

RNN

applekoong@naver.com 김정훈



- 003 numpy**
- 017 NLP**
- 044 Word Embedding**
- 086 사례정리**
- 098 nltk**
- 116 KoNLPy**
- 134 gensim**
- 137 RNN**
- 180 시계열**
- 191 Seq2Seq**



numpy
axis

3차원 배열

```
a = np.arange(32).reshape(4, 2, 4)  
print(a)
```

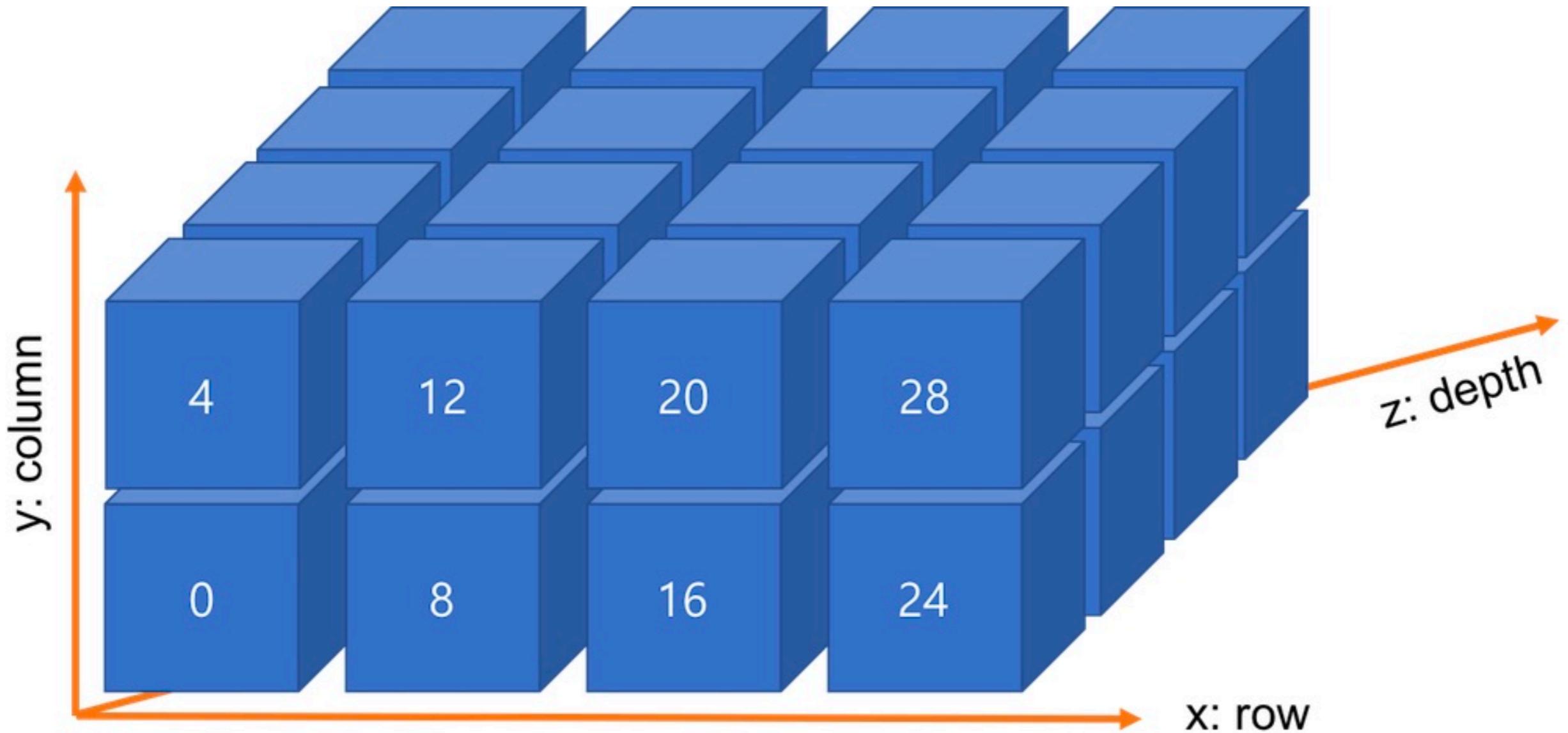
[출력 결과]

```
[[[ 0  1  2  3]  
 [ 4  5  6  7]  
  
 [[ 8  9 10 11]  
 [12 13 14 15]]  
  
 [[16 17 18 19]  
 [20 21 22 23]]  
  
 [[24 25 26 27]  
 [28 29 30 31]]]
```

3차원 배열

```
[  
  [ row: #1  
    [ depth: 1~4 [0 1 2 3], [ depth: 4~7 [4 5 6 7]  
  ], column: #1  
  [ row: #2  
    [ [8 9 10 11], [12 13 14 15]  
  ],  
  [ row: #3  
    [ [16 17 18 19], [20 21 22 23]  
  ],  
  [ row: #4  
    [ [24 25 26 27], [28 29 30 31]  
  ],  
]
```

3차원 배열



axis = 0

```
a = np.arange(32).reshape(4, 2, 4)  
s = np.sum(a, axis=0)  
print(s.shape)  
print(s)
```

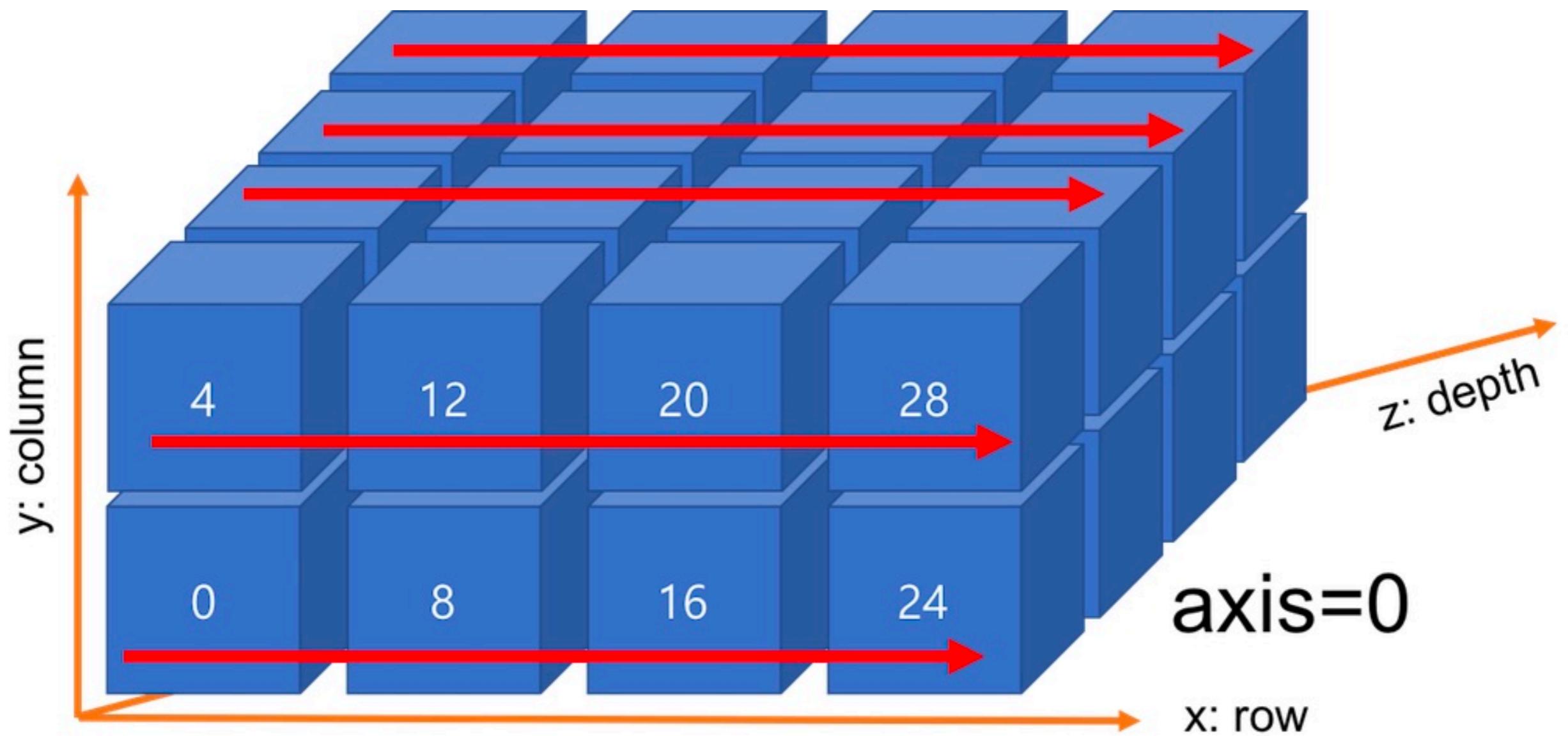
[출력결과]

(2, 4)

[[48 52 56 60]

[64 68 72 76]]

axis = 0



axis = 0

# row 구분 삭제	axis=0	shape=(2, 4)
[[0 1 2 3], [4 5 6 7]]	(4, 2, 4) => (2, 4)	[[48 52 56 60], [64 68 72 76]]
[[8 9 10 11], [12 13 14 15]]		
[[16 17 18 19], [20 21 22 23]]		
[[24 25 26 27], [28 29 30 31]]		

axis = 1

```
a = np.arange(32).reshape(4, 2, 4)
```

```
s = np.sum(a, axis=1)
```

```
print(s.shape)
```

```
print(s)
```

[출력결과]

(4, 4)

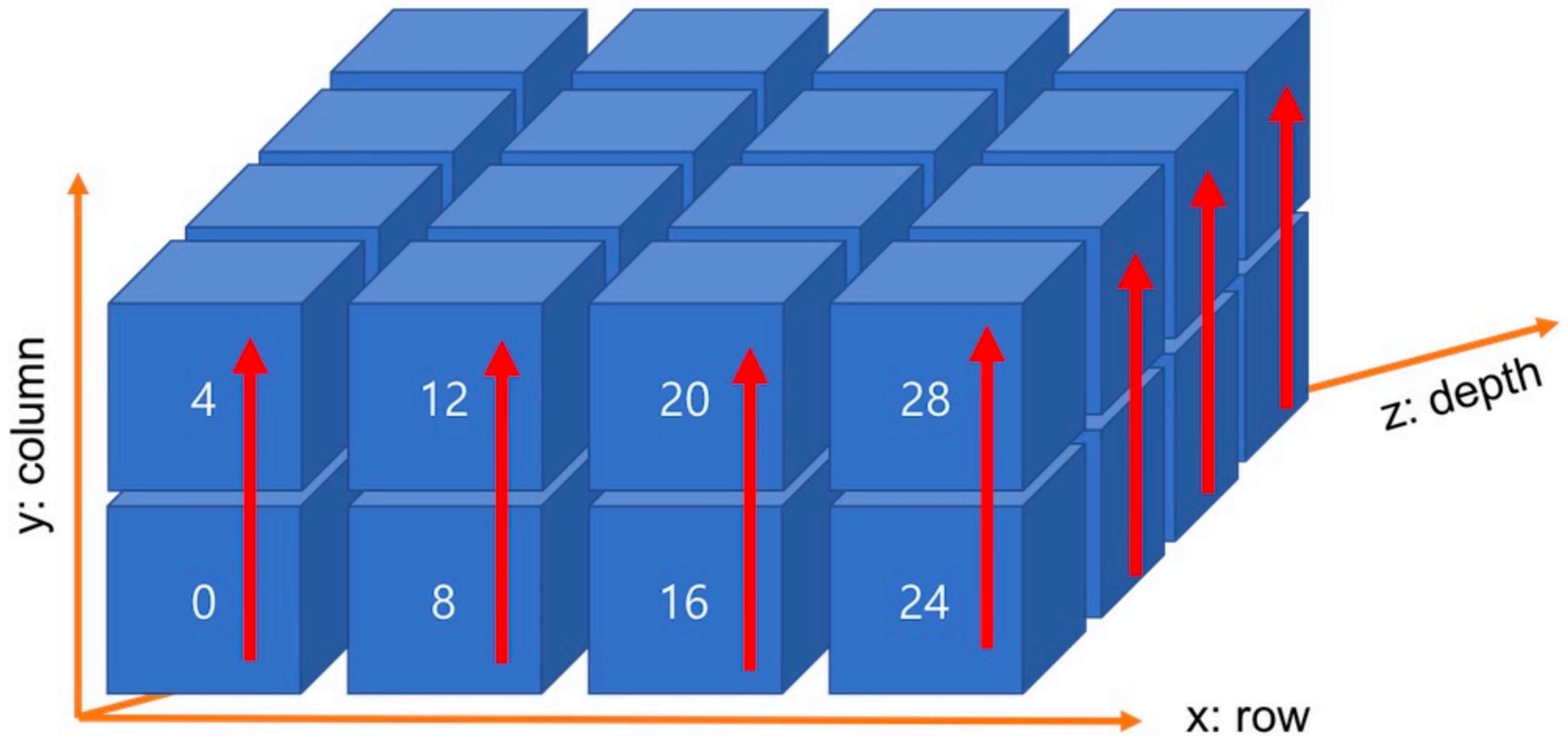
[[4 6 8 10]

[20 22 24 26]

[36 38 40 42]

[52 54 56 58]]

axis = 1



axis = 1

[# column 구분 삭제

[[0 1 2 3] , [4 5 6 7]]

[[8 9 10 11] , [12 13 14 15]]

[[16 17 18 19] , [20 21 22 23]]

[[24 25 26 27] , [28 29 30 31]]

axis=1

(4, 2, 4) => (4, 4)

[[4 6 8 10],
[20 22 24 26],
[36 38 40 42],
[52 54 56 58]]

shape=(4, 4)

axis = 2

```
a = np.arange(32).reshape(4, 2, 4)
```

```
s = np.sum(a, axis=2)
```

```
print(s.shape)
```

```
print(s)
```

[출력결과]

(4, 2)

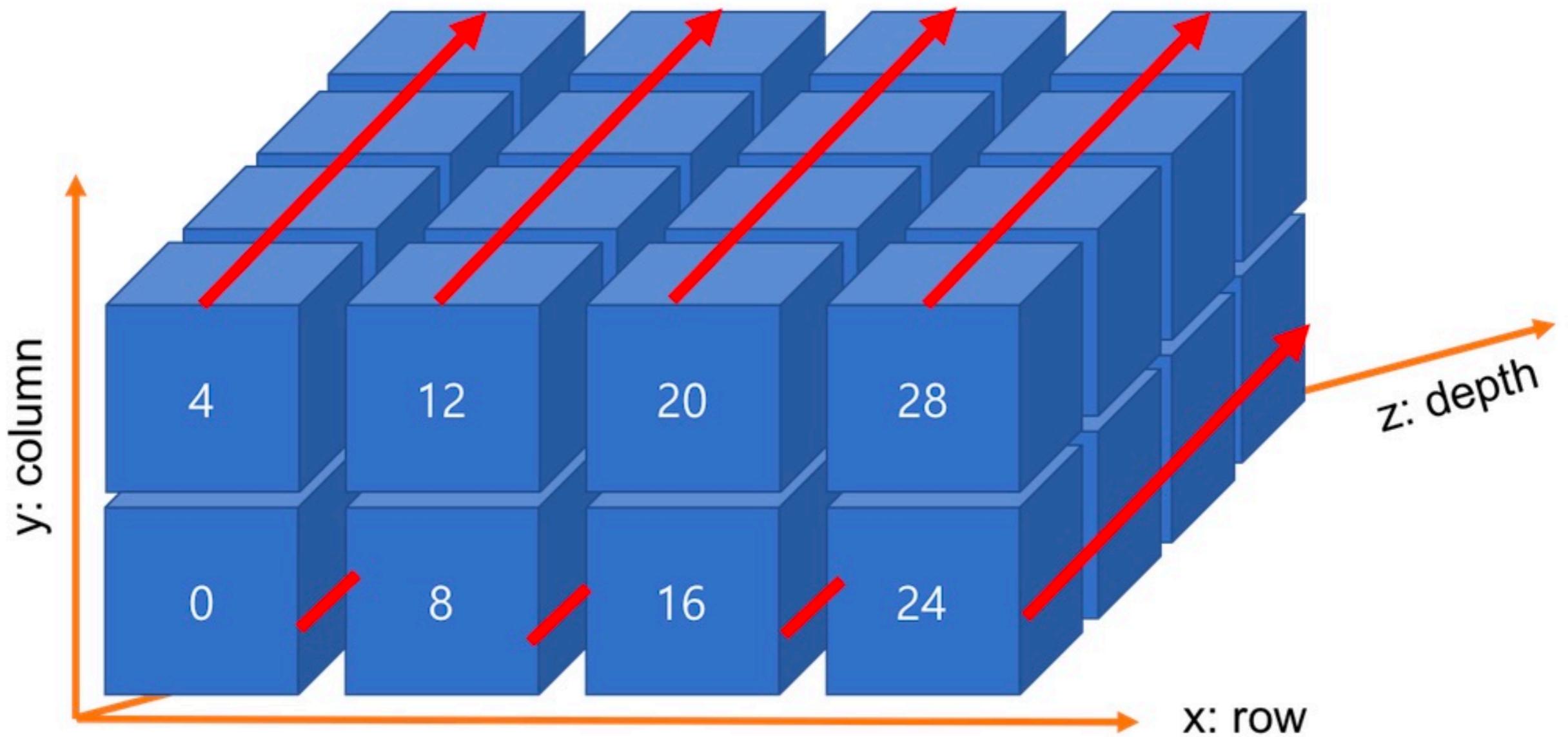
[[6 22]

[38 54]

[70 86]

[102 118]]

axis = 2



axis = 2

```
[ # depth 구분 삭제
[ [ 0 1 2 3 ] , [ 4 5 6 7 ] axis=2
], (4, 2, 4) => (4, 2)
[ [ 8 9 10 11 ] , [ 12 13 14 15 ] [
[ 6 22 ],
[ 28 54 ],
[ 70 86 ],
[ 102 118 ],
], ]
[ [ 16 17 18 19 ] , [ 20 21 22 23 ] ]
[ 24 25 26 27 ], [ 28 29 30 31 ]
]
]
```

정리

- axis=None은 기본값으로 모든 요소의 값을 합산하여 1개의 스칼라값 반환
- axis=0은 x축을 기준으로 여러 row를 한 개로 합치는 과정
- axis=1은 y축을 기준으로 row 별로 존재하는 column들의 값을 합쳐 1개로 축소하는 과정
- axis=2는 z축을 기준으로 column의 depth가 가진 값을 축소하는 과정



NLP

Natural language processing

자연어를 이해하는 방법

- 단어가 어떤 의미를 가지고 있는지를 알기 위해서는 문장의 컨텍스트(Context)를 이해해야 한다.
 - 사람은 Apple이 사과인지 회사인지 문맥을 통해 바로 구분할 수 있지만 기계는 할 수 없다.
 - '메시'로부터 '축구'와 '호나우도'처럼 단어간의 의미적 관계도 파악할 수 있지만 기계는 어렵다.
- 그렇기 때문에 단어를 수치적으로 표현하여 기계가 이해할 수 있도록 해야 한다.
- 이를 Word Embedding이라고 표현하는데, 간단히 말해서 텍스트를 숫자로 표현하는 것을 말하고, 같은 문자라고 하더라도 다른 수치로 표현할 수 있다.
- 단어를 사전을 사용하여 매핑하고 이를 벡터로 만드는 것이다.
ex) "the cat sat on the mat"
- [the, cat, sat, on, mat]의 단어별로 분류하고 이를 One hot encoding을 통해 0 또는 1로 나타낸다.
- 예를 들어 'cat'은 [0, 1, 0, 0, 0, 0]이며, 'the'의 경우 [1, 0, 0, 0, 0]으로 나타낸다.
- 벡터화를 시켜 숫자로 표현하면 분류나 회귀 등의 다양한 분석이 가능해진다.

문맥(CONTEXT)

- 유유상종

친구를 보면 그 사람을 안다.

- 단어의 주변을 보면 그 단어를 안다.

You shall know a word by the company it keeps.

-- J.R. Firth (1957)

예시 1

- 빈 칸에 맞는 말을 찾으세요.
 - 새로운 디자인의 선이 _____하다.
 - 야, 부엌 좀 _____히 하자.
 - 깜박하고 책상 위를 _____하게 치우지 않았다.

예시 2

- '매나니'라는 단어를 아시나요?
 - 그 녀석 무슨 배짱인지 '매나니'로 와서 일을 하겠다고 한다.
 - 삽이라도 있어야 땅을 파지 '매나니'로야 어떻게 하겠나?
 - SO만 있으면 코딩은 '매나니'로도 할 수 있다고 한다.

매나니

- 순 우리말(다음 사전)

일하는 데 아무런 도구나 연장이 없는 맨손

뜻/문법

고려대 ✓ 우리말샘

명사

(1) 일하는 데 아무런 도구나 연장이 없는 맨손.

| 그 녀석 무슨 배짱인지 매나니로 와서 일을 하겠다고 한다.

(2) 아무런 반찬이 없는 맨밥.

| 그는 입맛을 잃었다며 매나니로 요리를 하였다.

문맥

다시 말해, 단어의 의미는 해당 단어의 **문맥(context)**이 담고 있다.

영희가 철수에게 미안하다고 사과하면서

나무에서 갓 딴 맛있는 사과를 주었습니다

주변부 단어들이 “사과”的 문맥적 특성을 나타냄

- 문맥(context) := 정해진 구간(window) 또는 문장/문서 내의 단어들
 - 보통 영어의 구간은 좌우 1-8개 단어, 총 3-17개 단어의 문맥을 본다*
- 두 단어의 문맥이 비슷하면 "의미적"으로 유사한 단어
 - ex: "PyCon"은 "R"보다 "Python"에 가깝다?

언어 모델링

언어 모델은 주어진 문장에서 이전 단어들을 보고 다음 단어가 나올 확률을 계산해주는 모델이다. 언어 모델은 어떤 문장이 실제로 존재할 확률이 얼마나 되는지 계산해주기 때문에, 자동 번역의 출력값으로 어떤 문장을 내보내는 것이 더 좋은지 (실생활에서 높은 확률로 존재하는 문장들은 보통 문법적/의미적으로 올바르기 때문) 알려줄 수 있다.

문장에서 다음 단어가 나타날 확률을 계산해주는 주 목적 외의 부수적인 효과로 생성 (generative) 모델을 얻을 수 있는데, 출력 확률 분포에서 샘플링을 통해 문장의 다음 단어가 무엇이 되면 좋을지 정한다면 기존에 없던 새로운 문장을 생성할 수 있다. 또한, 학습 데이터에 따라 다양하고 재밌는 여러 가지를 만들어낼 수도 있다.

언어 모델에서의 입력값은 단어들의 시퀀스 (e.g. one-hot encoded 벡터 시퀀스)이고, 출력은 추측된 단어들의 시퀀스이다. 네트워크를 학습할 때에는 시간 스텝 t 에서의 출력값이 실제로 다음 입력 단어가 되도록 $o_t = x_{\{t+1\}}$ 로 정해준다.

언어 모델링

언어 모델(Language Model)은 짧게 요약하면 문장. 즉, 단어의 시퀀스의 확률을 예측하는 모델입니다. 언어 모델은 자연어 생성(Natural Language Generation)의 기반이 됩니다. 기본적으로 자연어 생성과 관련된 건 모두 언어 모델과 관련이 있습니다. 음성 인식, 기계 번역, OCR, 검색어 자동 완성 등과 같은 것은 전부 언어 모델을 통해 이루어 집니다.

언어 모델을 만드는 방법은 크게 통계를 이용한 방법과 인공 신경망을 이용한 방법으로 구분할 수 있습니다. 최근 자연어 처리에서 언어 모델에 대한 이야기를 빼놓을 수가 없는데, 최근 핫한 딥 러닝 자연어 처리의 신기술인 GPT나 BERT가 전부 언어 모델의 개념을 사용하여 만들어졌기 때문입니다.

인공 신경망의 일종인 RNN, LSTM 등을 통해 언어 모델을 만들 수 있고, seq2seq를 사용하면 기계 번역을 할 수도 있습니다.

언어 모델링 = 확률

- 기계 번역(Machine Translation):

$P(\text{나는 버스를 탔다}) > P(\text{나는 버스를 태운다})$

: 언어 모델은 두 문장을 비교하여 좌측의 문장의 확률이 더 높다고 판단합니다.

- 오타 교정(Spell Correction)

선생님이 교실로 부리나케

$P(\text{달려갔다}) > P(\text{잘려갔다})$

: 언어 모델은 두 문장을 비교하여 좌측의 문장의 확률이 더 높다고 판단합니다.

- 음성 인식(Speech Recognition)

$P(\text{나는 메롱을 먹는다}) < P(\text{나는 메론을 먹는다})$

: 언어 모델은 두 문장을 비교하여 우측의 문장의 확률이 더 높다고 판단합니다.

언어 모델링 = 예측

- 문장의 확률 예측

하나의 단어(word)를 w 라고 합시다. 단어의 시퀀스인 전체 문장(sentence)을 대문자 W 라고 합시다.

n 개의 w 로 구성된 문장 W 의 확률은 다음과 같이 표현할 수 있습니다.

$$P(W)=P(w_1, w_2, w_3, w_4, w_5, \dots, w_n)=\prod_{n=1}^n P(w_n)$$

- 다음 단어 예측

$n-1$ 개의 단어가 나열된 상태에서 n 번째 단어의 확률을 다음과 같이 표현할 수 있습니다.

$$P(w_n|w_1, \dots, w_{n-1}) \quad # | : 조건부 확률(conditional probability)$$

예를 들어 다섯번째 단어의 확률은 $P(w_5|w_1, w_2, w_3, w_4)$ 와 같습니다.

문장 즉, 전체 단어 시퀀스의 확률은 모든 단어가 예측되고 나서야 알 수 있으므로 결국 전체 단어 시퀀스의 예측은 다음과 같습니다.

$$P(W)=P(w_1, w_2, w_3, w_4, w_5, \dots, w_n)=\prod_{n=1}^n P(w_n|w_1, \dots, w_{n-1})$$

언어 모델

- Statistical Language Model
 - Conditional probability, count-based
- N-gram Language Model
 - 1-gram, 2-gram, 3-gram etc
- Neural Language Model
 - cbow, skip-gram
- Recurrent Language Model
 - lstm, gru

한국어 언어 모델

- 어순이 중요하지 않다.
 - 나는 운동을 합니다 체육관에서.
 - 나는 체육관에서 운동을 합니다.
 - 체육관에서 운동을 합니다.
 - 나는 운동을 체육관에서 합니다.
- 교착어
 - 띄어쓰기 단위로 토큰화를 할 경우에는 문장에서 발생가능한 단어의 수가 굉장히 늘어납니다. 대표적인 예로 교착어인 한국어에는 조사가 있습니다. 영어는 기본적으로 조사가 없습니다. 하지만 한국어에는 어떤 행동을 하는 동사의 주어나 목적어를 위해서 조사라는 것이 있습니다.
 - 가령 '그녀'라는 단어 하나만 해도 그녀가, 그녀를, 그녀의, 그녀와, 그녀로, 그녀께서, 그녀처럼 등과 같이 다양한 경우가 존재합니다. 그렇기 때문에, 한국어에서는 토큰화를 통해 접사나 조사 등을 분리하는 것은 중요한 작업이 되기도 합니다.
- 띄어쓰기가 지켜지지 않는다.
 - 띄어쓰기를 제대로 하지 않아도 의미가 전달되며, 띄어쓰기 규칙 또한 상대적으로 까다로운 언어이기 때문에 자연어 처리를 하는 것에 있어서 한국어 코퍼스는 띄어쓰기가 제대로 지켜지지 않는 경우가 많습니다. 이 경우에 토큰이 제대로 분리 되지 않아 언어 모델은 제대로 동작하지 않습니다.

Conditional Probability

- 조건부 확률은 두 확률 $P(A), P(B)$ 에 대해서 아래와 같은 관계를 갖습니다.

$$p(B|A) = P(A, B) / P(A)$$

$$P(A, B) = P(A)P(B|A)$$

- 4개의 확률이 조건부 확률의 관계를 가질 때, 아래와 같이 표현할 수 있습니다.

$$P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$$

- 조건부 확률의 연쇄 법칙(chain rule) 일반화

$$P(x_1, x_2, x_3 \dots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1 \dots x_{n-1})$$

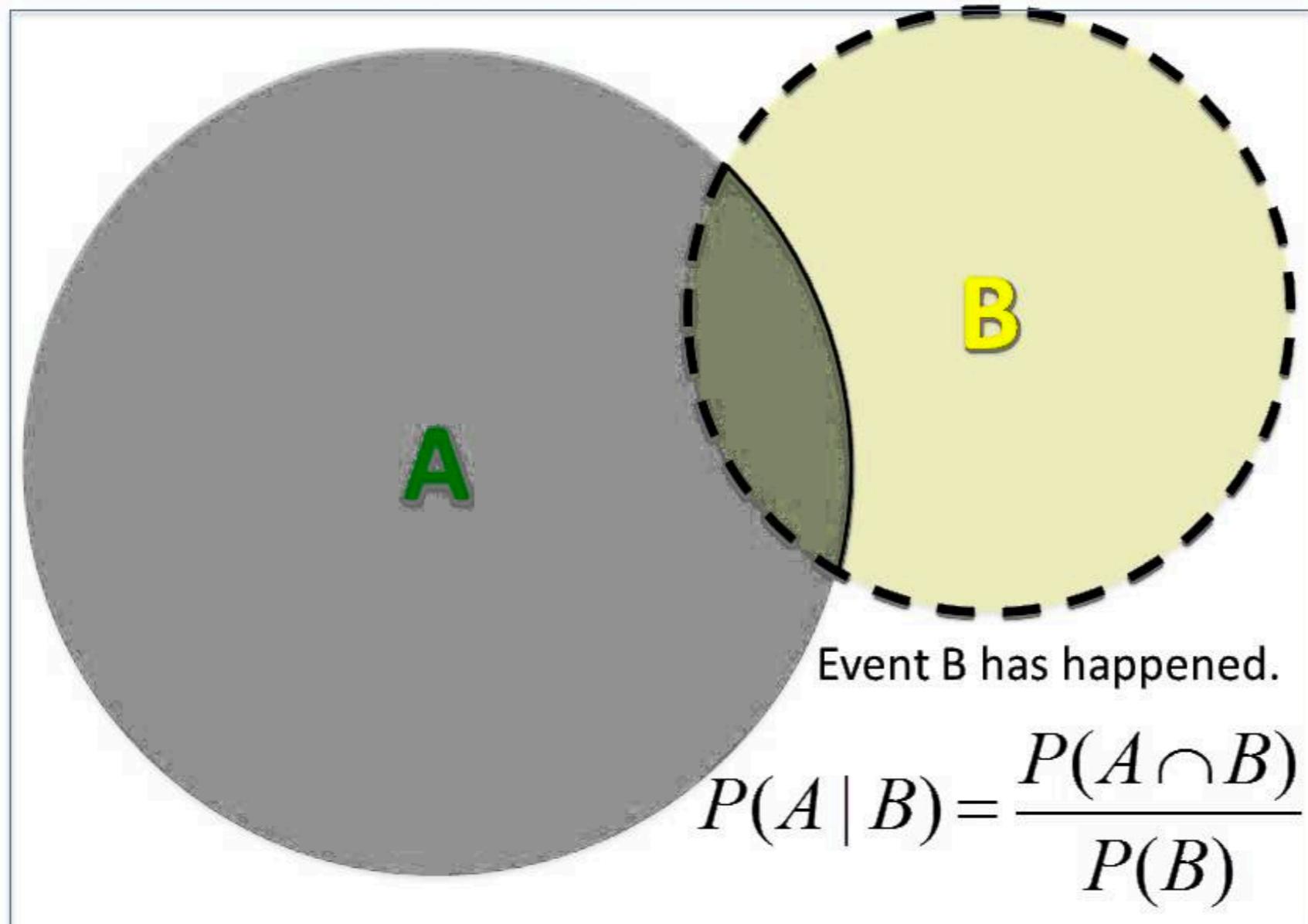
조건부 확률

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Likelihood **Class Prior Probability**
↓ ↑
Posterior Probability **Predictor Prior Probability**

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

조건부 확률



Given that B has happened, the probability that A will happen, too, is just the area ratio of the banana-shaped region to the B circle.

조건부 확률

확률, probability

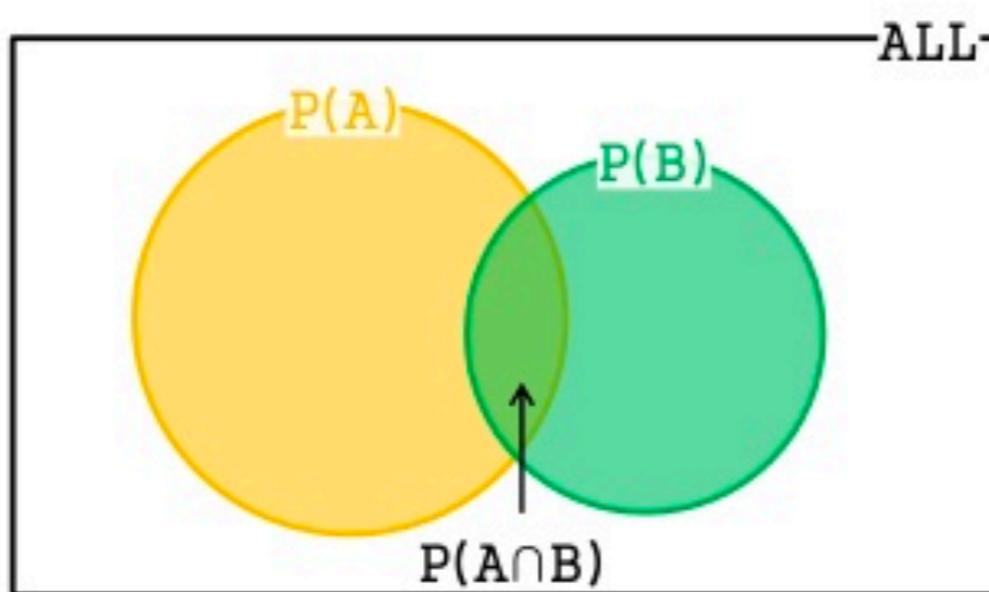
확률, probability

P(A), A가 일어날 확률 (probability of A)

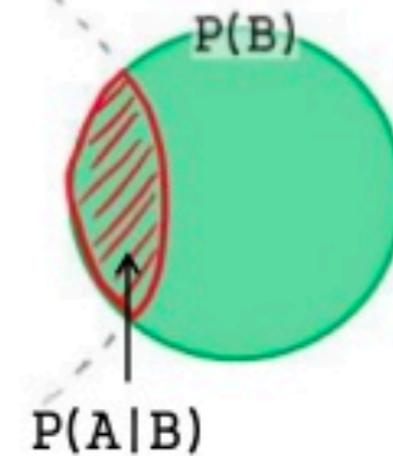
조건부 확률, conditional probability

P(A|B), B가 주어졌을 때, A가 일어날 확률 (probability of A given B)

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

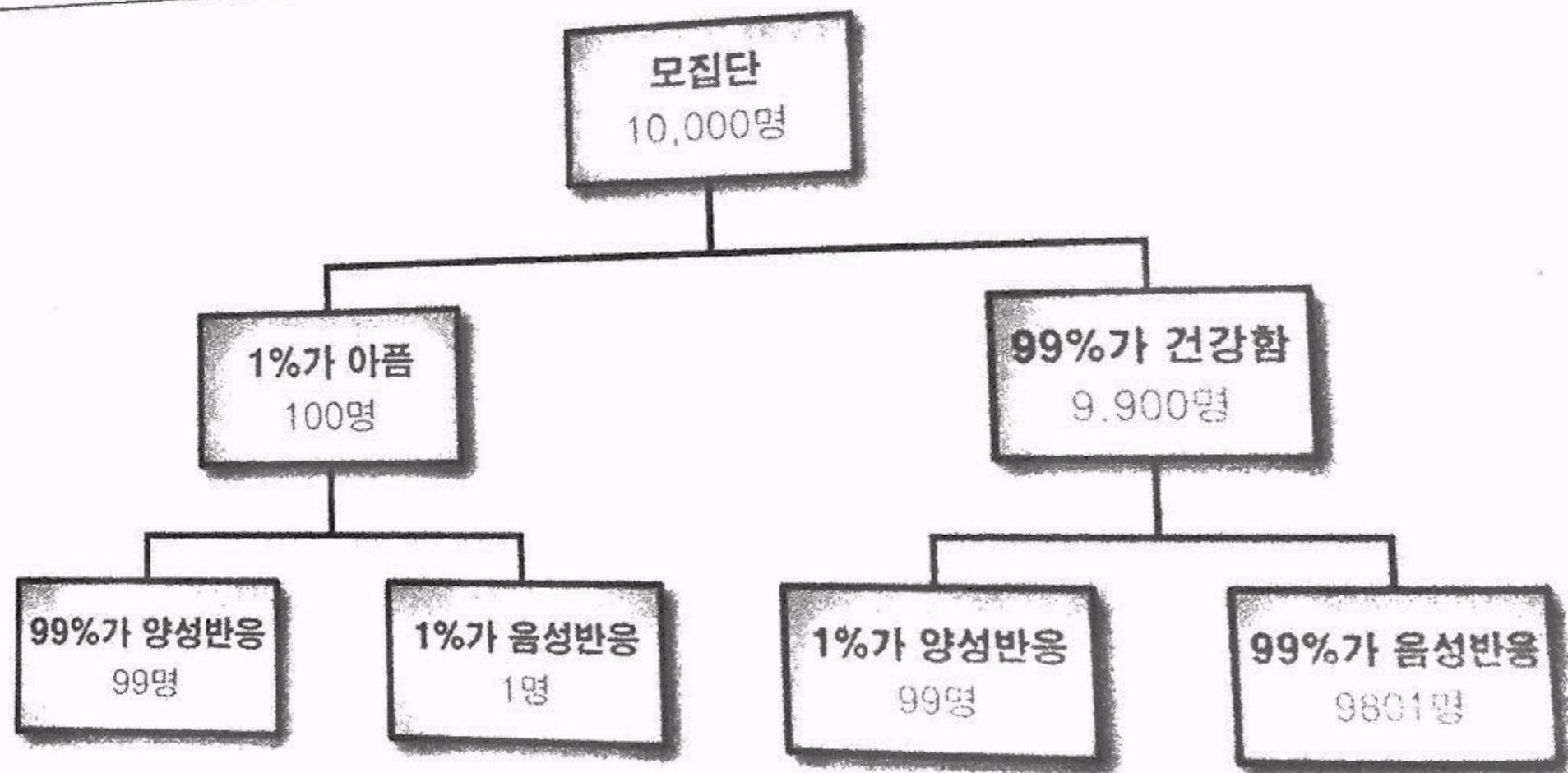


$P(B)$ 가 주어졌을 때, ($P(B)$ 의 확률이 1이라 보고)
붉은 부분의 확률이 $P(A|B)$ 이다



따라서 키가 180 cm일 때 남자일 확률은, $P(A|B)$
 $0.1 / 0.3 = 0.33$, 즉 33.3%

어떤 사람이 양성을 보였다면, 이 사람이 정말로 병에 걸렸을 확률은?



