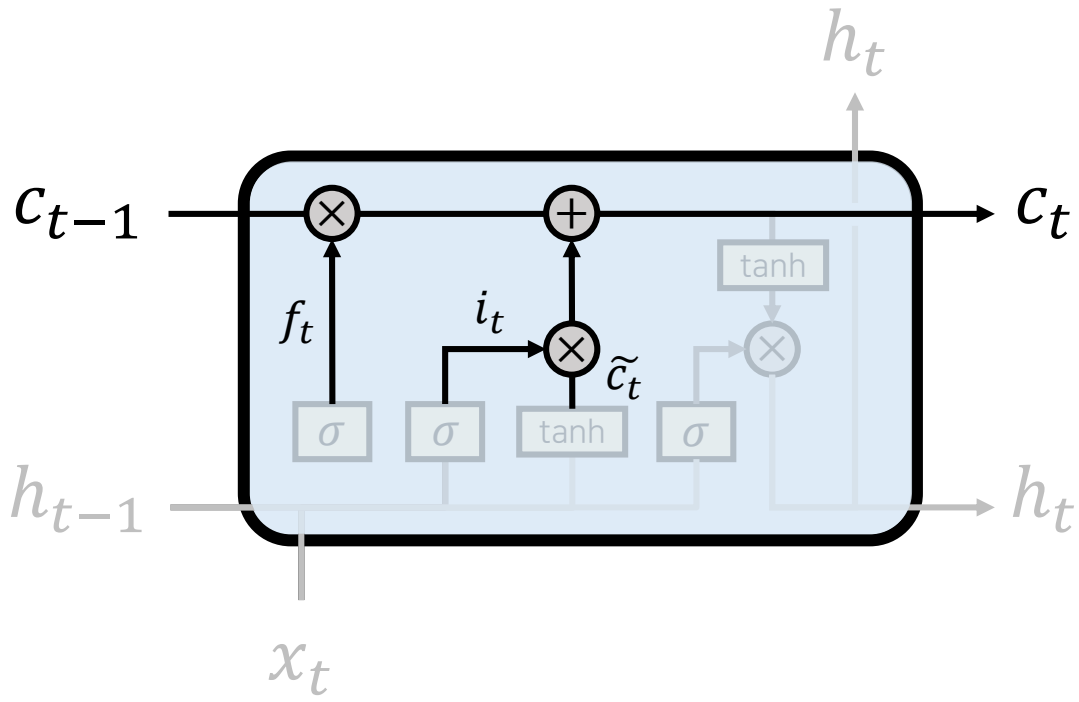
Formulation

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1})$$

$$\tilde{c}_t = tanh(W_{cx}x_t + W_{ch}h_{t-1})$$

LSTM 핵심 아이디어: 게이트(Gates)

- **장기 기억(Cell State)**은 Forget Gate와 Input Gate를 이용하여 업데이트됩니다.

