

# NLP팀

**2팀**

박상훈  
곽동길  
박윤아  
김수진  
신민서

# INDEX

---

1. 자연어

2. 단어 표현 방법

3. 언어 모델링

4. 기계 번역

1

자연어

## 자연어

### 자연어

의식적인 계획이나 의도 없이 사용, 반복, 변화의 과정을 거쳐  
인간의 공동체에서 자연스럽게 발생하는 모든 언어



자연어 처리를 통해 컴퓨터가 문서 내 언어의 문맥적 뉘앙스를 포함하여  
문서 내용을 이해할 수 있도록 하고자 함

## 자연어의 특징

### 순차성

자연어의 가장 대표적인 특징으로 순서가 달라지는 경우 의미가 손상됨

### 다의성

하나의 단어나 문장이 상황에 따라 여러 가지 의미를 가질 수 있음

### 함축성

언어는 효율적 극대화를 위해 커뮤니케이션 과정에서  
많은 정보가 생략될 수 있음

## 자연어의 특징

### 순차성

자연어의 가장 대표적인 특징으로 순서가 달라지는 경우 의미가 손상됨

나는 밥을 먹었다

밥이 나를 먹었다

문장의 순서가 바뀌면 문장의 의미가 전혀 달라지게 됨

## 자연어의 특징

### 다의성

하나의 단어나 문장이 상황에 따라 여러 가지 의미를 가질 수 있음

**배**가 달달하게 맛있네!



ню앙스를 통해 뜻을 파악하기 때문에 **문맥**이 매우 중요

## 자연어의 특징

### 함축성

언어는 효율적 극대화를 위해 커뮤니케이션 과정에서  
많은 정보가 생략될 수 있음

내가 한 거 아니야!  
무엇을 하지 않았는지 생략

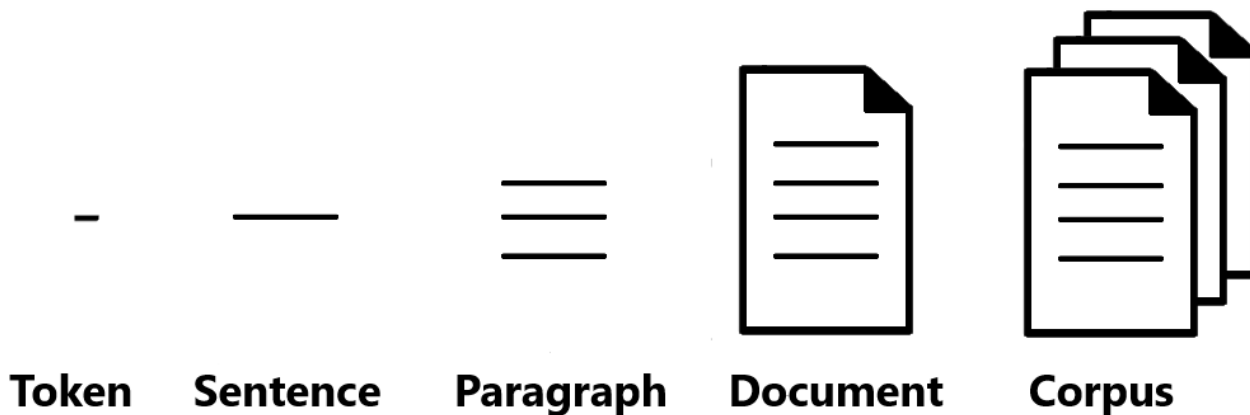
정보의 생략이 많을수록 의미 파악이 어려워짐



## 자연어의 특징

텍스트 데이터는 사용 가능한 데이터 중  
가장 구조화되어 있지 않은 형식 중 하나이며, 매우 복잡함

<텍스트 데이터의 계층 구조>



## 자연어의 특징

텍스트 데이터는 사용 가능한 데이터 중  
가장 구조화되어 있지 않은 형식 중 하나이며, 매우 복잡함

<텍스트 데이터의 계층 구조>

자연어의 특성을 반영할 수 있는 텍스트 데이터의 **변환 방법**을 찾아보자!