예제 1

질병 A의 발병률은 0.1%로 알려져있다.
이 질병이 실제로 있을 때 질병이 있다고 검진할 확률(민감도)은 99%,
질병이 없을 때 없다고 실제로 질병이 없다고 검진할 확률(특이도)은
98%라고 하자.

만약 어떤 사람이 질병에 걸렸다고 검진받았을 때,
이 사람이 정말로 질병에 걸렸을 확률은?

정답 0.047

예제 2

제약사에서 환자가 특정한 병에 걸린지 확인하는 시약을 만들었다.

그 병에 걸린 환자에게 시약을 테스트한 결과

99%의 확률로 양성 반응을 보였다.

병에 걸렸는지 확인이 되지 않은 어떤 환자가

이 시약을 테스트한 결과 양성 반응을 보였다면

이 환자가 그 병에 걸려 있을 확률은 얼마인가?

- 이 병은 전체 인구 중 걸린 사람이 0.2%인 희귀병이다.
- 이 병에 걸리지 않은 사람에게 시약 검사를 했을 때, 양성 반응, 즉 잘못된 결과(False Positive)가 나타난 확률이 5%다.

정답 0.038

Count-Based

- An adorable little boy가 나왔을 때,
is가 나올 확률인 $P(\text{is}|\text{An adorable little boy})$ 를 계산한다고 해봅시다.
- 그 확률은 다음과 같습니다.
예를 들어 기계가 학습한 코퍼스 데이터에서
An adorable little boy가 100번 등장했는데 그 다음에 is가 등장한 경우는 30번이
라고 합시다.
이때 기계에게 $P(\text{is}|\text{An adorable little boy})$ 를 묻는다면 30%가 됩니다.

$$P(\text{is}|\text{An adorable little boy}) = \frac{\text{count}(\text{An adorable little boy is})}{\text{count}(\text{An adorable little boy })}$$

n-gram

- n-gram은 n개의 연속적인 단어 나열을 의미합니다.
갖고 있는 코퍼스에서 n개의 단어 뭉치 단위로 끊어서 이를 하나의 토큰으로 간주합니다.
예를 들어서 문장 An adorable little boy is spreading smiles이 있을 때,
각 n에 대해서 n-gram을 전부 구해보면 다음과 같습니다.
- unigrams : an, adorable, little, boy, is, spreading, smiles
- bigrams : an adorable, adorable little, little boy, boy is, is spreading,
spreading smiles
- trigrams : an adorable little, adorable little boy, little boy is, boy is spreading,
is spreading smiles
- 4-grams : an adorable little boy, adorable little boy is, little boy is spreading,
boy is spreading smiles

n-gram

- n-gram을 사용할 때는 n이 1일 때는 유니그램(unigram), 2일 때는 바이그램(bigram), 3일 때는 트라이그램(trigram)이라고 명명하고 n이 4 이상일 때는 gram 앞에 그대로 숫자를 붙여서 명명합니다. 출처에 따라서는 유니그램, 바이그램, 트라이그램 또한 각각 1-gram, 2-gram, 3-gram이라고 하기도 합니다.
- n-gram을 통한 언어 모델에서는 다음에 나올 단어의 예측은 오직 n-1개의 단어에만 의존합니다. 예를 들어 'An adorable little boy is spreading' 다음에 나올 단어를 예측하고 싶다고 할 때, n=4라고 한 4-gram을 이용한 언어 모델을 사용한다고 합시다. 이 경우, spreading 다음에 올 단어를 예측하는 것은 n-1에 해당되는 앞의 3개의 단어만을 고려합니다.

~~An adorable little~~ boy is spreading ?

무시됨!

n-1개의 단어

n-gram

- 만약 갖고있는 코퍼스에서 boy is spreading가 1,000번 등장했다고 합시다. 그리고 boy is spreading insults가 500번 등장했으며, boy is spreading smiles가 200번 등장했다고 합시다. 그렇게 되면 boy is spreading 다음에 insults가 등장할 확률은 50%이며, smiles가 등장할 확률은 20%입니다. 확률적 선택에 따라 우리는 insults가 더 맞다고 판단하게 됩니다.
- $P(\text{insults}|\text{boy is spreading})=0.500$
 $P(\text{smiles}|\text{boy is spreading})=0.200$

$$P(w|\text{boy is spreading}) = \frac{\text{count}(\text{boy is spreading } w)}{\text{count}(\text{boy is spreading})}$$

Perplexity : 언어 모델 평가

- Perplexity는 언어 모델을 평가하기 위한 내부 평가 지표입니다. 보통 줄여서 PPL이라고 표현합니다. 왜 perplexity라는 용어를 사용했을까요? 영어에서 'perplexed'는 '헷갈리는'과 유사한 의미를 가집니다. 그러니까 여기서 PPL은 '헷갈리는 정도'로 이해해 봅시다. PPL를 처음 배울때 다소 낯설게 느껴질 수 있는 점이 있다면, PPL은 수치가 높을수록 좋은 성능을 의미하는 것이 아니라, '낮을 수록' 언어 모델의 성능이 좋다는 것을 의미한다는 점입니다.
- PPL은 단어의 수로 정규화(normalization) 된 테스트 데이터에 대한 확률의 역수입니다. PPL을 최소화한다는 것은 문장의 확률을 최대화하는 것과 같습니다.

Perplexity : 언어 모델 평가

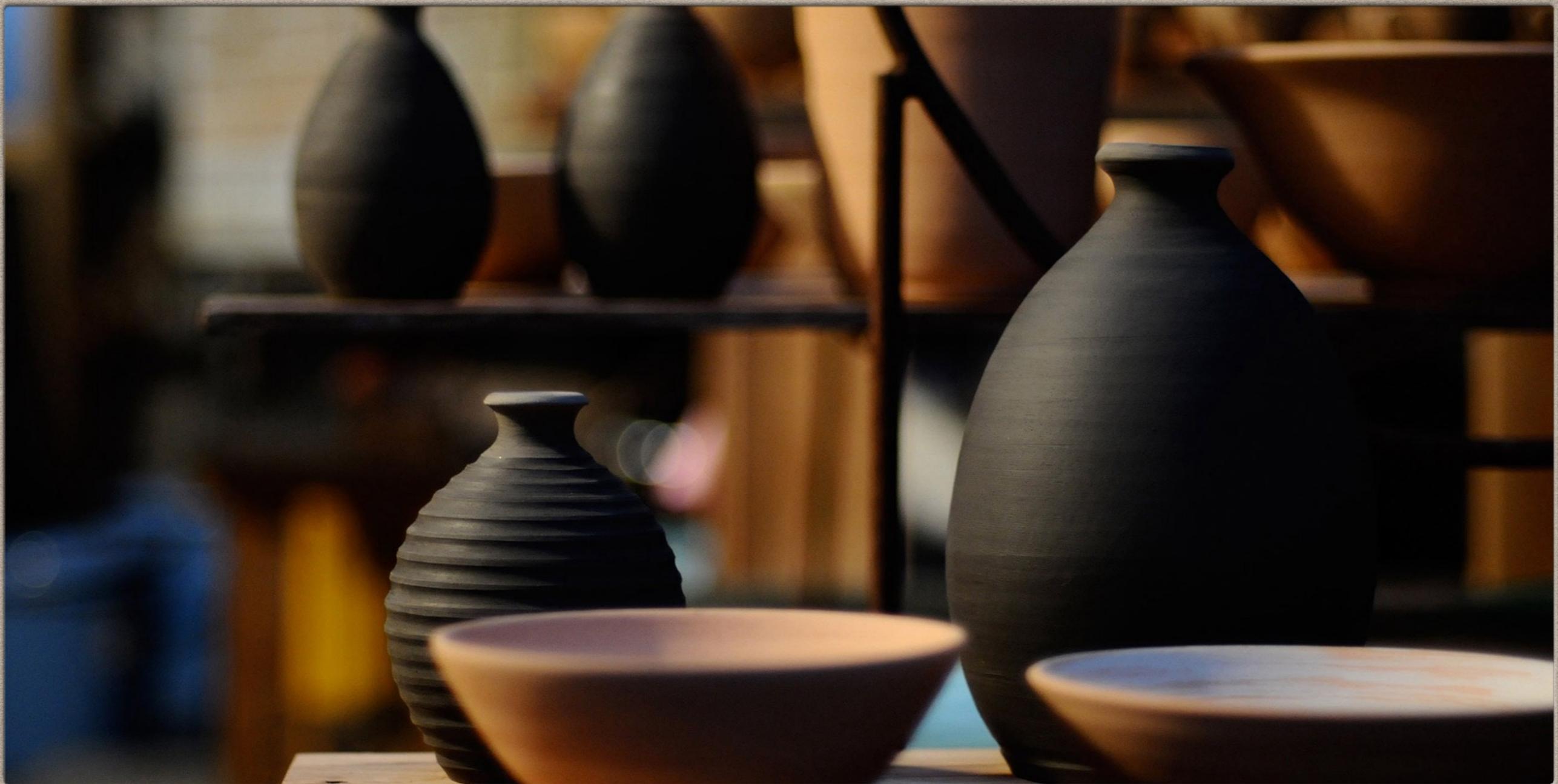
- PPL은 선택할 수 있는 가능한 경우의 수를 의미하는 분기계수(branching factor)입니다. PPL은 이 언어 모델이 특정 시점에서 평균적으로 몇 개의 선택지를 가지고 고민하고 있는지를 의미합니다. 가령, 언어 모델에 어떤 테스트 데이터를 주고 측정했더니 PPL이 10이 나왔다고 해봅시다. 그렇다면 해당 언어 모델은 테스트 데이터에 대해서 다음 단어를 예측하는 모든 시점(time-step)마다 평균적으로 10개의 단어를 가지고 어떤 것이 정답인지 고민하고 있다고 볼 수 있습니다. 같은 테스트 데이터에 대해서 두 언어 모델의 PPL을 각각 계산 후에 PPL의 값을 비교하면, 두 언어 모델 중 어떤 것이 성능이 좋은지도 판단이 가능합니다. 당연히 PPL이 더 낮은 언어 모델의 성능이 더 좋다고 볼 수 있습니다.
- 단, 평가 방법에 있어서 주의할 점은 PPL의 값이 낮다는 것은 테스트 데이터 상에서 높은 정확도를 보인다는 것인지, 사람이 직접 느끼기에 좋은 언어 모델이라는 것을 반드시 의미하진 않는다는 점입니다. 또한 언어 모델의 PPL은 테스트 데이터에 의존하므로 두 개 이상의 언어 모델을 비교할 때는 정량적으로 양이 많고, 또한 도메인에 알맞은 동일한 테스트 데이터를 사용해야 신뢰도가 높다는 것입니다.

$$PPL(W) = P(w_1, w_2, w_3, \dots, w_N)^{-\frac{1}{N}} = \left(\frac{1}{10}\right)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

Perplexity : 언어 모델 평가

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

페이스북 AI 연구팀 발표자료

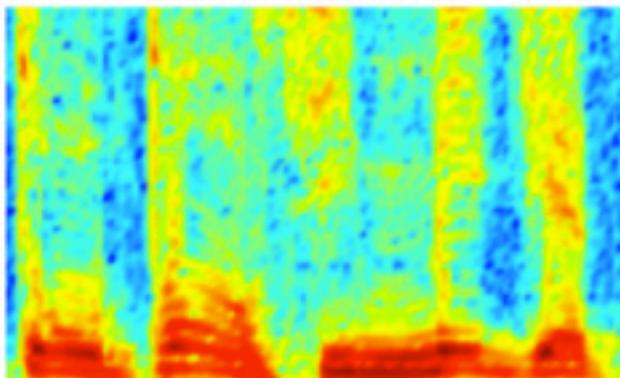


WORD EMBEDDING

Word2Vec, GloVe, FastText

DENSE & SPARSE

AUDIO



Audio Spectrogram

DENSE

IMAGES

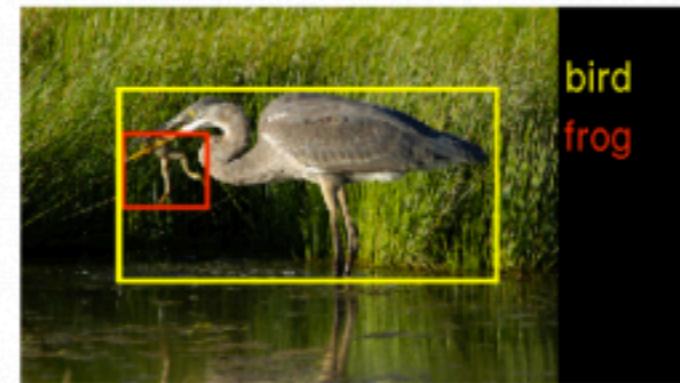


Image pixels

DENSE

TEXT

0	0	0	0.2	0	0.7	0	0	0
---	---	---	-----	---	-----	---	---	---	-----	-----

Word, context, or
document vectors

SPARSE

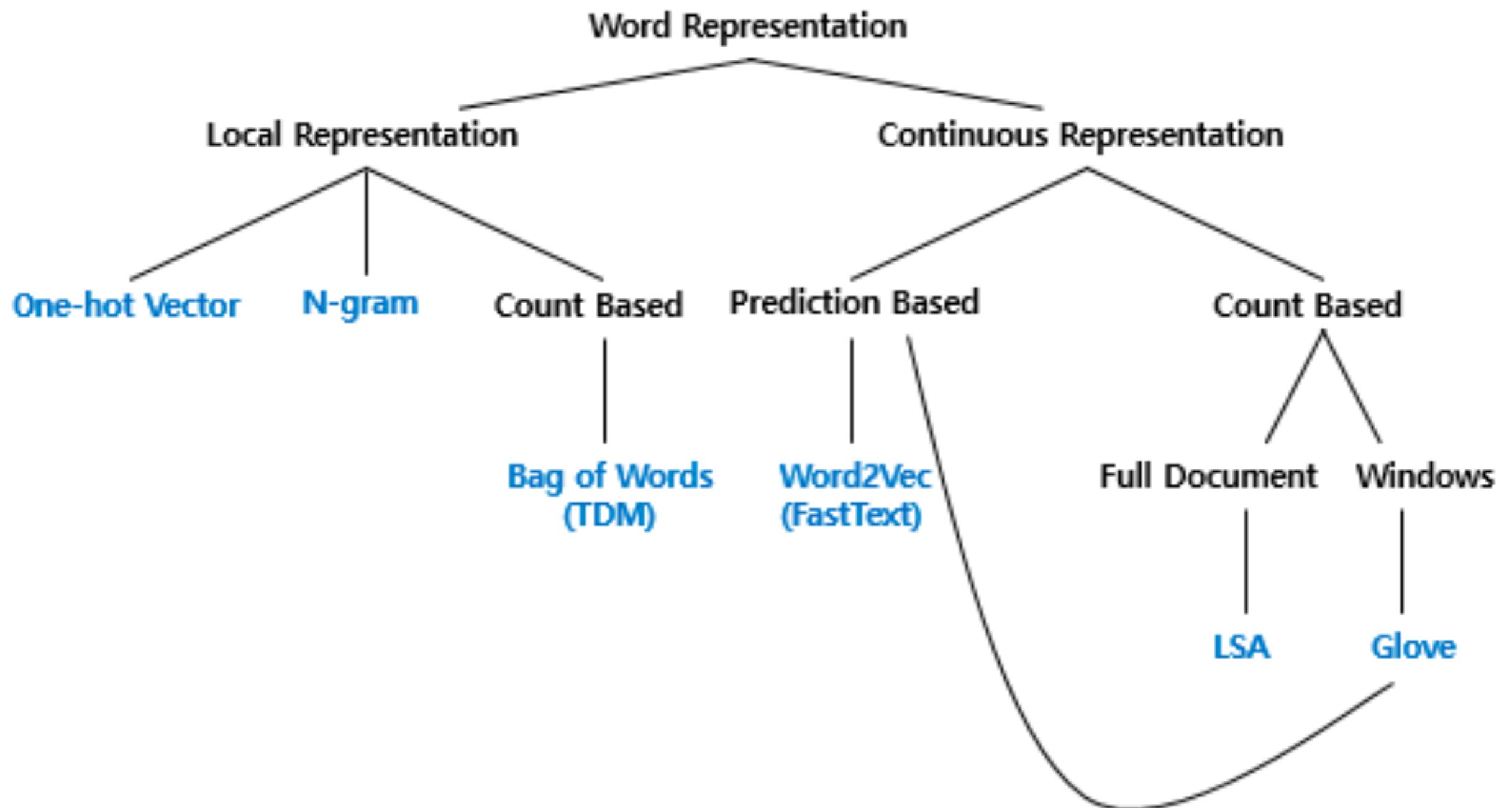
VECTOR SPACE MODEL

- VSM은 자연어 처리(NLP)에서 오랜기간 사용되어 왔다. 이 방법은 같은 context에 있는 단어는 같은 semantic meaning을 공유한다고 가정한다. 이런 가정을 Distributional Hypothesis라고 한다.
- Distributional Hypothesis
 1. 비슷한 분포를 가진 단어들은 비슷한 의미를 가진다.
 2. 비슷한 분포를 가졌다는 것은 기본적으로 단어들이 같은 문맥에서 등장한다는 것을 의미
- 종류
 1. count-based methods (e.g. Latent Semantic Analysis)
어떤 단어가 이웃 단어들과 같이 등장한 횟수를 계산하고 이 통계를 small, dense vector로 맵핑.
 2. predictive methods (e.g. neural probabilistic language models)
small, dense embedding vectors로 표현된 이웃 단어들을 이용해서 직접적으로 단어를 예측.

WORD EMBEDDINGS

- 오디오와 이미지와 같은 고차원의 데이터는 raw data로부터 인코딩(encoding)된 벡터로 특징을 뽑아내는 것이 쉽다.
- 텍스트의 경우, 이런식의 처리가 어렵기 때문에 전통적인 방법에서는 하나의 단어를 discrete atomic symbol로 표현
- 예시
cat = Id537, dog = Id143
- vector representation을 통해 위의 아래와 같은 두 가지 문제점을 해결할 수 있다.
 - 1.인코딩이 무작위적이고 데이터간의 관계를 보여주지 않는다. 따라서 모델이 “cats”라는 단어로부터 배운 특징을 “dogs”라는 단어를 처리할 때 적절하게 이용할 수 없다.
 - 2.단어들을 discrete ID로 맵핑함으로써 데이터를 sparse하게 만든다. 따라서 통계적 모델을 성공적으로 트레이닝 하려면 많은 데이터가 필요하다.

WORD REPRESENTATION



BAG OF WORDS

- Bag of Words란 단어들의 순서는 전혀 고려하지 않고, 단어들의 출현 빈도 (frequency)에만 집중하는 텍스트 데이터의 수치화 표현 방법입니다. Bag of Words를 직역하면 단어들의 가방이라는 의미입니다. 단어들이 들어있는 가방을 상상해봅시다. 갖고있는 어떤 텍스트 문서에 있는 단어들을 가방에다가 전부 넣습니다. 그리고나서 이 가방을 흔들어 단어들을 섞습니다. 만약, 해당 문서 내에서 특정 단어가 N번 등장했다면, 이 가방에는 그 특정 단어가 N개 있게됩니다. 또한 가방을 흔들어서 단어를 섞었기 때문에 더 이상 단어의 순서는 중요하지 않습니다.
- BoW를 만드는 과정
 1. 각 단어에 고유한 인덱스(Index) 부여
 2. 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터(Vector) 생성

문서 단어 행렬

- Bag of Words 확장
- 문서 단어 행렬(Document-Term Matrix, DTM)이란 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것을 말합니다. 쉽게 생각하면 각 문서에 대한 BoW를 하나의 행렬로 만든 것으로 생각할 수 있으며, BoW와 다른 표현 방법이 아니라 BoW 표현을 행렬로 표현하고 부르는 용어입니다.
- 행(Row)은 각각의 문서를 나타내고 열은 각각의 고유 토큰(Token)을 나타낸다.
- 단점
 - 희소 표현(Sparse representation)
대부분의 값이 0이 되는 특징을 갖는다
 - 단순 빈도 수 기반 접근
빈도만으로 표현하게 되면 문서들의 비교 및 분석이 어려워진다

문서 단어 행렬

- D1 : He is a lazy boy. She is also lazy.
- D2 : Neeraj is a lazy person.
- 토큰 목록 : He, She, Lazy, Boy, Neeraj, person
- D = 2, N = 6

	He	She	lazy	boy	Neeraj	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

TF-IDF

- 문서 단어 행렬 확장
- Term Frequency-Inverse Document Frequency
- 단어의 빈도와 역 문서 빈도(문서의 빈도에 특정 식을 취함)를 사용하여 DTM 내의 각 단어들마다 중요한 정도를 가중치로 주는 방법입니다. 사용 방법은 우선 DTM을 만든 후에, 거기에 TF-IDF 가중치를 주면됩니다.
- TF-IDF는 주로 문서의 유사도를 구하는 작업, 검색 시스템에서 검색 결과의 중요도를 정하는 작업, 문서 내에서 특정 단어의 중요도를 구하는 작업 등에 쓰일 수 있습니다.
- TF-IDF는 TF와 IDF를 곱한 값을 의미하는데 이를 식으로 표현해보겠습니다. 문서를 d , 단어를 t , 문서의 총 개수를 n 이라고 표현할 때 TF, DF, IDF는 각각 다음 페이지의 내용처럼 정의할 수 있습니다.

TF-IDF

1. $tf(d,t)$: 특정 문서 d 에서의 특정 단어 t 의 등장 횟수.

생소한 글자때문에 어려워보일 수 있지만, 잘 생각해보면 TF는 이미 앞에서 구한 적이 있습니다. TF는 앞에서 배운 DTM의 예제에서 각 단어들이 가진 값들입니다. DTM이 각 문서에서의 각 단어의 등장 빈도를 나타내는 값이었기 때문입니다.

2. $df(t)$: 특정 단어 t 가 등장한 문서의 수.

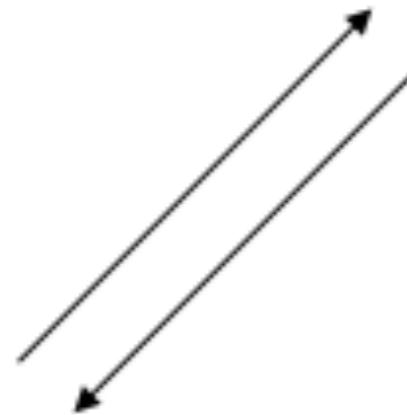
여기서 특정 단어가 각 문서, 또는 문서들에서 몇 번 등장했는지는 관심가지지 않으며 오직 특정 단어 t 가 등장한 문서의 수에만 관심을 가집니다. 앞서 배운 DTM에서 바나나는 문서2와 문서3에서 등장했습니다. 즉, 바나나의 df 는 2입니다. 문서3에서 바나나가 두 번 등장했지만, 그것은 중요한 게 아닙니다. 심지어 바나나란 단어가 문서2에서 100번 등장했고, 문서3에서 200번 등장했다고 하더라도 바나나의 df 는 2가 됩니다.

3. $idf(d, t)$: $df(t)$ 에 반비례하는 수.

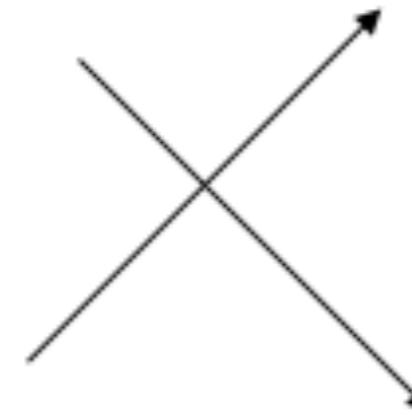
IDF라는 이름을 보고 DF의 역수가 아닐까 생각했다면, IDF는 DF의 역수를 취하고 싶은 것 이 맞습니다. 그런데 \log 와 분모에 1을 더해주는 식에 의아하실 수 있습니다. \log 를 사용하지 않았을 때, IDF를 DF의 역수($ndf(t)$ 라는 식)로 사용한다면 총 문서의 수 n 이 커질 수록, IDF의 값은 기하급수적으로 커지게 됩니다. 그렇기 때문에 \log 를 사용합니다.

코사인 유사도(Cosine Similarity)

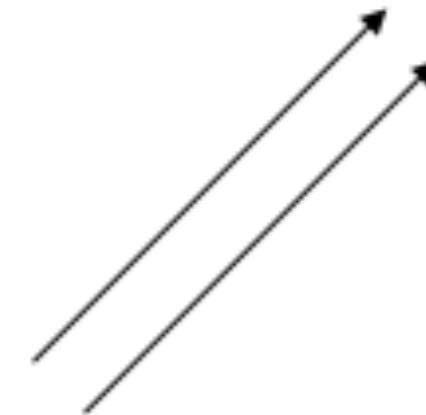
- 두 벡터 간의 코사인 각도를 이용하여 구할 수 있는 두 벡터의 유사도를 의미합니다.
- 두 벡터의 방향이 완전히 동일한 경우에는 1의 값을 가지며, 90° 의 각을 이루면 0, 180° 로 반대의 방향을 가지면 -1의 값을 갖게 됩니다.
- 결국 코사인 유사도는 -1 이상 1 이하의 값을 가지며 값이 1에 가까울수록 유사도가 높다고 판단할 수 있습니다. 이를 직관적으로 이해하면 두 벡터가 가리키는 방향이 얼마나 유사한가를 의미합니다.



코사인 유사도 : 1

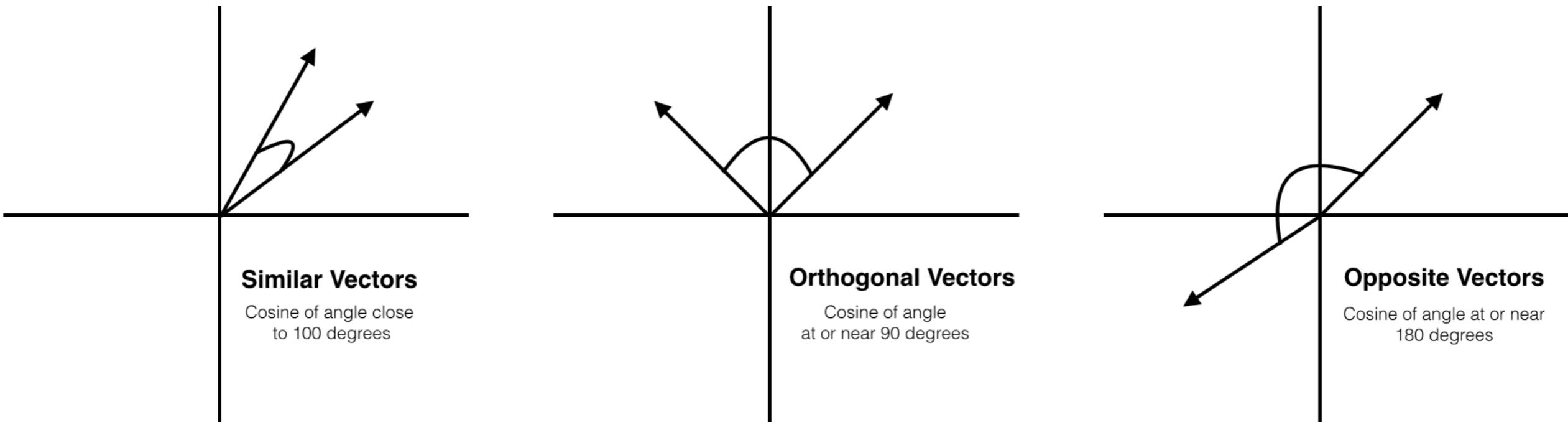


코사인 유사도 : -1



코사인 유사도 : 1

코사인 유사도(Cosine Similarity)



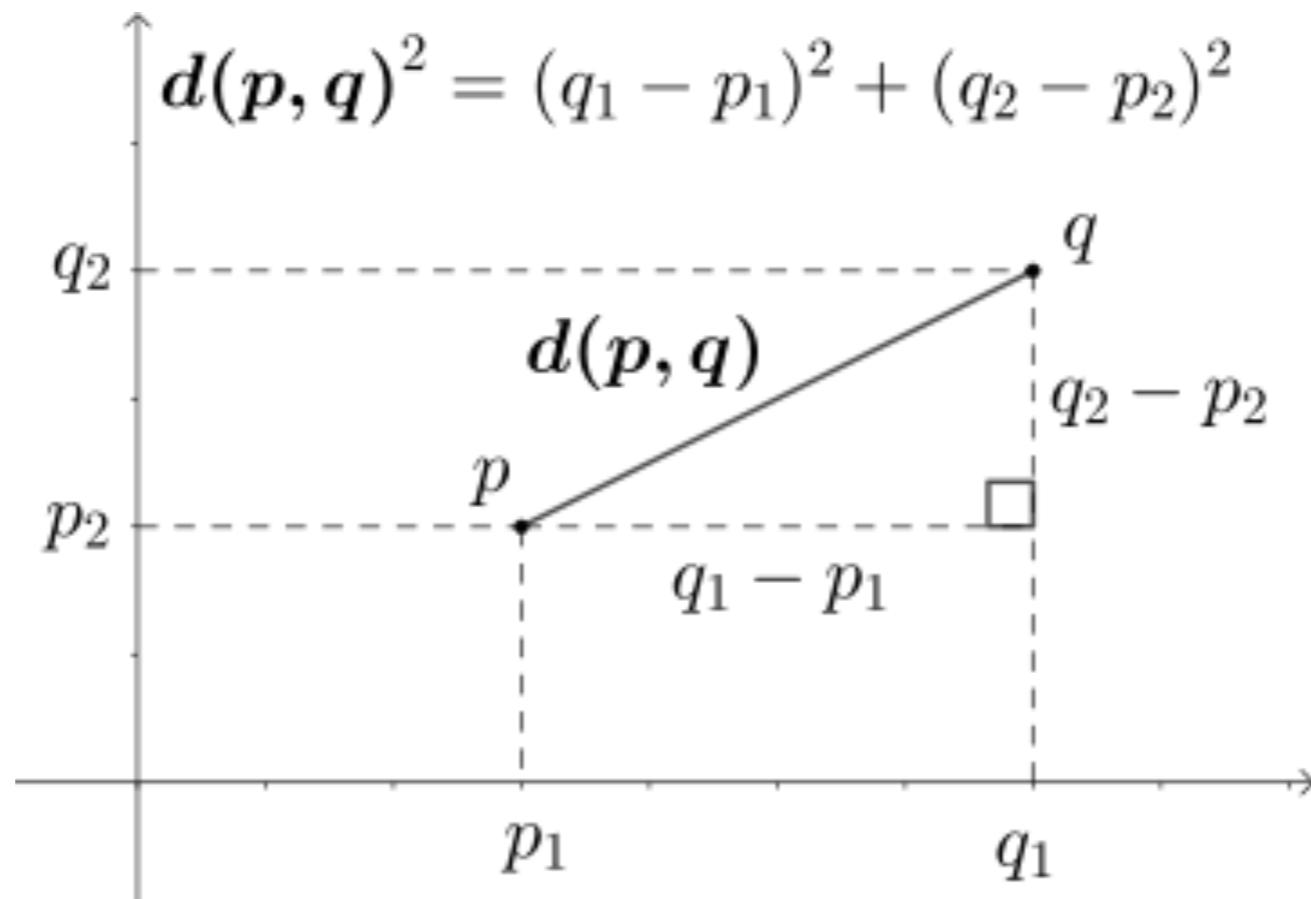
유클리드 거리(Euclidean Distance)

- 문서의 유사도를 구할 때 코사인이나 자카드 유사도만큼 유용하지는 않습니다.
- 기하학적으로의 거리가 가깝다는 것을 유사하다라고 판단할 수 있습니다.
- 여러 문서에 대해서 유사도를 구하고자 유클리드 거리 공식을 사용한다는 것은, 다음 페이지의 그림에 있는 2차원을 단어의 총 개수만큼의 차원으로 확장하는 것과 같습니다.
- 다차원 공간에서 두개의 점 p 와 q 가 각각 $p=(p_1,p_2,p_3,\dots,p_n)$ 과 $q=(q_1,q_2,q_3,\dots,q_n)$ 의 좌표를 가질 때 두 점 사이의 거리를 계산하는 유클리드 거리 공식은 다음과 같습니다.

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

유클리드 거리(Euclidean Distance)

- 2차원 좌표 평면 상에서 두 점 p와 q사이의 직선 거리를 구하는 문제입니다. 위의 경우에는 직각 삼각형으로 표현이 가능하므로, 중학교 수학 과정인 피타고라스의 정리를 통해 p와 q 사이의 거리를 계산할 수 있습니다. 즉, 2차원 좌표 평면에서 두 점 사이의 유클리드 거리 공식은 피타고라스의 정리를 통해 두 점 사이의 거리를 구하는 것과 동일합니다.



자카드 유사도(Jaccard Similarity)

- A와 B 두개의 집합이 있다고 합시다. 이때 교집합은 두 개의 집합에서 공통으로 가지고 있는 원소들의 집합을 말합니다. 즉, 합집합에서 교집합의 비율을 구한다면 두 집합 A와 B의 유사도를 구할 수 있다는 것이 자카드 유사도(jaccard similarity)의 아이디어입니다. 자카드 유사도는 0과 1사이의 값을 가지며, 만약 두 집합이 동일하다면 1의 값을 가지고, 두 집합의 공통 원소가 없다면 0의 값을 갖습니다. 자카드 유사도를 구하는 함수를 J라고 하였을 때, 자카드 유사도 함수 J는 아래와 같습니다.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

자카드 유사도(Jaccard Similarity)

- 두 개의 비교할 문서를 각각 doc1, doc2라고 했을 때 doc1과 doc2의 문서의 유사도를 구하기 위한 자카드 유사도는 아래와 같습니다.
- 즉, 두 문서 doc1, doc2 사이의 자카드 유사도 $J(doc_1, doc_2)$ 는 두 집합의 교집합 크기를 두 집합의 합집합 크기로 나눈 값으로 정의됩니다.

$$J(doc_1, doc_2) = \frac{doc_1 \cap doc_2}{doc_1 \cup doc_2}$$

토픽 모델링

- 토픽(Topic)은 한국어로는 주제라고 합니다. 토픽 모델링(Topic Modeling)이란 기계 학습 및 자연어 처리 분야에서 토픽이라는 문서 집합의 추상적인 주제를 발견하기 위한 통계적 모델 중 하나로, 텍스트 본문의 숨겨진 의미 구조를 발견하기 위해 사용되는 텍스트 마이닝 기법입니다.
- 검색 엔진, 고객 민원 시스템 등과 같이 문서의 주제를 알아내는 일이 중요한 곳에서 사용됩니다.

잠재 의미 분석

- Latent Semantic Analysis (LSA)
- LSA는 정확히는 토픽 모델링을 위해 최적화 된 알고리즘은 아니지만, 토픽 모델링이라는 분야에 아이디어를 제공한 알고리즘이라고 볼 수 있습니다. 뒤에 나오는 LDA는 LSA의 단점을 개선해가며 탄생한 알고리즘으로 토픽 모델링에 보다 적합한 알고리즘입니다.
- BoW에 기반한 DTM이나 TF-IDF는 기본적으로 단어의 빈도 수를 이용한 수치화 방법이기 때문에 단어의 의미를 고려하지 못한다는 단점이 있습니다. (이를 토픽 모델링 관점에서는 단어의 토픽을 고려하지 못한다고도 합니다.) 이를 위한 대안으로 DTM의 잠재된(Latent) 의미를 이끌어내는 방법으로 잠재 의미 분석(Latent Semantic Analysis, LSA)이라는 방법이 있습니다. 잠재 의미 분석(Latent Semantic Indexing, LSI)이라고 부르기도 합니다.
- 이 방법을 이해하기 위해서는 선형대수학의 특이값 분해(Singular Value Decomposition, SVD)를 이해할 필요가 있습니다.

특이값 분해

- Singular Value Decomposition (SVD)
- 대표적인 차원 축소 알고리즘
- A 가 $m \times n$ 행렬일 때, 2개의 직교 행렬과 1개의 대각 행렬의 곱으로 분해하는 것
- 직교행렬(orthogonal matrix)
자신과 자신의 전치 행렬(transposed matrix)의 곱 또는 이를 반대로 곱한 결과가 단위행렬(identity matrix)이 되는 행렬
- 대각행렬(diagonal matrix)
주대각선을 제외한 곳의 원소가 모두 0인 행렬
- 대각 행렬의 대각 원소의 값을 행렬 A 의 특이값(singular value)이라고 합니다.