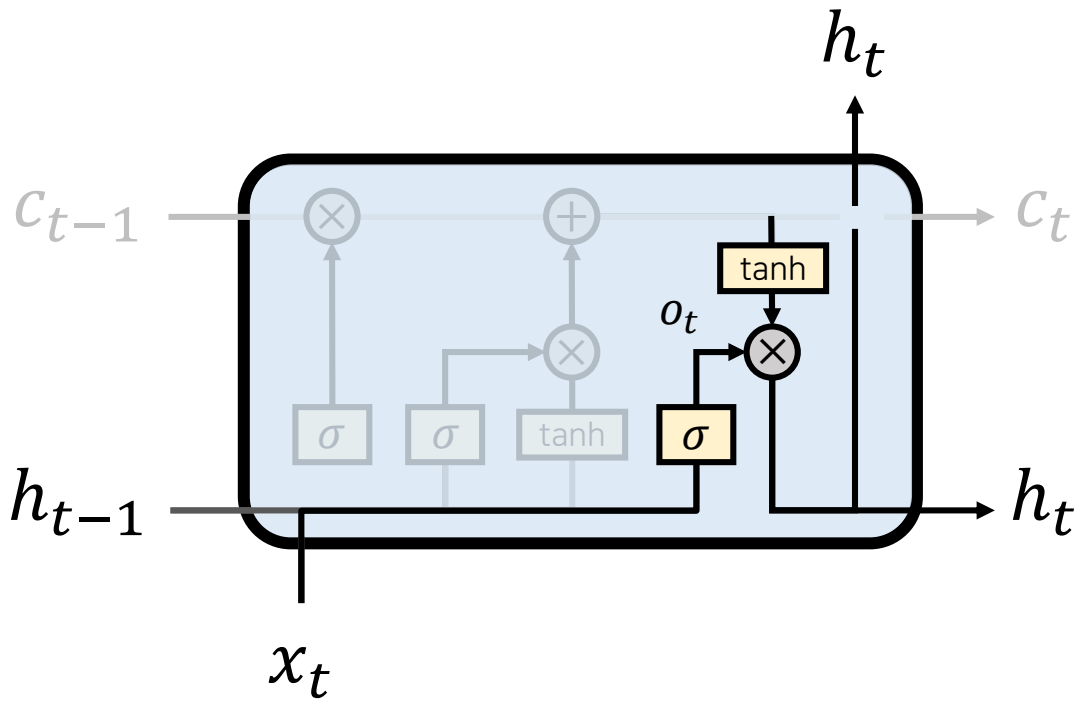


Formulation

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

LSTM 핵심 아이디어: 게이트(Gates)

- Output Gate는 장기 기억과 현재의 데이터를 이용해 단기 기억(Hidden State)을 갱신합니다.



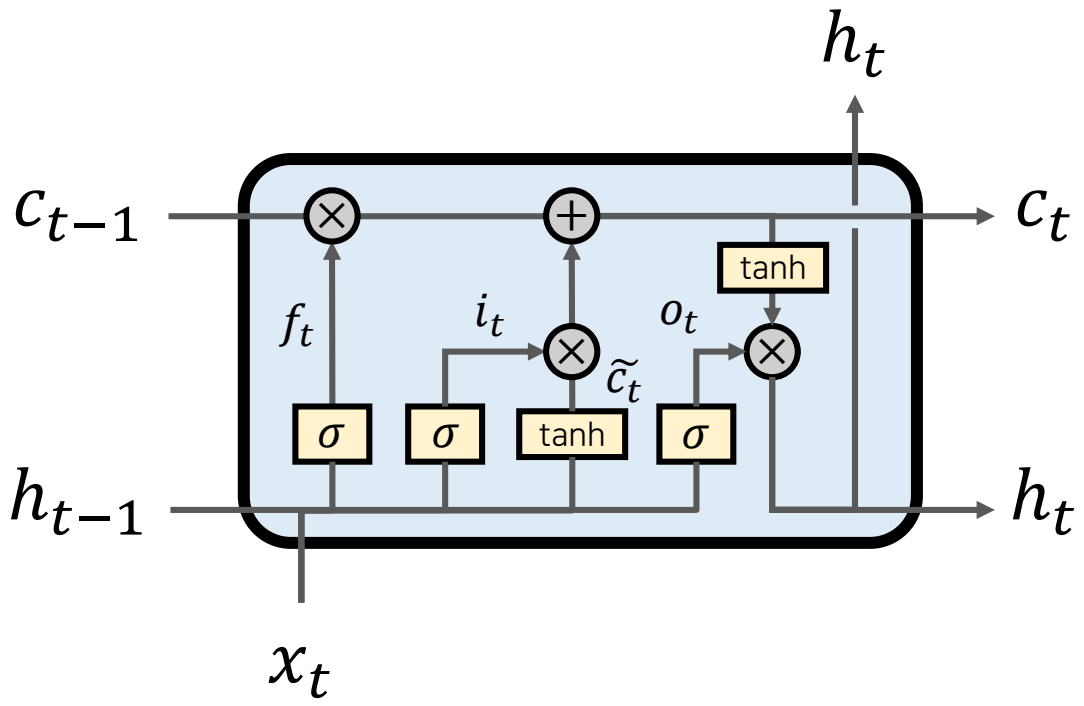
Formulation

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1})$$

$$h_t = o_t * tanh(c_t)$$

LSTM 전체 공식

- LSTM 전체 공식은 다음과 같습니다.
  - 공식에 등장하는 **모든 가중치(weight)는 공유**됩니다.



