2022-1 겨울방학
딥러닝 분반

분반장: 구은아, 김혜림

자연어 처리(NLP)

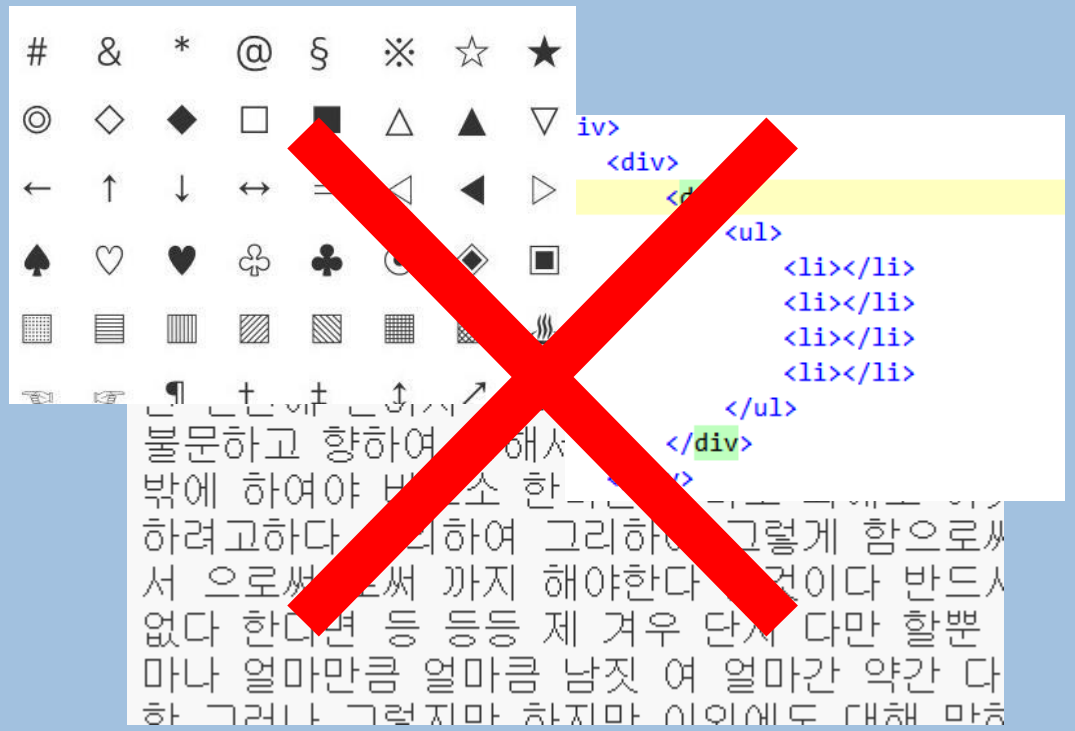
NLP: 인간의 언어를 컴퓨터로 분석하는 기술

ex. 기계 번역, 텍스트 분류, 감성 분석, 텍스트 요약, 음성 텍스트 분석 등

자연어 전처리

클렌징 (Cleansing)

**텍스트를 변환할 때 방해되거나 의미가 없어서
불필요한 것들을 제거하는 작업**



클렌징의 예시

- 특수문자, 기호 제거
- HTML 태그 제거(HTML 문서의 경우)
- 대소문자 구분 제거
- stopword 제거

토큰화

This is a sentence. This is a sample.

전체 텍스트를 그대로 input하면 컴퓨터는 이 텍스트를 하나의 덩어리로 인식

문장 토큰화

This is a sentence

This is a sample

텍스트를 일정한 단위로 분리하여 그 내용을 인식할 수 있도록 해주어야 함.

토큰화

This is a sentence. This is a sample.

단어 토큰화 ⬇

This | is | a | sentence

This | is | a | sample

한국어 토큰화

사과의 놀라운 효능이라는 글을 봤어. 그래서 오늘 사과를 먹으려고 했는데 사과가 썩어서 슈퍼에 가서 사과랑 오렌지 사왔어

띄어쓰기를 기준으로 단어 토큰화

['사과의', '놀라운', '효능이라는', '글을', '봤어.', '그래서', '오늘', '사과를', '먹으려고', '했는데', '사과가', '썩어서', '슈퍼에', '가서', '사과랑', '오렌지', '사왔어']

실제로 모두 같은 '사과'를 의미하는 토큰이지만 컴퓨터는 다른 단어로 인식

한국어 토큰화

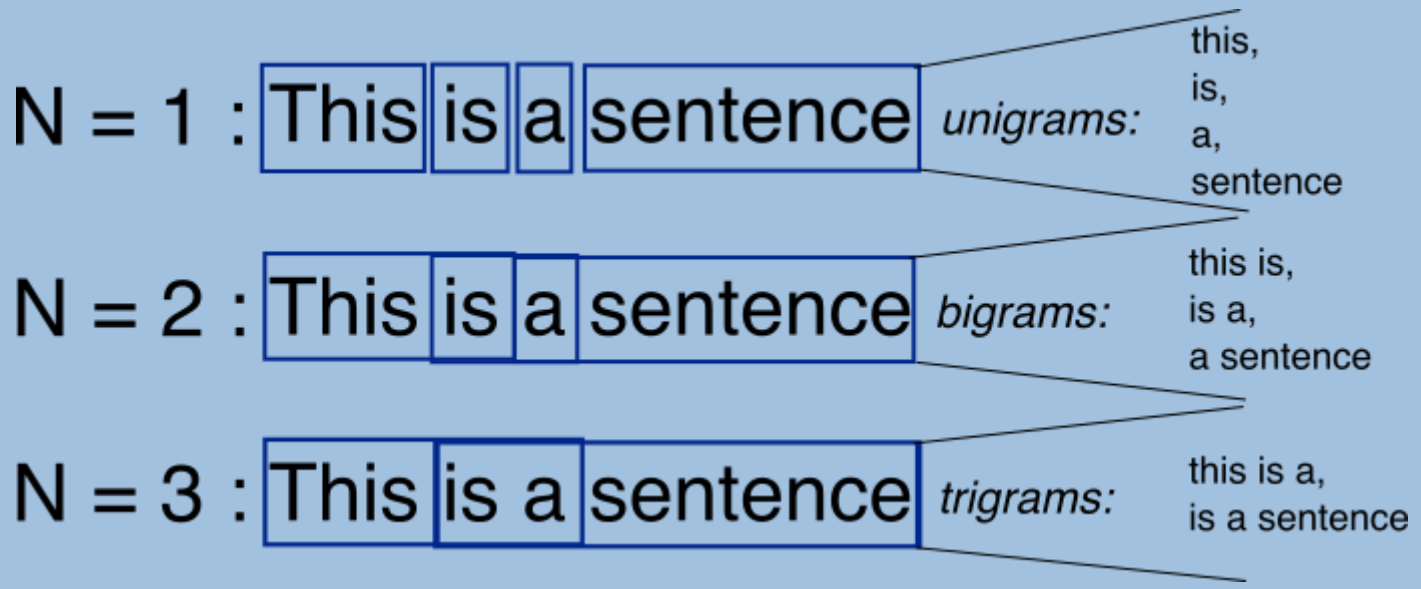
사과의 놀라운 효능이라는 글을 봤어. 그래서 오늘 사과를 먹으려고 했는데 사과가 썩어서 슈퍼에 가서 사과랑 오렌지 사왔어