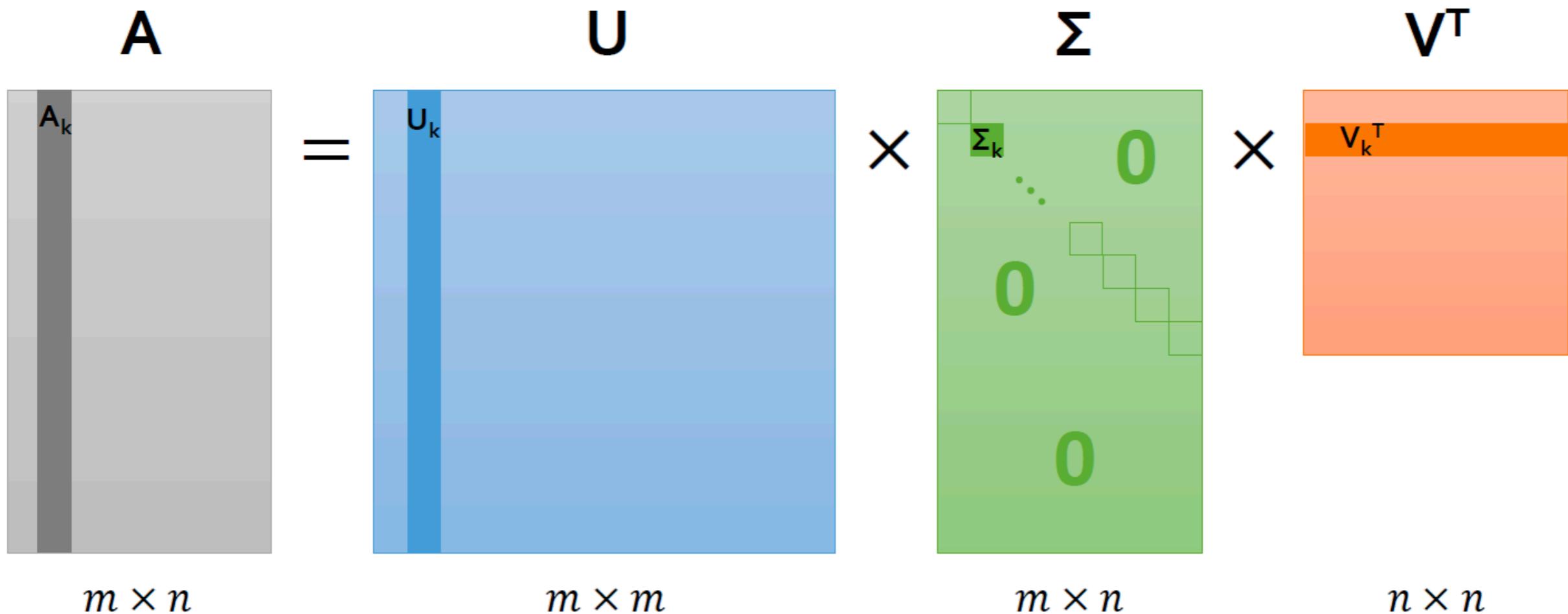
$$A = U\Sigma V^T$$

여기서 각 3개의 행렬은 다음과 같은 조건을 만족합니다.

$U : m \times m$ 직교행렬 ($AA^T = U(\Sigma\Sigma^T)U^T$)

$V : n \times n$ 직교행렬 ($A^TA = V(\Sigma^T\Sigma)V^T$)

$\Sigma : m \times n$ 직사각 대각행렬



잠재 디리클레 할당

- Latent Dirichlet Allocation (LDA)
- 토픽 모델링의 대표적인 알고리즘
- LDA는 문서들은 토픽들의 혼합으로 구성되어져 있으며, 토픽들은 확률 분포에 기반하여 단어들을 생성한다고 가정합니다. 데이터가 주어지면, LDA는 문서가 생성된 과정을 역추적합니다.

잠재 디리클레 할당

1. 사용자는 알고리즘에게 토픽의 개수 k 를 알려줍니다.

앞서 말하였듯이 LDA에게 토픽의 개수를 알려주는 역할은 사용자의 역할입니다. LDA는 토픽의 개수 k 를 입력받으면, k 개의 토픽이 M 개의 전체 문서에 걸쳐 분포되어 있다고 가정합니다.

2. 모든 단어를 k 개 중 하나의 토픽에 할당합니다.

이제 LDA는 모든 문서의 모든 단어에 대해서 k 개 중 하나의 토픽을 랜덤으로 할당합니다. 이 작업이 끝나면 각 문서는 토픽을 가지며, 토픽은 단어 분포를 가지는 상태입니다. 물론 랜덤으로 할당하였기 때문에 사실 이 결과는 전부 틀린 상태입니다. 만약 한 단어가 한 문서에서 2회 이상 등장하였다면, 각 단어는 서로 다른 토픽에 할당되었을 수도 있습니다.

3. 모든 문서의 모든 단어에 대해서 아래의 사항을 반복 진행합니다. (iterative)

어떤 문서의 각 단어 w 는 자신은 잘못된 토픽에 할당되어져 있지만, 다른 단어들은 전부 올바른 토픽에 할당되어져 있는 상태라고 가정하고, 단어 w 는 두 가지 기준에 따라 토픽을 재할당합니다.

- $p(\text{topic } t \mid \text{document } d)$: 문서 d 의 단어들 중 토픽 t 에 해당하는 단어들의 비율
- $p(\text{word } w \mid \text{topic } t)$: 단어 w 를 갖고 있는 모든 문서들 중 토픽 t 가 할당된 비율

LSA VS. LDA

- LSA
DTM을 차원 축소하여 축소 차원에서 근접 단어들을 토픽으로 묶는다.
- LDA
단어가 특정 토픽에 존재할 확률과
문서에 특정 토픽이 존재할 확률을 결합 확률로 추정하여 토픽을 추출한다.

WORD2VEC

- raw text로부터 word embedding을 학습하는 계산 효율성이 좋은 predictive model.

1. CBOW(Continuous Bag-Of-Words) model

- 소스 컨텍스트에서 타겟 단어 예측
- 예를 들어, 'the cat sits on the'라는 소스 컨텍스트로부터 'mat'이라는 타겟 단어 예측
- CBOW는 smaller 데이터셋에 적합

2. Skip-Gram model

- 타겟 단어로부터 소스 컨텍스트 예측
- 예를 들어, 'mat'이라는 타겟 단어로부터 'the cat sits on the'라는 소스 컨텍스트 예측
- Skip-Gram model은 larger 데이터셋에 적합

WORD2VEC

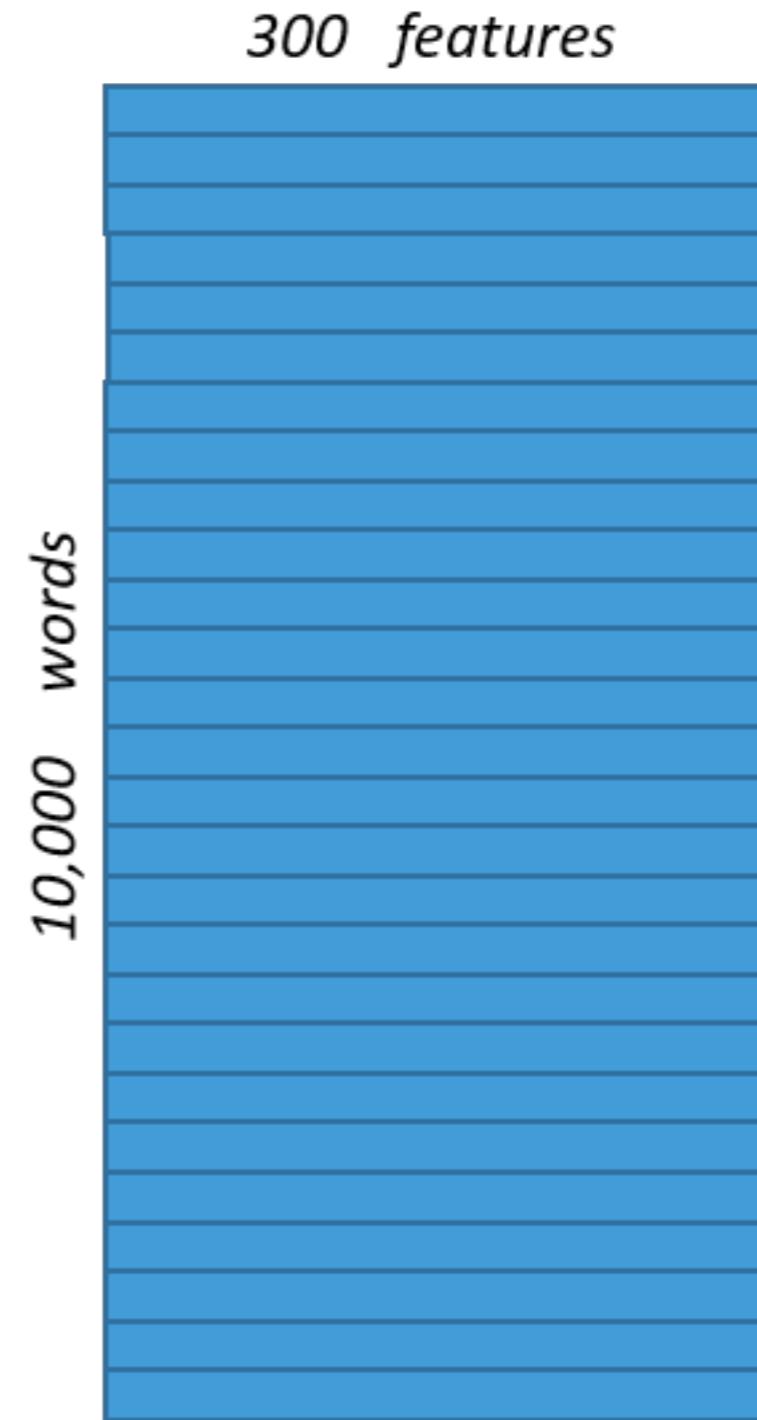
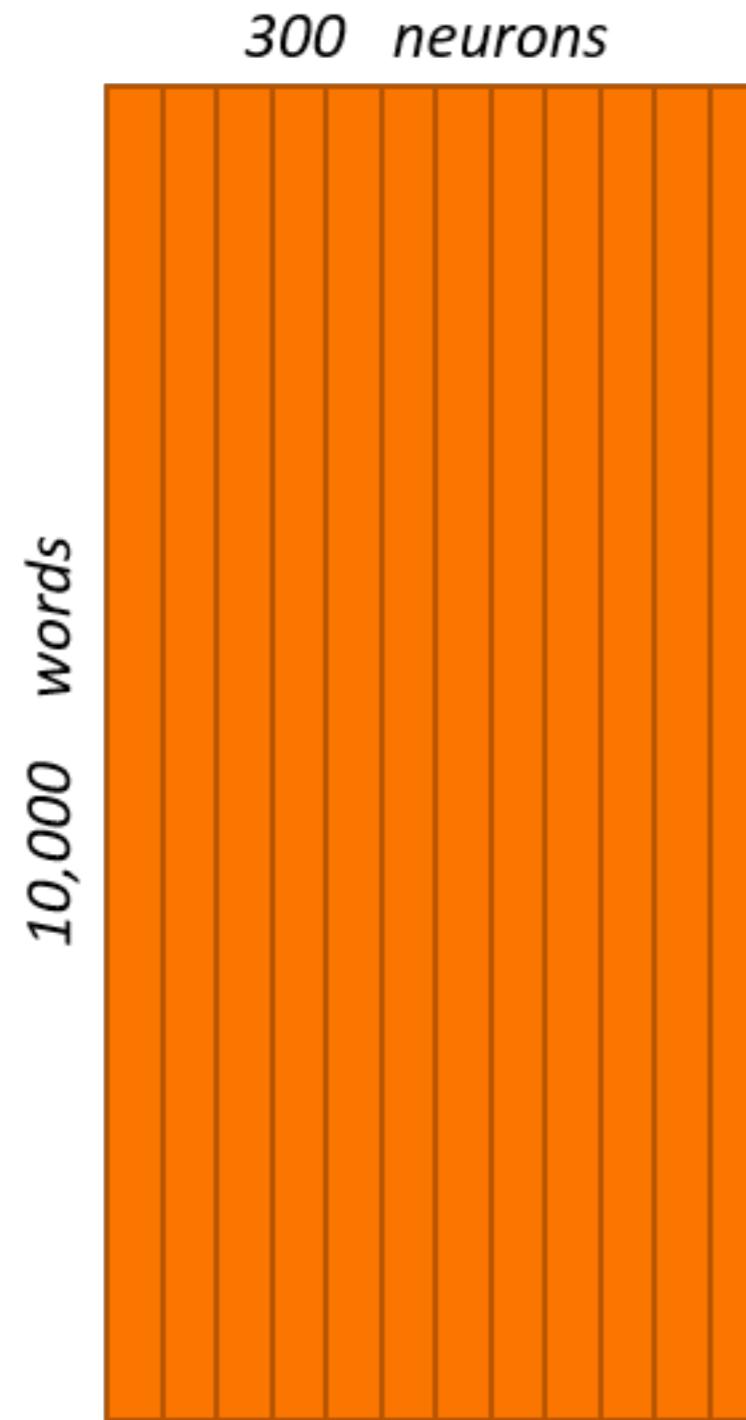
- Word2Vec 아키텍처는 중심단어로 주변단어를 맞추거나, 주변단어로 중심단어를 더 잘 맞추기 위해 가중치행렬인 W, W' 을 조금씩 업데이트하면서 학습이 이뤄지는 구조입니다. 그런데 여기서 흥미로운 점은 W 가 one-hot-encoding된 입력벡터와 은닉층을 이어주는 가중치행렬임과 동시에 Word2Vec의 최종 결과물인 임베딩 단어벡터의 모음이기도 하다는 사실입니다.
- 아래와 같이 단어가 5개뿐인 말뭉치에서 Word2Vec을 수행한다고 가정해 봅시다. 사전 등장 순서 기준으로 네번째 단어를 입력으로 하는 은닉층 값은 아래처럼 계산됩니다. 보시다시피 Word2Vec의 은닉층을 계산하는 작업은 사실상 가중치행렬 W 에서 해당 단어에 해당하는 행벡터를 참조(lookup)해 오는 방식과 똑같습니다.

$$\begin{bmatrix} 0 & 0 & 0 & \boxed{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & \boxed{12} & \boxed{19} \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

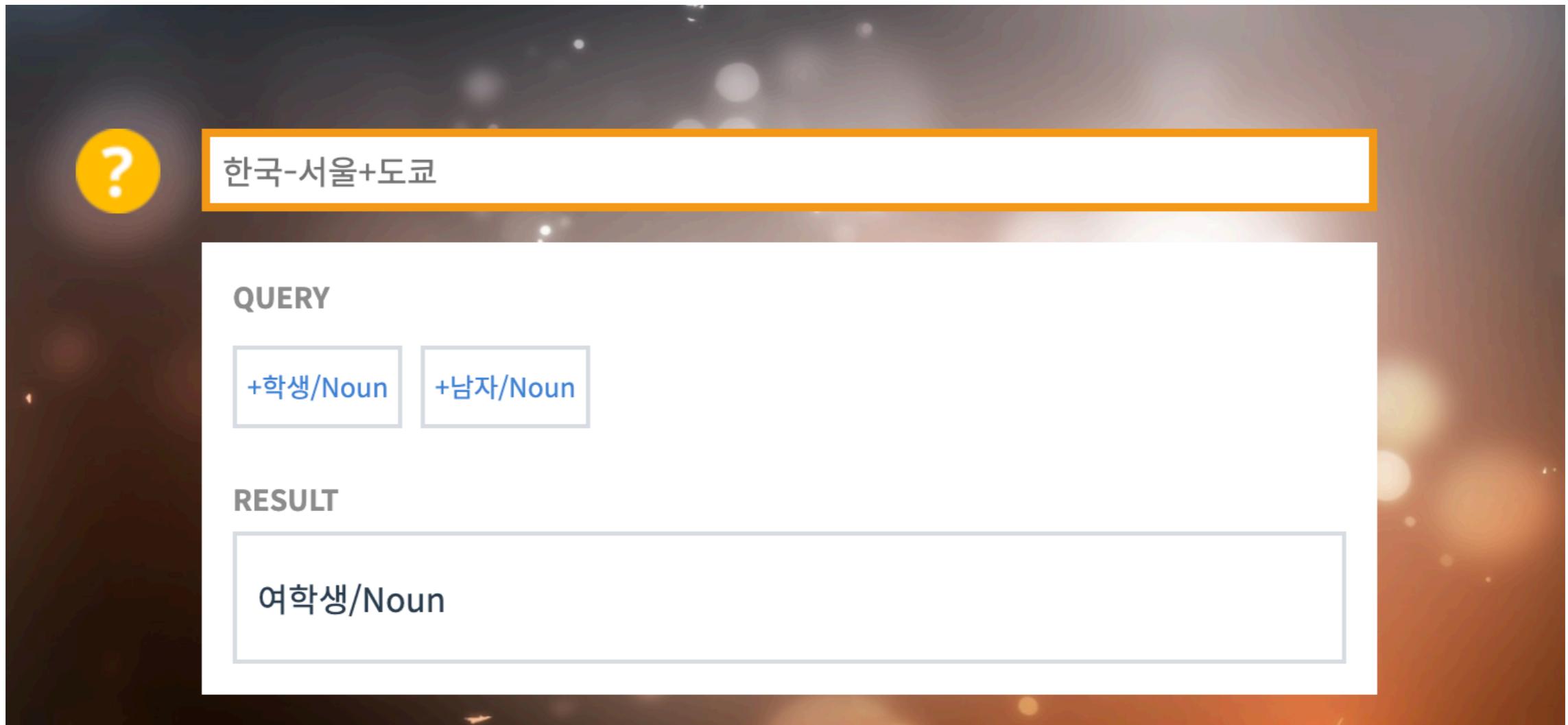
Hidden Layer Weight Matrix



Word Vector Lookup Table!



WORD2VEC



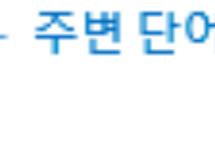
<http://word2vec.kr/search/>

CBOW

중심 단어



주변 단어



The **fat** cat sat on the mat

The **fat** cat sat on the mat

The fat **cat** sat **on** the mat

The fat **cat** **sat** **on** the mat

The fat **cat** sat **on** the mat

The fat cat **sat** **on** the **mat**

The fat cat sat **on** the **mat**

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

SKIP-GRAM

Source Text

Training Samples

The quick brown fox jumps over the lazy dog. →

(the, quick)
(the, brown)

The quick brown fox jumps over the lazy dog. →

(quick, the)
(quick, brown)
(quick, fox)

The quick brown fox jumps over the lazy dog. →

(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)

The quick brown fox jumps over the lazy dog. →

(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

Subsampling

- **subsampling은 학습량을 효과적으로 줄여 계산량을 감소시키는 전략입니다.**
- Word2Vec의 파라미터 크기가 $V \times N, N \times V$ 인데요. 보통 말뭉치에 등장하는 단어수가 10만 개 안팎이라는 점을 고려하면 N (임베딩 차원수, 사용자 지정)이 100차원만 되어도 2000만개 ($2 \times 10\text{만} \times 100$)나 되는 많은 숫자들을 구해야 합니다. 단어 수가 늘어날수록 계산량이 폭증하는 구조입니다.
- 이 때문에 Word2Vec 연구진은 말뭉치에서 자주 등장하는 단어는 학습량을 확률적인 방식으로 줄이기로 했습니다. 등장빈도만큼 업데이트 될 기회가 많기 때문입니다.
- 아래 식에서 $f(w_i)$ 는 해당 단어가 말뭉치에 등장한 비율(해당 단어 빈도/전체 단어수)을 말합니다. t 는 사용자가 지정해주는 값인데요, 연구팀에선 0.00001을 권하고 있습니다.
- 만일 $f(w_i)$ 가 0.01로 나타나는 빈도 높은 단어(예컨대 조사 '은/는')는 $P(w_i)$ 가 0.9684나 되어서 100번의 학습 기회 가운데 96번 정도는 학습에서 제외하게 됩니다. 반대로 등장 비율이 적어 $P(w_i)$ 가 0에 가깝다면 해당 단어가 나올 때마다 빼놓지 않고 학습을 시키는 구조입니다.

i번째 단어를 학습에서 제외시키기 위한 확률

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Negative sampling

- Word2Vec은 출력층이 내놓는 스코어값에 소프트맥스 함수를 적용해 확률값으로 변환한 후 이를 정답과 비교해 역전파(backpropagation)하는 구조입니다.
- 그런데 소프트맥스를 적용하려면 분모에 해당하는 값, 즉 중심단어와 나머지 모든 단어의 내적을 한 뒤, 이를 다시 \exp 를 취해줘야 합니다. 보통 전체 단어가 10만개 안팎으로 주어지니까 계산량이 어마어마해지죠.
- 이 때문에 소프트맥스 확률을 구할 때 전체 단어를 대상으로 구하지 않고, 일부 단어만 뽑아서 계산을 하게 되는데요. 이것이 바로 negative sampling입니다. negative sampling은 학습 자체를 아예 스kip하는 subsampling이랑은 다르다는 점에 유의하셔야 합니다.
- negative sampling의 절차는 이렇습니다. 사용자가 지정한 윈도우 사이즈 내에 등장하지 않는 단어(negative sample)를 5~20개 정도 뽑습니다. 이를 정답단어와 합쳐 전체 단어처럼 소프트맥스 확률을 구하는 것입니다. 바꿔 말하면 윈도우 사이즈가 5일 경우 최대 25개 단어를 대상으로만 소프트맥스 확률을 계산하고, 파라미터 업데이트도 25개 대상으로만 이뤄진다는 이야기입니다.
- 참고로 subsampling과 negative sampling에 쓰는 확률값들은 고정된 값이기 때문에 학습을 시작할 때 미리 구해놓게 됩니다.

윈도우 내에 등장하지 않은 단어가 뽑힐 확률

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^n f(w_j)^{3/4}}$$

GloVe

- 글로브(Global Vectors for Word Representation, GloVe)는 카운트 기반과 예측 기반을 모두 사용하는 방법론으로 2014년에 미국 스탠포드대학에서 개발한 단어 임베딩 방법론입니다. 카운트 기반의 LSA(Latent Semantic Analysis)와 예측 기반의 Word2Vec의 단점을 지적하며 이를 보완한다는 목적으로 나왔고, 실제로도 Word2Vec만큼이나 뛰어난 성능을 보여줍니다. 현재까지의 연구에 따르면 단정적으로 Word2Vec과 GloVe 중에서 어떤 것이 더 뛰어나다고 말할 수는 없고, 이 두 가지 전부를 사용해보고 성능이 더 좋은 것을 사용하는 것이 바람직합니다.
- LSA는 DTM이나 TF-IDF 행렬과 같이 각 문서에서의 각 단어의 빈도수를 카운트 한 행렬이라는 전체적인 통계 정보를 입력으로 받아 차원을 축소(Truncated SVD)하여 잠재된 의미를 끌어내는 방법론이 있습니다. 반면, Word2Vec는 실제값과 예측값에 대한 오차를 손실 함수를 통해 줄여나가며 학습하는 예측 기반의 방법론이었습니다. 서로 다른 방법을 사용하는 이 두 방법론은 각각 장, 단점이 있습니다.
- LSA는 카운트 기반으로 코퍼스의 전체적인 통계 정보를 고려하기는 하지만, 왕:남자 = 여왕:? (정답은 여자)과 같은 단어 의미의 유추 작업(Analogy task)에는 성능이 떨어집니다. Word2Vec은 예측 기반으로 단어 간 유추 작업에는 LSA보다 뛰어나지만, 임베딩 벡터가 윈도우 크기 내에서만 주변 단어를 고려하기 때문에 코퍼스의 전체적인 통계 정보를 반영하지 못합니다. GloVe는 이러한 기존 방법론들의 각각의 한계를 지적하며, LSA의 메커니즘이었던 카운트 기반의 방법과 Word2Vec의 메커니즘이었던 예측 기반의 방법론 두 가지를 모두 사용합니다.

동시 발생

"Co-occurrence"

두 단어가 정해진 구간 내에서 동시에 등장함

Word2Vec 이후의 워드 임베딩은 동시 발생 개념을 모두 포함하고 있다.

동시 발생 행렬

Window based Co-occurrence Matrix

Quick Brown Fox Jump Over The Lazy Dog

Quick Brown Fox Jump Over The Lazy Dog

He is not lazy He is intelligent He is smart

He is not lazy He is intelligent He is smart

He is not lazy He is intelligent He is smart

He is not lazy He is intelligent He is smart

동시 발생 행렬

Window based Co-occurrence Matrix

- He is not lazy. He is intelligent. He is smart

	He	is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

동시 등장 확률 (Co-occurrence Probability)

- 동시 등장 확률 $P(k | i)$ 는 동시 등장 행렬로부터 특정 단어 i 의 전체 등장 횟수를 카운트하고, 특정 단어 i 가 등장했을 때 어떤 단어 k 가 등장한 횟수를 카운트하여 계산한 조건부 확률입니다.
- $P(k | i)$ 에서 i 를 중심 단어(Center Word), k 를 주변 단어(Context Word)라고 했을 때, 위에서 배운 동시 등장 행렬에서 중심 단어 i 의 행의 모든 값을 더한 값을 분모로 하고 i 행 k 열의 값을 분자로 한 값이라고 볼 수 있겠습니다.

동시 등장 확률 (Co-occurrence Probability)

동시 등장 확률과 크기 관계 비(ratio)	k=solid	k=gas	k=water	k=fasion
$P(k \mid \text{ice})$	0.00019	0.000066	0.003	0.000017
$P(k \mid \text{steam})$	0.000022	0.00078	0.0022	0.000018
$P(k \mid \text{ice}) / P(k \mid \text{steam})$	8.9	0.085	1.36	0.96

- 위의 표를 통해 알 수 있는 사실은 solid가 등장했을 때 ice가 등장할 확률 0.00019 는 solid가 등장했을 때 steam이 등장할 확률인 0.000022보다 약 8.9배 크다는 겁니다. 그도 그럴 것이 solid는 '단단한'이라는 의미를 가졌으니까 '증기'라는 의미를 가지는 steam보다는 당연히 '얼음'이라는 의미를 가지는 ice라는 단어와 더 자주 등장할 겁니다.

동시 등장 확률 (Co-occurrence Probability)

- 수식적으로 다시 정리하여 언급하면 k 가 solid일 때, $P(\text{solid} | \text{ice}) / P(\text{solid} | \text{steam})$ 를 계산한 값은 8.9가 나옵니다. 이 값은 1보다는 매우 큰 값입니다. 왜냐면 $P(\text{solid} | \text{ice})$ 의 값은 크고, $P(\text{solid} | \text{steam})$ 의 값은 작기 때문입니다.
- 그런데 k 를 solid가 아니라 gas로 바꾸면 얘기는 완전히 달라집니다. gas는 ice보다는 steam과 더 자주 등장하므로, $P(\text{gas} | \text{ice}) / P(\text{gas} | \text{steam})$ 를 계산한 값은 1보다 훨씬 작은 값인 0.085가 나옵니다. 반면, k 가 water인 경우에는 solid와 steam 두 단어 모두 와 동시 등장하는 경우가 많으므로 1에 가까운 값이 나오고, k 가 fasion인 경우에는 solid 와 steam 두 단어 모두와 동시 등장하는 경우가 적으므로 1에 가까운 값이 나옵니다.

단어의 동시 등장 확률과 크기 관계 비(ratio)	$k=\text{solid}$	$k=\text{gas}$	$k=\text{water}$	$k=\text{fashion}$
$P(k \text{ice})$	큰 값	작은 값	큰 값	작은 값
$P(k \text{steam})$	작은 값	큰 값	큰 값	작은 값
$P(k \text{ice}) / P(k \text{steam})$	큰 값	작은 값	1에 가까움	1에 가까움

GloVe

- GloVe의 아이디어를 한 줄로 요약하면
'임베딩 된 중심 단어와 주변 단어 벡터의 내적이
전체 코퍼스에서의 동시 등장 확률이 되도록 만드는 것'입니다.
즉, 이를 만족하도록 임베딩 벡터를 만드는 것이 목표입니다.

FastText

- 단어를 벡터로 만드는 또 다른 방법으로는 페이스북에서 개발한 패스트텍스트(FastText)가 있습니다. Word2Vec 이후에 나온 것이기 때문에, 메커니즘 자체는 Word2Vec의 확장이라고 볼 수 있습니다. Word2Vec와 패스트텍스트와의 가장 큰 차이점이라면 Word2Vec는 단어를 쪼개질 수 없는 단위로 생각한다면, 패스트텍스트는 하나의 단어 안에도 여러 단어들이 존재하는 것으로 간주합니다. 즉 내부 단어(subword)를 고려하여 학습합니다.

- 패스트텍스트에서는 우선 각 단어는 글자들의 n-gram으로 나타냅니다. n을 몇으로 결정하는지에 따라서 단어들이 얼마나 분리되는지 결정됩니다. 예를 들어서 n을 3으로 잡은 트라이그램(tri-gram)의 경우, apple은 app, ppl, ple로 분리하고 이들 또한 임베딩을 합니다. 더 정확히는 시작과 끝을 의미하는 <, >를 도입하여 아래의 5개를 임베딩합니다.

<ap, app, ppl, ple, le> #n=3이므로 길이가 3

- 그리고 여기에 추가적으로 하나를 더 임베딩합니다.

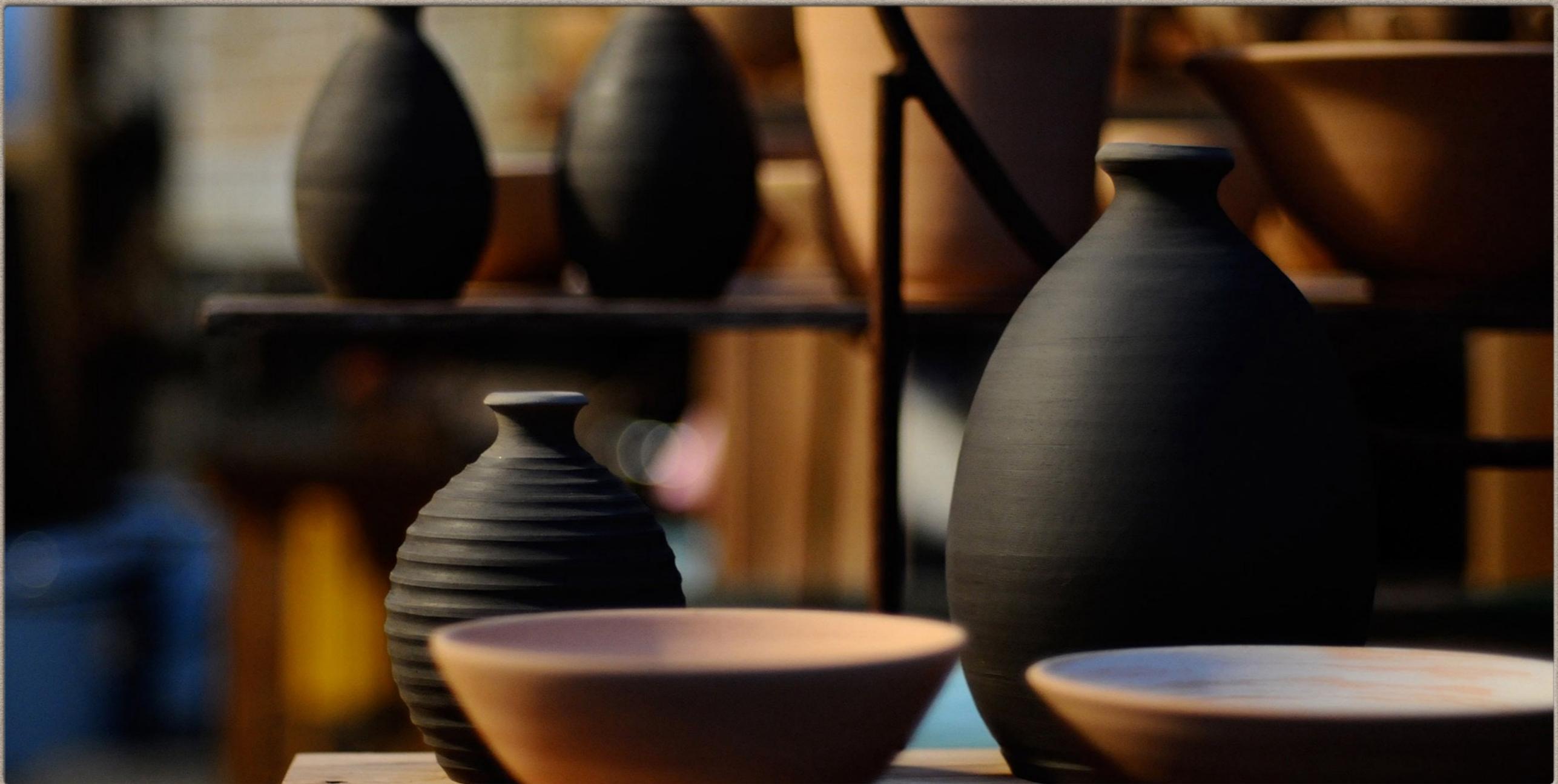
<apple> # 특별 토큰

FastText

- 실제 사용할 때는 n의 최소값과 최대값을 설정할 수 있는데, 기본값으로는 각각 3과 6으로 설정되어져 있습니다. 패스트텍스트의 인공 신경망을 학습한 후에는 데이터셋의 모든 단어의 각 n-gram에 대해서 워드 임베딩이 됩니다. 이렇게 되면 장점은 데이터셋만 충분한다면 위와 같은 내부 단어(Subword)를 통해 모르는 단어(Out Of Vocabulary, OOV)에 대해서도 다른 단어와의 유사도를 계산할 수 있다는 점입니다.
- 가령, 패스트텍스트에서 birthplace(출생지)란 단어를 학습하지 않은 상태라고 해봅시다. 하지만 다른 단어의 n-gram으로서 birth와 place를 학습한 적이 있다면 birthplace의 임베딩 벡터(Embedding vector)를 만들어낼 수 있습니다. 이는 모르는 단어에 대처할 수 없었던 Word2Vec와는 다른 점입니다.

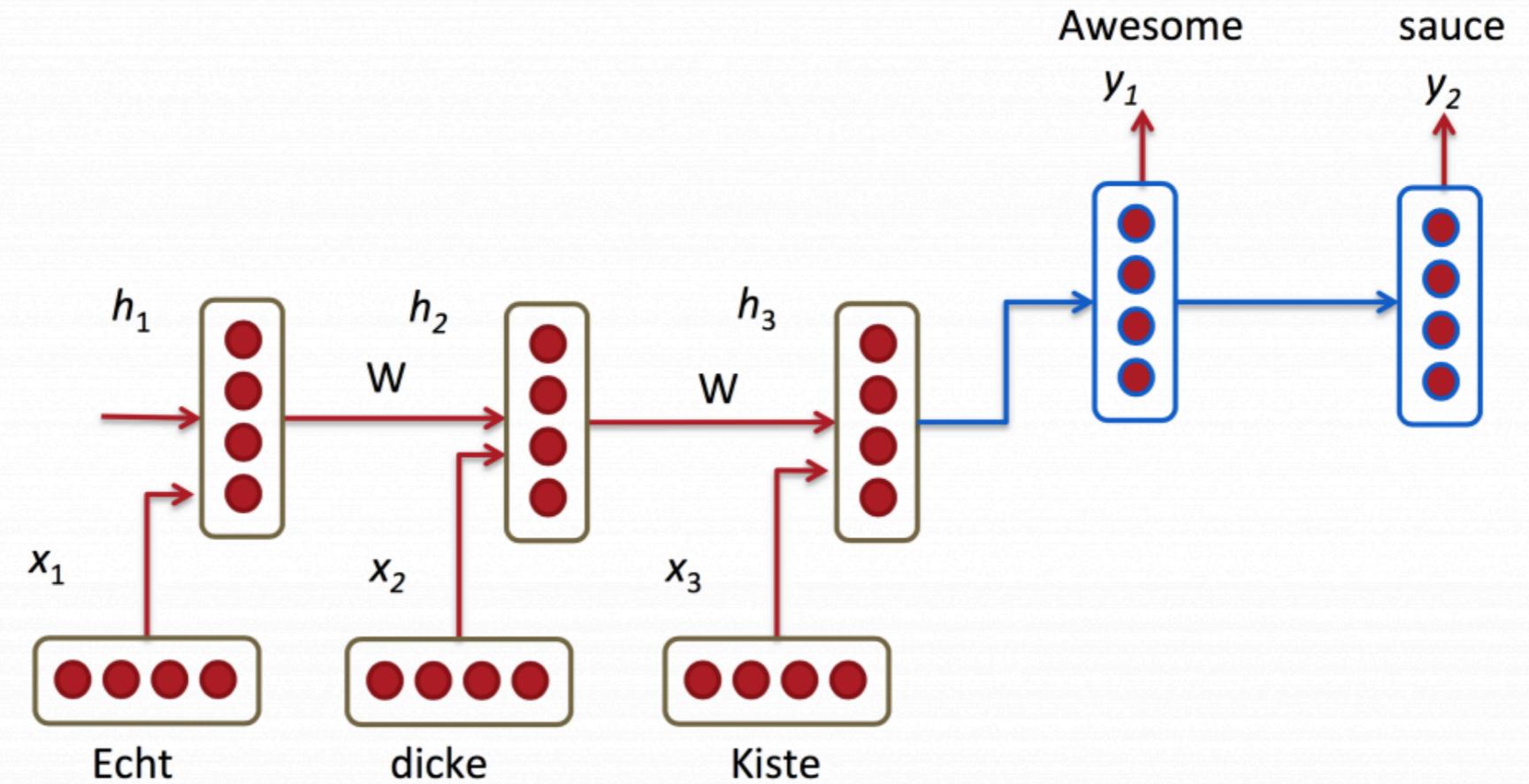
FastText

- Word2Vec의 경우에는 단어 집합에서 등장 빈도 수가 높으면 비교적 정확하게 임베딩이 되지만, 등장 빈도 수가 적은 단어(Rare word)에 대해서는 임베딩의 정확도가 높지 않다는 단점이 있습니다. 참고할 수 있는 경우의 수가 적다보니 정확하게 임베딩이 되지 않는 것입니다.
- 하지만 패스트텍스트는 등장 빈도 수가 적은 단어라 하더라도, n-gram으로 임베딩을 하는 특성상 참고할 수 있는 경우의 수가 많아지므로 Word2Vec와 비교하여 정확도가 높은 경향이 있습니다.
- 패스트텍스트가 노이즈가 많은 코퍼스에서 강점을 가진 것 또한 이와 같은 이유입니다. 모든 훈련 코퍼스에 오타(Typo)나 맞춤법이 틀린 단어가 없으면 이상적이겠지만, 실제 많은 비정형 데이터에는 오타가 섞여있습니다. 그리고 오타가 섞인 단어는 당연히 등장 빈도수가 매우 적으므로 일종의 희귀 단어가 됩니다. 즉, Word2Vec에서는 오타가 섞인 단어는 임베딩이 제대로 되지 않지만 패스트텍스트는 이에 대해서도 일정 수준의 성능을 보입니다.