Formulation

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1})$$

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1})$$

$$\tilde{c}_t = tanh(W_{cx}x_t + W_{ch}h_{t-1})$$

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1})$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

$$h_t = o_t * tanh(c_t)$$

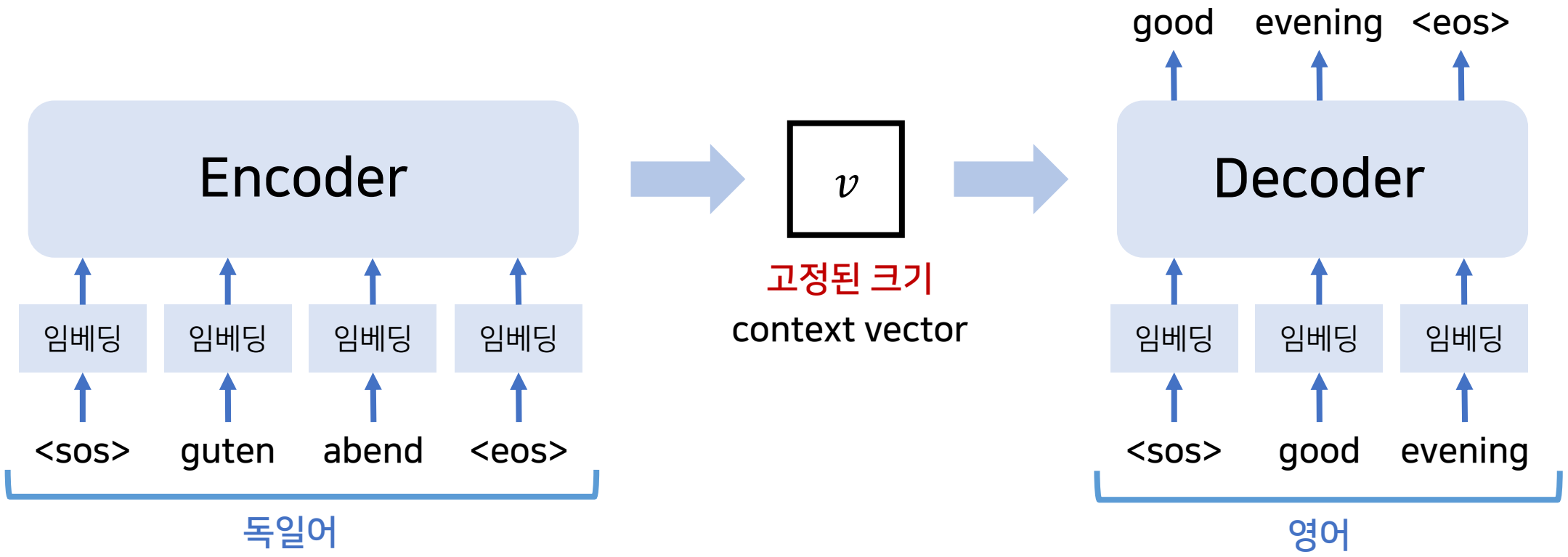
[실습] PyTorch를 활용한 RNN과 LSTM 실습

- PyTorch RNN을 활용한 주기 함수 학습
  - RNN은 연속적인 형태의 데이터를 효과적으로 학습할 수 있습니다.
- PyTorch LSTM을 활용한 주기 함수 학습
  - LSTM은 연속적인 형태의 데이터를 효과적으로 학습할 수 있습니다.