형태소 분석

['사과', '의', '놀라운', '효능', '이', '라는', '글', '을', '봤', '어', '.', '그래서', '오늘', '사과', '를', '먹', '으려고', '했', '는데', '사과', '가', '썩', '어서', '슈퍼', '에', '가', '서', '사과', '랑', '오렌지', '사', '왔', '어']

형태소를 기준으로 문법적 도구와 의미를 가지는 부분을 잘 나누어서 토큰화해야 한다.

n-gram 토큰화



단어 토큰화를 진행하면 원문장에서 단어가 가지는 문맥적 의미가 무시된다.
이 문제를 해결하기 위해 연속된 n개의 단어를 묶어 한 개의 토큰으로 만드는 것을
n-gram 토큰화라고 한다.

단어 집합

단어	번호
…	
사과	11
를	12
먹	13
으려고	14
…	

‘사과를 먹으려고’

[11, 12, 13, 14]로 인식

Stemming, Lemmatization

Stemming

문법 요소에 따라 변환된 단어의 원형(어근)을 추출하기 위해 어간을 추출하는 방법.

일정한 규칙을 갖고 단어의 어미를 자르기 때문에 간단하나 섬세하지 않아서 정확성이 떨어짐.

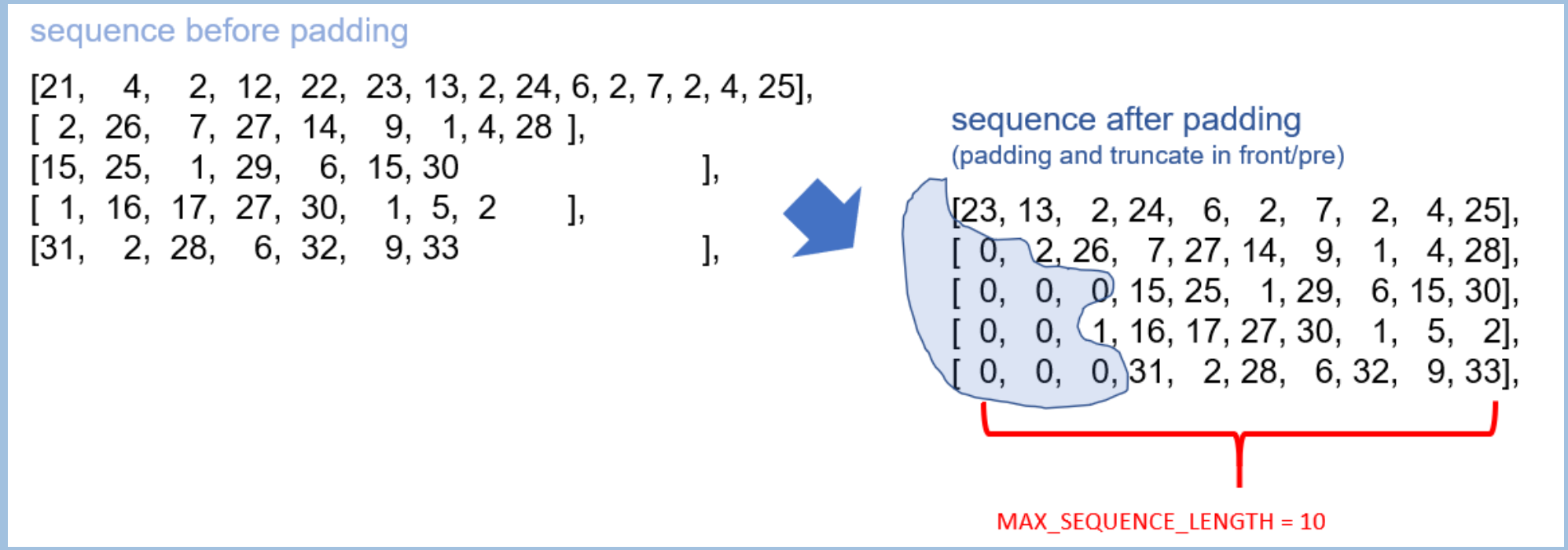
Lemmatization

특정한 규칙을 따르는 것이 아니라 문법 요소와 의미 요소를 감안해서 어근을 추출하는 방법.

Stemming에 비해 높은 정확성을 보이는 대신 시간이 오래 걸림.

ex. ‘am’, ‘are’, ‘is’를 ‘be’로 추출

패딩 (padding)



길이가 다른 문장을 모두 동일한 길이로 바꾸는 작업.
길이가 모두 같으면 컴퓨터가 텍스트를 처리하는 작업을 병렬적으로 연산할 수 있으므로 더 효율적이다.