사례 정리

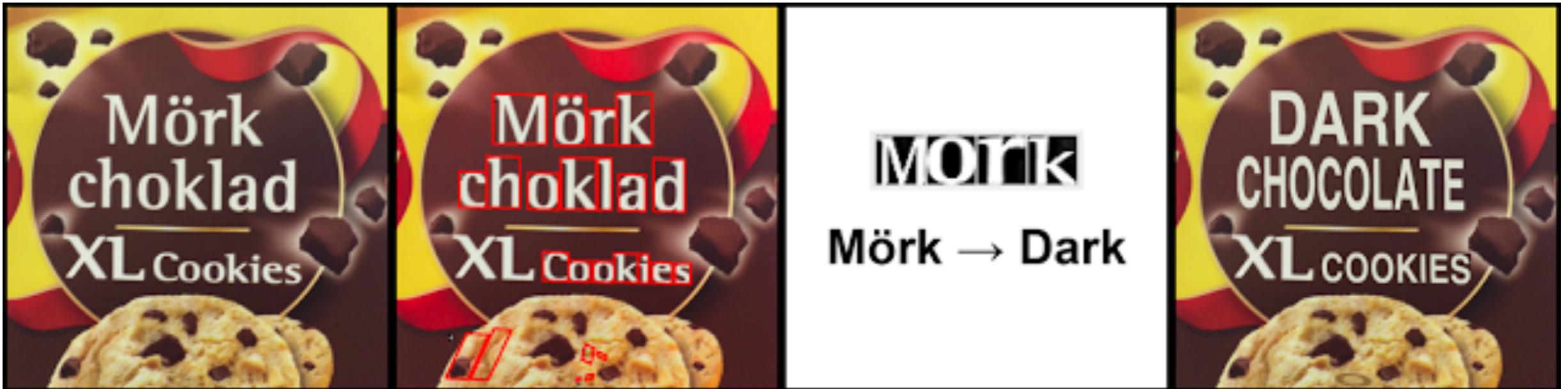
번역



기계 번역 문제는 입력이 단어들의 시퀀스라는 점에서 언어 모델링과 비슷하지만, 출력값이 다른 언어로 되어있는 단어들의 시퀀스라는 점에서 차이가 있다. 네트워크 상에서의 중요한 차이점은, 입력값을 전부 다 받아들인 다음에서야 네트워크가 출력값을 내보낸다는 점에 있는데, 번역 문제에서는 어순이 다른 문제 등이 있기 때문에 대상 언어의 문장의 첫 단어를 알기 위해선 번역할 문장 전체를 봐야 할 수도 있기 때문이다.

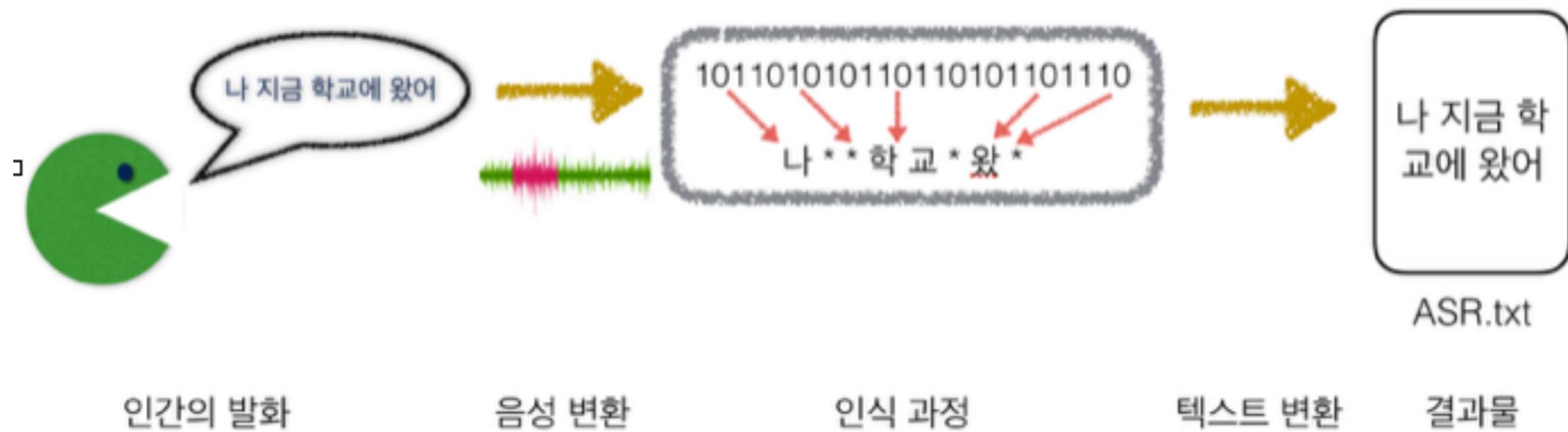
번역

구글 번역기는 자동으로 사진 속 외국어를 당신이 원하는 나라의 언어로 번역해줍니다. 당신이 알고 싶은 물체 위에 카메라만 대고 있으면 이미지 속 외국어를 번역해 줍니다. 언어는 점점 더 진입장벽이 낮아질 것이고 우리는 세계 각국의 많은 사람들과 대화가 가능해질 것입니다.



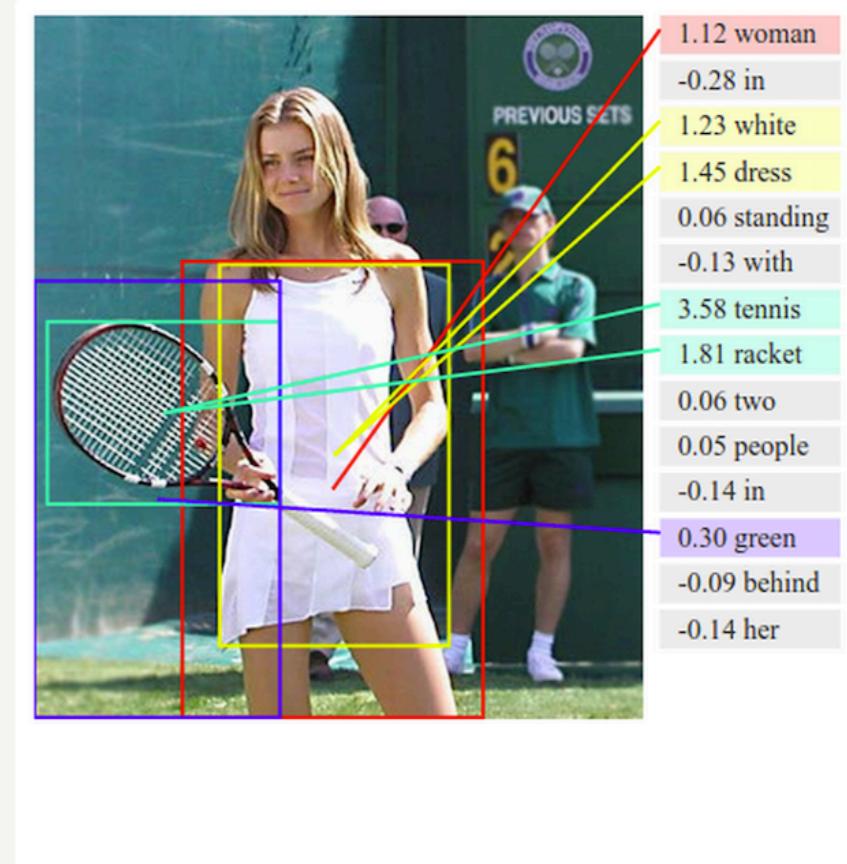
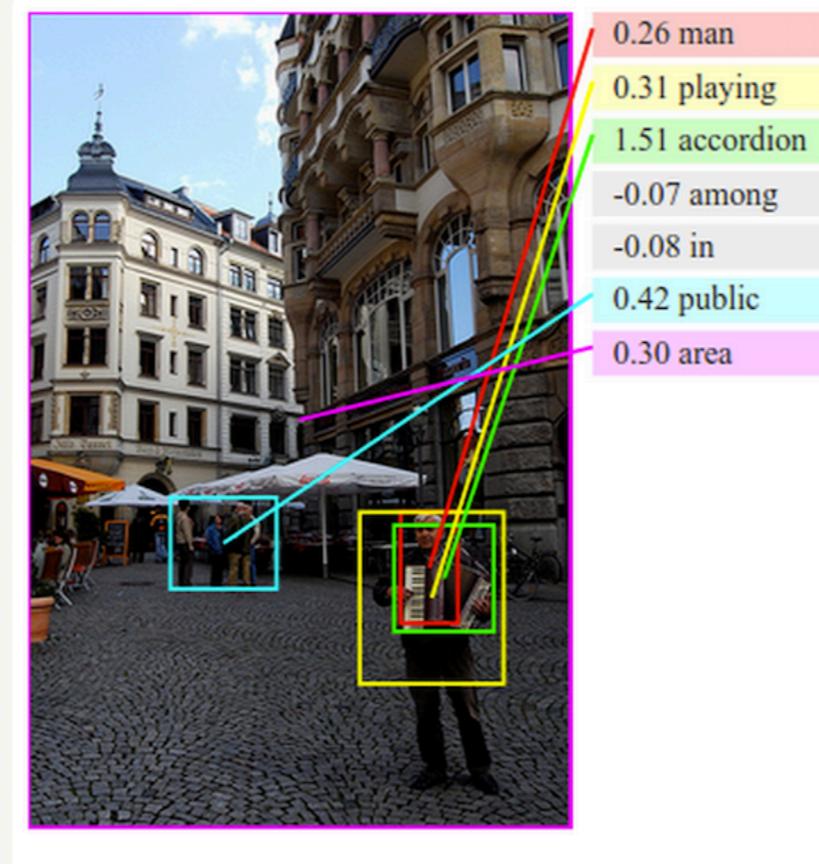
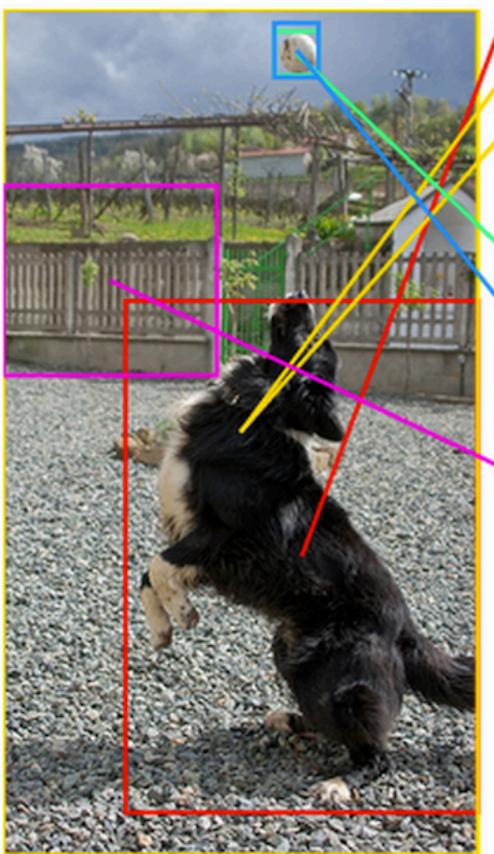
음성 인식

사운드 웨이브의 음향 신호(acoustic signal)를 입력으로 받아들이고, 출력으로는 음소(phonetic segment)들의 시퀀스와 각각의 음소별 확률 분포를 추측할 수 있다.



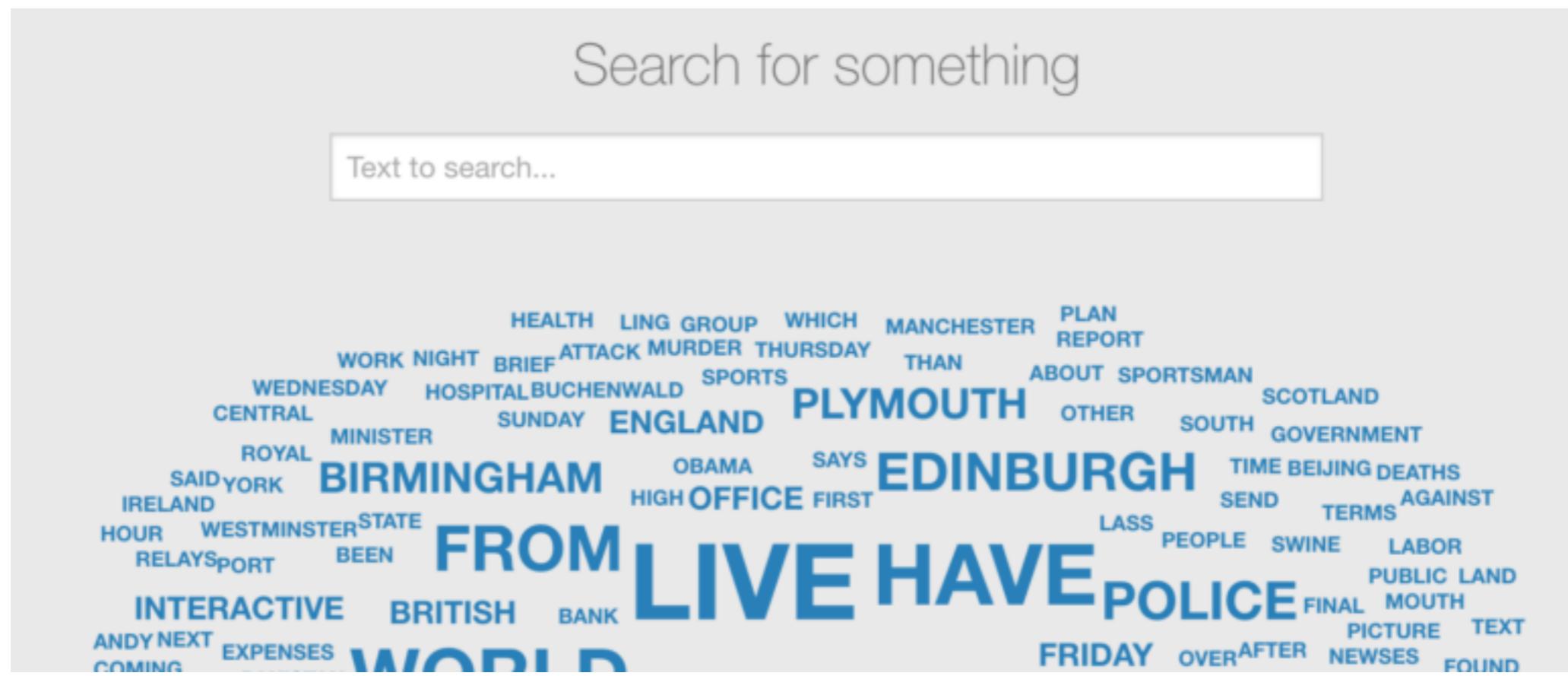
이미지 캡션

컴퓨터 비전에서 활발하게 사용된 convolutional neural network(CNN)과 RNN을 함께 사용한다면, 임의의 이미지를 텍스트로 설명해주는 시스템을 만드는 것도 가능하다. 실제로 어떻게 왜 동작하는지는 상당히 신기하다. CNN과 RNN을 합친 모델은 이미지로부터 얻어낸 주요 단어들과 이미지의 각 부분을 매칭해줄 수도 있다.



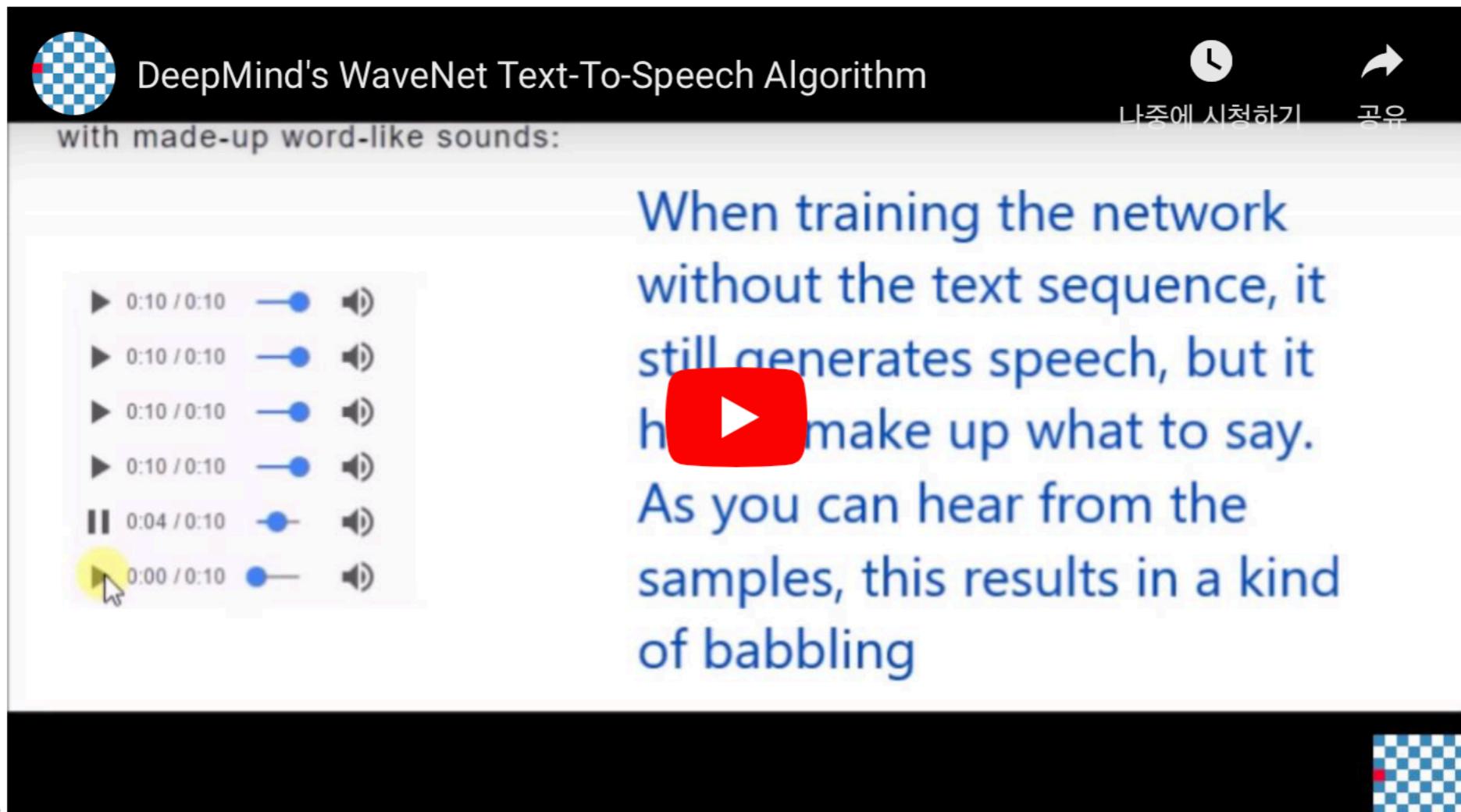
텍스트 읽기

Oxford Visual Geometry group은 딥러닝을 이용하여 “무방비한 상태의 문자 형태를 읽어냅니다.” 이는 사진이나 동영상의 텍스트를 읽어내는 시도로써 BBC 뉴스의 비디오에서 텍스트를 읽기 위한 노력입니다. 아래 예시를 참고해보세요.



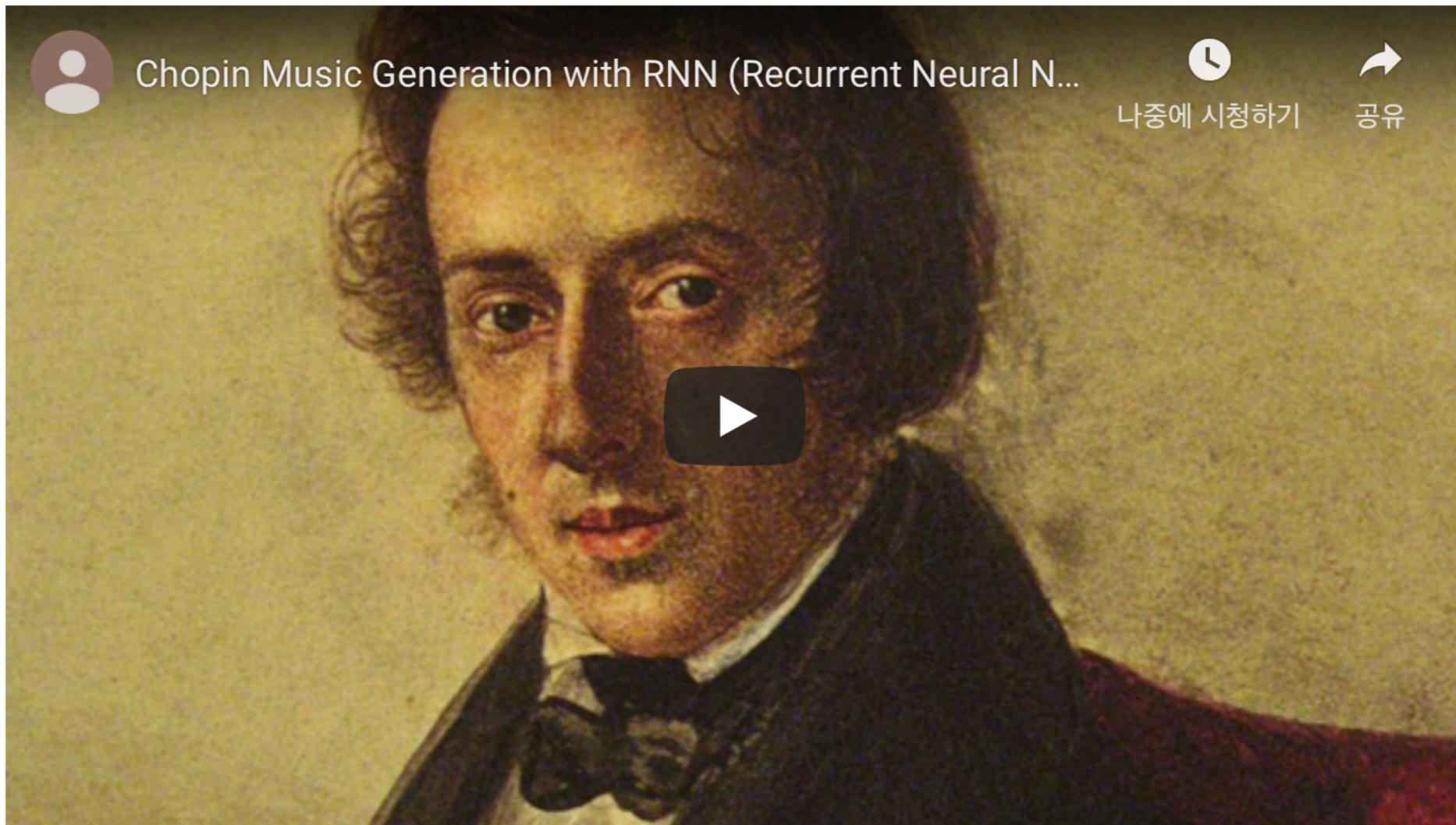
음성 생성

작년에 구글은 WaveNet을 출시하고 Baidu는 Deep Speech를 출시하였습니다. 두 가지 다 딥러닝을 이용하여 목소리를 자동 생성합니다. ‘그래서 뭐 어떡하라고?’라고 물어보실 수 있습니다. 시리와 알렉사도 충분히 말을 할 줄 아니까요. text2voice 시스템은 아직까진 완벽하게 자율적이지 않습니다. 새로운 소리를 내기에는 그렇게 훈련을 받아야지만 가능합니다. 요즘의 시스템은 시간에 걸쳐 스스로 훈련을 하면서 사람의 목소리를 흉내 냅니다.



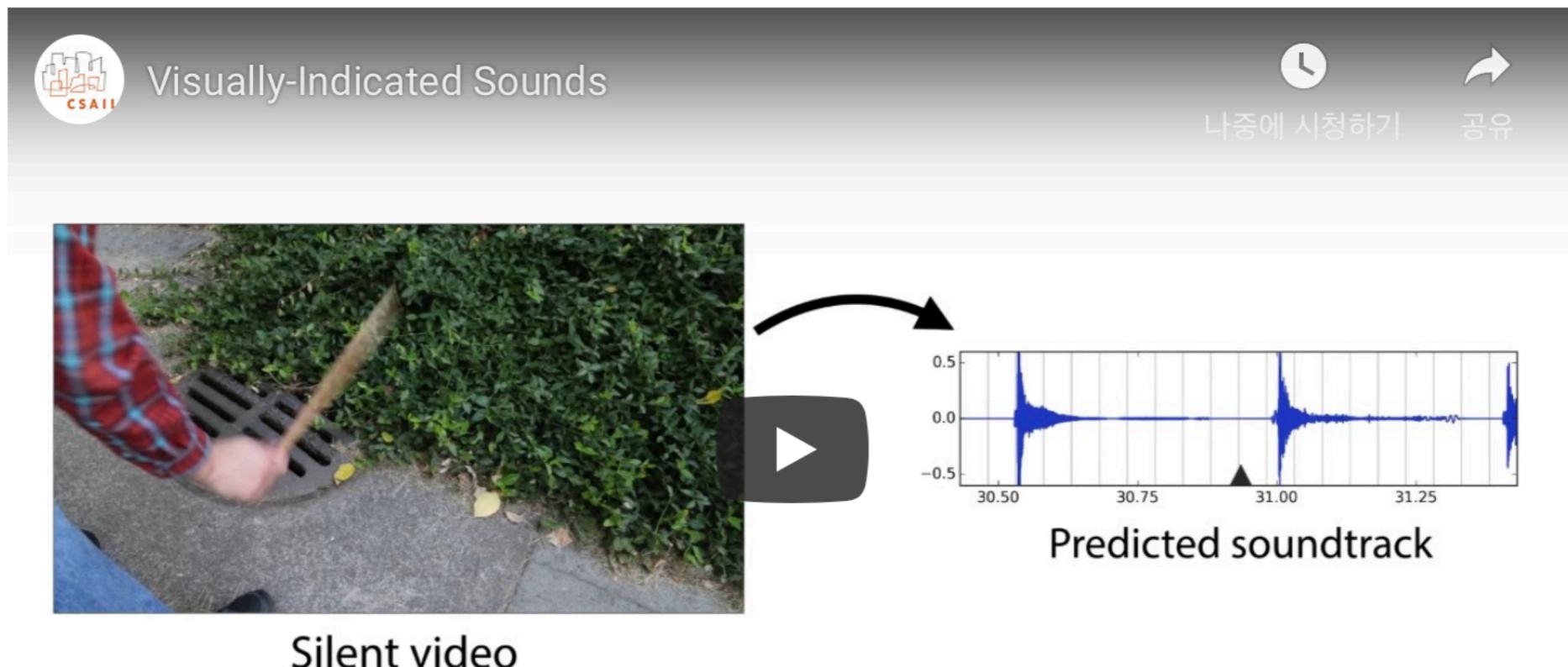
작곡

음성 인식과 같은 기술로 음악 작곡도 가능합니다. 아래는 Francesco Marchesani(쇼팽의 음악과 유사한 곡을 작곡시키기 훈련한 장본인)의 예시입니다. 컴퓨터는 쇼팽 음악 특유의 패턴과 통계정보를 바탕으로 학습하고, 기존에 없었던 새로운 작품을 생성하였습니다.



소리 복원

음소거된 영상에서 소리를 복원한다는 것이 믿기지 않을 수 있지만, 사람은 다른 사람의 입모양을 읽을 수 있다는 점을 기억하세요! Owens et al 은 영상 속 사람들이 드럼스틱으로 물체를 치고 긁는 소리를 학습에 있어 몇 번의 반복 작업 후, 과학자들은 영상의 소리를 끄고 컴퓨터에게 예상되는 소리를 만들라 지시하였습니다. 결과물은 인상적입니다.



입모양 읽기

이 사실이 아직 당신을 만족시키지 않는다면, 컴퓨터가 사람의 입모양을 읽는 건 어떻게 생각하시나요? 이것이 LipNet(옥스포드와 구글 Deepmind 과학자들의 작업)이 행할 수 있는 것입니다. LipNet은 사람의 입술을 읽어내는 일에 93%의 성공률을 기록하고 있습니다. 반면 실제 사람은 52%의 성공률을 나타내고요.



창작

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul.

Breaking and stro
The earth and tho

DUKE VINCENTIO:

Well, your wit is

Second Lord:

They would be rul
my fair nues begu
Whose noble souls

Clown:

Come, sir, I will

VIOLA:

I'll drink it.

세익스피어

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[<http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm> Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

위키피디아

손글씨

이 손글씨는 손이 아닌 컴퓨터가 쓴 문장입니다. Alex Graves의 블로그에서 당신의 문장을 손글씨로 만들어보세요.

컴퓨터가 만든 문장이나 예술 작품을 제가 계속 보여주고 있지만, 컴퓨터는 현재 손글씨까지도 가능합니다. 토론토 대학의 Alex Graves는 컴퓨터에게 다양한 스타일의 손글씨를 가질 수 있도록 훈련시켰습니다

recurrent neural network handwriting generation demo



nltk

Natural Language Toolkit

nltk

- Natural Language Toolkit
- 교육용으로 개발된 자연어 처리 및 문서 분석용 파이썬 패키지
- 다양한 기능 및 예제 포함
- 연구를 비롯해서 현업에서도 많이 사용

주요 기능

- 말뭉치 corpus, corpora(corpus의 복수형)
- 토큰 생성 tokenizing
- 형태소 분석 morphological analysis
- 품사 태깅 POS tagging

CORPORA

- WordNet
`nltk.corpus.wordnet` - English Lexical Database
- Guttenberg
`nltk.corpus.gutenberg` - Books from the Gutenberg Project
- WebText
`nltk.corpus.webtext` - User generated content on the web
- Brown
`nltk.corpus.brown` - Text categorized by genre
- Reuters
`nltk.corpus.reuters` - News Corpus
- Words
`nltk.corpus.words` - English Vocabulary
- SentiWordNet
`nltk.corpus.sentiwordnet` - Sentiment polarities mapped over WordNet structure

GUTTENBERG CORPUS

- 'austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt',
'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt',
'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt',
'edgeworth-parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt',
'shakespeare-caesar.txt', 'shakespeare-hamlet.txt',
'shakespeare-macbeth.txt', 'whitman-leaves.txt'

AUSTEN-EMMA.TXT

- 제인 오스틴(Jane Austen)
- 1816년에 집필한 소설
- 유명한 영국의 소설가
- 1775년 12월 16일 - 1817년 7월 18일
- 섬세한 시선과 재치있는 문체로 18세기 영국 중·상류층 여성들의 삶을 다루는 것이 특징
- 생전에는 그리 유명하지 않았으나, 20세기에 들어와서는 작품 중 《오만과 편견》, 《이성과 감성》등은 여러 번 영화화되는 등 인기를 끌고 있다.



Universal History Archive via Getty Images

토큰 생성

- 자연어 분석을 위해서는 긴 문자열을 작은 단위로 분할
- 분할된 작은 문자열을 토큰(token)이라고 부른다.
- 토큰으로 나누는 작업은 토큰 생성(tokenizing)이라고 부른다.

TOKENIZER

- **simple**
split() 함수 기반으로 동작하는 형태소 분리 모듈
- **word_tokenize()**
단어 단위로 형태소를 분리하는 함수
- **sent_tokenize()**
문장 단위로 형태소를 분리하는 함수
- **RegexpTokenizer()**
정규표현식을 사용해서 형태소를 분리하는 클래스

형태소 분석

- morphological analysis
- 형태소(morpheme)
언어학에서 일정한 의미가 있는 가장 작은 말의 단위
- 보통 자연어 처리에서는 토큰으로 형태소를 이용한다.
- 단어로부터 어근, 접두사, 접미사, 품사 등 다양한 언어적 속성을 파악하고 이를 이용하여 형태소를 찾아내거나 처리하는 작업
- 형태소 분석의 예
 - 어간 추출(stemming)
 - 원형 복원(lemmatizing)
 - 품사 부착(Part-Of-Speech tagging)

어간 추출

- 어간 추출(stemming)은 여러가지 이유로 변화된 단어의 접미사나 어미를 제거하여 같은 의미를 가지는 형태소의 실제 형태를 동일하게 만드는 방법이다. (Stemmers remove morphological affixes from words, leaving only the word stem.)
- NLTK는 PorterStemmer LancasterStemmer 등을 제공한다.
- 자세한 어간 추출 알고리즘은 다음 웹사이트를 참고한다.
<http://snowball.tartarus.org/algorithms/porter/stemmer.html>
- 어간 추출은 원형 복원(lemmatizing)의 일종이다.
- 원형 복원은 같은 의미를 가지는 여러 단어를
가장 근본적인 형태, 즉 사전형으로 통일하는 작업이다.

POS TAGGING

- 품사(POS, part-of-speech)는 낱말을 문법적인 기능이나 형태, 뜻에 따라 구분한 것이다.
- 품사의 구분은 언어마다 그리고 학자마다 다르다.
- NLTK에서는 펜 트리뱅크 태그세트(Penn Treebank Tagset)라는 것을 이용한다.
- 다음은 펜 트리뱅크 태그세트에서 사용하는 품사의 예이다.
 - NN 명사(단수형 혹은 집합형)
 - PRP 인칭대명사
 - CD 서수
 - DT 관형사
 - VBP 동사 현재형

품사 태그셋 (영어)

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP\$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP\$	Possessive wh-pronoun
WRB	Wh-adverb

국내 태그셋

- "21세기 세종계획 품사 태그세트"를 비롯하여 다양한 품사 태그세트가 있다.
- 세종계획 품사 태그세트의 자세한 내용은 "(21세기 세종계획)국어 기초자료 구축" 보고서의 "어절 분석 표지 표준안"을 참조한다.

언어 종류

- 고립어

대부분의 형태소가 그 자체로 낱말이 되는 언어인데, 중국어, 베트남어 등이 이에 속한다.
한자 하나가 나, 친구, 물, 말 등의 단어를 뜻하는 경우이다.

예) 중국어

- 굴절어

종합어의 한 분류로서, 한 문장 속의 기능에 따라 단어 형태 자체가 변한다.

예) 영어

- 교착어

굴절어와 고립어의 중간 단계이다. 단어의 중심이 되는 형태소에 접두사/접미사와 다른 형태소들이 덧붙어 단어가 구성된다.

예) 한국어

언어 사례

- 고립어 : 我, 我的, 모양이 변하는 일 없다.
 - 굴절어 : I, Me, My, 모양 자체가 변한다.
 - 교착어 : 나는, 내가, 나를, 나의, 어간+어미 = 명사+조사
-
- 고립어 : 去, 去了, 모양이 변하는 일 없다.
 - 굴절어 : Go, went, 모양 자체가 변한다.
 - 교착어 : 가다, 갔다, 어간+어미

나이브 베이즈

- 언어 모델에 쉽게 적용할 수 있는 머신러닝 알고리즘의 하나
- 기계 학습분야에서, '나이브 베이즈 분류(Naïve Bayes Classification)'는 특성들 사이의 독립을 가정하는 베이즈 정리를 적용한 확률 분류기의 일종으로 1950년대 이후 광범위하게 연구되고 있다.
- 통계 및 컴퓨터 과학 문헌에서, 나이브 베이즈는 단순 베이즈, 독립 베이즈를 포함한 다양한 이름으로 알려져 있으며, 1960년대 초에 텍스트 검색 커뮤니티에 다른 이름으로 소개되기도 하였다.
- 나이브 베이즈 분류는 텍스트 분류에 사용됨으로써 문서를 여러 범주 (예: 스팸, 스포츠, 정치) 중 하나로 판단하는 문제에 대한 대중적인 방법으로 남아있다. 또한, 자동 의료 진단 분야에서의 응용사례를 보면, 적절한 전처리를 하면 더 진보된 방법들 (예: 서포트 벡터 머신 (Support Vector Machine))과도 충분한 경쟁력을 보임을 알 수 있다.