기계 번역(Machine Translation)

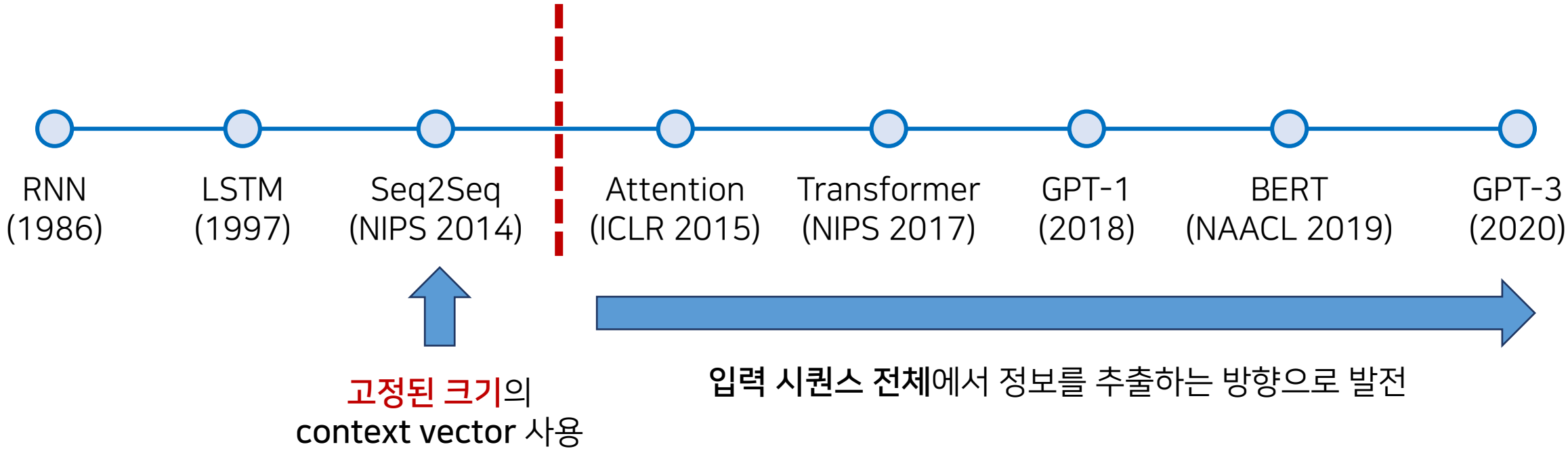
Sequence to Sequence Learning with Neural Networks (NIPS 2014)

- 본 논문에서는 LSTM을 활용한 효율적인 **Seq2Seq** 기계 번역 아키텍처를 제안합니다.
  - Seq2Seq는 딥러닝 기반 기계 번역의 돌파구와 같은 역할을 수행했습니다.
  - Transformer(2017)가 나오기 전까지 state-of-the-art로 사용되었습니다.