임베딩

One-hot Encoding

I am a student

I → [1, 0, 0, 0]

a → [0, 0, 1, 0]

am → [0, 1, 0, 0]

student → [0, 0, 0, 1]

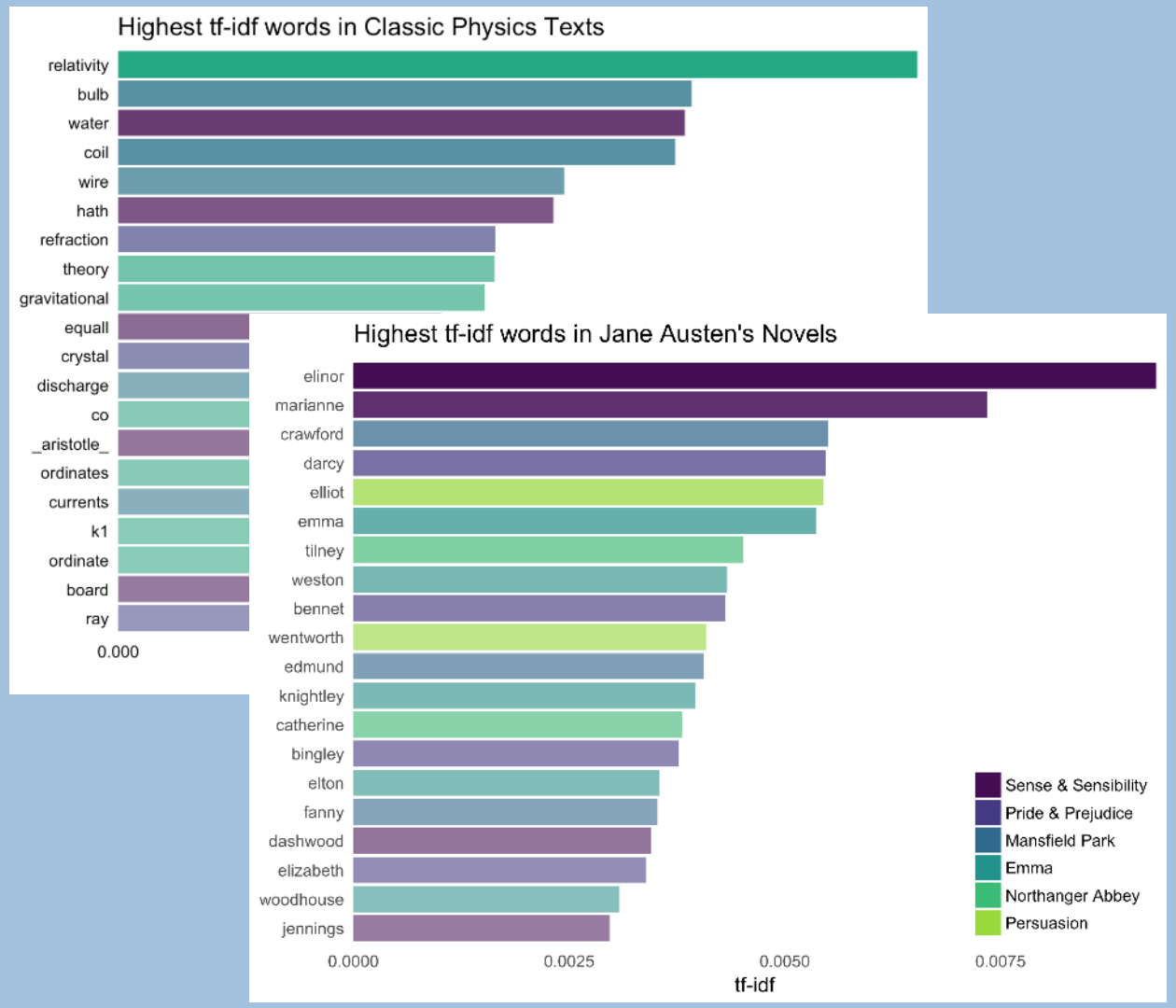
One-hot Encoding의 문제점

	cat	the	quick	brown	fox	jumped	over	dog	bird	flew	...	kangaroo	house
time	0	1	0	0	0	0	0	0	0	0	...	0	0
	0	0	1	0	0	0	0	0	0	0	...	0	0
	0	0	0	1	0	0	0	0	0	0	...	0	0
	0	0	0	0	1	0	0	0	0	0	...	0	0
	0	0	0	0	0	1	0	0	0	0	...	0	0
	0	0	0	0	0	0	1	0	0	0	...	0	0
	0	1	0	0	0	0	0	0	0	0	...	0	0

- 단어가 다양해질수록 벡터의 차원이 너무 커진다.
- 벡터에 0으로 채워진 빈 공간이 너무 많은 sparse 벡터를 만들기 때문에 공간 낭비가 심하다.
- 단어 간 유사도를 표현하지 못한다.

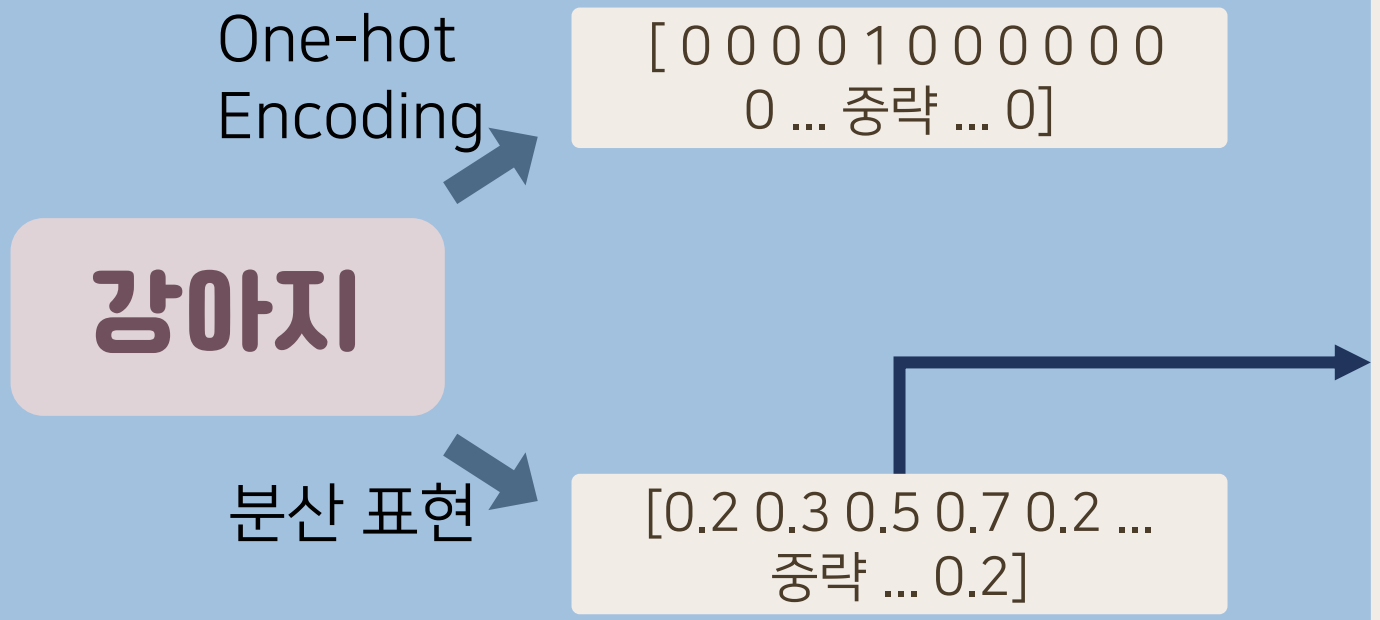
텍스트를 dense 벡터로 표현할 수 있는 방법 필요!
-> 워드 임베딩