Token

Sentence

Paragraph

Document

Corpus

## 자연어의 전처리 : 토큰화

### 토큰화

주어진 말뭉치(corpus)를 **토큰**이라 불리는 단위로 나누는 작업

의미 있는 단위로 토큰을 정의

“This book is for deep learning learners”

토큰화



## 자연어의 전처리 : 토큰화

### 토큰화

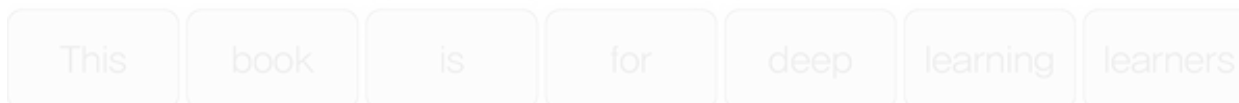
주어진 말뭉치(corpus)를 토큰이라 불리는 단위로 나누는 작업

의미 있는 단위로 토큰을 정의

기준이 정확히 정해져 있지 않은 만큼,

필요에 따라 그 방법을 잘 선택해야 함

토큰화



## 자연어의 전처리 : 토큰화

*“Hey Amazon - my package never arrived [https://www.amazon.com/gp/css/order-history?ref\\_nav\\_orders\\_first](https://www.amazon.com/gp/css/order-history?ref_nav_orders_first) PLEASE FIX ASAP! @amazonhelp”*



공백기준으로 분리

```
['Hey', 'Amazon', '-', 'my', 'package', 'never', 'arrived',  
'https://www.amazon.com/gp/css/order-history?ref_nav_orders_first',  
'PLEASE', 'FIX', 'ASAP!', '@amazonhelp']
```

## 자연어의 전처리 : 토큰화

*“Hey Amazon - my package never arrived [https://www.amazon.com/gp/css/order-history?ref\\_nav\\_orders\\_first](https://www.amazon.com/gp/css/order-history?ref_nav_orders_first) PLEASE FIX ASAP! @amazonhelp”*



nlTK의 word\_tokenize 함수

```
['Hey', 'Amazon', '-', 'my', 'package', 'never', 'arrived', 'https', ':',  
'//www.amazon.com/gp/css/order-history', '?', 'ref_nav_orders_first',  
'PLEASE', 'FIX', 'ASAP', '!', '@', 'amazonhelp']
```

문장 부호와 특수문자(":", "-", "!") 토큰화

## 자연어의 전처리 : 토큰화

*“Hey Amazon - my package never arrived [https://www.amazon.com/gp/css/order-history?ref\\_nav\\_orders\\_first](https://www.amazon.com/gp/css/order-history?ref_nav_orders_first) PLEASE FIX ASAP! @amazonhelp”*



Keras의 text\_to\_word\_sequence 함수

```
['hey', 'amazon', 'my', 'package', 'never', 'arrived', 'https', 'www',  
'amazon', 'com', 'gp', 'css', 'order', 'history', 'ref', 'nav', 'orders',  
'first', 'please', 'fix', 'asap', 'amazonhelp']
```

문장 부호와 특수 문자를 모두 제거하고, 대문자는 소문자로 변환

## 자연어의 전처리 : 정제

정제

주어진 말뭉치(corpus)에서 불필요한 노이즈 데이터를 제거하는 것

등장 빈도가 적은  
단어 제거

길이가 짧은 단어  
제거

불용어 제거





자연어의 전처리 : 정제

불용어

정제

조사나 접미사처럼 문장 내에 자주 등장하지만,  
큰 의미를 가지지 않는 단어



등장 빈도가  
적은 단어 제거

길이가 짧은  
단어 제거

불용어  
제거

불용어 목록은 일반적으로 정해진 것이 없기 때문에,  
중요하지 않다고 판단되는 단어들을 스스로 판단하여 제거

## 자연어의 전처리 : 정제

```
['Hey', 'Amazon', '-', 'my', 'package', 'never', 'arrived', 'https', ':',  
'//www.amazon.com/gp/css/order-history', '?', 'ref_=nav_orders_first',  
'PLEASE', 'FIX', 'ASAP', '!', '@', 'amazonhelp']
```

## 자연어의 전처리 : 정제

`['Hey', 'Amazon', '-', 'my', 'package', 'never', 'arrived', 'https', ':',  
'//www.amazon.com/css/order-history', '?', 'ref_nav_orders_first',  
'PLEASE', 'FIX', 'ASAP', '!', '@', 'amazonhelp']`