나이브 베이즈

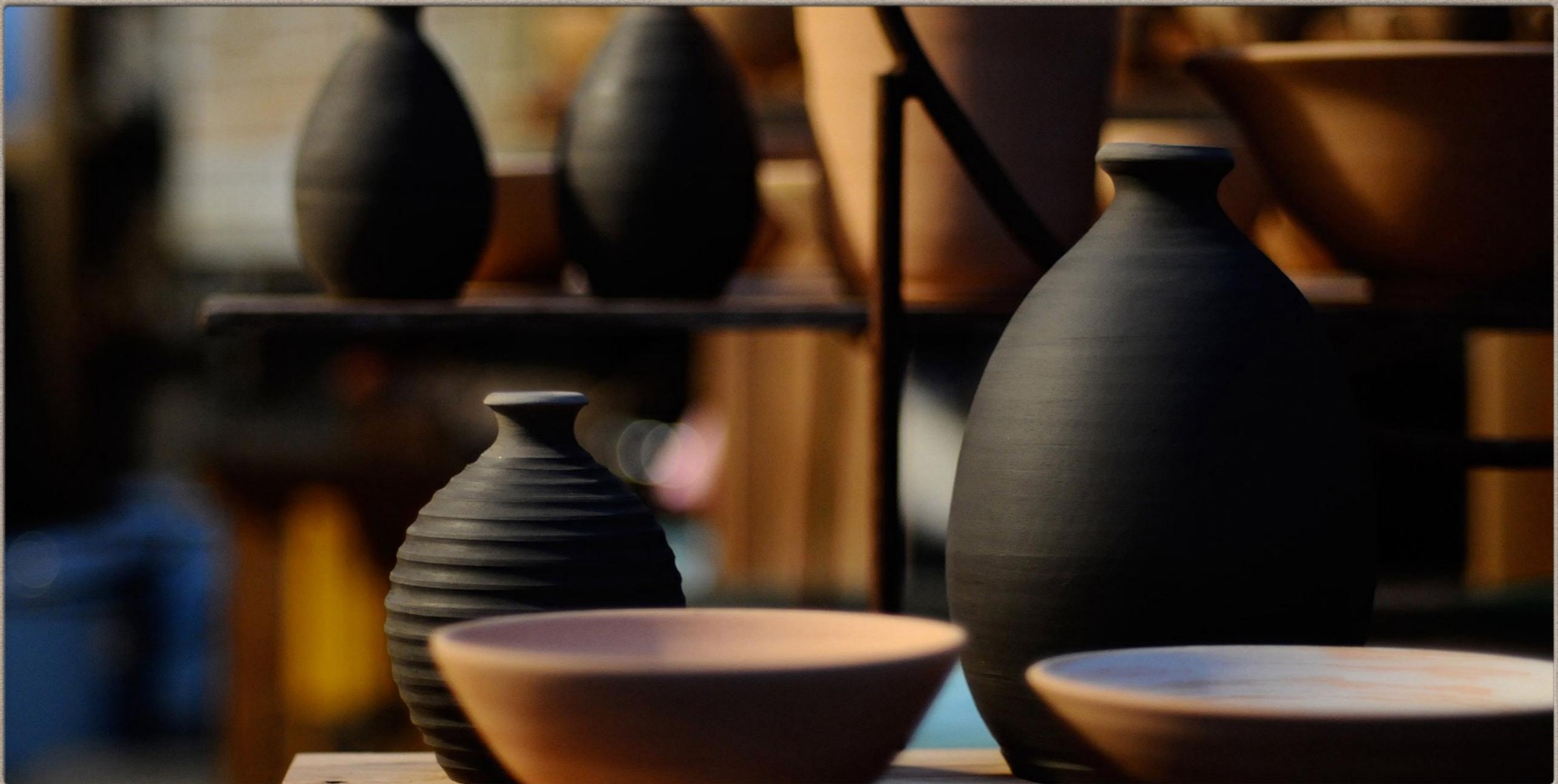
- 나이브 베이즈는 분류기를 만들 수 있는 간단한 기술로써 단일 알고리즘을 통한 훈련이 아닌 일반적인 원칙에 근거한 여러 알고리즘들을 이용하여 훈련된다. 모든 나이브 베이즈 분류기는 공통적으로 모든 특성 값은 서로 독립임을 가정한다. 예를 들어, 특정 과일을 사과로 분류 가능하게 하는 특성들 (둥글다, 빨갛다, 지름 10cm)은 나이브 베이즈 분류기에서 특성들 사이에서 발생할 수 있는 연관성이 없음을 가정하고 각각의 특성들이 특정 과일이 사과일 확률에 독립적으로 기여 하는 것으로 간주한다.
- 나이브 베이즈의 장점은 다음과 같다. 첫째, 일부의 확률 모델에서 나이브 베이즈 분류는지도 학습 (Supervised Learning) 환경에서 매우 효율적으로 훈련 될 수 있다. 많은 실제 응용에서, 나이브 베이즈 모델의 파라미터 추정은 최대우도방법 (Maximum Likelihood Estimation (MLE))을 사용하며, 베이즈 확률론이나 베이지안 방법들은 이용하지 않고도 훈련이 가능하다. 둘째, 분류에 필요한 파라미터를 추정하기 위한 트레이닝 데이터의 양이 매우 적다는 것이다. 셋째, 간단한 디자인과 단순한 가정에도 불구하고, 나이브 베이즈 분류는 많은 복잡한 실제 상황에서 잘 작동한다. 2004년의 한 분석은 나이브 베이즈 분류의 이러한 능력에 명확한 이론적인 이유가 있음을 보여 주었다. 또한 2006년에는 다른 분류 알고리즘과의 포괄적인 비교를 통하여 베이지안 분류는 부스트 트리 또는 랜덤 포레스트와 같은 다른 접근 방식을 넘어섰다는 것이 밝혀졌다.

나이브 베이즈

```
train_set, test_set = make_train_test(labeled_names)
```

```
clf = nltk.NaiveBayesClassifier.train(train_set)
```

```
acc = nltk.classify.accuracy(clf, test_set)
```



KoNLPy

Korean NLP in Python

KoNLPy

- KoNLPy is a Python package for Korean natural language processing.
한국어 정보처리를 위한 파이썬 패키지
- 지원 기능
 - 한국어 말뭉치
kolaw, kobill
 - 한국어 처리 유틸리티
한글이 리스트나 딕셔너리의 내부에 있을 때도 한글 글자 모양을 정상적으로 보여주는 pprint 유틸리티 함수 제공.
 - 형태소 분석 및 품사 태깅
tag 서브패키지에서 형태소 분석을 위한 5개의 클래스 제공
morphs(), nouns(), pos()

KoNLPy 말뭉치

- kolaw
 - 한국 법률 말뭉치
 - constitution.txt
- kobill
 - 대한민국 국회 의안 말뭉치. 파일 ID는 의안 번호를 의미
 - 1809890.txt - 1809899.txt
- kolaw 헌법 파일 경로
`/usr/local/Cellar/python3/3.6.3/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/konlpy/data/corpus/kolaw/constitution.txt`

KoNLPy 지원 형태소 분석기

- 사전은 대부분 말뭉치를 이용해 구축되었으며 형태소 분석 및 품사 태깅에 사용됨.
- Kkma
<http://kkma.snu.ac.kr/>
- Hannanum
<http://semanticweb.kaist.ac.kr/hannanum/>
- Twitter
<https://github.com/twitter/twitter-korean-text/>
- Komoran
http://www.shineware.co.kr/?page_id=835
- Mecab
<https://bitbucket.org/eunjeon/mecab-ko-dic>

형태소 분석기 성능 비교

- Kkma: 5.6988 secs
- Komoran: 5.4866 secs
- Hannanum: 0.6591 secs
- Twitter: 1.4870 secs
- Mecab: 0.0007 secs
- 실행시간: 10만 문자의 문서를 대상으로 각 클래스의 pos 메소드를 실행하는데 소요되는 시간.

형태소 분석기 성능 비교

- Kkma: 35.7163 secs
- Komoran: 25.6008 secs
- Hannanum: 8.8251 secs
- Twitter: 2.4714 secs
- Mecab: 0.2838 secs
- 문자의 개수를 늘려감에 따라 모든 클래스의 실행 시간은 기하급수적으로 증가합니다.

Hannanum 형태소 분석기

- 한나눔 시스템 사전은 KAIST 말뭉치를 이용해 생성 (4.7MB)

- 위치

./konlpy/java/data/kE/dic_system.txt

- 예시

나라경제 ncn

나라기획 nqq

나라기획회장 ncn

나라꽃 ncn

나라님 ncn

나라도둑 ncn

나라따르 pvg

- 사용자 사전에 새로운 항목을 추가하려면 ./konlpy/java/data/kE/dic_user.txt 수정

Kkma 형태소 분석기

- Kkma 시스템 사전은 세종 말뭉치를 이용해 생성 (32MB)
- 위치
꼬꼬마 형태소 분석기의 .jar 파일 안에 위치. 사전 파일을 직접 보기 위해서는 꼬꼬마 미러 사용
- 예시
아니/IC
후우/IC
그래서/MAC
그러나/MAC
그러니까/MAC
그러면/MAC
그러므로/MAC

MeCab-ko 형태소 분석기

- Mecab 시스템 사전은 세종 말뭉치로 만들어진 CSV 형태의 사전 (346MB)
- 위치
컴파일된 사전은 /usr/local/lib/mecab/dic/mecab-ko-dic (또는 MeCab 설치시 지정 경로)
- 예시

가오토,0,0,0,NNG,* ,F,가오토,* ,*,*,*,*

갑툭튀,0,0,0,NNG,* ,F,갑툭튀,* ,*,*,*,*

강퇴,0,0,0,NNG,* ,F,강퇴,* ,*,*,*,*

개드립,0,0,0,NNG,* ,T,개드립,* ,*,*,*,*

갠소,0,0,0,NNG,* ,F,갠소,* ,*,*,*,*

고퀄,0,0,0,NNG,* ,T,고퀄,* ,*,*,*,*

광삭,0,0,0,NNG,* ,T,광삭,* ,*,*,*,*
- KoNLPy의 Mecab() 클래스는 윈도우에서 지원되지 않습니다.

MeCab

- 오픈소스 형태소 분석 엔진 MeCab(메카브)
- MeCab, originally a Japanese morphological analyzer and POS tagger developed by the Graduate School of Informatics in Kyoto University, was modified to MeCab-ko by the Eunjeon Project to adapt to the Korean language.
- 학습용 데이터와 분석 사전을 준비하면 다른 언어 분석 가능
- MeCab를 이용한 한국어 형태소 분석
<http://porocise.sakura.ne.jp/wiki/korean/mecab.ko>
- 한국어 버전
<https://bitbucket.org/eunjeon/mecab-ko>

MeCab 도구

- **한국어 학습 보조 도구**

입력된 현대한국어 문장을 분석하여 단어의 학습 수준을 표시하거나 한자 표기로 변환하는 등 한국어 학습에 도움이 되는 정보를 제공합니다. 단어마다 웹 사전으로의 링크를 생성할 수도 있습니다.

<http://porocise.sakura.ne.jp/korean/mecab/main.html>

- **형태소 분석 / 한자 표기 변환**

현대한국어 문장을 분석하여 품사를 표시하거나 문장 중의 한자어를 한자 표기로 바꿔 줍니다.

<http://porocise.sakura.ne.jp/korean/mecab/analyizer.html>

- **히라가나의 한글 전사**

입력된 히라가나 표기를 한글로 전사합니다. 또한 한자가 섞인 일본어 문장도 MeCab로 분석한 다음에 한글로 전사할 수 있습니다.

<http://porocise.sakura.ne.jp/korean/hira2han/index.html>

MeCab 콘솔

1. 터미널에서 직접 사용할 수 있다.
2. mecab 명령만 입력하면 형태소 분석 모드로 들어간다.
3. 분석하고 싶은 문장 쓰고 엔터. 분석 결과를 보여주고 EOS 출력
4. 도움말 mecab -h
5. 분석 모드를 탈출하는 명령은 없다. 종료는 ctrl + c.

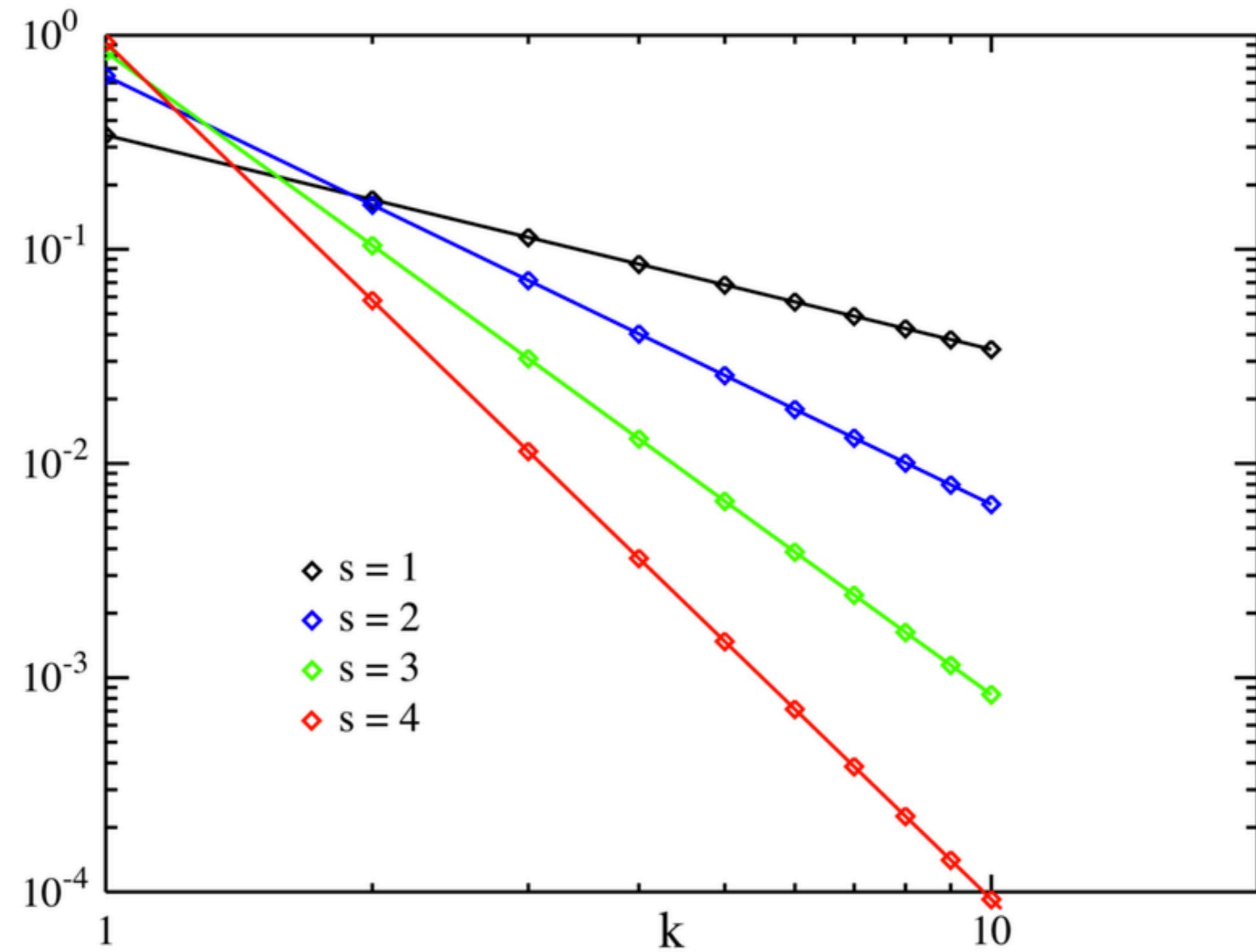
MeCab test

```
import konlpy  
  
mecab = konlpy.tag.Mecab()  
  
print(mecab.morphs('영등포구청역에 있는 맛집 좀 알려주세요.'))  
# ['영등포구', '청역', '에', '있', '는', '맛집', '좀', '알려', '주', '세요', '.']  
  
print(mecab.nouns('우리나라에는 무릎 치료를 잘하는 정형외과가 없는가!'))  
# ['우리', '나라', '무릎', '치료', '정형외과']  
  
print(mecab.pos('자연주의 쇼핑몰은 어떤 곳인가?'))  
# [('자연', 'NNG'), ('주', 'NNG'), ('의', 'JKG'), ('쇼핑몰', 'NNG'), ('은', 'JX'), ('어떤', 'MM'),  
('곳', 'NNG'), ('인가', 'VCP+EF'), ('?', 'SF')]
```

지프의 법칙(Zipf's law)

- 수학적 통계를 바탕으로 밝혀진 경험적 법칙
- 물리 및 사회과학 분야에서 연구된 많은 종류의 정보들이 지프 분포에 가까운 경향을 보이는 것을 뜻한다.
- 지프 분포는 이산 벡터법칙 확률분포와 관계된 확률분포의 하나이다.
- 미국의 언어학자인 조지 킹슬리 지프가 최초로 이 법칙을 공식 제안(Zipf 1935, 1949).
- 지프의 법칙에 따르면 어떠한 자연어 말뭉치 표현에 나타나는 단어들을 그 사용 빈도가 높은 순서대로 나열하였을 때, 모든 단어의 사용 빈도는 해당 단어의 순위에 반비례한다. 따라서 가장 사용 빈도가 높은 단어는 두 번째 단어보다 빈도가 약 두 배 높으며, 세 번째 단어보다는 빈도가 세 배 높다. 예를 들어, 브라운 대학교 현대 미국 영어 표준 말뭉치의 경우, 가장 사용 빈도가 높은 단어는 영어 정관사 "the"이며 전체 문서에서 7%의 빈도(약 백만 개 남짓의 전체 사용 단어 중 69,971회)를 차지한다. 두 번째로 사용 빈도가 높은 단어는 "of"로 약 3.5% 남짓(36,411회)한 빈도를 차지하며, 세 번째로 사용 빈도가 높은 단어는 "and"(28,852회)로, 지프의 법칙에 정확히 들어 맞는다. 약 135개 항목의 어휘만으로 브라운 대학 말뭉치의 절반을 나타낼 수 있다.
- 지프의 법칙은 데이터의 순위와 빈도를 각 축에 로그 스케일로 나타낸 그래프를 통해 쉽게 확인할 수 있다.

지프의 법칙(Zipf's law)



세종 태그셋 (1)

- 한글 형태소 품사(POS) 태그표

<http://kkma.snu.ac.kr/documents/?doc=postag>

대분류	세종 품사 태그		심광섭 품사 태그		KKMA 단일 태그 V 1.0						확률태 그	저장사전
	태그	설명	Class	설명	묶음 1	묶음 2	태그	설명				
체언	NNG	일반 명사	NN	명사	N	NN	NNG	보통 명사		NNA	noun.dic	
	NNP	고유 명사					NNP	고유 명사				
	NNB	의존 명사	NX	의존 명사			NNB	일반 의존 명사		NNB	simple.dic	
				단위 명사			NNM	단위 의존 명사				
	NR	수사	NU	수사		NR	NR	수사		NR		
용언	NP	대명사	NP	대명사		NP	NP	대명사		NP		
	VV	동사	VV	동사	V	VV	VV	동사		VV	verb.dic	
	VA	형용사	AJ	형용사		VA	VA	형용사		VA		
	VX	보조 용언	VX	보조 동사		VX	VXV	보조 동사		VX		
			AX	보조 형용사		VXA	VXA	보조 형용사		VXA		
관형사	VCP	긍정 지정사	CP	서술격 조사 '이다'	M	VC	VCP	긍정 지정사, 서술격 조사 '이다'		VCP	raw.dic	
	VCN	부정 지정사				VC	VCN	부정 지정사, 형용사 '아니다'		VCN		
	MM	관형사	DT	일반 관형사		MD	MDT	일반 관형사		MD		
			DN	수 관형사		MD	MDN	수 관형사		MD		
부사	MAG	일반 부사	AD	부사		MA	MAG	일반 부사		MAG	simple.dic	
	MAJ	접속 부사				MA	MAC	접속 부사		MAC		

세종 태그셋 (2)

감탄사	IC	감탄사	EX	감탄사	I	IC	IC	감탄사	IC				
조사	JKS	주격 조사	JO	조사	J	JK	JKS	주격 조사	JKS	raw.dic			
	JKC	보격 조사					JKC	보격 조사	JKC				
	JKG	관형격 조사					JKG	관형격 조사	JKG				
	JKO	목적격 조사					JKO	목적격 조사	JKO				
	JKB	부사격 조사					JKM	부사격 조사	JKM				
	JKV	호격 조사					JKI	호격 조사	JKI				
	JKQ	인용격 조사					JKQ	인용격 조사	JKQ				
	JX	보조사					JX	보조사	JX				
	JC	접속 조사					JC	접속 조사	JC				
선어말 어미	EP	선어말 어미	EP	선어말 어미	E	EP	EPH	존칭 선어말 어미	EP	raw.dic			
어말 어미	EF	종결 어미	EM	어말 어미			EPT	시제 선어말 어미					
	EC	연결 어미					EPP	공손 선어말 어미					
	ETN	명사형 전성 어미		EF		EFN	평서형 종결 어미	EF					
	ETM	관형형 전성 어미				EFQ	의문형 종결 어미						
						EFO	명령형 종결 어미						
어말 어미			EM			어말 어미					EFA	청유형 종결 어미	
											EFI	감탄형 종결 어미	
											EFR	존칭형 종결 어미	
				EC		ECE	대등 연결 어미	EC					
							ECD				의존적 연결 어미		
							ECS				보조적 연결 어미		
						ET	ETN	명사형 전성 어미	ETN				
							ETD	관형형 전성 어미	ETD				

세종 태그셋 (3)

접두사	XPN	체언 접두사	PF	접두사	X	XP	XPN	체언 접두사	XPN	simple.dic	
							XPV	용언 접두사	XPV		
접미사	XSN	명사 파생 접미사	SN	명사화 접미사		XS	XSN	명사 파생 접미사	XSN		
	XSV	동사 파생 접미사	SV	동사화 접미사			XSV	동사 파생 접미사	XSV		
	XSA	형용사 파생 접미사	SJ	형용사화 접미사			XSA	형용사 파생 접미사	XSA		
			SA	부사화 접미사			XSM	부사 파생 접미사	XSM		
			SF	기타 접미사			XSO	기타 접미사	XSO		
어근	XR	어근	XR			XR	XR	어근	XR		
부호	SF	마침표물음표,느낌표	SY	부호 의래어		SF	SF	마침표물음표,느낌표	SF	Symbol class	
	SP	쉼표,가운뎃점,콜론,빗금				SP	SP	쉼표,가운뎃점,콜론,빗금	SP		
	SS	따옴표,괄호표,줄표				SS	SS	따옴표,괄호표,줄표	SS		
	SE	줄임표				SE	SE	줄임표	SE		
	SO	불임표(물결,숨김,빠짐)				SO	SO	불임표(물결,숨김,빠짐)	SO		
	SW	기타기호 (논리수학기호,화폐기호)				SW	SW	기타기호 (논리수학기호,화폐기호)	SW		
분석 불능	NF	명사추정범주	NR	미등록어	U	UN	UN	명사추정범주	NNA	N/A	
	NV	용언추정범주				UV	UV	용언추정범주	N/A		
	NA	분석불능범주				UE	UE	분석불능범주	N/A		
한글 이외	SL	외국어			O	OL	OL	외국어	NNA		
	SH	한자				OH	OH	한자	NNA		
	SN	숫자				ON	ON	숫자	NR		



gensim
NLP Library

개요

- gensim은 자연어 처리 라이브러리
- 자연어 처리에서 일반적인 기능을 갖춘 nltk에 비해 토픽 모델링에 특화
- word embedding
 - 딥러닝 기반의 자연어 처리의 기본
 - word2vec, glove 등 다양한 알고리즘 존재
 - tensorflow를 사용해서 직접 구현 가능
 - 실제 사용은 word2vec에서만 큼은 케라스보다 쉽고 유용
 - clustering, cosine similarity 등에서 압도적인 우위

개요

- Gensim
 - Open-source vector space modeling and topic modeling toolkit implemented in Python
 - It uses NumPy, SciPy and optionally Cython for performance.
 - Specifically designed to handle large text collections, using data streaming and efficient incremental algorithms
 - It differentiates from most other scientific software packages that only target batch and in-memory processing
- Implementations
 - tf-idf
 - random projections
 - word2vec
 - document2vec
 - hierarchical Dirichlet processes (HDP)
 - latent semantic analysis (LSA, LSI, SVD)
 - latent Dirichlet allocation (LDA)



RNN

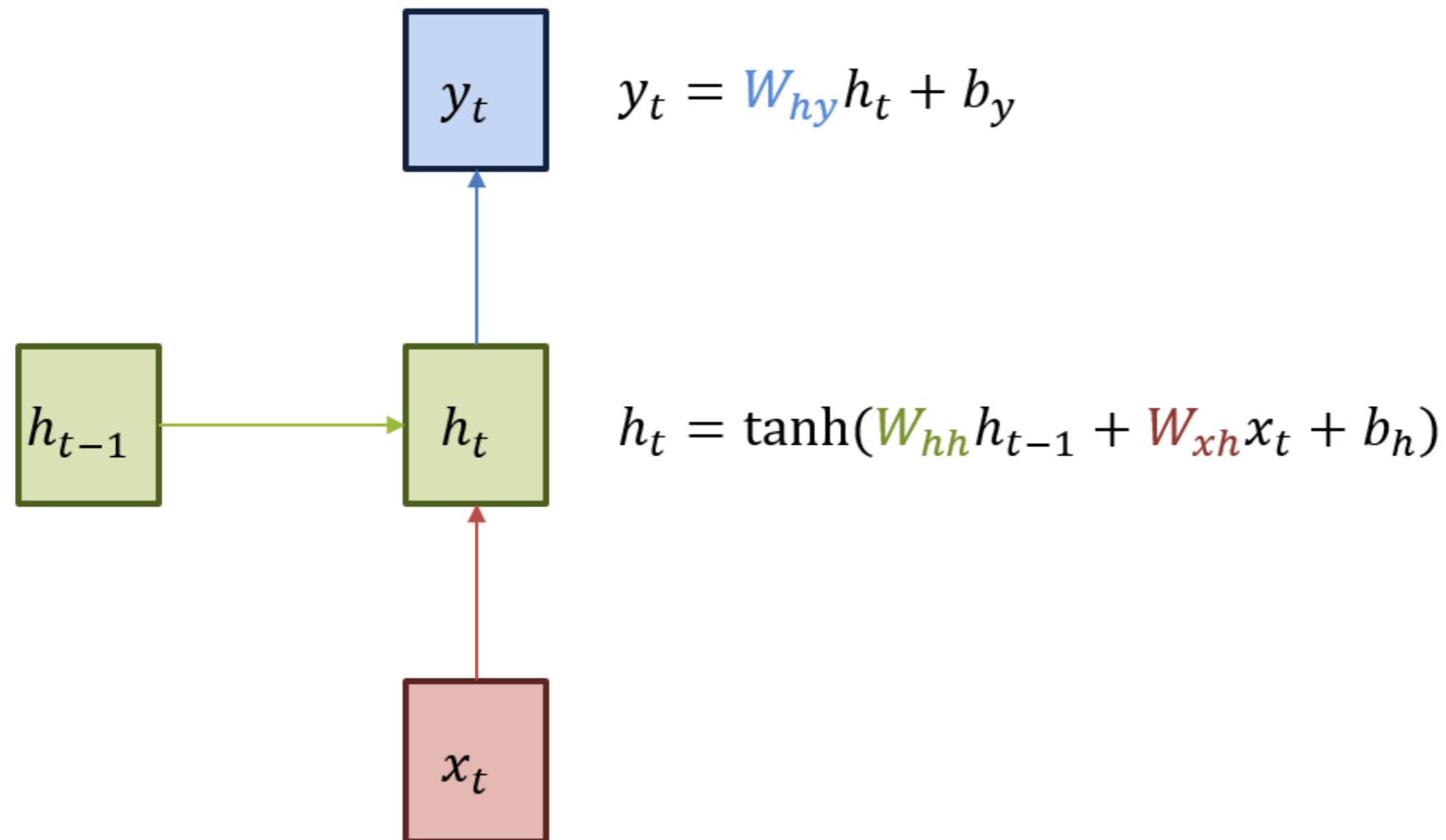
Recurrent, LSTM, GRU

OVERVIEW

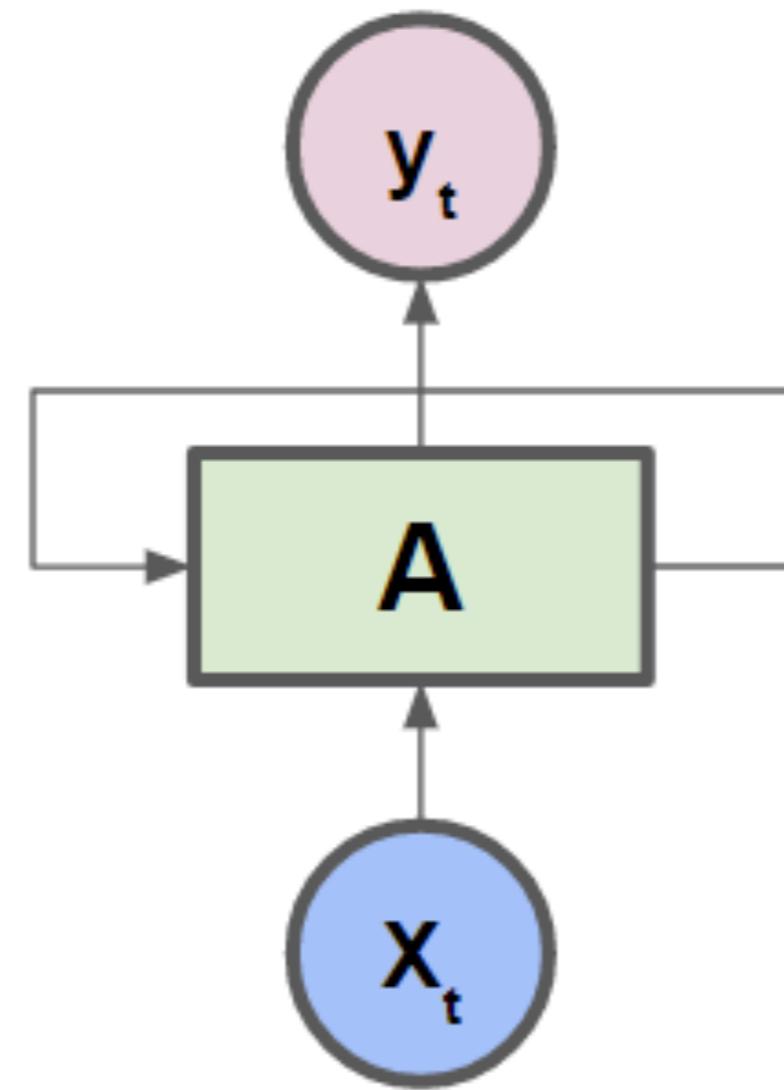
- Recurrent Neural Network
- 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류
- 음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델
- Convolutional Neural Networks(CNN)과 더불어 최근 들어 각광 받고 있는 알고리즘
- 시퀀스 길이에 관계없이 입력과 출력을 받아들일 수 있는 네트워크 구조
- 필요에 따라 다양하고 유연하게 구조를 만들 수 있다는 점이 RNN의 가장 큰 장점

RNN 구조

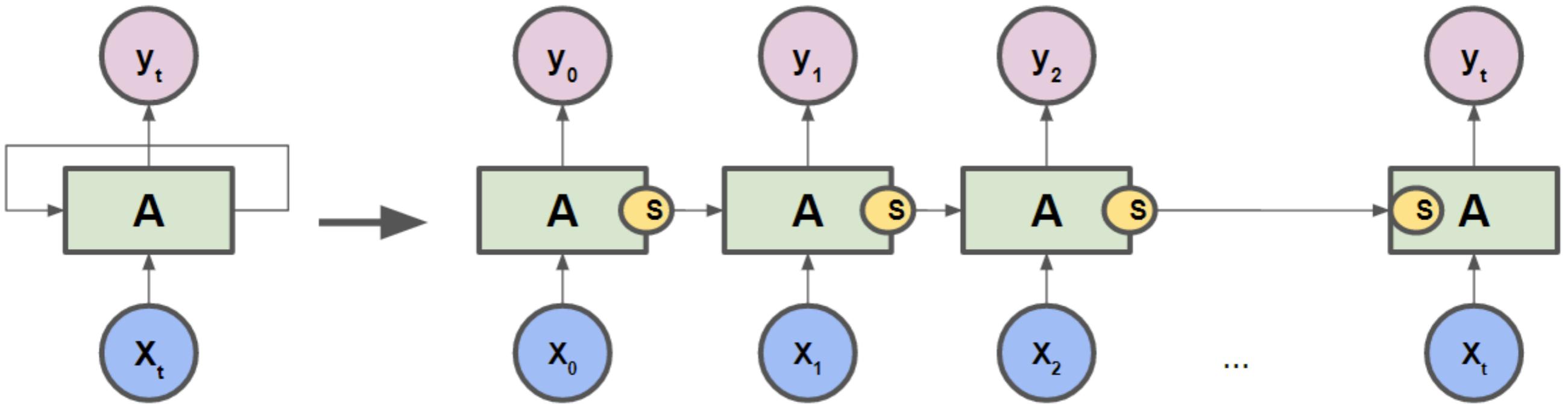
- 녹색 박스는 히든 state를 의미. 빨간 박스는 인풋 x , 파란 박스는 아웃풋 y .
현재 상태의 히든 state h_t 는 직전 시점의 히든 state h_{t-1} 를 받아 갱신됩니다.
- 현재 상태의 아웃풋 y_t 는 h_t 를 전달받아 갱신되는 구조입니다.
히든 state의 활성함수는 비선형 함수인 하이퍼볼릭탄젠트(tanh)입니다.