TF-IDF



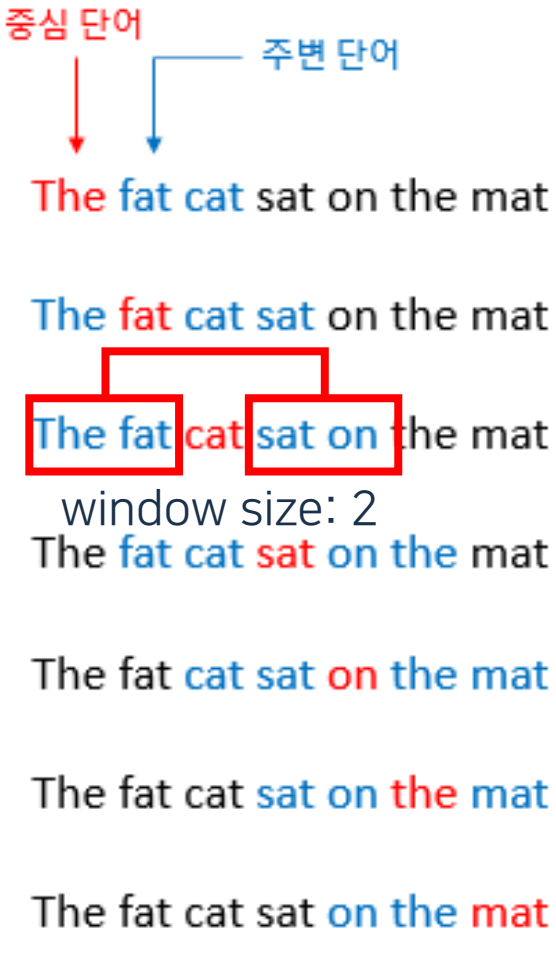
- 자주 나타나는 단어에 큰 가중치를 부여하는 카운팅 기반 임베딩 방법. 가중치가 클수록 중요한 단어라고 판단.
- 그러나 모든 문서에서 전반적으로 나타나는 단어는 서술 상 자주 쓰일 뿐 실제로는 중요하지 않을 가능성이 높다. 그래서 이러한 단어에는 패널티를 주어 중요성을 떨어뜨린다.

예측(추론) 기반 임베딩



- 분산 표현 방법의 경우, 벡터의 모든 element가 조합되어 강아지라는 의미를 표현.
- 비슷한 단어는 비슷한 형태의 벡터를 가지므로 단어의 유사도 계산 가능.
- 이러한 방식의 벡터화를 맥락 기반의 벡터화라고 하며, 대표적인 방법으로 Word2Vec이 있다.

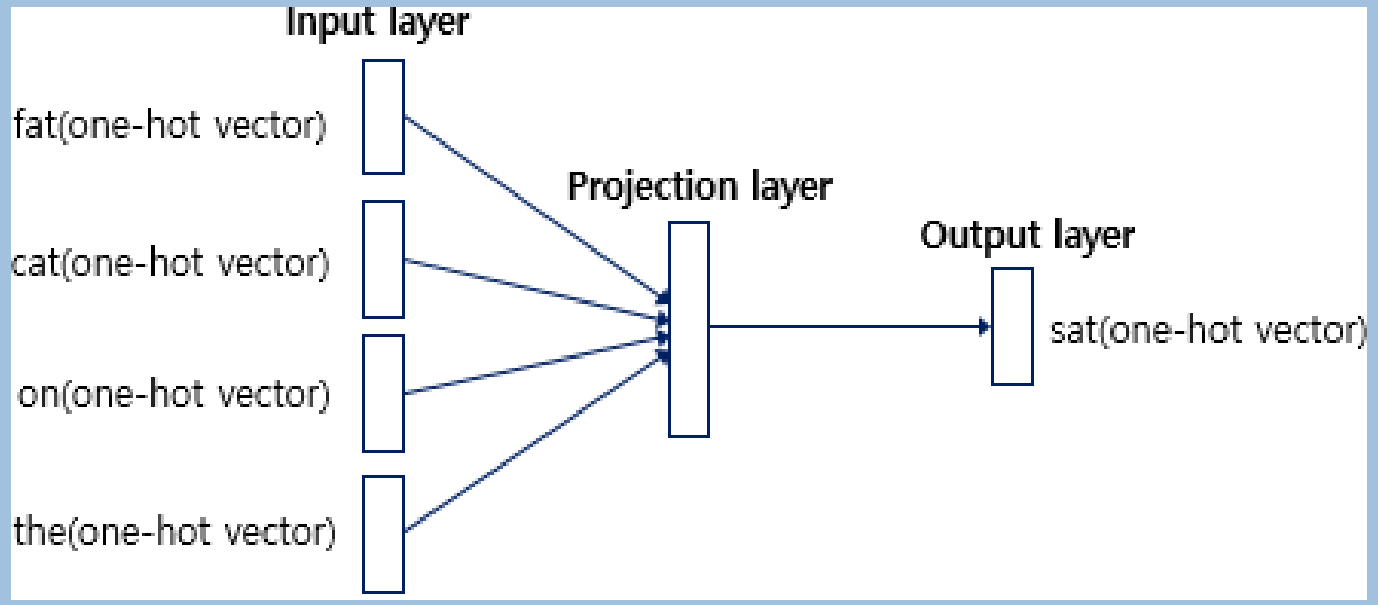
Word2Vec



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

- Word2Vec으로 벡터화하기 위해서는 중심 단어와 주변 단어로 이루어진 데이터셋을 새롭게 만들어야 한다.
- window size: 중심 단어를 기준으로 양쪽으로 몇 개를 주변 단어로 할 지 설정하는 파라미터.
- window size를 정했다면 window를 옮기면서 데이터셋을 만든다. 이를 슬라이딩 윈도우라고 한다.