등장 빈도가 적은 단어 제거

`['Hey', 'Amazon', '-', 'my', 'package', 'never', 'arrived', 'PLEASE', 'FIX',  
'ASAP', '!', '@', 'amazonhelp']`

웹사이트 주소는 등장 빈도가 적고, 유의미한 정보를 주지 못하므로 제거

## 자연어의 전처리 : 정제

['Hey', 'Amazon', 'X', 'my', 'package', 'never', 'arrived', 'https', 'X',  
'//www.amazon.com/css/order-history', 'X', 'ref\_=nav\_orders\_first',  
'PLEASE', 'FIX', 'ASAP', 'X', 'X', 'amazonhelp']



### 길이가 짧은 단어 제거

['Hey', 'Amazon', 'my', 'package', 'never', 'arrived', 'PLEASE', 'FIX', 'ASAP',  
'amazonhelp']

문장 부호와 특수 문자 제거

## 자연어의 전처리 : 정제

['Hey', 'Amazon', 'I', 'just', 'package', 'never', 'arrived', 'https', '://www.amazon.com/orders/css/order-history', 'ref\_nav\_orders\_first', 'PLEASE', 'FIX', 'ASAP', 'amazonhelp']



### 불용어(stopword) 제거

['Hey', 'Amazon', 'package', 'never', 'arrived', 'PLEASE', 'FIX', 'ASAP', 'amazonhelp']

Nltk 패키지의 stopwords 목록을 이용해 문장 부호와 특수 문자 등 제거

## 자연어의 전처리 : 정규화

### 정규화

표현 방법이 다른 단어들을 통합해서 **같은 단어로 만들어주는 것**

대소문자 통일

표제어 추출

어간 추출

## 자연어의 전처리 : 정규화

```
['Hey', 'Amazon', 'package', 'never', 'arrived', 'PLEASE', 'FIX', 'ASAP',  
'amazonhelp']
```



### 대소문자 통일

```
['hey', 'amazon', 'package', 'never', 'arrived', 'please', 'fix', 'asap',  
'amazonhelp']
```

대문자와 소문자가 의미 차이를 만들지 않는 경우 통일

PLEASE → please

## 자연어의 전처리 : 정규화

### 표제어 추출

```
['hey', 'amazon', 'package', 'never', 'arrive', 'please', 'fix', 'asap',  
'amazonhelp']
```

“기본 사전형 단어”로 변환 arrived → arrive

### 어간 추출

```
['hey', 'amazon', 'packag', 'never', 'arriv', 'pleas', 'fix', 'asap',  
'amazonhelp']
```

어간 추출을 통해 arrived → arriv



# 2

## 단어 표현 방법

## 단어 표현의 필요성

아무리 정제되어 있는 데이터라도  
컴퓨터는 텍스트 데이터를 직접 처리할 수 없음

➡ 텍스트 데이터를 수치형 데이터로 변환해야 함

### 희소 표현

값이 대부분 0인 벡터를 사용하여  
이산적으로 단어의 의미 표현

### 밀집 표현

사용자가 설정한 차원의  
실수 벡터로 단어를 표현

book 과 books도 다른 단어로 간주

## 희소 표현

## One-Hot vector

N개의 단어를 N차원의 표준 단위 벡터로 나타낸 것

John likes to watch movies. Mary likes movies too.

John	1	0	0	0	0	0	0
likes	0	1	0	0	0	0	0
to	0	0	1	0	0	0	0
watch	0	0	0	1	0	0	0
movies	0	0	0	0	1	0	0
<u>mary</u>	0	0	0	0	0	1	0
too	0	0	0	0	0	0	1

단어가 많아질수록 그 차원이 점점 커지게 됨

## 희소 표현

### Bag of Words (BoW)

문서 내 단어들에 대해 각 단어의 **등장 빈도 수**를 세는 방법

John likes to watch movies. Mary likes movies too.

```
{"John":0, "likes":1, "to":2, "watch":3, "movies":4, "Mary":5, "too":6}
```

```
Word to Frequencies : [1, 2, 1, 1, 2, 1, 1]
```

단어들에 대해 인덱스를 각각 부여해준 뒤 각 단어가 빈도수로 표현됨  
단어의 순서는 고려하지 않음

## 희소 표현

### Document-Term Matrix (DTM)

문서가 여러 개일 경우 각 문서마다 단어 빈도를 행렬로 나타내는 방법

Document1: John likes to watch movies. Mary likes movies too.