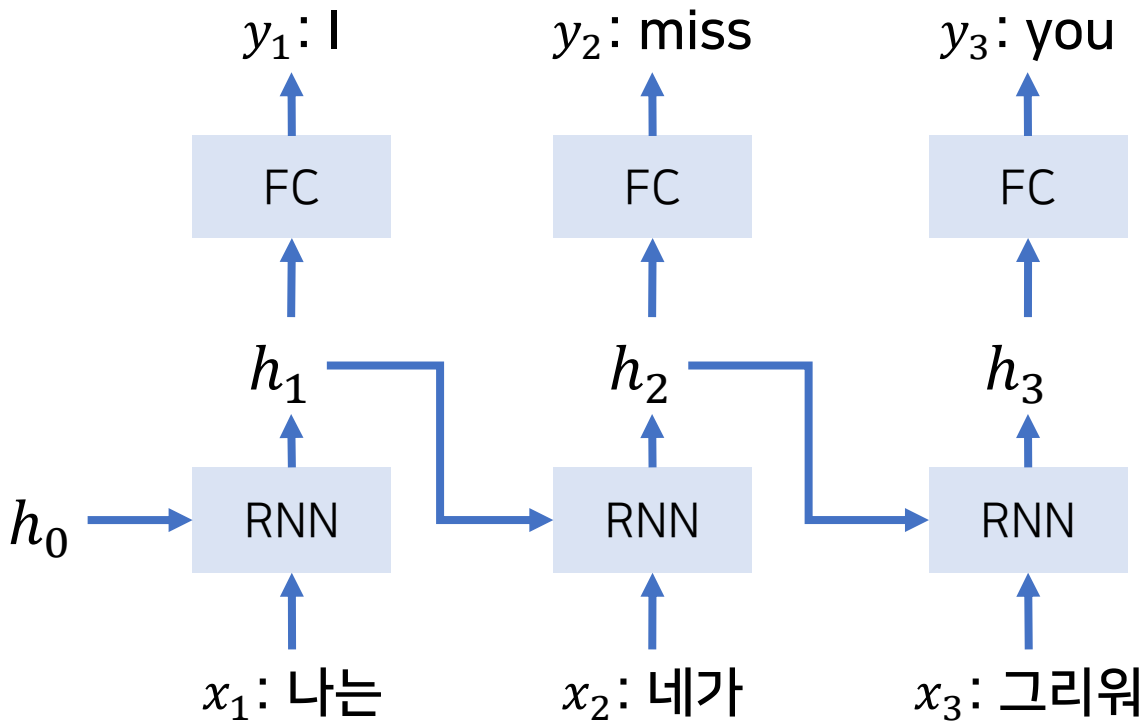
딥러닝 기반의 기계 번역 발전 과정

- 2021년 기준으로 최신 고성능 모델들은 Transformer 아키텍처를 기반으로 하고 있습니다.
  - **GPT**: Transformer의 **디코더(Decoder)** 아키텍처를 활용
  - **BERT**: Transformer의 **인코더(Encoder)** 아키텍처를 활용



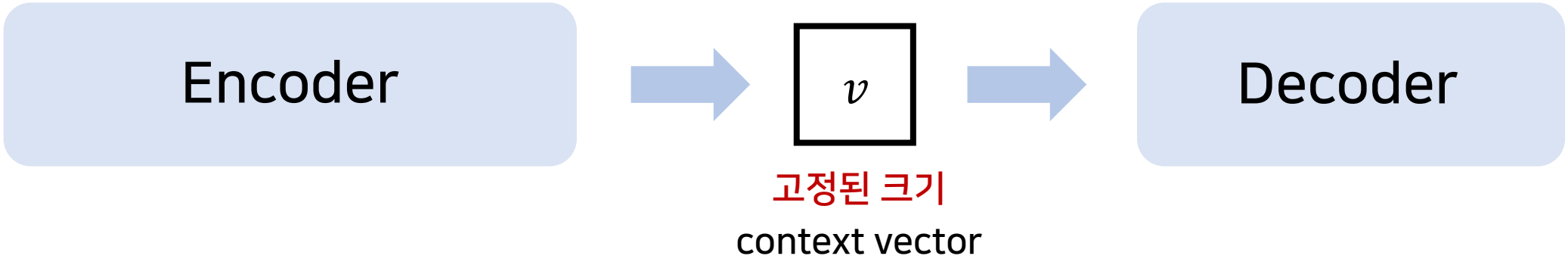
전통적인 RNN 기반의 번역 과정

- 전통적인 초창기 RNN 기반의 언어 모델에서 번역이 이루어지는 과정은 다음과 같습니다.
- 전통적인 RNN 기반의 기계 번역은 입력과 출력의 크기가 같다고 가정합니다.
  - **입력**: $(x_1, \dots, x_T)$
  - **출력**: $(y_1, \dots, y_T)$
    - $h_t = sigmoid(W^{hx}x_t + W^{hh}h_{t-1})$
    - $y_t = W^{yh}h_t$