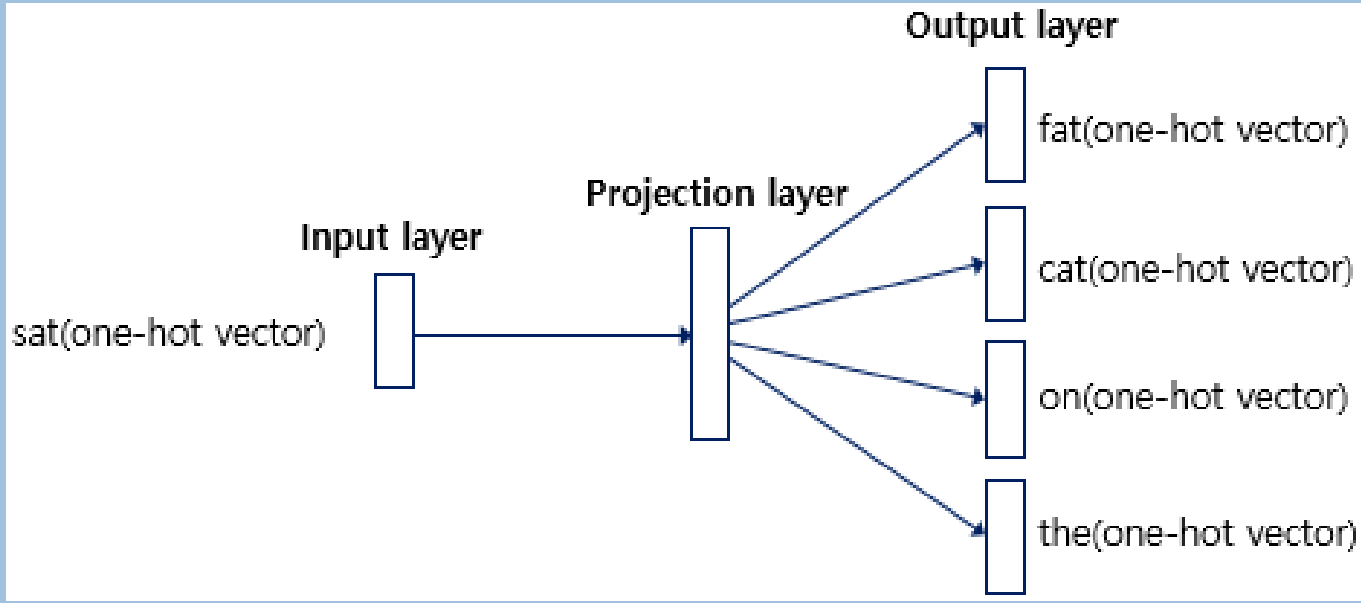
CBOW 기반 Word2Vec



CBOW 기반:
주변 단어들을 통해
중심 단어를 예측

중심 단어	주변 단어
output	[0, 1, 0, 0, … , 0, 0] ← input
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]