Document2: I think that movie was awful, but John likes that movie.

Document3: John likes Mary.

	John	likes	to	watch	movies	Mary	too	I	think	that	movie	was	awful
Doc1	1	2	1	1	2	1	1	0	0	0	0	0	0
Doc2	1	1	0	0	0	0	0	2	1	2	2	1	1
Doc3	1	1	0	0	0	1	0	0	0	0	0	0	0

John 의 경우 벡터 [1 1 1]이 됨



## 희소 표현

## DTM 방법의 문제점

### Document-Term Matrix (DTM)

문서가 여러 개일 경우 각 문서마다 단어 빈도를 행렬로 나타내는 방법  
불용어처럼 자주 나오는 단어라면,

그 단어의 중요도에 상관없이 높은 빈도수를 가지게 됨

Document1: John likes to watch movies. Mary likes movies too.

Document2: I think that movie was awful, but John likes that movie.

Document3: John likes Mary.



	John	likes	to	watch	movies	Mary	too	I	think	that	movie	was	awful
Doc1	1	2	1	1	2	1	1	0	0	0	0	0	0
Doc2	1	1	0	0	0	0	0	2	1	2	2	1	1
Doc3	1	1	0	0	0	1	0	0	0	0	0	0	0

중요도가 낮은 단어들에 대해 조치를 취해줘야 함

John 의 경우 [1 1 1]로 나타남

## 희소 표현

TF-IDF

단어의 빈도와 역 문서 빈도를 곱하는 방식

중요도 가중치를 두어 기존의 DTM 방법을 개선함



모든 문서에 통틀어 자주 등장하는 단어는  
중요하지 않을 것이므로 낮은 가중치를 주자!

Ex) The, a, an ...

## 희소 표현

TF, IDF 계산 공식

[1] 단어의 빈도  $tf(t, d) = \frac{\text{Term}(t) \text{ counts in document } (d)}{\text{Total \# of terms in document } (d)}$

[2] 역 문서 빈도  $idf(t) = \log \left( \frac{\text{Total number of documents}}{\# \text{ of documents containing the term}(t)} \right)$

$$TF-IDF(t, d) = tf(t, d) \times idf(t)$$

← 문서 내의 단어
← 개별 문서

각 단어마다 문서별 TF-IDF값을 계산하고 기존의 DTM 행렬의 값들을 대체



## 희소 표현의 문제점



### 공간의 낭비

희소 표현 방법에서는 대부분의 값이 0으로  
정보를 가지고 있지 않기 때문에 비효율성 초래



### 단어 간 의미 관계 부재

One-Hot Encoding , Bow 모두  
모두 단어의 존재 유무, 단어의 빈도 수를 정수형으로 나타냄  
의미적 유사성을 학습하는 데 많은 제약이 있음

## 희소 표현의 문제점



### 공간의 낭비

희소 표현 방법에서는 대부분의 값이 0으로  
정보를 가지고 있지 않기 때문에 비효율성 초래

단어의 **잠재 의미를 반영**하는 학습 방법은 없을까?



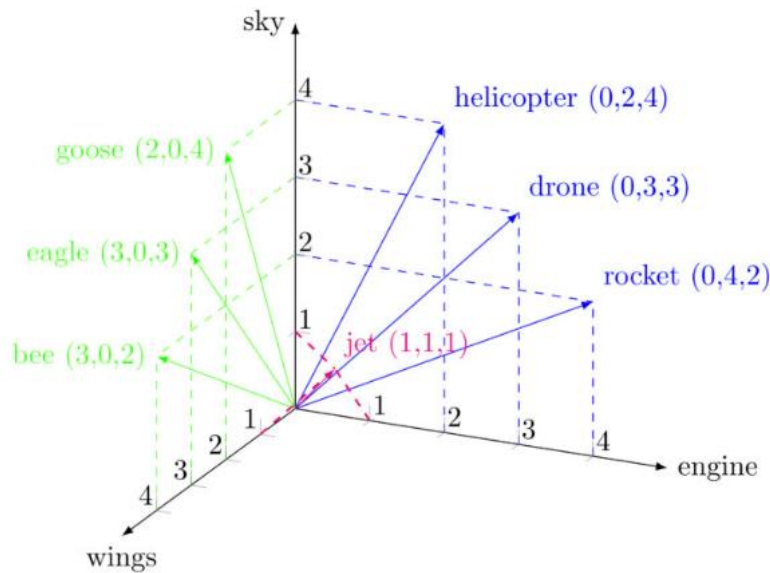
### 단어 간 의미 관계 부재

One-Hot Encoding , Bow 모두  
모두 단어의 존재 유무, 단어의 빈도 수를 정수형으로 나타냄  
의미적 유사성을 학습하는 데 많은 제약이 있음