RNN 기반의 Sequence to Sequence 개요

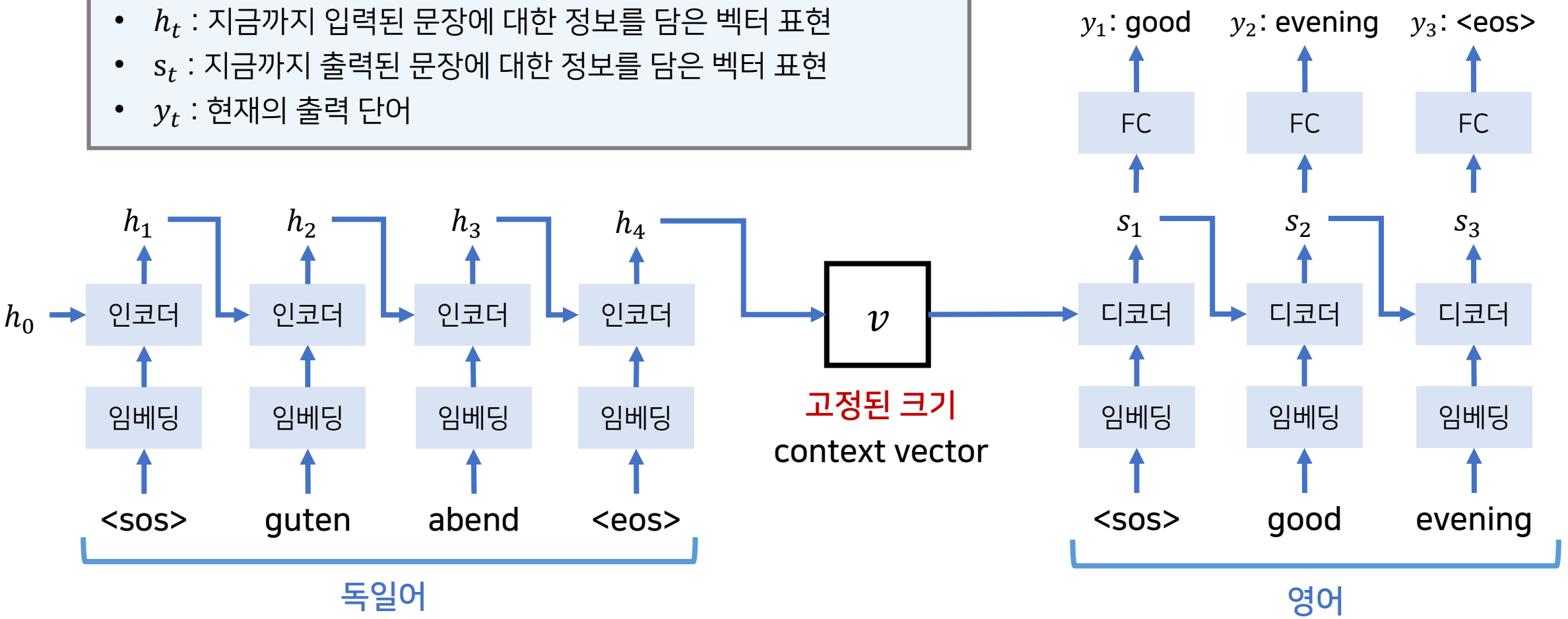
- 전통적인 초창기 RNN 기반의 언어 모델은 다양한 한계점을 가지고 있습니다.
  - 이를 해결하기 위해 **인코더**가 고정된 크기의 **문맥 벡터(context vector)**를 추출하도록 합니다.
  - 이후에 문맥 벡터로부터 **디코더**가 번역 결과를 추론합니다.
  - 본 Seq2Seq 논문에서는 LSTM를 이용해 문맥 벡터를 추출하도록 하여 성능을 향상시킵니다.
    - 인코더의 마지막 hidden state만을 context vector로 사용합니다.



- **인코더**와 **디코더**는 **서로 다른 파라미터(가중치)**를 가집니다.

RNN 기반의 Sequence to Sequence 자세히 살펴보기

- x_t : 현재의 입력 단어
- h_t : 지금까지 입력된 문장에 대한 정보를 담은 벡터 표현
- s_t : 지금까지 출력된 문장에 대한 정보를 담은 벡터 표현
- y_t : 현재의 출력 단어