Skip-gram 기반 Word2Vec

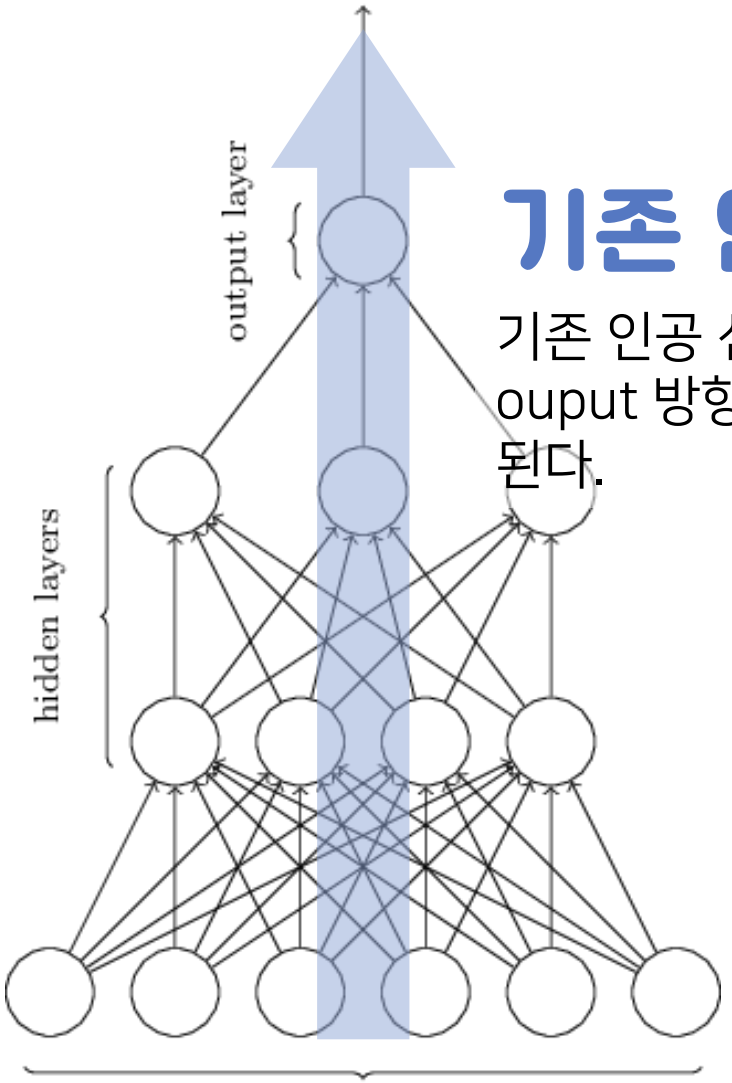


Skip-gram 기반:
중심 단어를 통해
주변 단어들을 예측

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

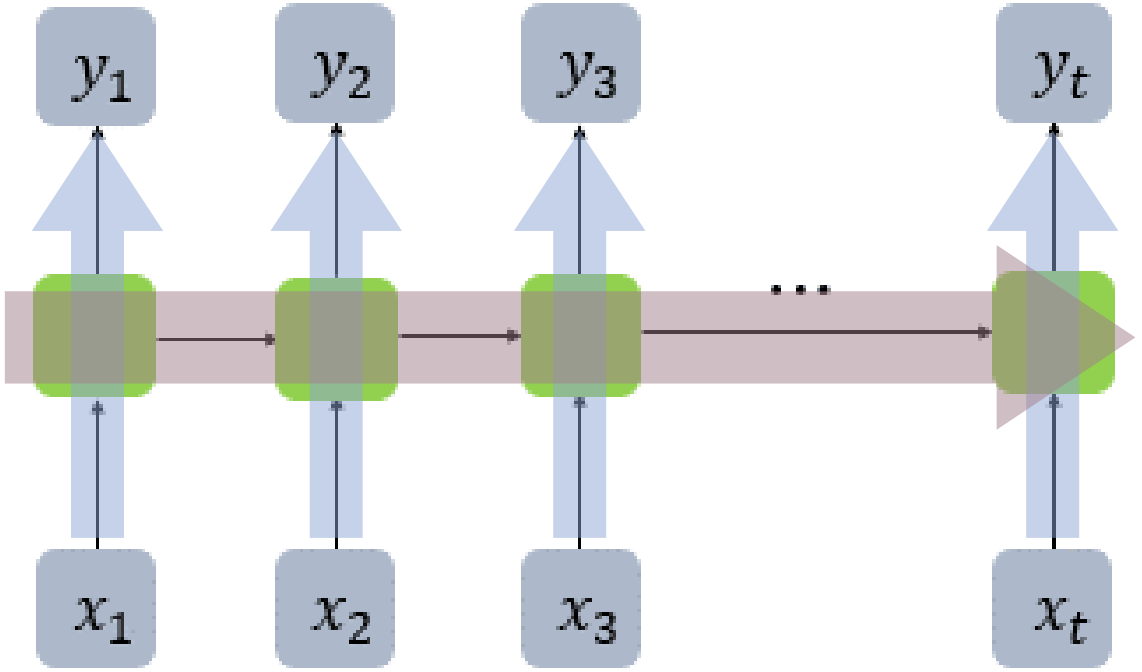
RNN

RNN



기존 인공 신경망

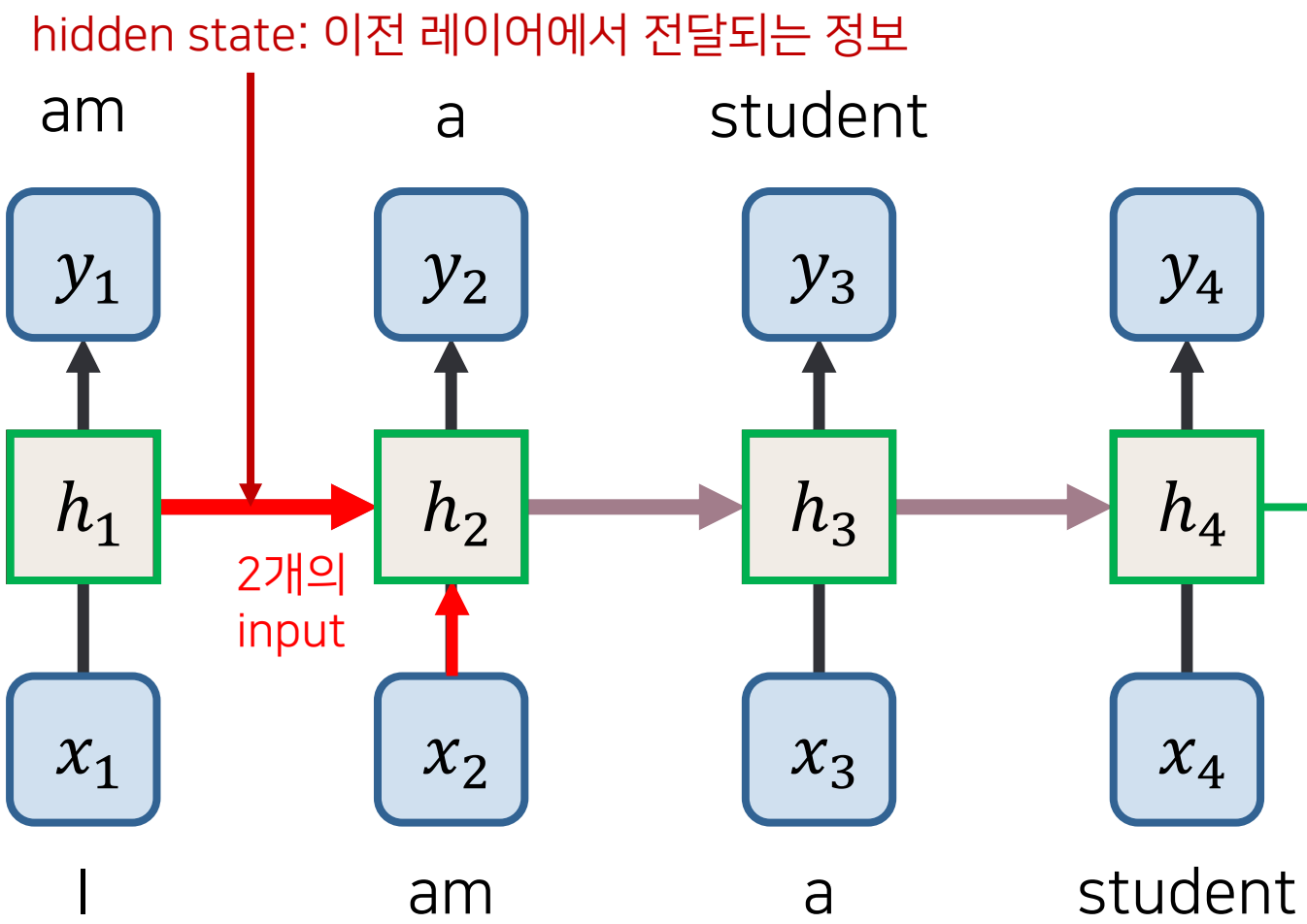
기존 인공 신경망은 input에서 ouput 방향으로만 학습이 진행된다.



RNN (순환 신경망)

학습이 한 방향으로 진행되지 않는다.
Input->output 방향 이외에 다음 time step의 hidden layer 방향으로도 학습이 진행된다.