RNN CELL



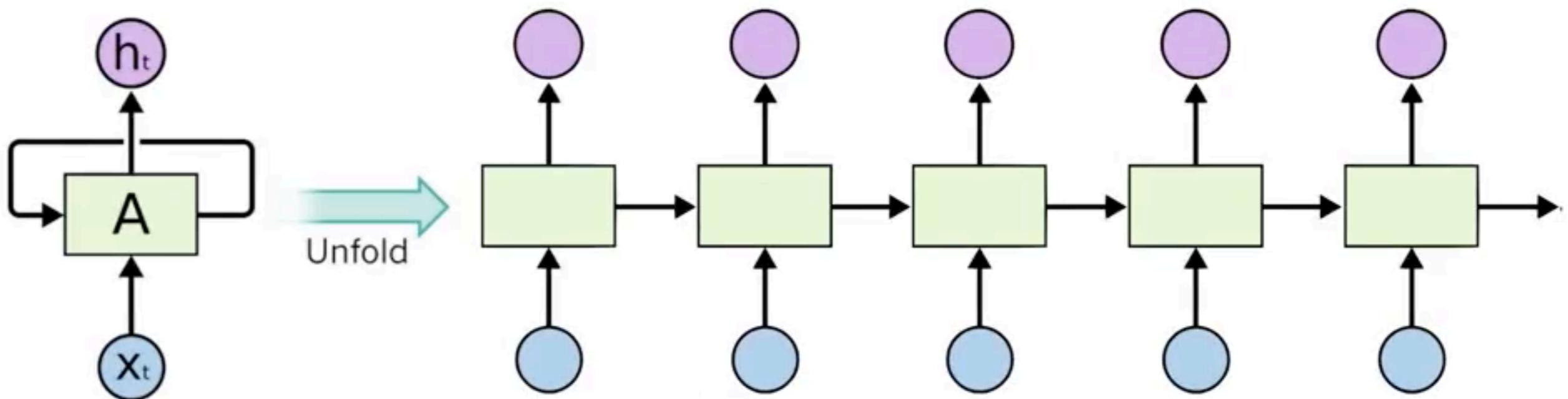
전개 : STATE



전개 : batch_size

sequence_length, hidden_size = 5, 2

shape=(1,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]]]

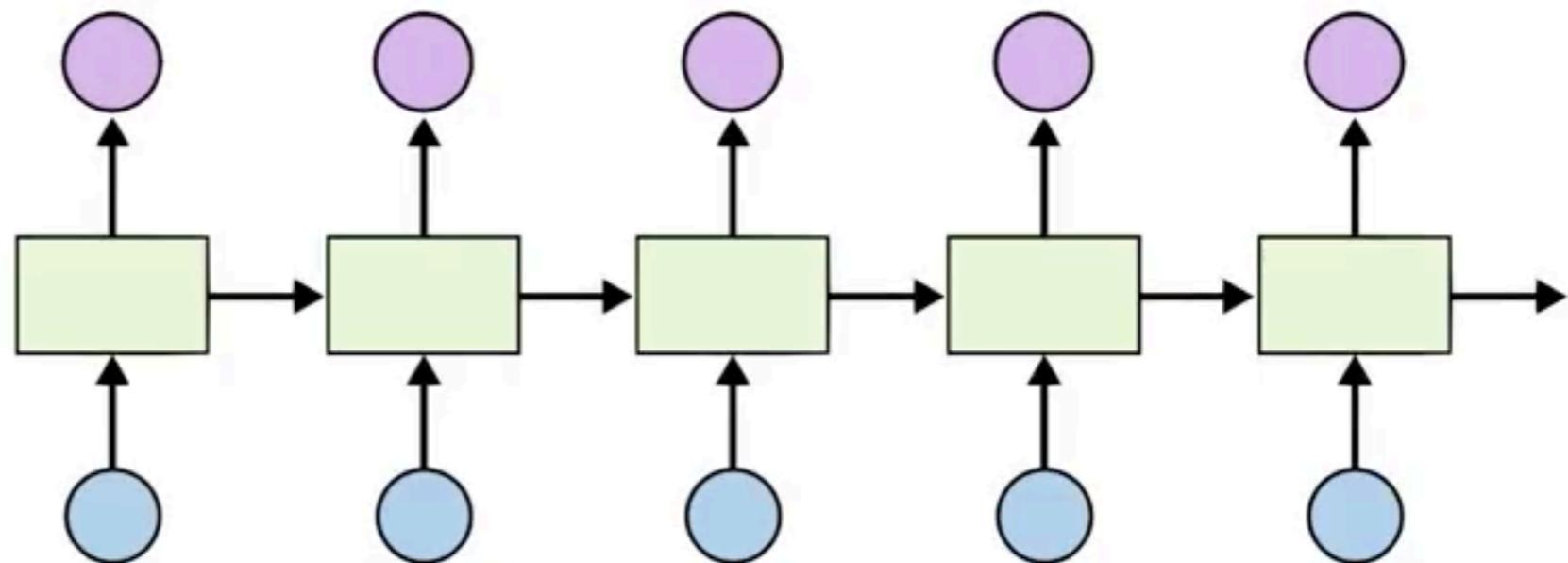


shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
h e l l o

전개 : batch_size

batch_size, sequence_length, hidden_size = 3, 5, 2

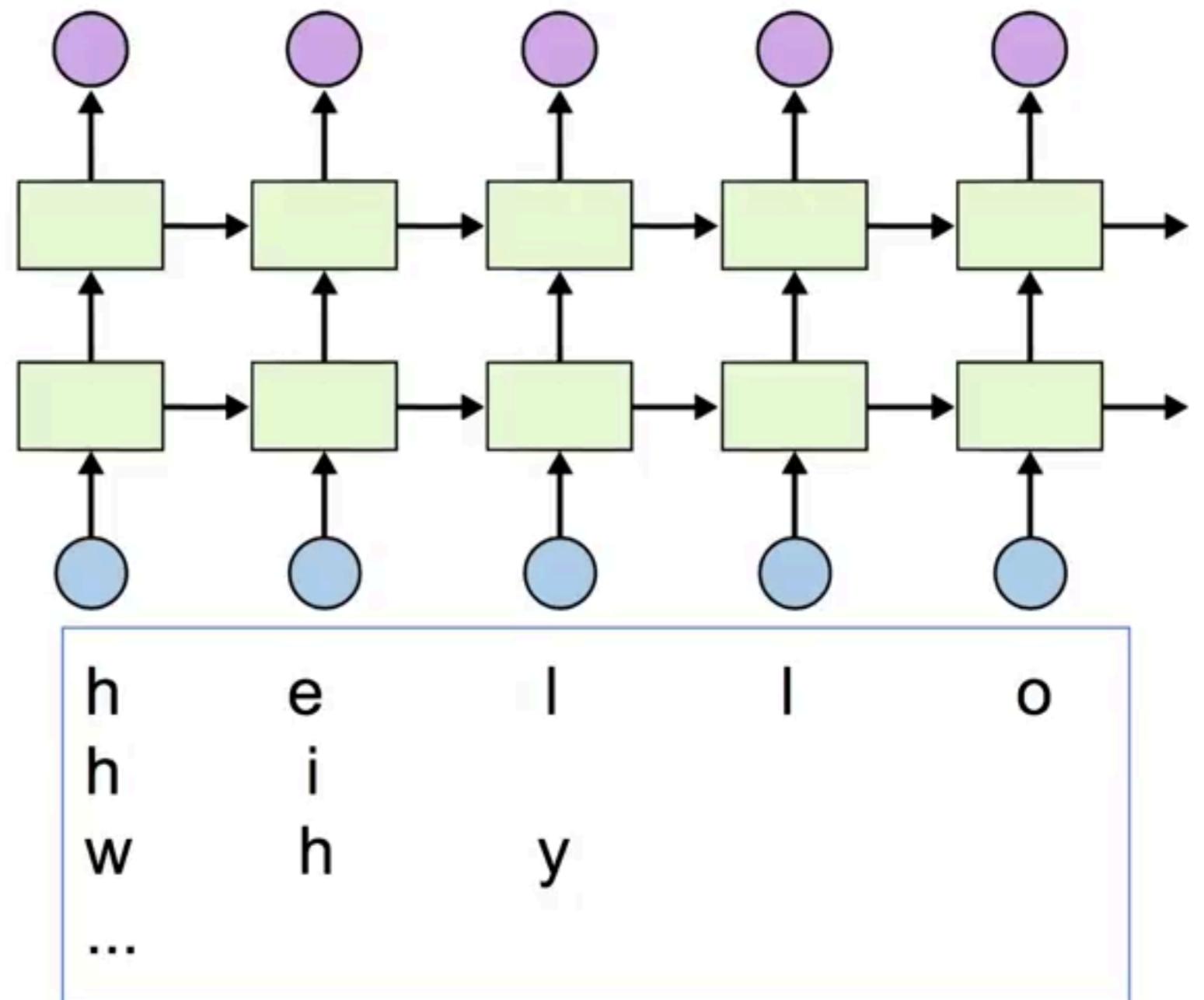
```
shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],  
[[[x,x], [x,x], [x,x], [x,x], [x,x]],  
[[x,x], [x,x], [x,x], [x,x], [x,x]]]
```



```
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello  
[[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]] # eol11  
[[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # 1leel
```

dynamic_rnn()

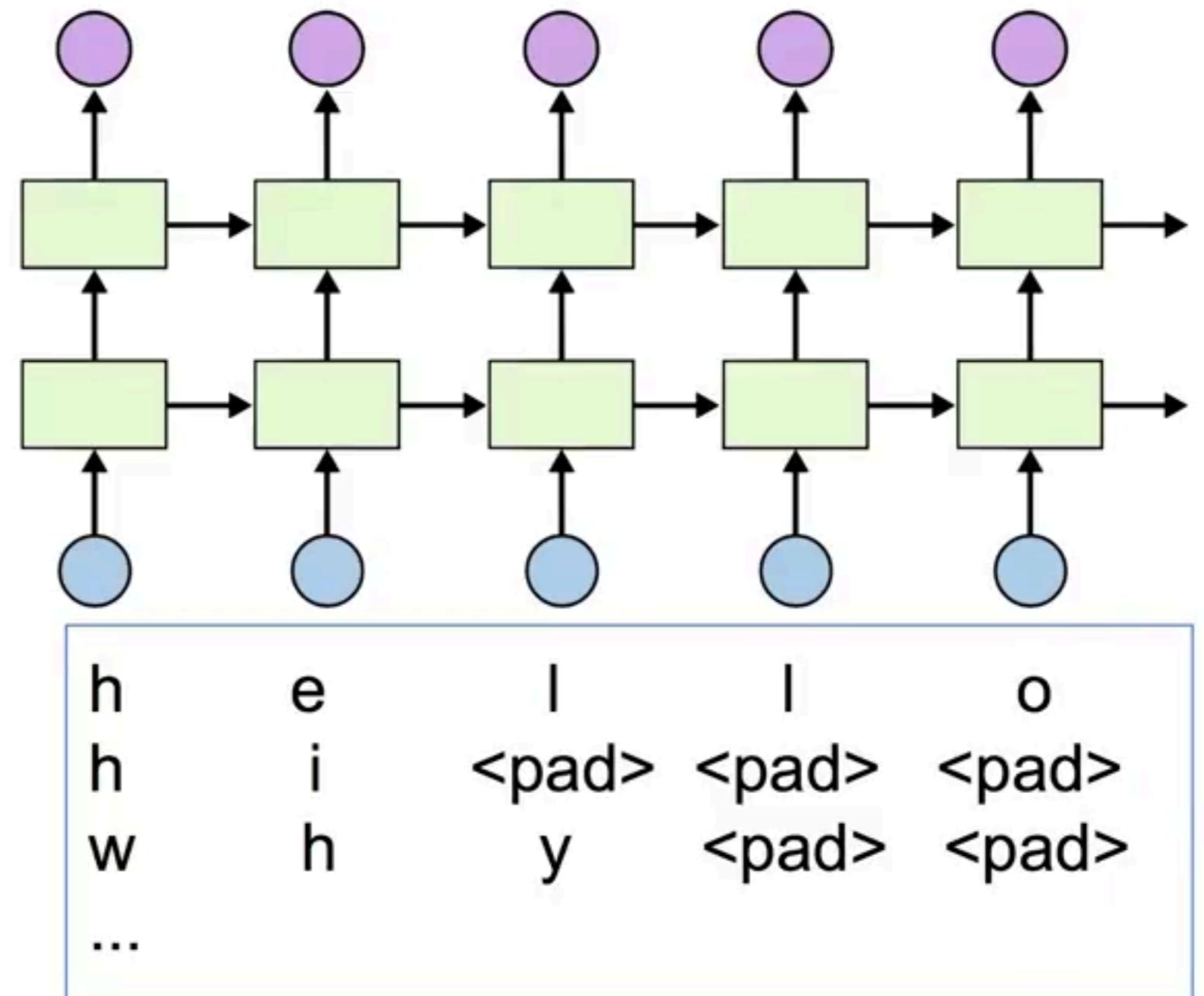
길이가 다른 문자열이
들어온다면?



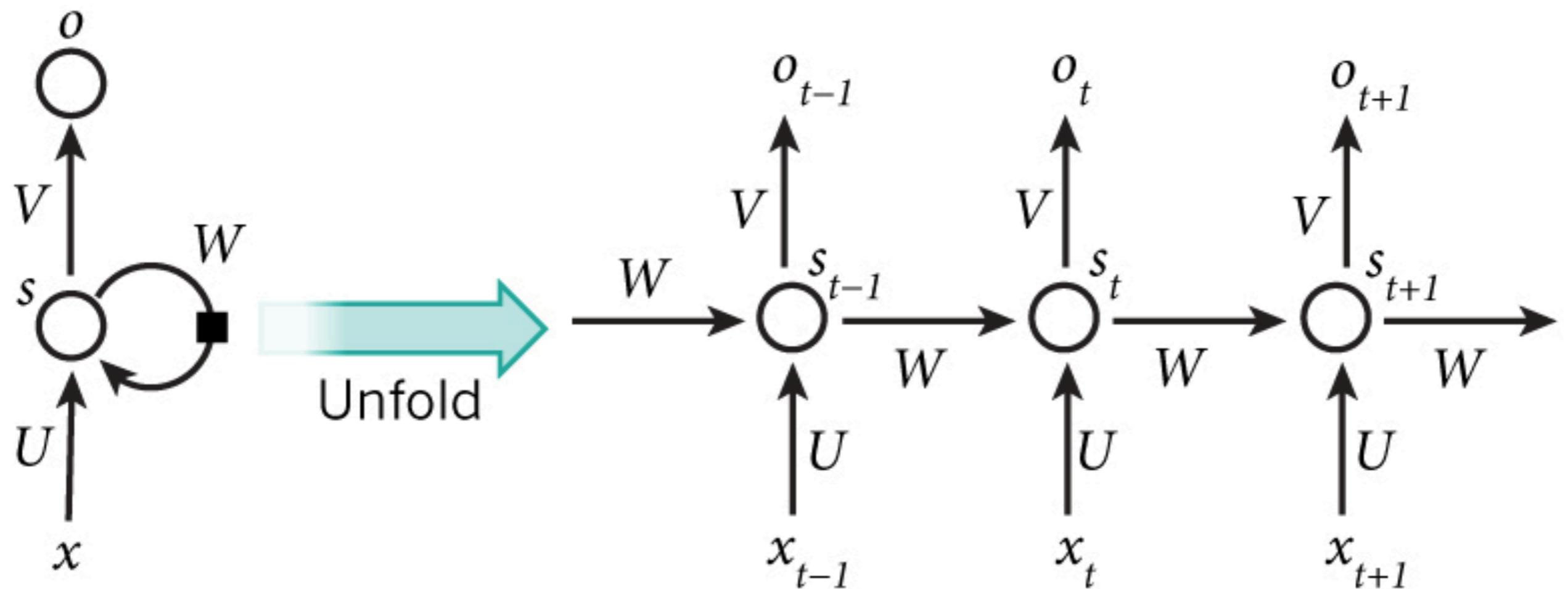
dynamic_rnn()

<pad>와 같은
식별자 추가

sequence_length
옵션으로 해결



UNFOLD



<http://aikorea.org/blog/rnn-tutorial-1>

UNFOLD

- x_t 는 시간 스텝(time step) t에서의 입력값이다.
- s_t 는 시간 스텝 t에서의 hidden state이다. 네트워크의 "메모리" 부분으로서, 이전 시간 스텝의 hidden state 값과 현재 시간 스텝의 입력값에 의해 계산된다:
$$s_t = f(Ux_t + Ws_{t-1})$$
.
비선형 함수 f 는 보통 tanh나 ReLU가 사용되고,
첫 hidden state를 계산하기 위한 s_{-1} 은 보통 0으로 초기화시킨다.
- o_t 는 시간 스텝 t에서의 출력값이다. 예를 들어,
문장에서 다음 단어를 추측하고 싶다면 단어 수만큼의 차원의 확률 벡터가 될 것이다.
$$o_t = \text{softmax}(Vs_t)$$

UNFOLD

- Hidden state s_t 는 네트워크의 메모리라고 생각할 수 있다. s_t 는 과거의 시간 스텝들에서 일어난 일들에 대한 정보를 전부 담고 있고, 출력값 o_t 는 오로지 현재 시간 스텝 t 의 메모리에만 의존한다. 하지만 위에서 잠깐 언급했듯이, 실제 구현에서는 너무 먼 과거에 일어난 일들은 잘 기억하지 못한다.
- 각 layer마다의 파라미터 값들이 전부 다 다른 기존의 deep한 신경망 구조와 달리, RNN은 모든 시간 스텝에 대해 파라미터 값을 전부 공유하고 있다 (위 그림의 U, V, W). 이는 RNN이 각 스텝마다 입력값만 다를 뿐 거의 똑같은 계산을 하고 있다는 것을 보여준다. 이는 학습해야 하는 파라미터 수를 많이 줄여준다.
- 위 다이어그램에서는 매 시간 스텝마다 출력값을 내지만, 문제에 따라 달라질 수도 있다. 예를 들어, 문장에서 긍정/부정적인 감정을 추측하고 싶다면 굳이 모든 단어 위치에 대해 추측값을 내지 않고 최종 추측값 하나만 내서 판단하는 것이 더 유용할 수도 있다. 마찬가지로, 입력값 역시 매 시간 스텝마다 꼭 다 필요한 것은 아니다. RNN에서의 핵심은 시퀀스 정보에 대해 어떠한 정보를 추출해 주는 hidden state이기 때문이다.

학습

- RNN 네트워크를 학습하는 것은 기존의 신경망 모델 학습과 매우 유사하다.
- 네트워크의 각 시간 스텝마다 파라미터들이 공유되기 때문에 펼쳐진 네트워크에서 기존의 backpropagation 알고리즘을 그대로 사용하진 못한다.
- Backpropagation Through Time (BPTT)라는 약간 변형된 알고리즘을 사용한다. 그 이유는, 각 출력 부분에서의 gradient가 현재 시간 스텝에만 의존하지 않고 이전 시간 스텝들에도 의존하기 때문이다. 즉, $t=4$ 에서의 gradient를 계산하기 위해서는 시간 스텝 3 개 이전부터 gradient를 전부 더해주어야 한다.
- vanishing/exploding gradient라는 문제 등에 의해서 단순한 RNN을 BPTT로 학습시키는 것은 긴 시퀀스를 다루기 어렵다. 이를 해결하기 위한 여러 트릭들이 존재하고, LSTM 등이 문제를 해결하기 위한 다양한 변종(확장된) RNN 모델들도 존재한다.

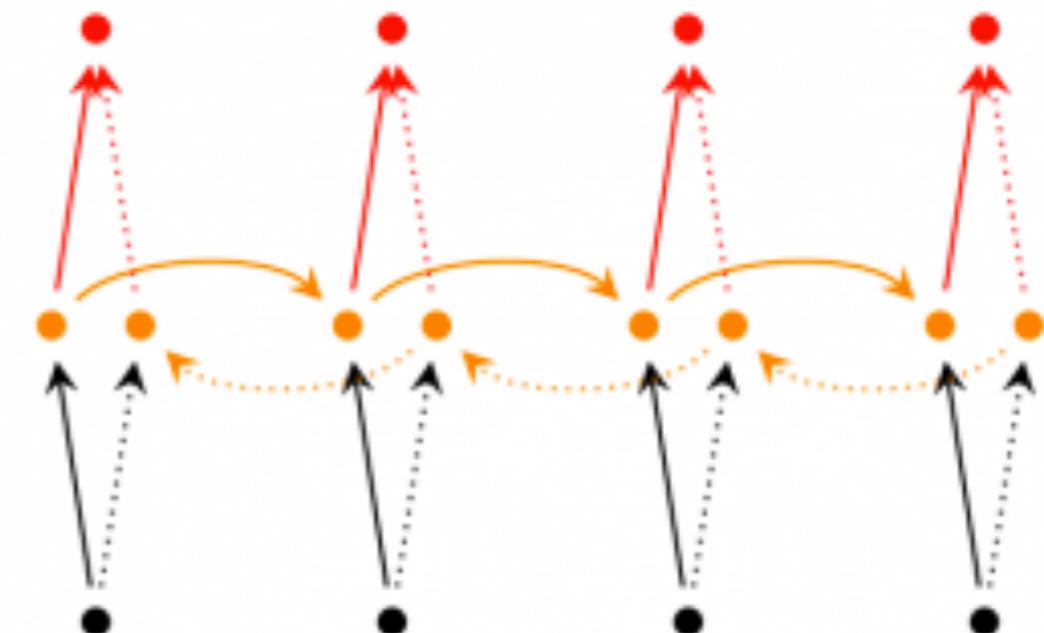
확장 모델

- Bidirectional RNN

시간 스텝 t 에서의 출력값이 이전 시간 스텝 외에, 이후의 시간 스텝에서 들어오는 입력값에도 영향을 받을 수 있다는 아이디어에 기반한다.

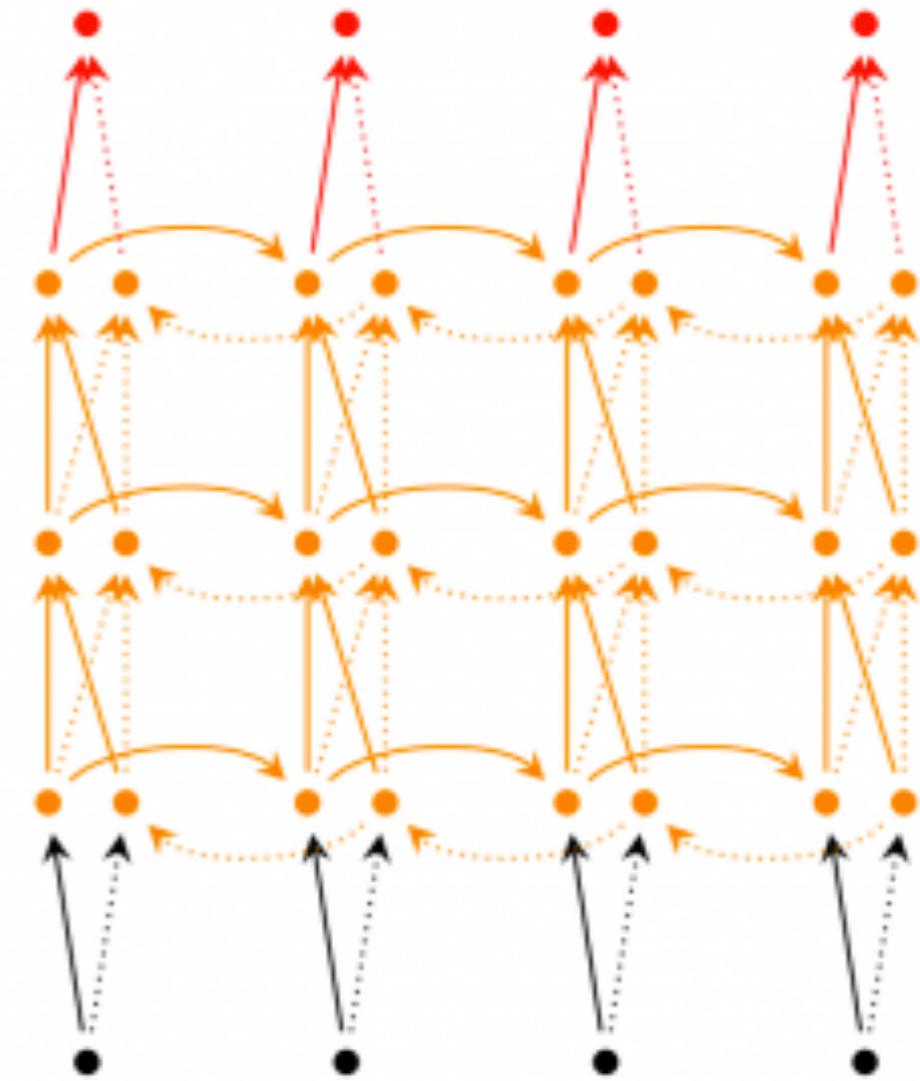
예를 들어, 영어 문제에서 빈칸에 가장 알맞는 단어를 채우기 위해서는 빈칸보다 앞쪽 문장들도 봐야겠지만, 빈칸 이후의 단어들도 문맥을 파악하는데 도움이 될 것이기 때문이다.

네트워크 구조는 RNN에서 단순히 확장되어서, 그림처럼 두 개의 RNN이 동시에 존재하고, 출력값은 두 RNN의 hidden state에 모두 의존하도록 계산된다.



확장 모델

- Deep (Bidirectional) RNN
앞의 구조와 비슷하지만, 매 시간 스텝
마다 여러 layer가 있다.
실제 구현에서는 이러한 구조가 학습
할 수 있는 capacity가 크다.
(당연히, 학습 데이터는 훨씬 더 많이
필요하다).



언어 모델링

- m 개의 단어로 이루어져 있는 문장이 있다고 하면,
언어 모델에서 이 문장이 특정 데이터셋에서 나타날 확률은 다음과 같다.
- 수식
$$P(w_1, \dots, w_m) = \prod_{i=1}^m P(w_i | w_1, \dots, w_{i-1})$$
- 문장이 나타날 확률은 이전 단어를 알고 있을 때 각 단어가 나타날 확률의 곱이 된다. 따라서 "He went to buy some chocolate"라는 문장의 확률은 "He went to buy some"이 주어졌을 때 "chocolate"의 확률 곱하기 "He went to buy"가 주어졌을 때 "some"의 확률 곱하기 ... 문장의 시작에서 아무것도 안 주어졌을 때 "He"의 확률까지의 곱이 된다.
- 앞의 수식에서 각 단어들의 확률은 이전에 나왔던 모든 단어들에 의존하고 있다. 하지만 실제 구현에서는 많은 모델들이 계산량, 메모리 문제 등으로 인해 long-term dependency 를 효과적으로 다루지 못해서 긴 시퀀스는 처리하는 것이 힘들다. 이론적으로 RNN 모델은 임의의 길이의 시퀀스를 전부 기억할 수 있지만 실제로는 조금 더 복잡하다.

언어 모델 필요성

- 점수를 매기는 메커니즘으로 활용될 수 있다. 예를 들어, 자동 기계 번역 시스템은 보통 하나의 입력 문장에 대해 여러 개의 후보 답안 문장을 생성한다. 여기서 언어 모델로 가장 확률이 높은 문장을 고를 수 있을 것이다. 직관적으로 보면, 가장 확률이 높은 문장은 문법적으로도 더 맞을 확률이 높다. 음성 인식 시스템에서도 비슷한 방식으로 점수를 매기는데 활용된다.
- 언어 모델 문제를 풀다보면 상당히 재미있는 부산물이 나타난다. 문장에서 이전 위치에 나타나는 단어들을 알 때 다음 단어가 나타날 확률을 얻을 수 있기 때문에, 이를 기반으로 새로운 텍스트를 생성해낼 수도 있는 것이다.
즉, 생성 모델 (generative model)이 나타난다. 현재 갖고 있는 단어들의 시퀀스를 주고 결과로 얻은 단어들의 확률 분포에서 다음 단어를 샘플링하고, 문장이 완성될 때까지 계속 이 과정을 반복할 수 있다.
- Andrej Karpathy가 블로그 포스트에 언어 모델이 어떤 일들을 할 수 있는지에 대해 훌륭하게 정리해 주었다. Karpathy의 모델은 단어 기준이 아니라 글자(character) 단위로 학습되었고, 셰익스피어부터 리눅스 소스 코드까지 전부 다 생성해낼 수 있다.

데이터 전처리

1. 텍스트의 토큰화 (Tokenize Text)

텍스트 데이터에서 단어 단위로 예측을 하기 위해서는 댓글을 문장으로 토큰화 하고, 문장을 단어 단위로 쪼개야 한다. 단순히 공백(스페이스바)을 기준으로 자를 수도 있겠지만, 이는 문장 부호들을 제대로 처리하지 못하게 된다. 예시로, "He left!"라는 문장은 3개의 토큰 - "He", "left", "!" - 으로 이루어져야 한다. 여기서는 NLTK의 `word_tokenize`와 `sent_tokenize` 방식을 사용하였다.

데이터 전처리

2. 빈도수가 낮은 단어들 없애기

데이터셋에 있는 대부분의 단어들은 한 번 내지 두 번 정도 등장한다. 이렇게 드문드문 나타나는 단어들은 없애는 것이 더 도움이 된다. 기억해야 할 단어의 종류가 너무 커지면 모델을 학습하는데 시간이 더 오래 걸리고, 빈도수가 낮은 단어들의 경우에는 어떤 상황에서 이런 단어들이 나타나는지에 대한 예시가 별로 없어서 학습하기도 힘들다. 사람이 배우는 것과도 비슷한데, 어떤 단어의 의미를 제대로 파악하려면 여러 상황에서 활용된 예시문을 봐야 할 것이다. 텍스트에 등장하는 빈도순으로 `vocabulary_size` 변수만큼으로 단어 수를 제한한다. 단어장에 없는 단어들은 전부 `UNKNOWN_TOKEN`으로 바꿔준다. 예시로, 단어장에 "`nonlinearities`"라는 단어가 없다면, "`nonlinearities are important in neural networks`"라는 문장은 "`UNKNOWN_TOKEN are important in neural networks`"로 바뀔 것이다. `UNKNOWN_TOKEN`이라는 단어는 단어장에 추가되고, 다른 단어처럼 나타날 확률 예측도 하게 된다. 새로운 텍스트를 생성할 때는 `UNKNOWN_TOKEN`을 단어장에 없는 단어 중에서 아무거나 랜덤으로 뽑아서 대체할 수도 있고, 아니면 `UNKNOWN_TOKEN`이 나오기 전까지만 문장을 생성하는 방법도 있다.

데이터 전처리

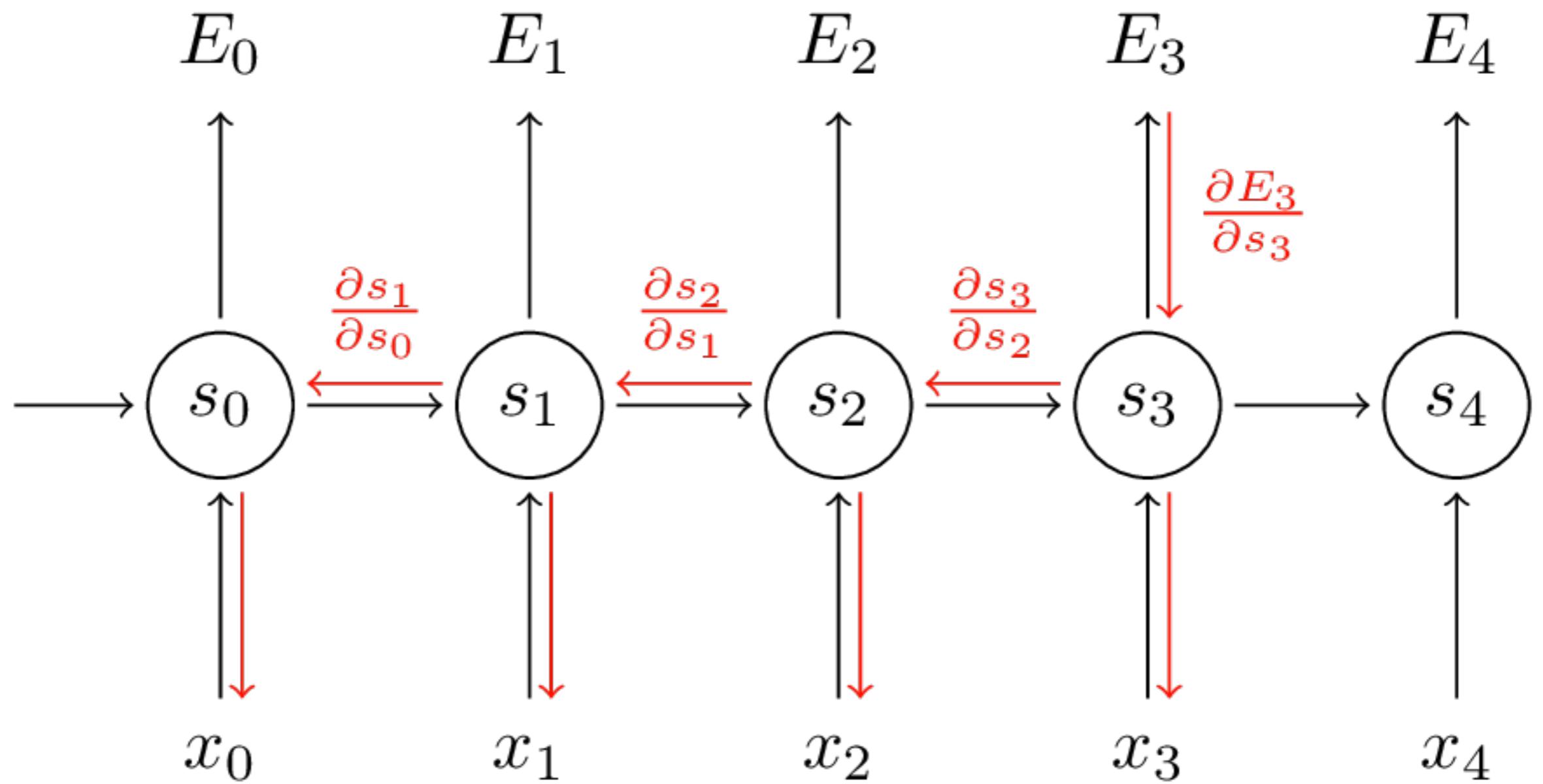
3. 시작 토큰과 끝 토큰 연결

언어 모델은 어떤 단어들이 문장의 맨 처음 나타나고, 어떤 것들이 맨 뒤에 나타나는지도 학습하고자 한다. 이를 위해서 특별히 SENTENCE_START 토큰을 문장의 맨 앞에 이어붙이고, SENTENCE_END 토큰을 문장의 맨 뒤에 붙일 것이다. 모든 문장에 대해 이 과정을 처리해 주고나면, 문제는 다음과 같이 바뀐다: 첫 번째 토큰이 SENTENCE_START일 때, 다음 단어는 무엇일까? (실제 문장의 첫 단어)

4. 학습 데이터 행렬 구성

RNN 모델의 입력은 문자열이 아니라 벡터이기 때문에 단어들과 인덱스들 사이의 매핑 - index_to_word와 word_to_index를 먼저 만든다. 예를 들어, "friendly"라는 단어는 2001번 위치에 있을 수 있다. 학습 데이터 x 는 $[0, 179, 341, 416]$ 과 같이 생겼을 것이고, 여기서 0은 SENTENCE_START를 뜻한다. 해당하는 정답 y 는 $[179, 341, 416, 1]$ 정도로 나타내질 것이다. 우리 목적은 다음 단어를 예측하는 것이기 때문에 y 는 단순히 x 벡터를 오른쪽으로 한 칸 옮기고 마지막 위치에 SENTENCE_END 토큰을 넣어준 것이어야 한다. 즉, 179 번 단어의 올바른 예측값은 실제 다음 단어인 341이 되어야 한다.

BPTT



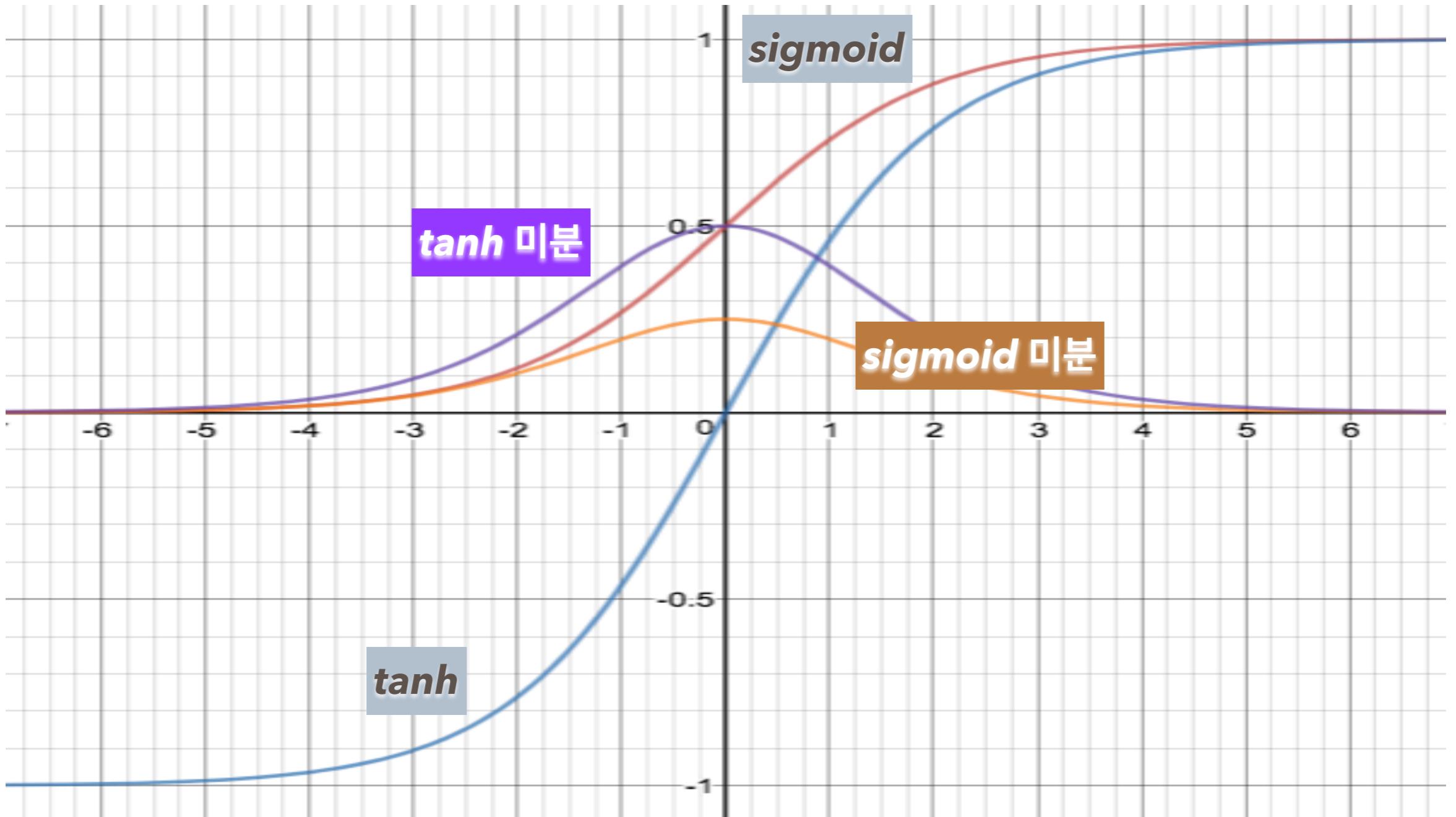
BPTT

- Back Propagation Through Time
- deep Feedforward Neural Network(피드백 연결이 없는 네트워크)에서 사용하는 원래의 backpropagation 알고리즘과 똑같다.
- 중요한 차이점은 매 시간 스텝마다 W 에 대한 gradient를 더해준다는 점이다. 기존의 신경망 구조에서는 layer별로 파라미터를 공유하지 않기 때문에 계산 결과들을 서로 더해줄 필요가 없다.

Vanishing Gradient

- RNN은 긴 시퀀스를 처리하는데 (long-range dependency를 처리하는데) 한계가 있다. 즉, 주요 단어들 사이에 여러 시간 스텝이 지났다면 잘 기억하지 못한다.
- 이것은 문장의 의미를 파악하는데 있어서 가까이 있지 않은 단어들이 밀접한 관련이 있을 수도 있기 때문에 문제가 된다.
- 예로, "The man who wore a wig on his head went inside."라는 문장을 보면, 이 문장은 "man"이 "inside"로 가는 것에 대한 문장이지 "wig(가발)"에 대한 것이 아니다. 그러나, 기본 RNN 모델은 남자보다 가발에 대한 정보를 더 잘 기억하게 된다.

Vanishing Gradient



Vanishing Gradient

- tanh 함수와 sigmoid 함수는 양쪽 끝에서 미분값이 0으로 수렴하는 것을 볼 수 있다.
- 이 현상이 발생할 때, 그 뉴런이 포화되었다고 말하는데, 이런 뉴런들은 gradient가 거의 0이기 때문에 곱해지는 이전 layer의 gradient들도 0으로 수렴하게 만든다.
- 따라서, 행렬에 작은 값들이 들어있고 여러 ($t-k$ 번) 행렬곱이 이루어지면 gradient는 지수 함수로 감소하고, 시간 스텝 몇 번만 지나도 사라져 버린다 (vanish!).
- 시퀀스에서 여러 시간 스텝이 떨어진 곳에서는 gradient가 전달되지 못하고, 먼 과거의 상태(state)는 현재 스텝의 학습에 아무 도움이 되지 못하게 된다. 즉, long-range dependency를 제대로 배우지 못한다.
- Vanishing gradient 문제는 RNN에서만 나타나는 것이 아니다. Deep Feedforward Neural Network에서도 마찬가지로 발생하지만, RNN은 보통 시간 스텝 횟수만큼 매우 깊은 구조이기 때문에 이 문제가 훨씬 잘 나타난다.