## 밀집 표현

## 모델 기반 Word Embedding

각 단어를 인공 신경망 학습을 통해 밀집 벡터로 만들어주는 작업



단어들이 저차원 공간의  
실수 값의 점으로  
간단히 표현됨 & **의미 보존**

Word Embedding으로 3차원 공간에 단어를 표현한 예시

## 밀집 표현



모델 기반 Word Embedding

Word Embedding으로 각 벡터의 성분 값은 어떻게 얻어질까?

단어를 랜덤 값을 가지는 밀집 벡터로 초기화한 후  
인공 신경망의 가중치를 학습하는 방식으로 단어 벡터 학습  
단, 밀집 벡터의 차원은 사용자가 설정

일반적으로 LSA, Word2Vec, FastText, Glove 등의 모델들이 학습한 값을 사용

Word Embedding의 차원이 3인 결과

## Word2Vec

Word2Vec

단어를 밀집(dense) 벡터로 표현하는 방법 중 하나로  
의미가 반영된 임베딩 벡터를 학습하기 위해 문맥을 활용함

## Word2Vec

### Word2Vec

단어를 밀집(dense) 벡터로 표현하는 방법 중 하나로  
**의미가 반영된** 임베딩 벡터를 학습하기 위해 **문맥을 활용함**

### <Word2Vec의 기본 아이디어>



단어는 함께 쓰이는 **주변 단어들을 통해** 그 의미를 알 수 있다!

... problems turning into banking crises as ...

정제된 corpus

## Word2Vec

### Word2Vec

단어를 밀집(dense) 벡터로 표현하는 방법 중 하나로  
**의미가 반영된** 임베딩 벡터를 학습하기 위해 **문맥을 활용함**

정제된 corpus 속 t번째 단어를  $w^{(t)}$  라고 할 때, 중심단어  $w^{(t)}$ 의  
주변 단어를  $w^{(t-d)}, \dots, w^{(t-1)}, w^{(t+1)}, \dots, w^{(t+d)}$  라고 정의하고 활용  
단, 간격 d는 사용자가 임의로 설정함

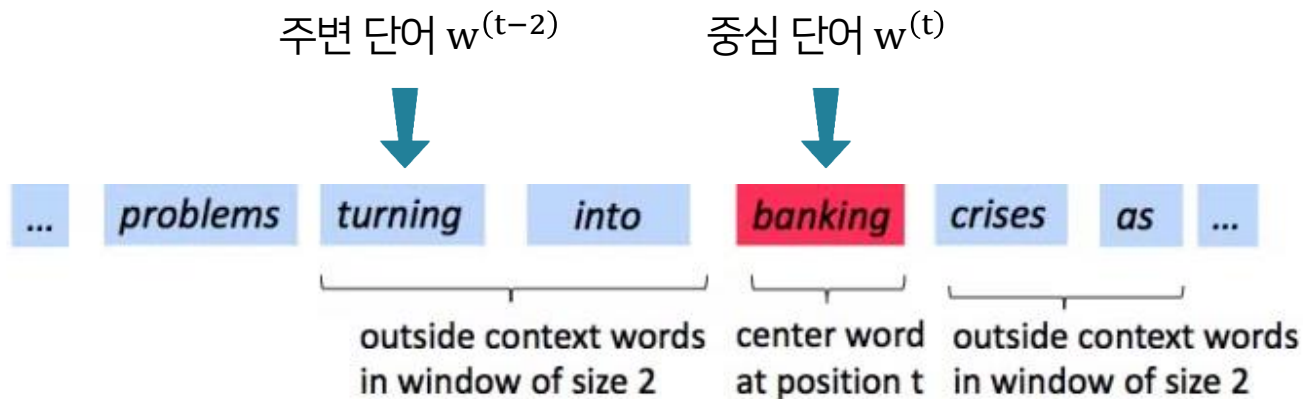
... problems turning into banking crises as ...