RNN

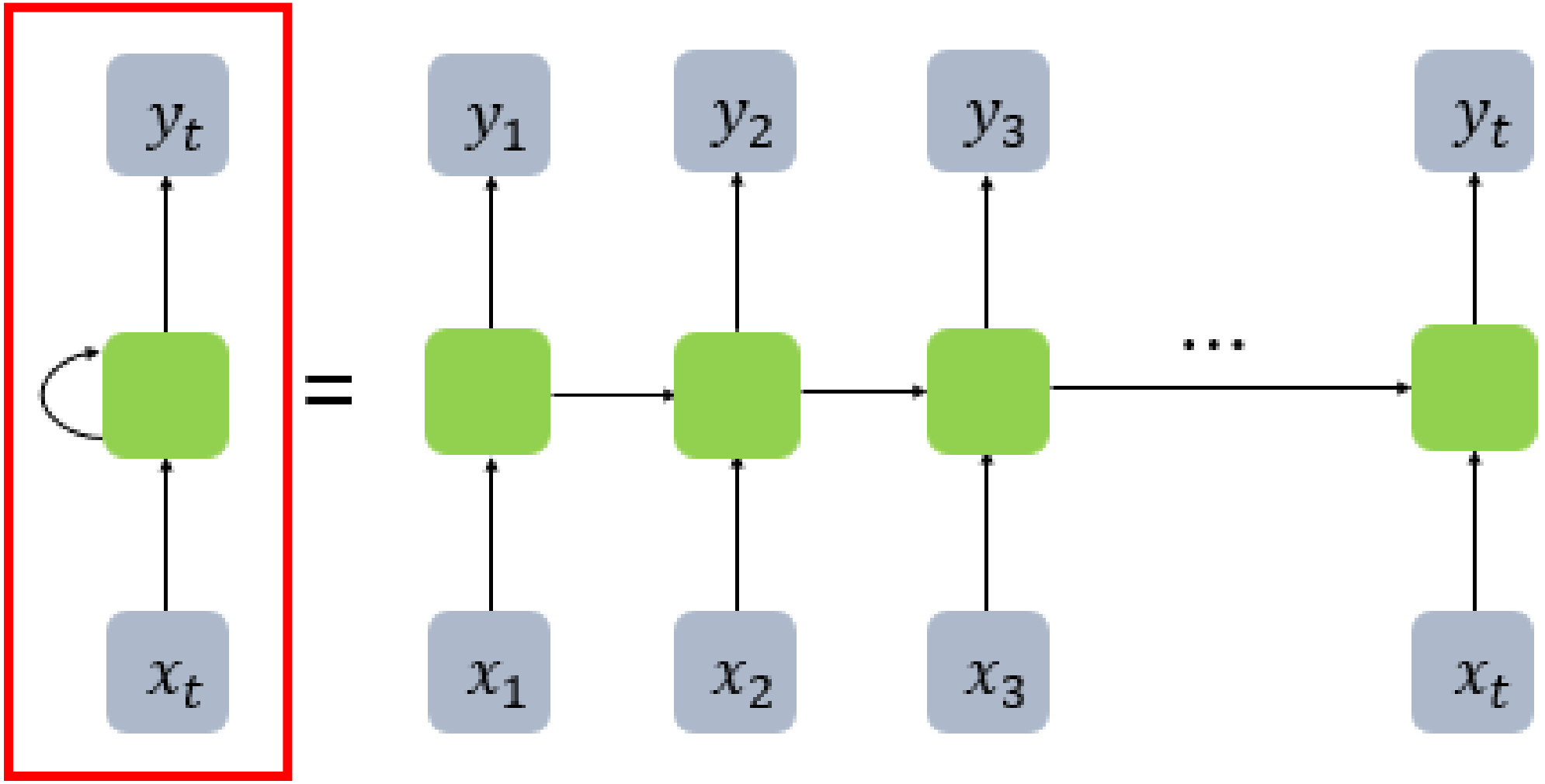


재귀적 학습: t-1번째 hidden layer가 t번째 hidden layer로 전달되는 학습.

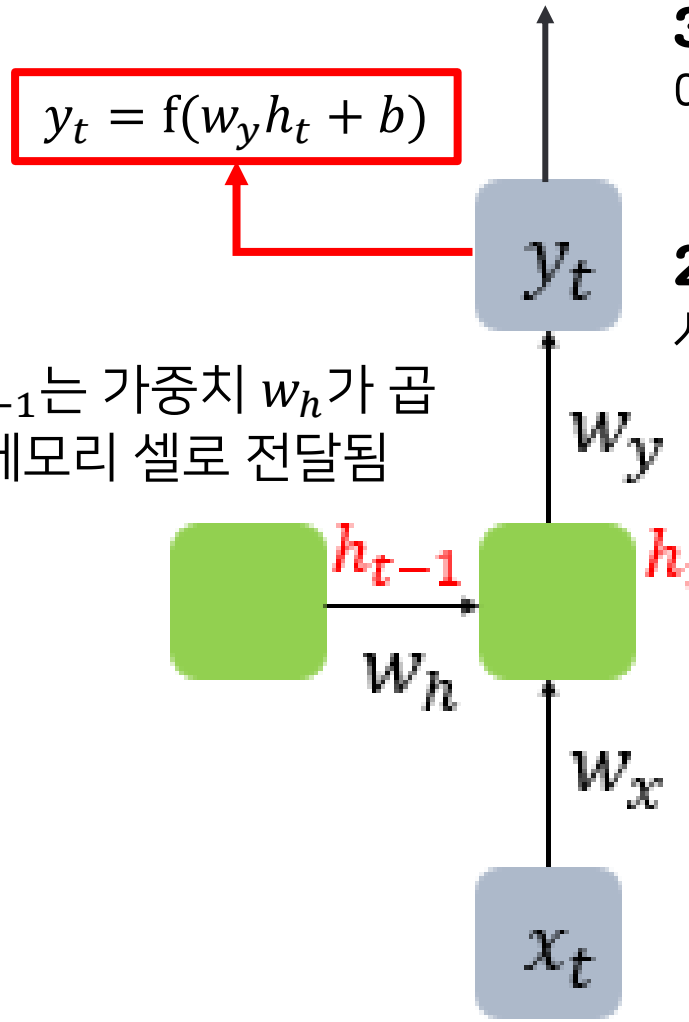
RNN은 해당 time step의 input+이전 time step hidden layer까지 포함해서 학습