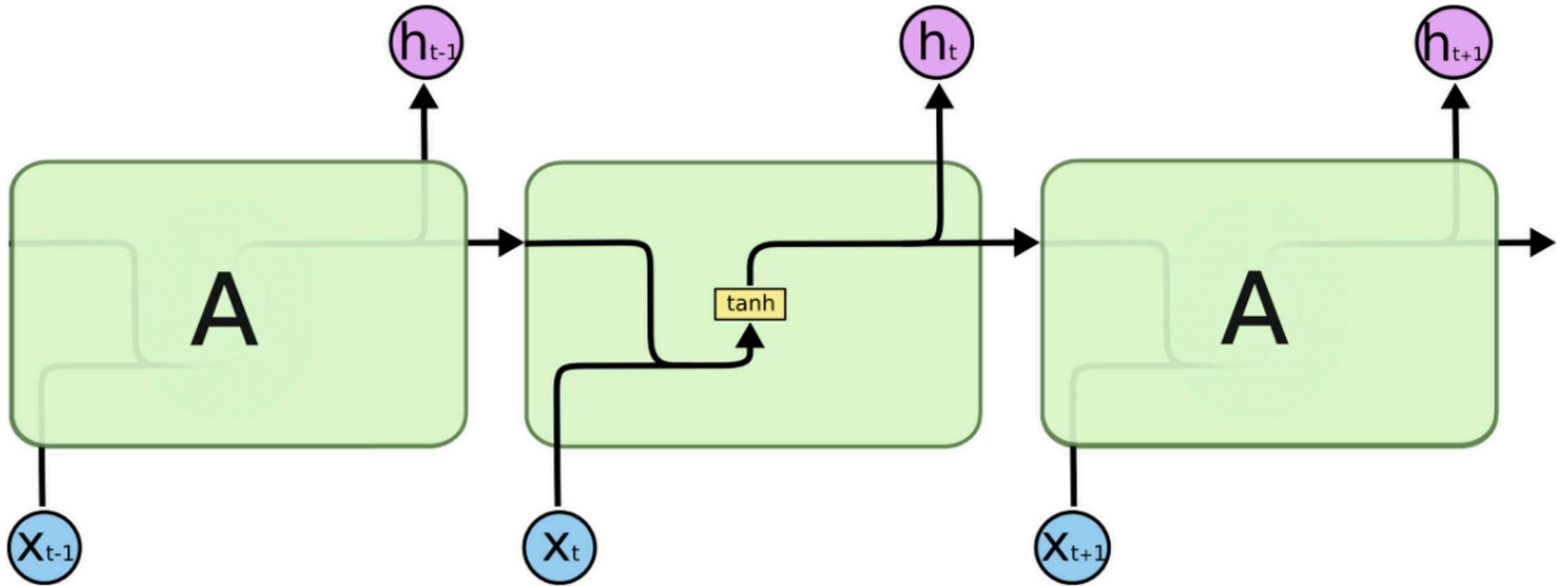
Vanishing Gradient

- 자코비안 행렬을 통한 Gradient 계산을 보면, 행렬 안의 값들이 크다면 activation 함수와 네트워크 파라미터 값에 따라 gradient가 사라지는게 아니라 지수 함수로 증가하는 경우도 충분히 상상해볼 수 있다
- 이 문제는 exploding gradient 문제로 알려져 있다.
- Vanishing gradient 문제가 더 많은 관심을 받는 이유
 - exploding gradient 문제는 쉽게 알아차릴 수 있다는 점이다. Gradient 값들이 NaN (not a number)이 될 것이고 프로그램이 죽을 것이기 때문이다.
 - gradient 값이 너무 크다면 미리 정해준 적당한 값으로 잘라버리는 방법으로 쉬우면서도 효율적으로 해결할 수 있기 때문이다.
 - Vanishing gradient 문제는 언제 발생하는지 바로 확인하기가 힘들고 간단한 해결법이 없기 때문에 문제였다.

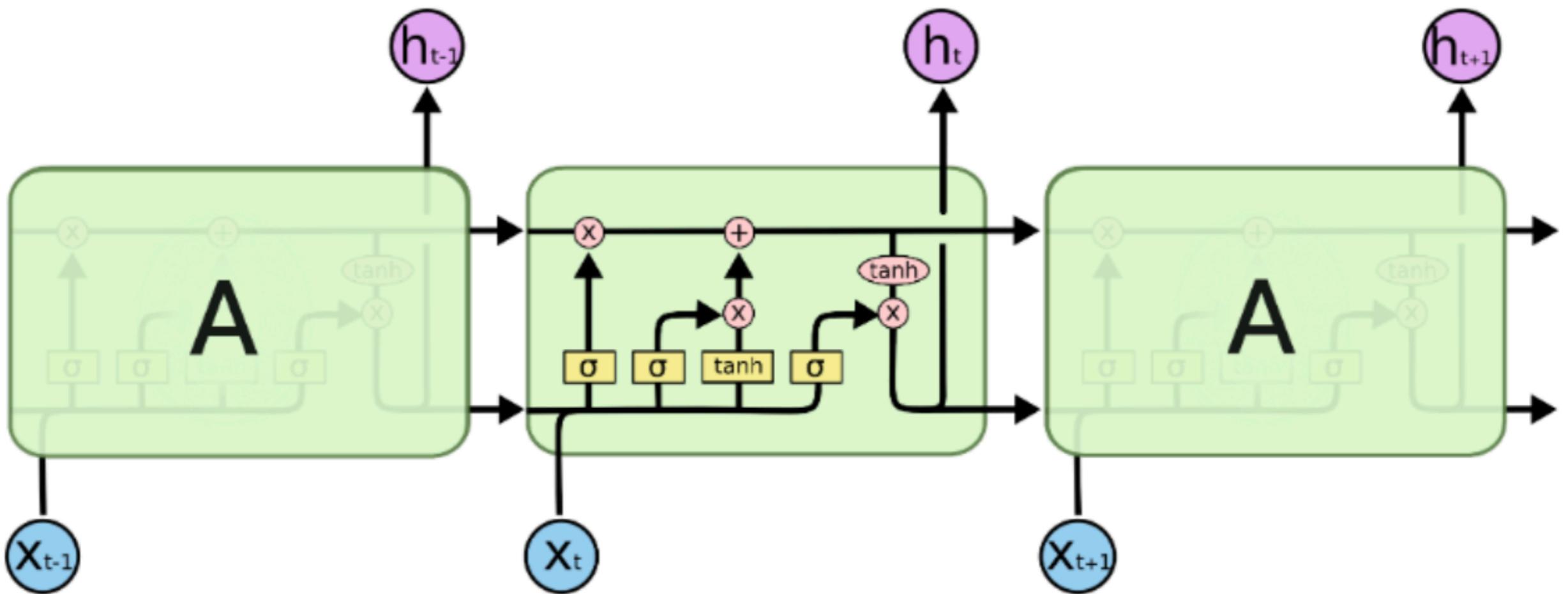
Vanishing Gradient 해결책

- W 행렬을 적당히 좋은 값으로 잘 초기화 해준다면 vanishing gradient의 영향을 줄일 수 있고, regularization을 잘 정해줘도 비슷한 효과를 볼 수 있다.
- 더 보편적으로 사용되는 방법은 tanh나 sigmoid activation 함수 말고 ReLU를 사용하는 것이다. ReLU는 미분값의 최대치가 1로 정해져있지 않기 때문에 gradient 값이 없어져버리는 일이 크게 줄어든다.
- 이보다 더 인기있는 해결책은 Long Short-Term Memory (LSTM)이나 Gated Recurrent Unit (GRU) 구조를 사용하는 방법이다.
 - LSTM은 1997년에 처음 제안되었고, 현재 자연어처리 분야에서 가장 널리 사용되는 모델 중 하나이다. GRU는 2014년에 처음 나왔고, LSTM을 간략화한 버전이다.
 - 두 가지 RNN의 변형 구조 모두 vanishing gradient 문제 해결을 위해 디자인되었고, 효과적으로 긴 시퀀스를 처리할 수 있다는 것을 보여줬다.

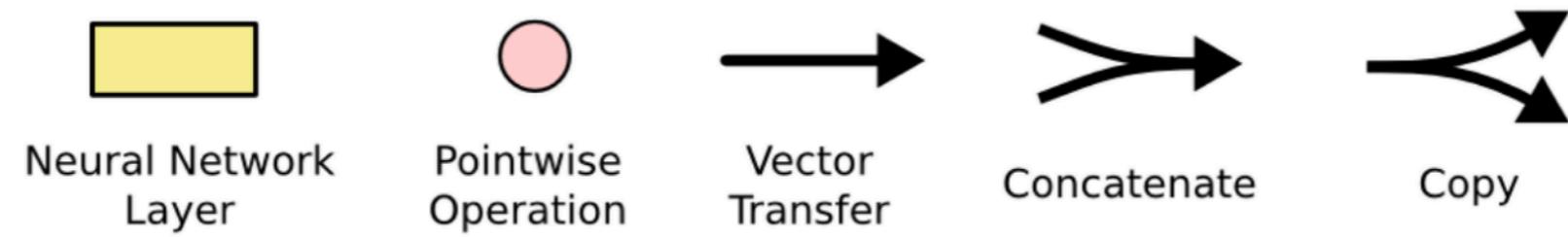
셀 내부 : 기본



셀 내부 : LSTM

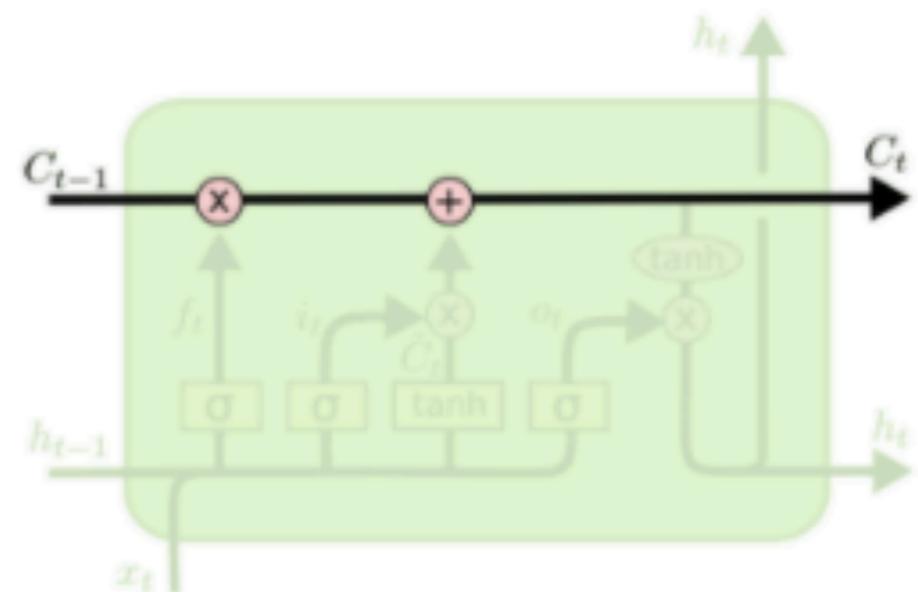


셀 내부 : LSTM



- 노란색 박스는 학습된 neural network layer를 나타낸다.
- 분홍색 동그라미는 vector 합과 같은 pointwise operation을 나타낸다.
- 각 선은 한 노드의 output을 다른 노드의 input으로, vector 전체를 보내는 흐름을 나타낸다.
- 합쳐지는 선은 concatenation을 의미하고,
- 갈라지는 선은 정보를 복사해서 다른 쪽으로 보내는 fork를 의미한다.

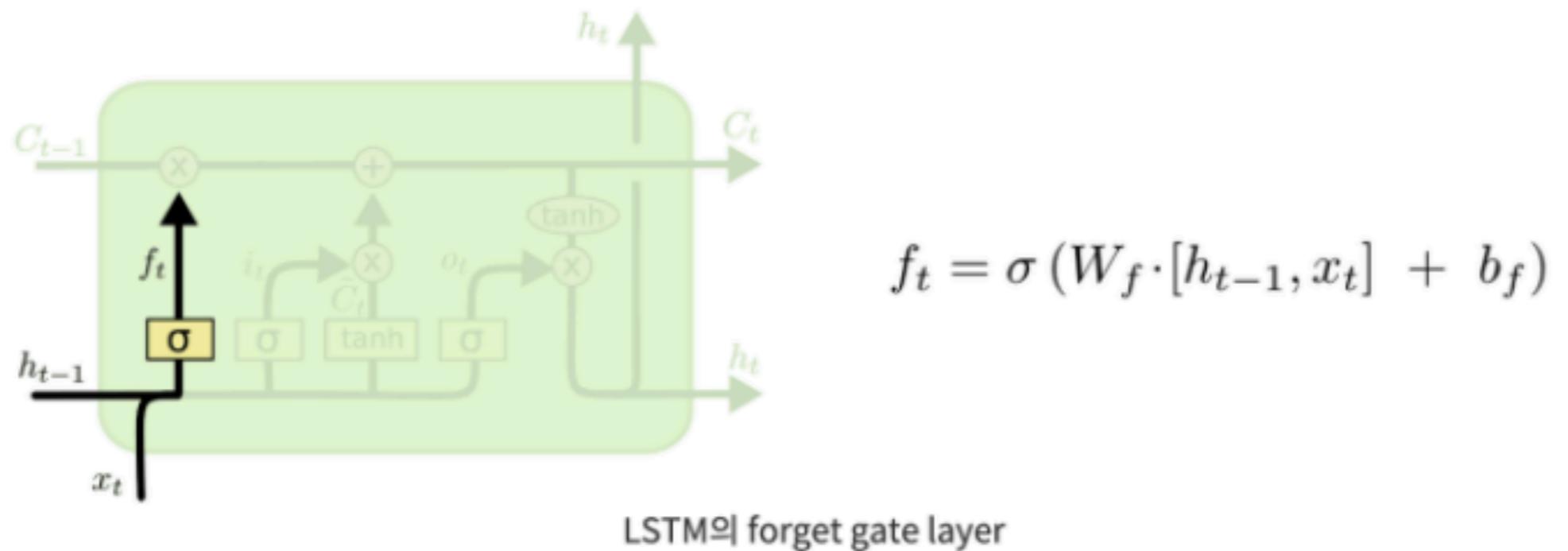
- Cell State



- LSTM의 핵심은 Cell State인데, 모듈 그림에서 수평으로 그어진 윗 선에 해당한다.
- 메모리와 같은 존재이며, 컨베이어 벨트와 같아서 State가 꽤 오래 경과하더라도 Gradient가 잘 전파된다.
- 또한, Cell State는 Gate라고 불리는 구조에 의해 정보가 추가되거나 제거되며
- Gate는 training을 통해 어떤 정보를 유지하고 어떤 정보를 버릴지 학습한다.

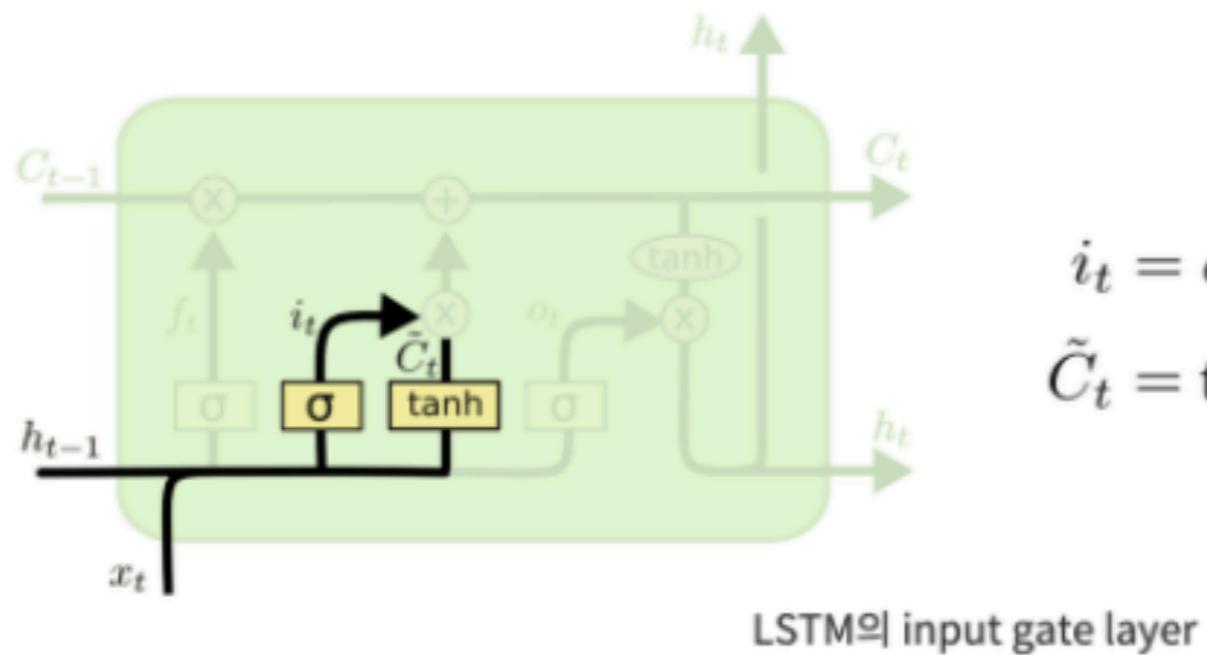
- **Gates**
 - LSTM은 3개의 Gate를 가지고 있고, 이 gate들은 cell state를 보호하고 제어한다.
 - forget gate (f) : 과거 정보를 잊기 위한 게이트
 - input gate (i) : 현재 정보를 기억하기 위한 게이트
 - output gate (o) : 최종 결과를 내보내기 위한 게이트
- **Sigmoid**
 - 모든 Gate는 시그모이드 함수를 사용하고, 시그모이드 함수의 출력 범위는 0~1이다.
 - 따라서, 그 값이 0이라면 "아무 것도 넘기지 말라"가 되고, 값이 1이라면 "모든 것을 넘겨드려라"가 된다.

- Forget Gate



- 과거 정보를 얼마나 잊을지에 대한 단계이며, "forget gate"라고 불린다.
- 이 단계에서는, $h(t-1)$ 와 $x(t)$ 를 받아서, sigmoid layer를 통해 0과 1 사이의 값을 $C(t-1)$ 에 보내준다.
- 그 값이 1이면 "이전 상태의 모든 정보를 보존해라"가 되고, 0이면 "이전 상태의 모든 정보를 버려라"가 된다.

- Input Gate

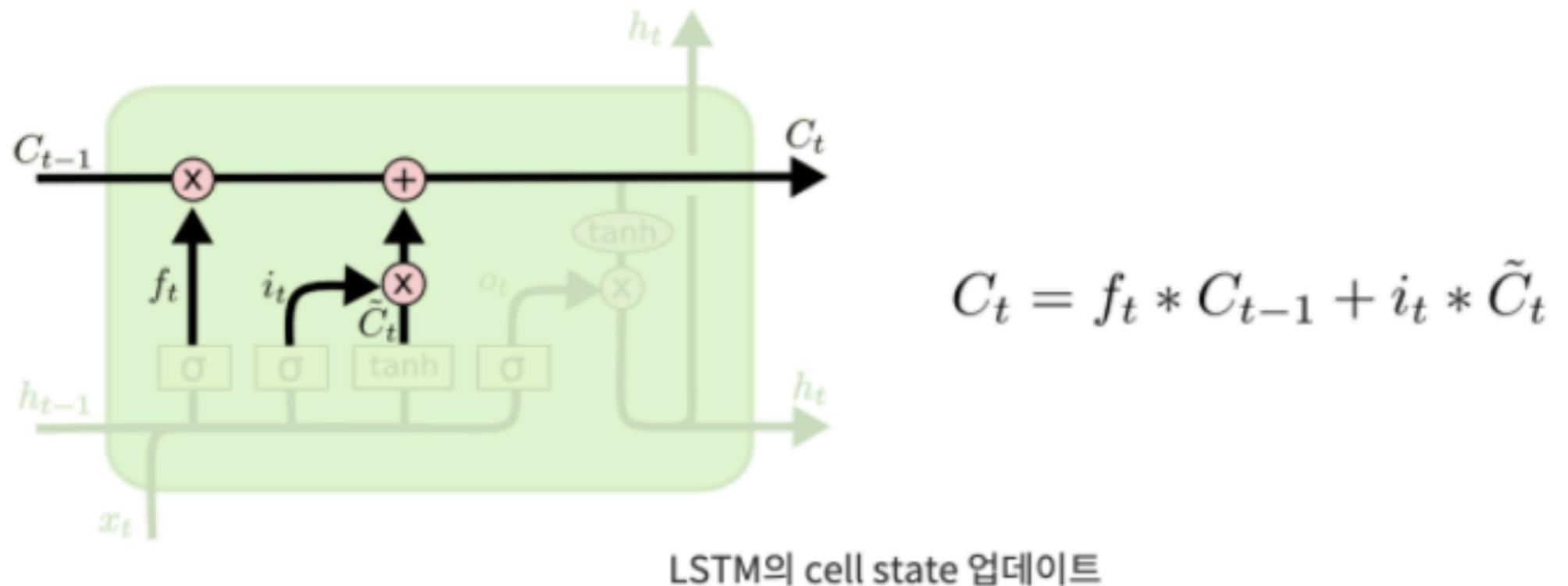


- 현재 정보를 얼마나 기억할 것인지에 대한 단계이며, "input gate"라고 불린다.
- 이 단계에서는, $h(t-1)$ 와 $x(t)$ 를 받아서, sigmoid layer를 통해 $i(t)$ 를 구한 뒤
- tanh layer를 통해 새로운 후보 값들인 $\tilde{C}(t)$ 라는 vector를 만들고,
- $i(t)$ 와 $\tilde{C}(t)$ 의 정보를 합쳐, cell state를 업데이트할 재료를 만들게 된다.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

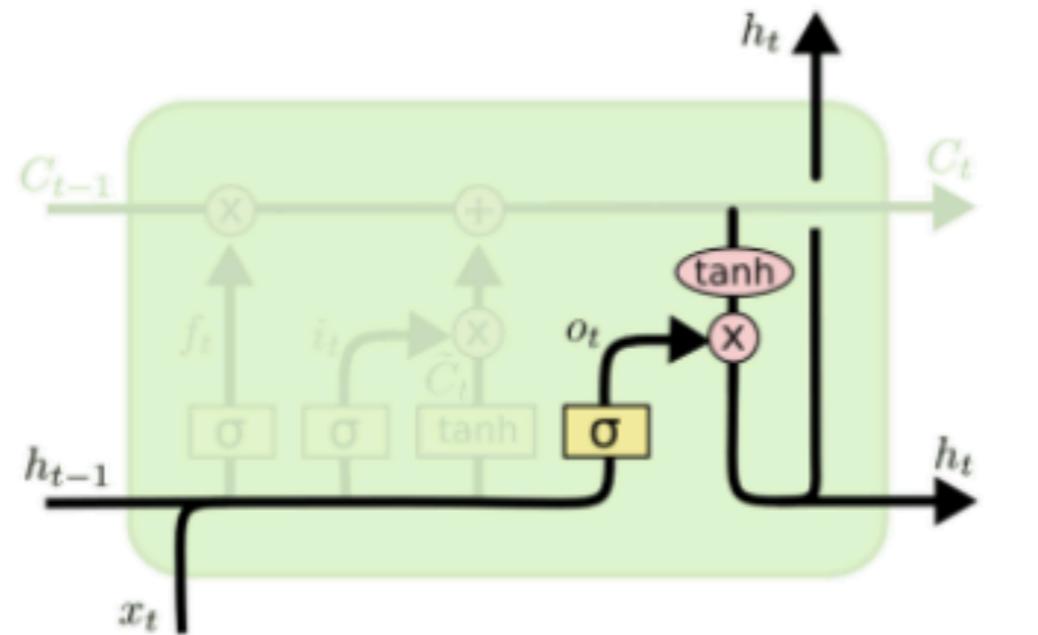
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- State Update



- 이전 cell state인 $C(t-1)$ 를 업데이트해서 새로운 cell state인 $C(t)$ 를 만드는 단계이다.
- 이전 cell state를 얼마나 잊을건지(forget gate) 곱셈
- 현재 cell state를 얼마나 반영할건지(input gate) 덧셈
- 최종 결과를 다음 상태의 cell state로 내보낸다.

- Output Gate



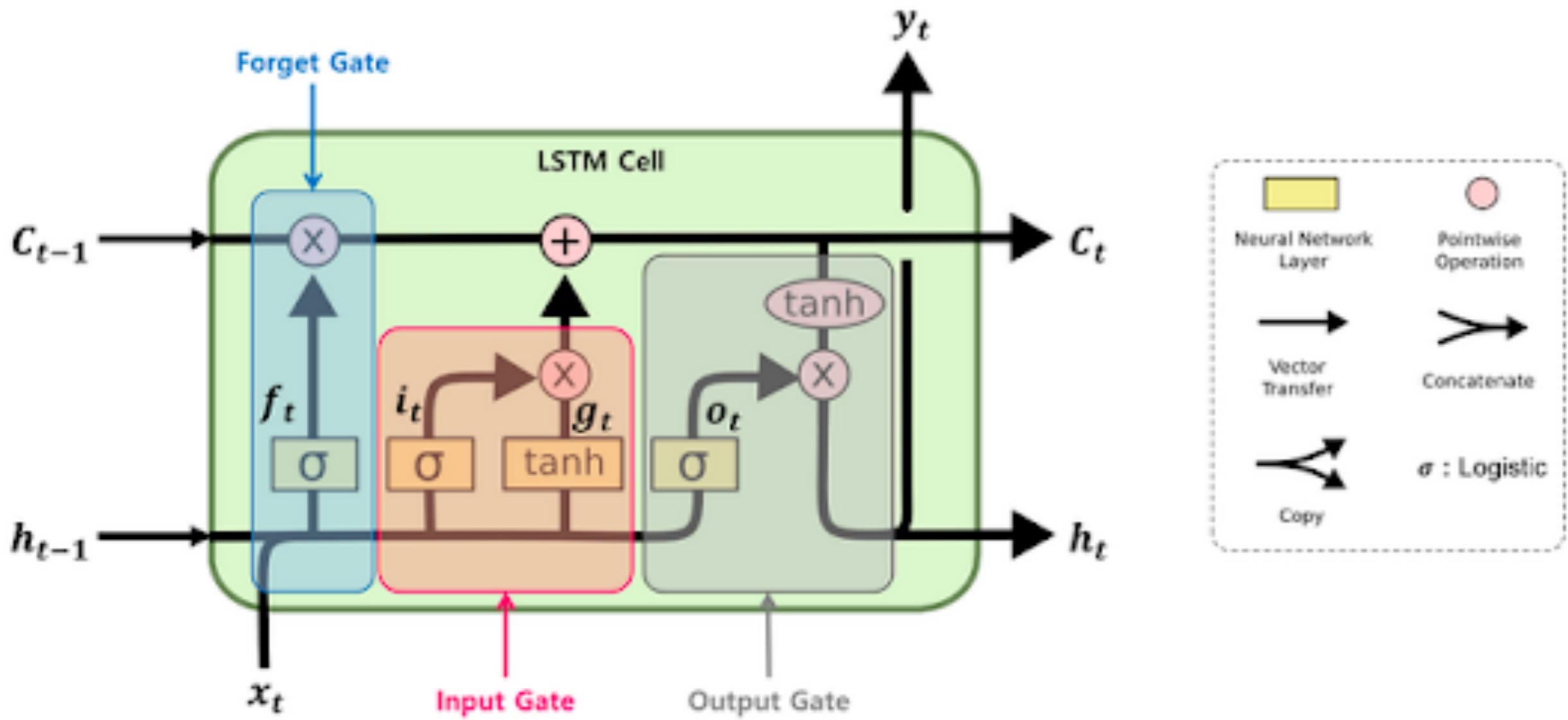
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

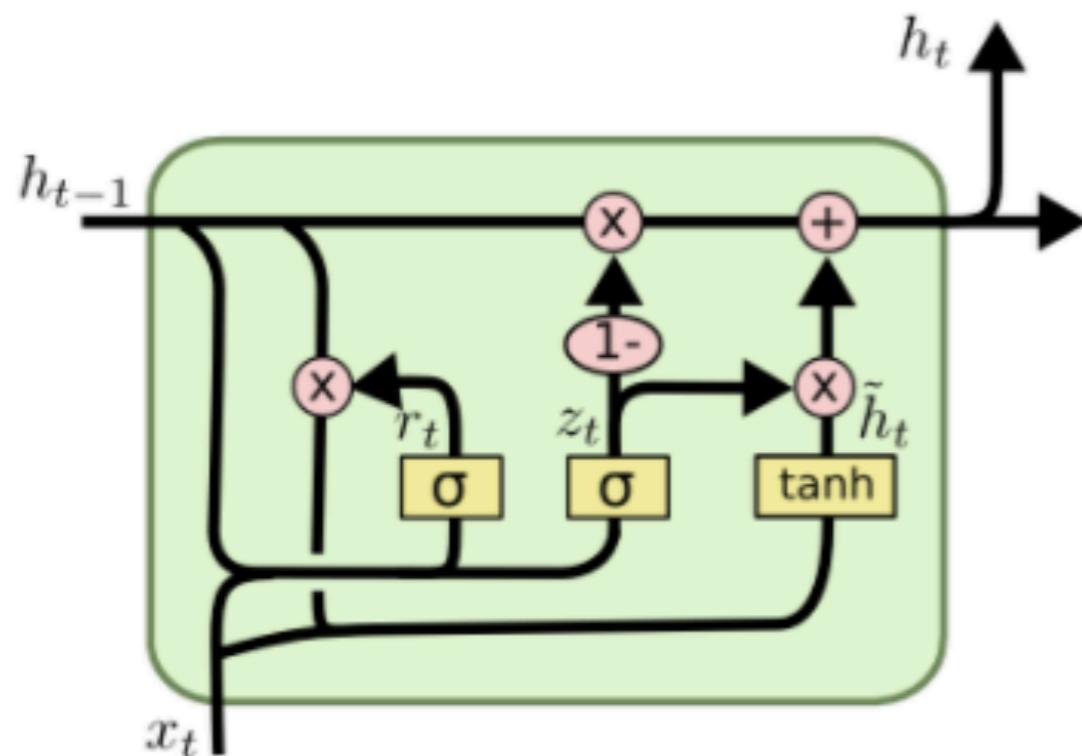
$$h_t = o_t * \tanh (C_t)$$

LSTM의 output gate layer

- 다음 State로 내보낼 output(hidden state)을 구하는 단계이다.
- cell state에 tanh를 취한 뒤
- $h(t-1)$ 와 $x(t)$ 를 받아 sigmoid layer를 통해 $O(t)$ 를 계산한 후
- $O(t)$ 와 tanh 결과를 곱해 output(hidden state)을 내보낸다.

셀 내부 : GRU





$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- GRU는 LSTM과 비슷한 이유로 만들어졌는데, LSTM을 구성하는 Time-Step의 Cell을 조금 더 간소화한 버전이다.
- GRU는 LSTM보다 학습 속도가 빠르다고 알려져있지만, 여러 평가에서 GRU는 LSTM과 비슷한 성능을 보인다고 알려져 있다.
- 데이터 양이 적을 때는, 매개 변수의 양이 적은 GRU가 조금 더 낫고,
- 데이터 양이 더 많으면 LSTM이 더 낫다고 알려져 있다.

- 차이점
 - GRU는 LSTM과 다르게 Gate가 2개이며, Reset Gate(r)과 Update Gate(z)로 이루어져 있다.
 - Reset Gate는 이전 상태를 얼마나 반영할지
 - Update Gate는 이전 상태와 현재 상태를 얼마만큼의 비율로 반영할지
 - 또한, LSTM에서의 Cell State와 Hidden State가 Hidden State로 통합되었고
 - Update Gate가 LSTM에서의 forget gate, input gate를 제어한다.
 - GRU에는 Output Gate가 없다.
- **Reset Gate**
 - 이전 상태의 hidden state와 현재 상태의 x 를 받아 sigmoid 처리
 - 이전 hidden state의 값을 얼마나 활용할 것인지에 대한 정보
- **Update Gate**
 - 이전 상태의 hidden state와 현재 상태의 x 를 받아 sigmoid 처리
 - LSTM의 forget gate, input gate와 비슷한 역할을 하며,
 - 이전 정보와 현재 정보를 각각 얼마나 반영할 것인지에 대한 비율을 구하는 것이 핵심이다.
 - 즉, update gate의 계산 한 번으로 LSTM의 forget gate + input gate의 역할을 대신할 수 있다.
 - 따라서, 최종 결과는 다음 상태의 hidden state로 보내지게 된다.

LSTM VS. GRU

- 게이팅 메커니즘을 통해 긴 시퀀스를 잘 기억하도록 해준다는 점에서 비슷하다.
- GRU는 게이트가 2개이고, LSTM은 3개입니다.
- GRU는 내부 메모리 값 (ct)이 외부에서 보게되는 hidden state 값과 다르지 않습니다. LSTM에 있는 output 게이트가 없기 때문입니다.
- input 게이트와 forget 게이트가 update 게이트 z 로 합쳐졌고, reset 게이트 r 은 이전 hidden state 값에 바로 적용됩니다. 따라서, LSTM의 forget 게이트의 역할이 r 과 z 양쪽으로 나눠졌다고 생각할 수 있습니다.
- 출력값을 계산할 때 추가적인 비선형 함수를 적용하지 않습니다.

LSTM VS. GRU

- 여러 논문에서의 실험들에 따르면,
두 모델 모두 좋은 성능을 보여주고 있어서 어떤 것이 좋다라고 얘기할 수는 없습니다.
- 레이어 크기같은 파라미터 튜닝을 잘 하는 것이 모델을 고르는 것보다 더 중요합니다.
- GRU는 파라미터 수가 적어서 (U 와 W 가 더 작다)
학습 시간이 짧고 적은 데이터로도 학습이 가능할 수 있습니다.
- 데이터가 충분한 경우에는
LSTM의 우수한 모델링 파워가 더 좋은 결과를 보여줄 수도 있을 것입니다.

성능 향상

- Rmsprop/Adam 등의 최적화 알고리즘 사용
- 임베딩 레이어 추가

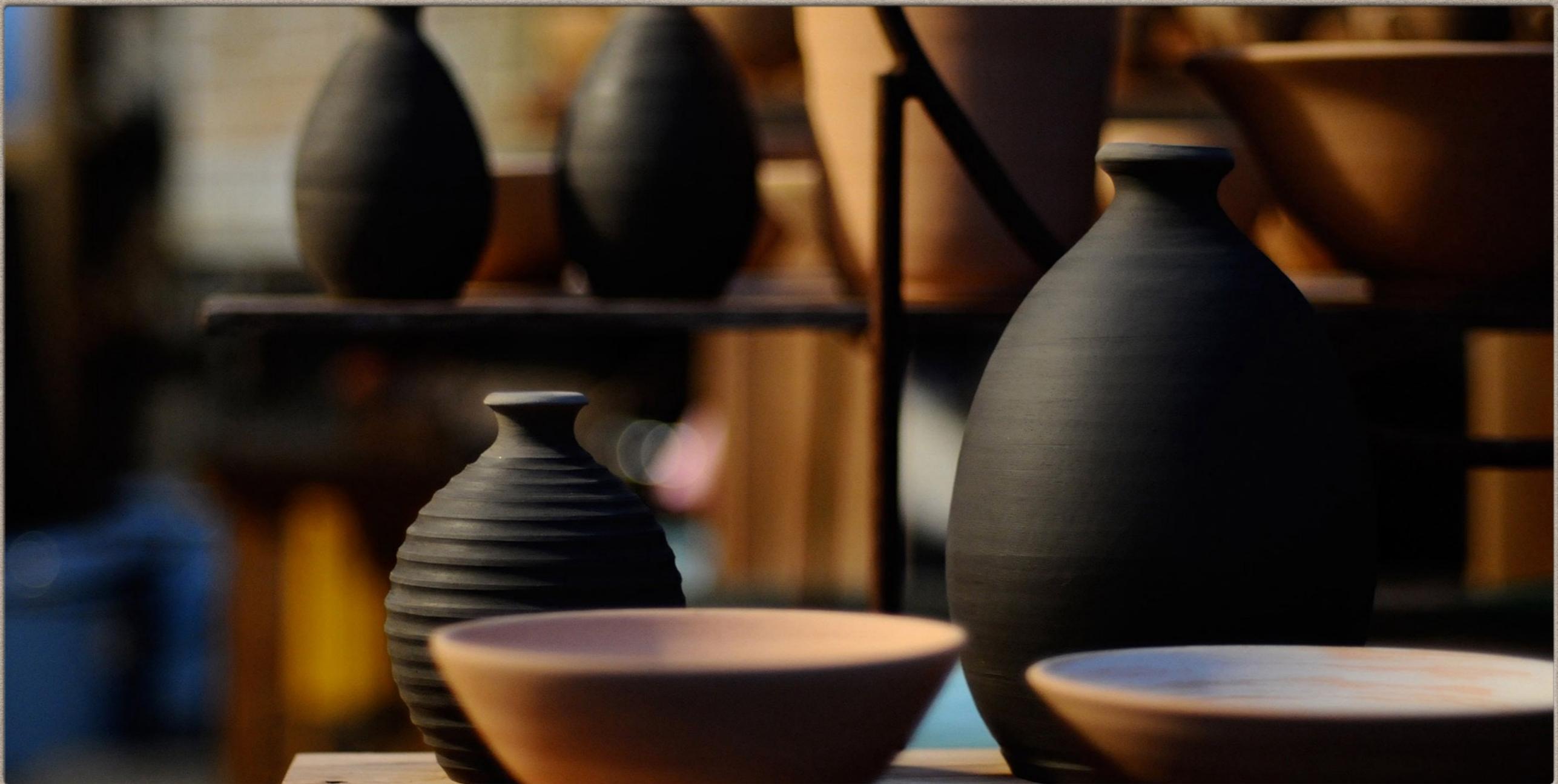
Word2vec이나 GloVe와 같은 단어 임베딩을 추가하는 것은 모델의 성능을 향상시키기 위해 자주 사용되는 방법입니다. 각 단어의 one-hot 벡터 표현법과 달리 word2vec이나 GloVe에서 학습된 낮은 차원의 벡터 표현은 그 단어의 의미 정보를 담게 됩니다. 즉, 비슷한 단어는 비슷한 벡터 값을 갖게 됩니다. 이것들을 사용하는 것은 pre-training (딥러닝에서 한번에 학습이 어렵기 때문에 단계별로 네트워크의 일부분을 미리 학습해 두는 작업)처럼 생각할 수 있습니다. 이 과정을 통해 네트워크가 언어에 대한 정보를 (미리 어느정도는 학습되어 있어서) 학습해야 될 부분이 줄어들게 되는 것입니다. Pre-train된 벡터들은 학습 할 데이터가 많지 않을 때 특히 유용한데, 네트워크가 사전에 보지 못한 단어들에 대해서도 일반화가 가능해지기 때문입니다.