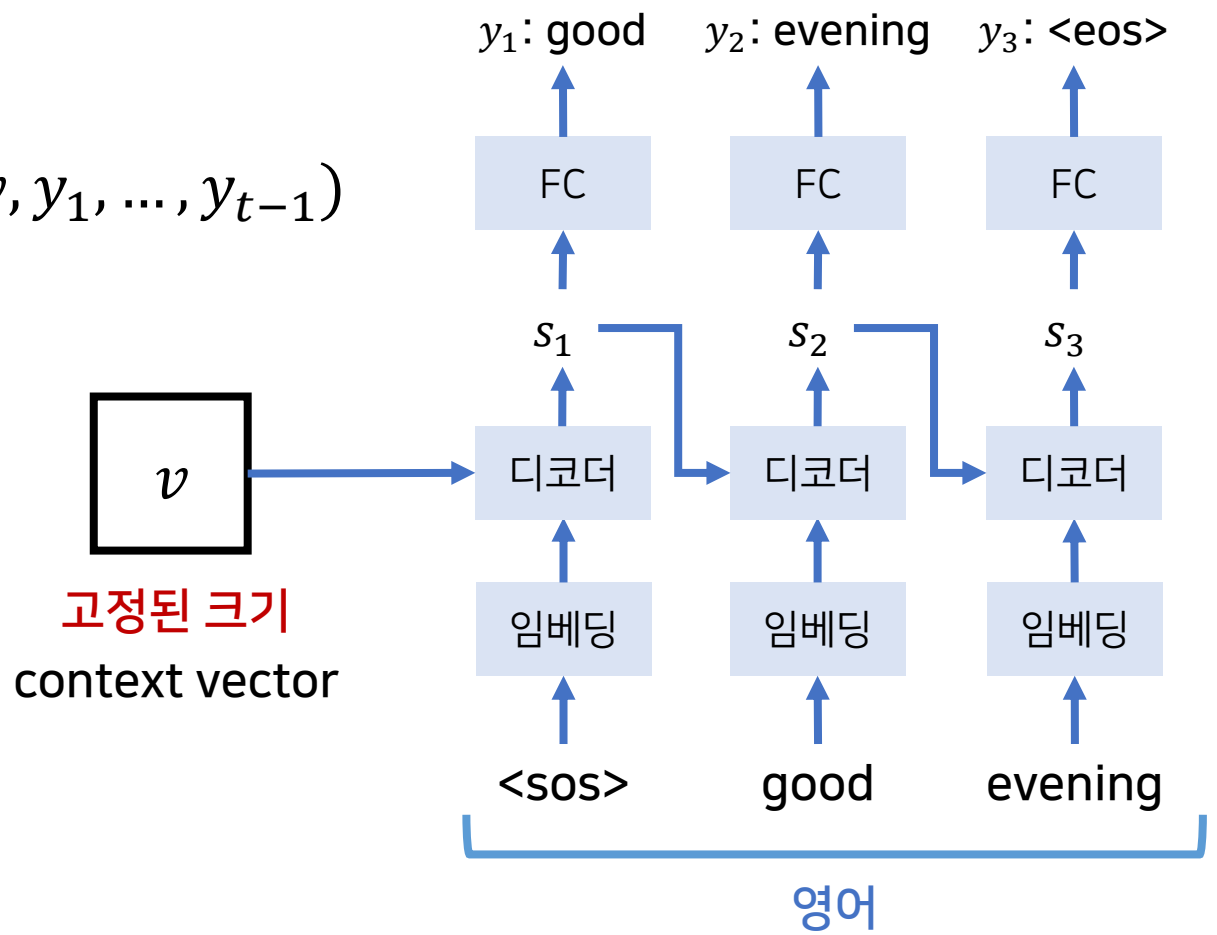


RNN 기반의 Sequence to Sequence 자세히 살펴보기

- RNN 기반 Seq2Seq 모델의 목표 공식(formulation)은 다음과 같습니다.
  - **종료 시점**: y_t = <eos>

$$p(y_1, \dots, y_{T'} | x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \dots, y_{t-1})$$

- v : $x_1, \dots, x_T$에 대한 정보를 담은 벡터 표현
- y_t : 현재의 출력 단어



Seq2Seq의 성능 개선 포인트: LSTM 활용 및 입력 문장의 순서 뒤집기

- 기본적인 RNN 대신에 **LSTM**을 활용했을 때 더 높은 정확도를 보입니다.
- 실제 학습 및 테스트 과정에서 **입력 문장의 순서를 거꾸로 했을 때 더 높은 정확도**를 보입니다.
  - 출력 문장의 순서는 바꾸지 않습니다.