메모리 셀: 이전 값을 기억하려는 특징을 가짐

RNN



RNN의 수식



3. y_t는 출력층 함수를 거쳐 예측값을 도출

2. h_t는 가중치 w_y가 곱해져서 출력층으로 전달됨

1-2. h_{t-1}는 가중치 w_h가 곱해져서 메모리 셀로 전달됨

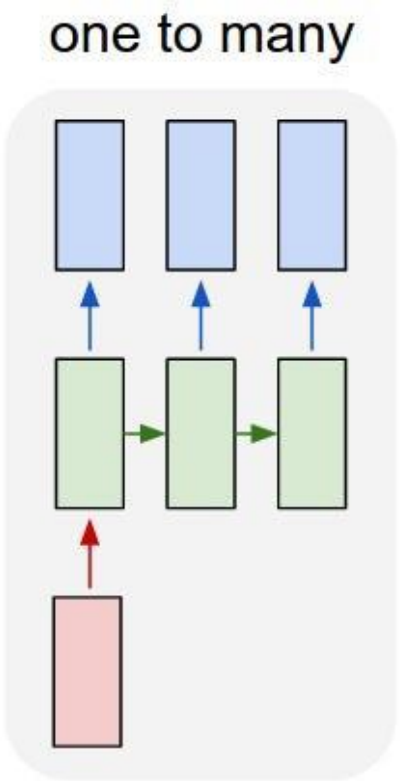
$$w_x x_t + w_h h_{t-1} + b$$
$$h_t = \tanh(w_x x_t + w_h h_{t-1} + b)$$

1-1. x_t는 가중치 w_x가 곱해져서 메모리 셀로 전달됨

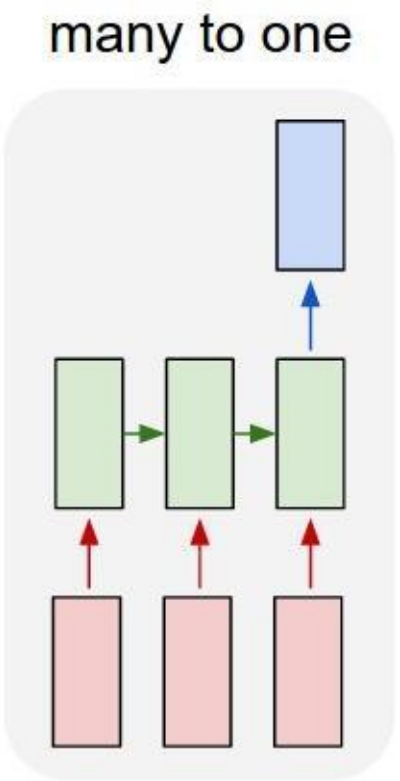
4. 도출된 예측값으로 loss를 구하고, 역전파를 통해 가중치 최적화

다양한 RNN 구조

RNN은 입력층과 출력층 길이를 다르게 설정하여 다양한 구조를 만들 수 있다.



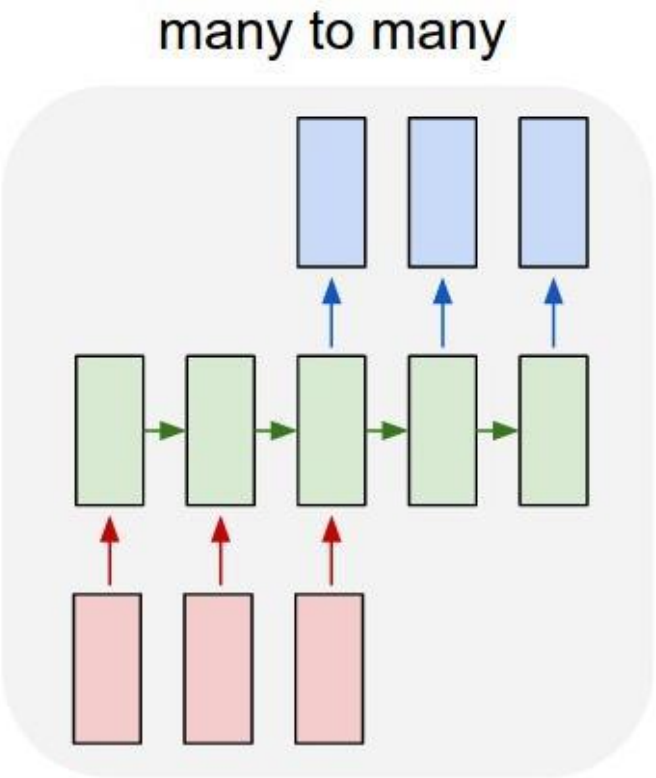
한 개의 input이 여러 개의 output(sequence)을 출력하는 구조.
image captioning 등에 사용할 수 있다.



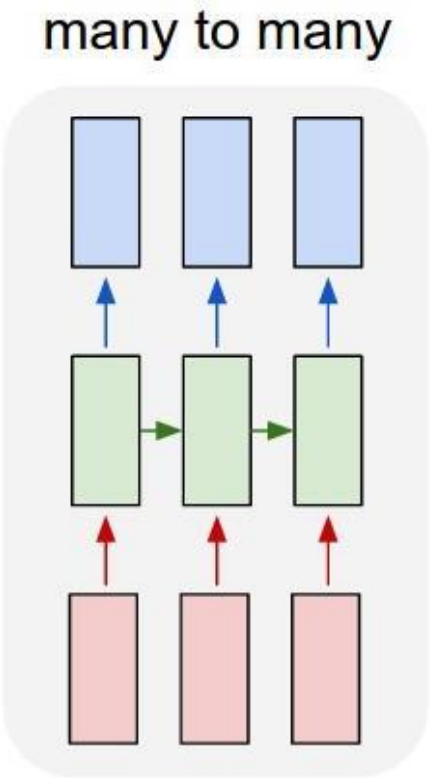
여러 개의 input(sequence)이 한 개의 output을 출력하는 구조.
감성 분석 등에 사용할 수 있다.

다양한 RNN 구조

RNN은 입력층과 출력층 길이를 다르게 설정하여 다양한 구조를 만들 수 있다.



여러 개의 input(sequence)이 여러 개의 output(sequence)을 출력하는 구조.
기계 번역 등에 사용할 수 있다.



many-to-many 모델 중에서도 각 input마다 output이 생성되는 모델.
출력이 지연되지 않기 때문에 실시간 처리가 필요한 작업에 사용할 수 있다.

수고하셨습니다!
다음 주에 마지막 내용으로 만나요~