- 두 번째 GRU 레이어 추가하기

두 번째 레이어를 추가하는 것은 고차원적인 정보를 담을 수 있게 해줍니다. 그러나, 2-3 레이어 이후부터는 성능이 더 안 좋아지는 것을 확인할 가능성이 큰데, 데이터가 많지 않은 이상 레이어가 많아진다고 해서 성능에 큰 차이가 없을 것이고, 오히려 과적합될 수 있습니다.

성능 향상

- 파라미터를 업데이트할 때 배치(batch)로 gradient를 합치는 것입니다.
- 한 번에 한 문장씩 학습을 하기보다는, 같은 길이의 문장들을 (또는 같은 길이가 되도록 zero-padding을 해서) 그룹으로 묶고, 큰 행렬 곱연산을 수행하여 그 배치에 대한 gradient들을 전부 더해줍니다. 이것이 효율적인 이유는, 큰 행렬들의 곱연산은 GPU를 통해 효율적으로 처리될 수 있기 때문인데, 이 방식을 취하지 않는다면 GPU를 사용하는 이점이 별로 없어져서 학습이 매우 느려질 것입니다.
- 큰 모델을 학습시키려면 성능에 대해 최적화가 잘 되어있는 현존하는 딥러닝 라이브러리 중 하나를 사용하는 것을 추천합니다.



시계열

Time Series Data

시계열 정의

- ❖ 시계열(time series) 데이터는 관측치가 시간적 순서를 가진 데이터이다. 이 데이터는 변수간의 상관성(correration)이 존재하는 데이터를 다루며, i.i.d, 연속(continuous)하거나 불규칙적(irregular)데이터는 다루지 않는다.
- ❖ 시계열 데이터는 과거의 데이터를 통해서 현재의 움직임 그리고 미래를 예측하는데 사용된다. 일반적인 label데이터는 input과 label간의 상관관계를 다루는 반면에 시간에 따라 어떻게 움직이는 과거의 자료를 가지고 예측하게 된다.

시계열 데이터 특징과 분석의 목적

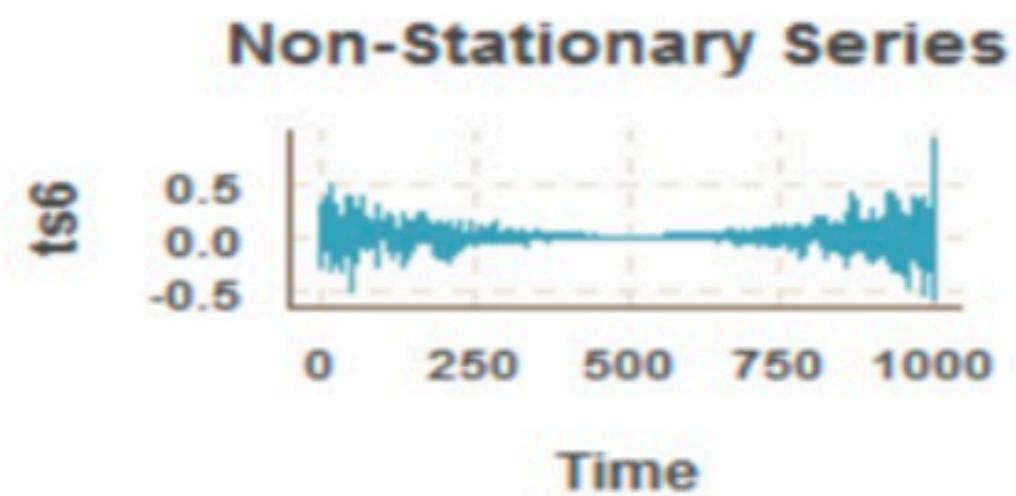
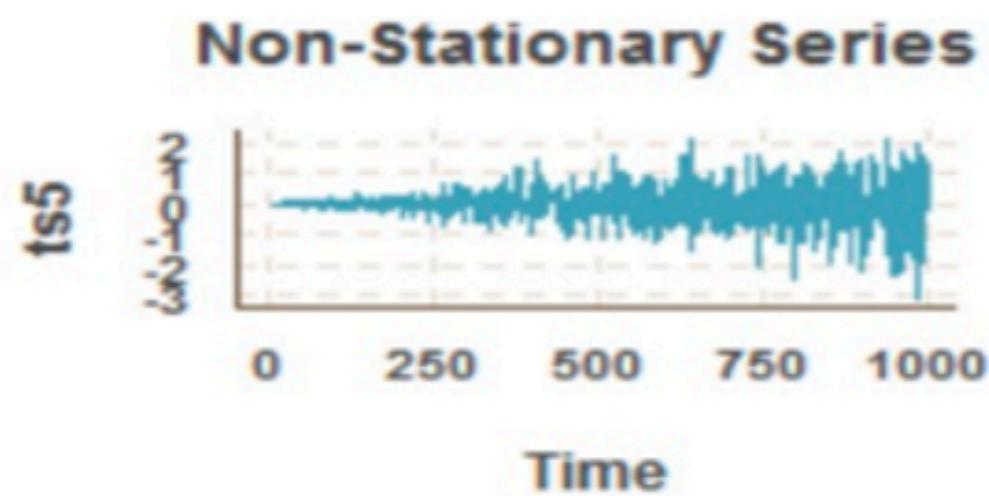
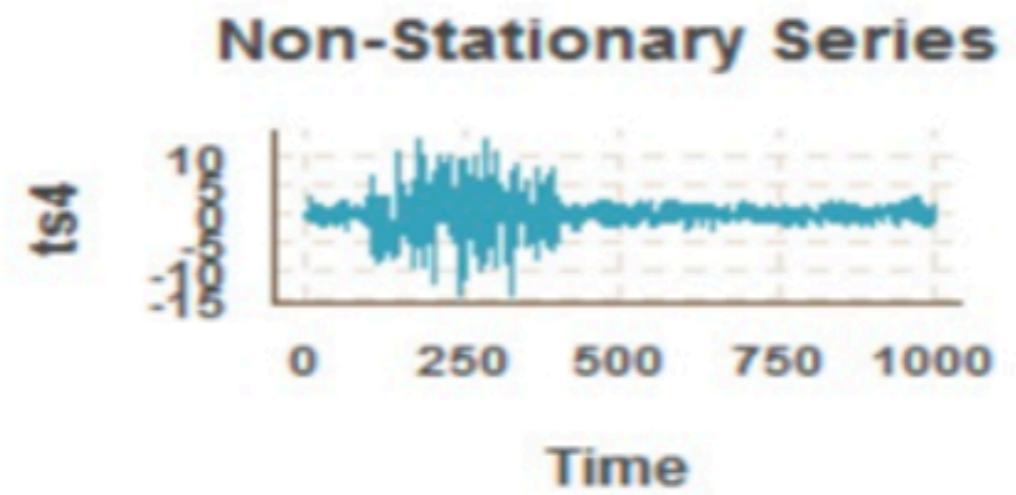
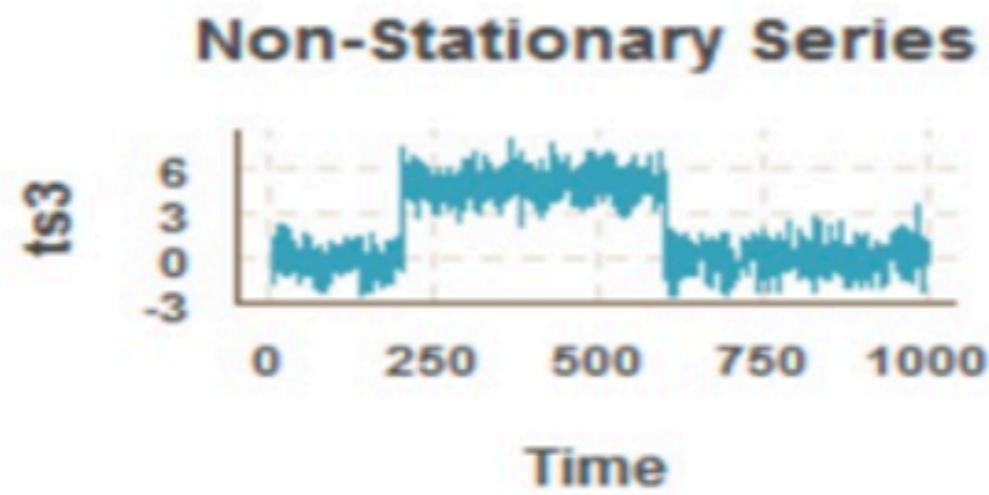
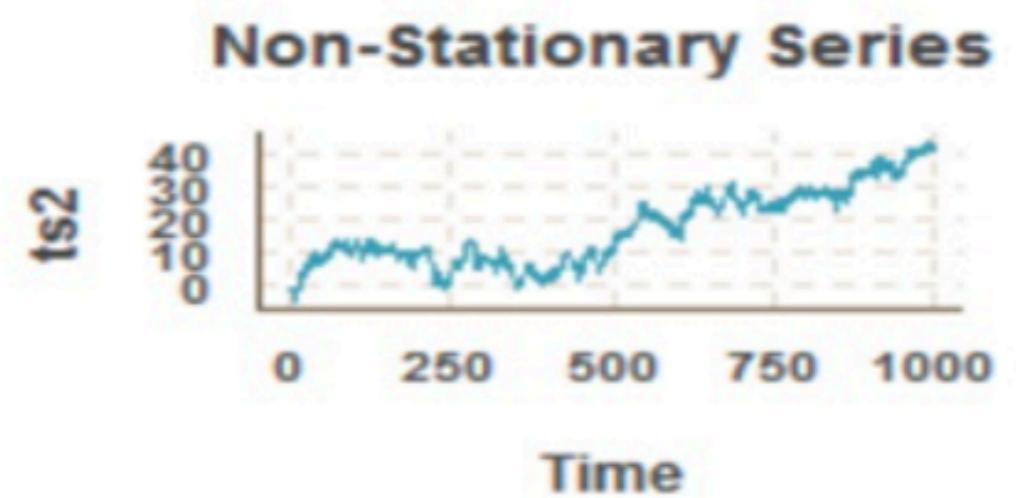
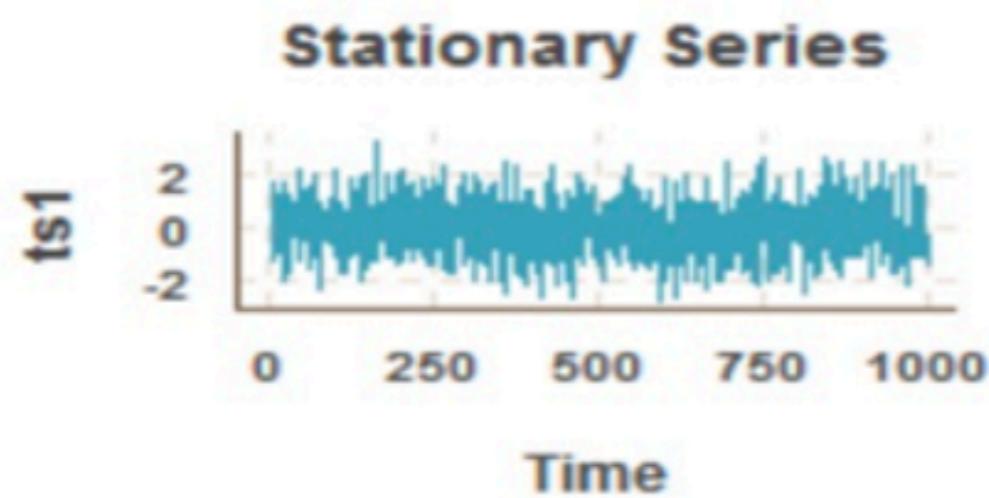
시계열 데이터란, 시간을 통해 순차적으로 발생한 관측치의 집합이라고 할 수 있다. 이렇게 순차적으로 발생한 연속적인 관측치는 서로 관련이 있다. 하지만 하나 알아두어야 할 점이 있다. 바로,

"이 때 시계열은 반드시 고정된(fixed) 시간 구간의 관측치어야 한다."

즉, 시계열이 불규칙적인 시간 구간이여서는 안된다는 것이다. 어떤 구간은 Daily 구간이였다가, 어떤 구간은 Monthly, 또 어떤 구간은 Yearly... 이렇게 다양한 구간이 동시에 존재해서는 안된다는 것이다.

그리고 시계열 데이터에 크게 두 가지로 나눌 수 있다.

- **정상시계열(Stationary)** : 평균과 표준편차가 일정하다는 조건이 선행되어야 분석이 가능하다. 대표적인 예시로는 **ARIMA모델**이 있다.
- **비정상시계열** : 차분이나 log함수를 써워 정상시계열로 변환 후 분석을 해야 한다.



분석 목적

그렇다면 이러한 시계열 데이터를 대체 왜 분석할까? 그 목적은 무엇일까? 결론부터 말하면 "미래 값을 예측하기 위해"서이다. 즉, 시계열이 갖고 있는 법칙성을 발견해 이를 모형화하고, 이 추정된 모형을 통해서 미래값을 예측(forecast)하기 위해 시계열 데이터를 분석하는 것이다.

참고로 한 가지 추가로 알아두어야 할 점은 시계열 구간을 작은 범위에서 큰 구간으로는 변환할 수 있지만 반대로는 불가능하다. 밑의 예시를 보고 이해하자.

- 'Monthly' -> 'Quarterly' -> 'Yearly' (변환 가능!)
- 'Yearly' -> 'Quarterly' -> 'Monthly' (변환 불가능!)

시계열 변동요인과 모형

세상에 존재하는 데이터가 우리가 가장 좋아하는 형태로 안정적이고 이쁜(?) 데이터면 얼마나 좋을까!? 하지만 현실은 그렇지 못하다. 다양한 또는 괴상한 변동이 존재하는 시계열 데이터가 있기 마련이다. 따라서 일정한 패턴에 따라 정의할 수 있는 시계열 변동 모형을 알아보자.

1.

추세(Trend) 변동 : 상승과 하락이 있는 변동

2.

계절(Season) 변동 : 1년안에 월, 분기로 반복되는 패턴

3.

순환(Circulation) 변동 : 경기변동이라고도 하며, 5년, 10년처럼 장기간 동안 간격을 두고 상승, 하락이 주기적으로 반복되는 패턴을 말한다. 이 때는 데이터가 크며 추세변동과 결합해 주로 분석을 진행

4.

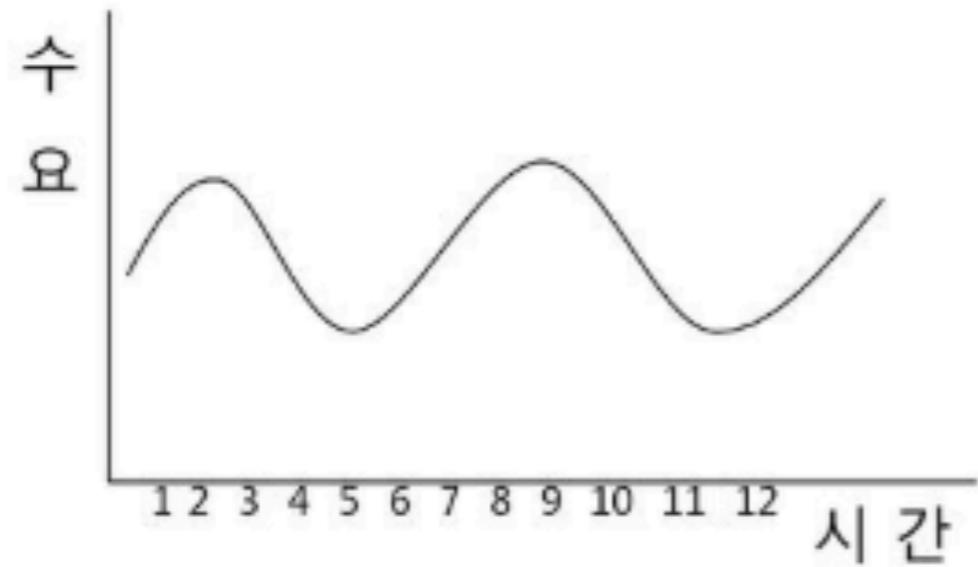
불규칙(irregular) 변동 : 1,2,3번 변동으로는 설명할 수 없는 패턴. 모형은 두가지로 나누어진다.

•

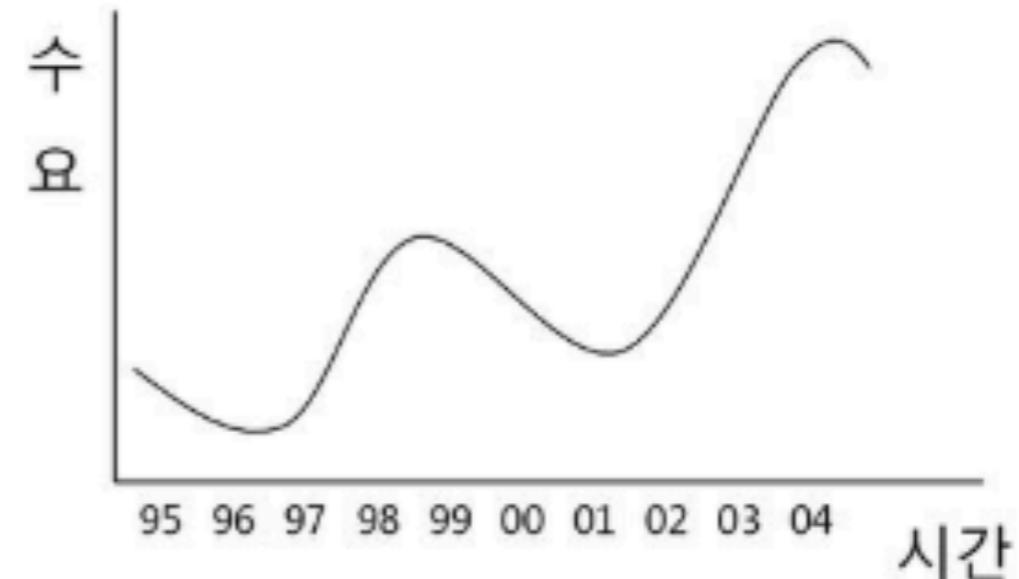
승법(곱셈) 모형 : 추세*계절*순환*불규칙 변동

•

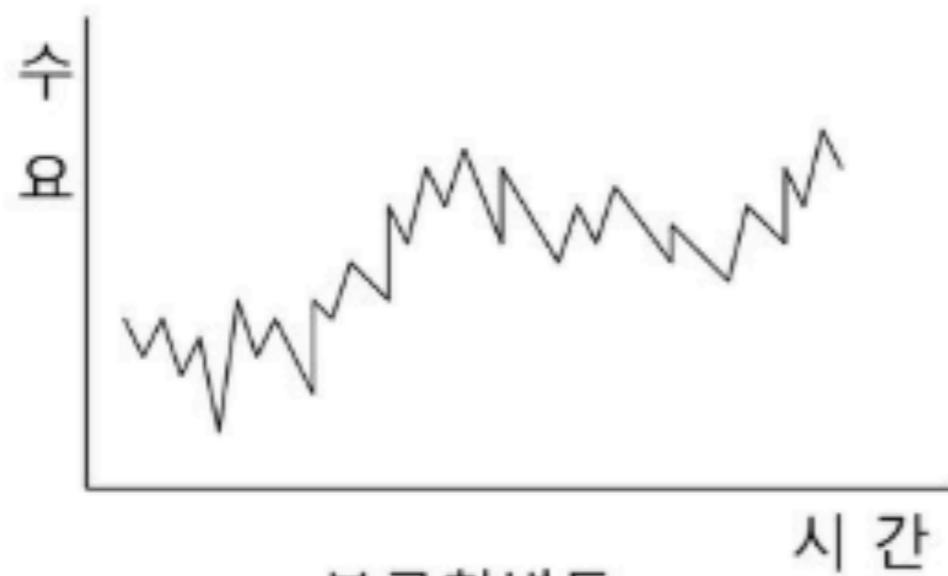
가법(덧셈) 모형 : 추세+계절+순환+불규칙 변동



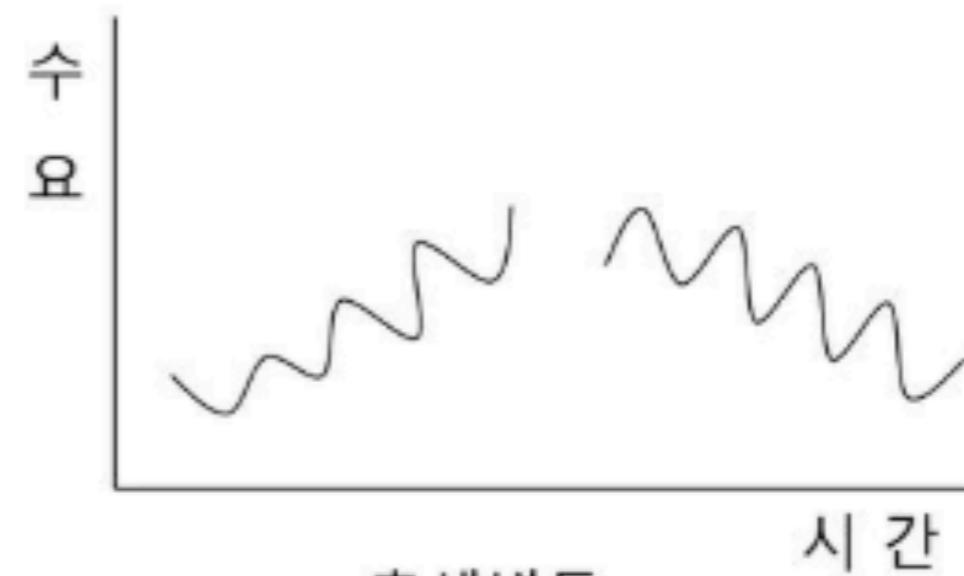
계절변동



순환변동



불규칙변동



추세변동

시계열 자료의 예측방법

다음은 주어진 시계열 자료를 이용해서 미래의 값을 예측하는 방법이다. 크게 양적, 질적 예측방법이 존재한다.

- **양적예측방법** : 과거에 대한 정보(양적 자료)를 이용해 예측에 필요한 경험적 법칙을 추정해 예측하는 방법이다. 쉽게 말해서 "과거의 패턴은 A였으니 미래에서도 A패턴이 지속될거야"라고 예측하는 것이다. 양적예측방법에는 또 두 가지로 나뉘는데,
- **평활법과 분해법** : 주어진 데이터를 잘 설명하는 것에 초점을 맞춘 방법이라 할 수 있다.
- **확률적 시계열 분석** : ARIMA(시간영역), Fourier 분석(주파수 영역) 총 2가지가 존재한다.
- **질적예측방법** : 미래 예측을 위해 전문가들의 주관적 견해를 사용한다. 또는 과거의 정보가 없거나 불충분한 경우에 사용을 한다. 대표적인 방법으로는 델파이 기법과, 시나리오 기법이 있다.

시계열 자료의 예측평가와 기준

위 그림을 보면서 2가지의 평가방법이 있다는 것을 알아두자. 첫 번째 방법은 사전평가와 모형추정을 같이 진행한다는 점, 두 번째 방법은 모형추정을 먼저하고 사전평가를 시행한다는 점이 큰 차이점이다. 그런데 필기에도 써놓았듯이 평가하는 방법이 머신러닝 모델의 성능을 평가할 때와 매우 비슷하다. 특히 두 번째 방법에서 사전평가를 중간단계에 둔 점은 머신러닝 모델의 성능을 중간평가하는 역할을 하는 Validation Data와 매우 비슷하다.

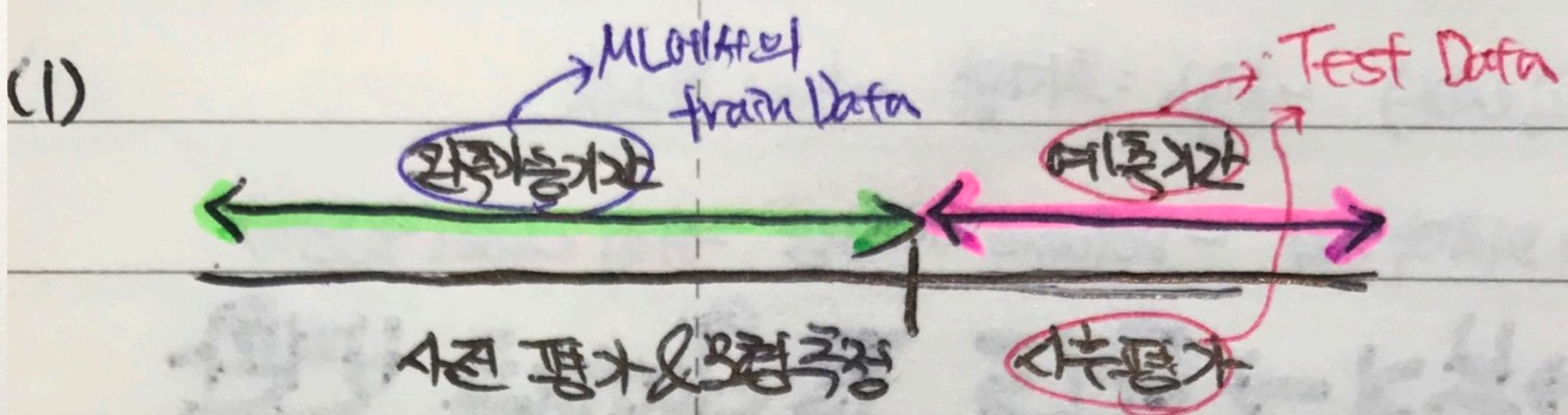
다음은 예측평가의 기준이다. 예측평가는 기본적으로 error값을 기준으로 한다. error란, 실제값에서 예측값을 뺀 차이이다.

$$\begin{array}{cc} \text{실제값} & \text{예측값} \\ \downarrow & \downarrow \\ e_t = y_t - \hat{y}_t \end{array}$$

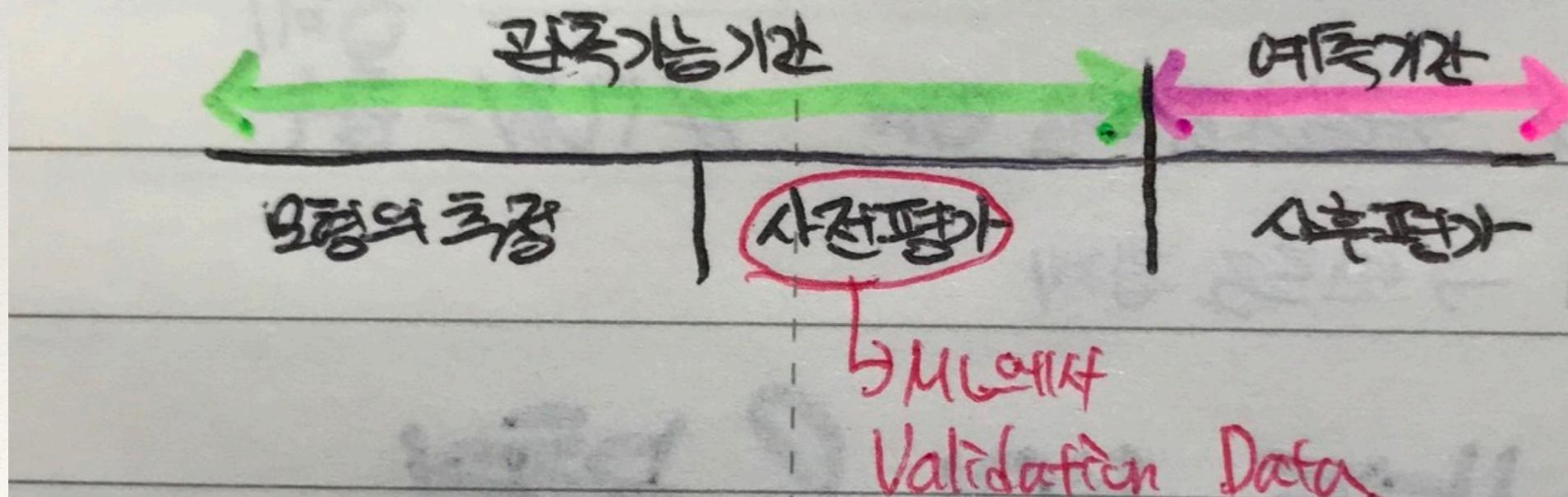
$$\text{Error} = \text{실제값} - \text{예측값}$$

6. 예측 평가

(1)



(2)



예측평가 기준은 크게 5가지가 있으며 눈여겨보고 넘어가자. 이 중에서 가장 많이 쓰이는 기준은 평균 제곱근 오차인 **RMSE**이다.

1.

평균 제곱 오차(MSE)

2.

평균 제곱근 오차(RMSE)

3.

평균 절대 오차(MAE)

4.

평균 절대 백분비 오차(MAPE)

5.

타일의 불일치계수

Mean squared error	$MSE = \frac{1}{n} \sum_{t=1}^n e_t^2$
Root mean squared error	$RMSE = \sqrt{\frac{1}{n} \sum_{t=1}^n e_t^2}$
Mean absolute error	$MAE = \frac{1}{n} \sum_{t=1}^n e_t $
Mean absolute percentage error	$MAPE = \frac{100\%}{n} \sum_{t=1}^n \left \frac{e_t}{y_t} \right $



Seq2seq

Translation, Chatbot

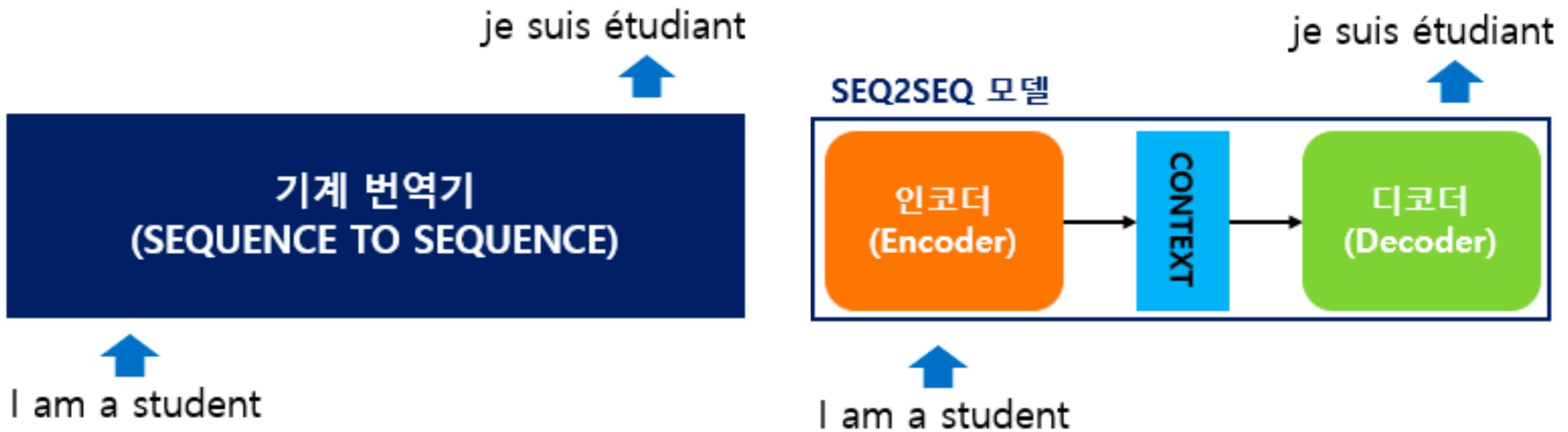
SEQ2SEQ

- 입력 시퀀스를 다른 도메인의 출력 시퀀스로 변환하는 모델
- 인코더와 디코더에 해당하는 두 개의 RNN 레이어로 구성
- 인코더의 출력은 사용하지 않고, 인코더에서 학습한 상태값을 디코더에 전달
- 디코더의 입력에는 시작 태그, 출력에는 종료 태그 연결
- 예측할 때는 디코더 입력을 패딩 태그로 대체

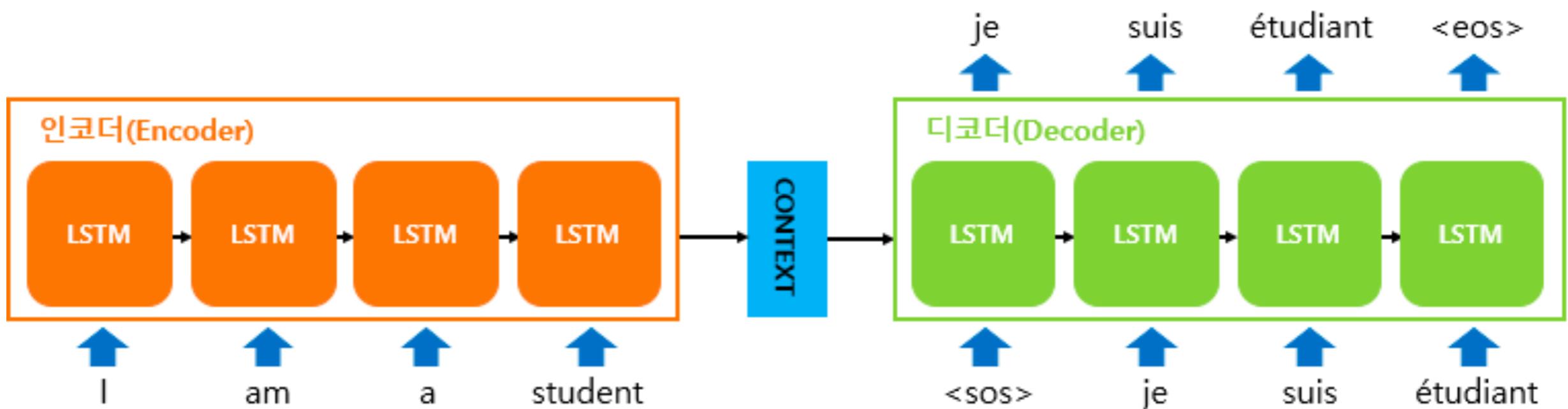
[활용 사례]

- 번역
입력을 한국어, 출력을 영어로 구성
- 챗봇
입력을 질문, 출력을 대답으로 구성
- 그외에도 내용 요약(Text Summarization)과 STT(Speech to Text)에서도 활용

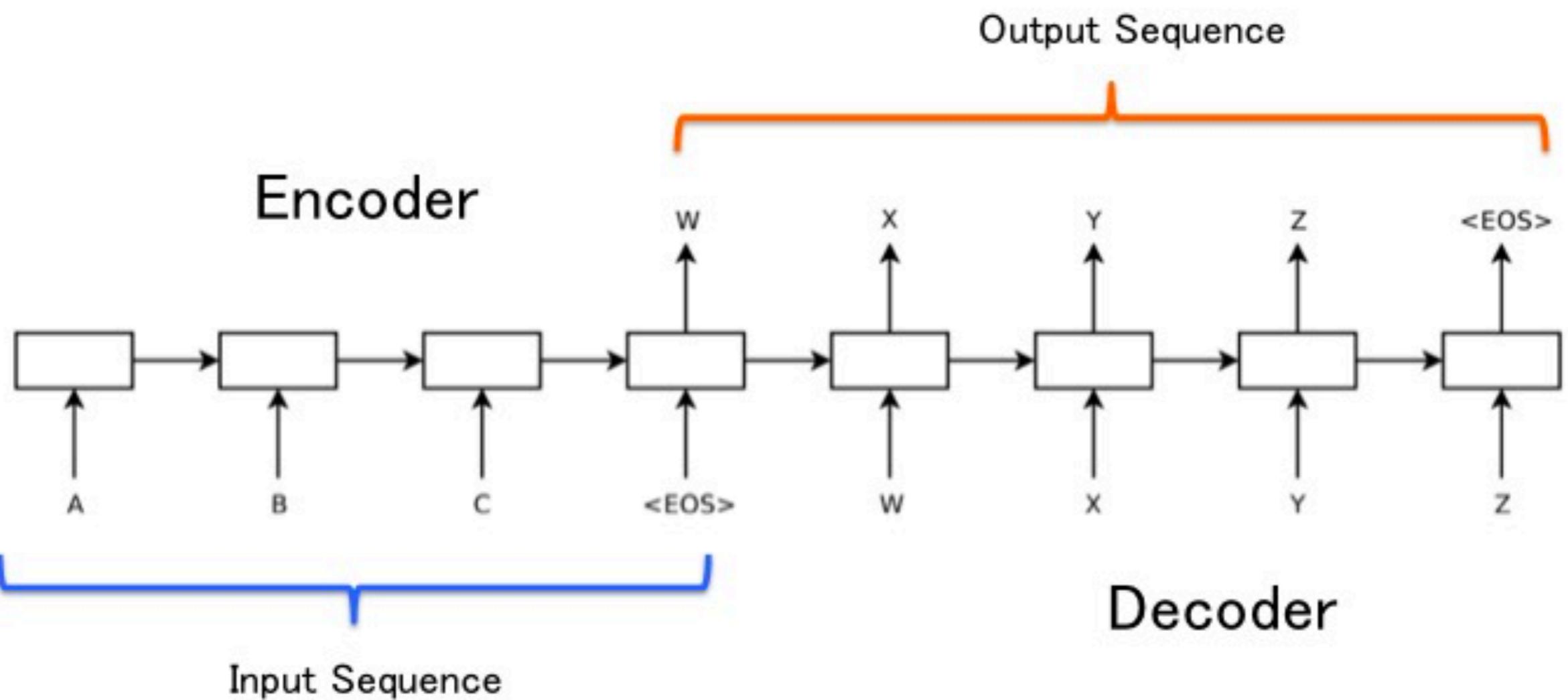
한글 -> 프랑스어



한글 -> 프랑스어



SEQ2SEQ



THE END.