정제된 corpus

## Word2Vec

### Word2Vec

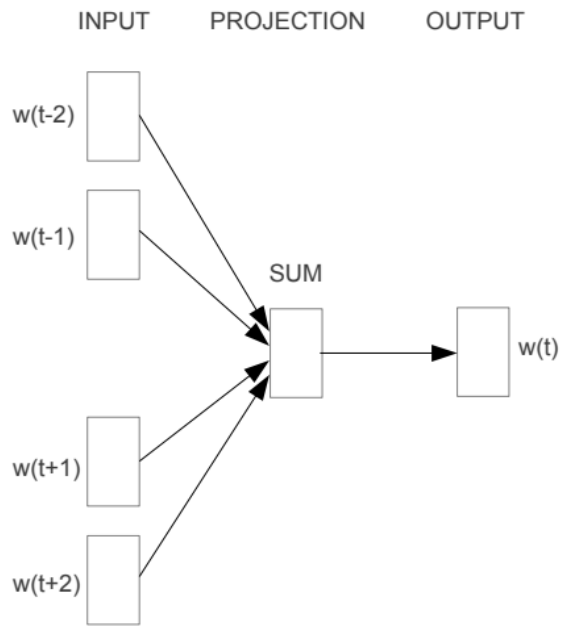
단어를 밀집(dense) 벡터로 표현하는 방법 중 하나로  
의미가 반영된 임베딩 벡터를 학습하기 위해 문맥을 활용함



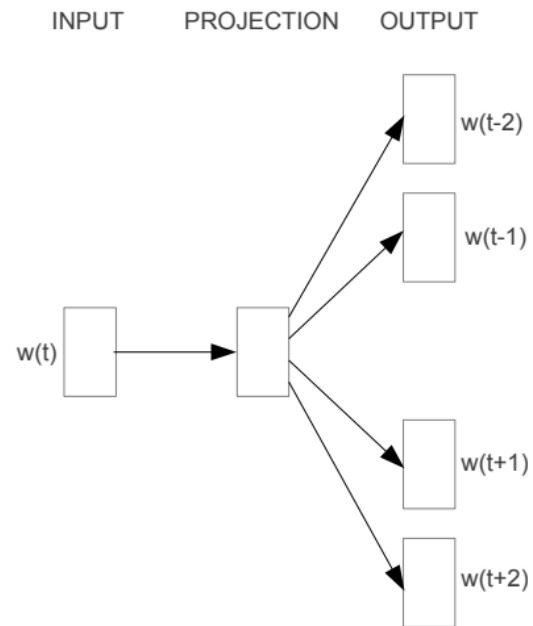
<중심 단어가 banking이고  $d=2$ 일 때>



## Word2Vec



CBOW



Skip-gram

문맥 활용 방법에 따라 CBOW 모델과 Skip-gram 모델로 분류됨

## Word2Vec

### CBOW 모델

중심 단어를 예측하기 위해 주변 단어들을 활용

$y_{(1)}, \dots, y_{(2d)}$  가 주변 단어로 주어졌을 때,  
그 중심에  $y$ 가 있을 확률

$$P_{cbow}(y | y_{(1)}, \dots, y_{(2d)})$$



Corpus의 총 토큰 수

$$\frac{1}{T} \sum_{t=1}^T \log P_{cbow}(w^{(t)} | w^{(t-d)}, \dots, w^{(t+d)})$$

위 Log likelihood를 최대화하는 모델이 문맥을 제대로 보존한다고 가정!

## Word2Vec

### Skip-gram 모델

주변 단어들을 예측하기 위해 중심 단어를 활용

$y$ 가 중심 단어로 주어졌을 때,  
그 주변에  $y_{(1)}, \dots, y_{(2d)}$  가 있을 확률

$$P_{skip}(y_{(1)}^{(t-d)}, \dots, y_{(2d)}^{(t+d)} | y^{(t)})$$



Corpus의 총 토큰 수

$$\frac{1}{T} \sum_{t=1}^T \log P_{skip}(w^{(t-d)}, \dots, w^{(t+d)} | w^{(t)})$$

위 Log likelihood를 최대화하는 모델이 문맥을 제대로 보존한다고 가정!

## Word2Vec

### Skip-gram 모델

주변 단어들을 예측하기 위해 중심 단어를 활용

y가 중심 단어로 주어졌을 때,  
중심 단어가 결정된 상태에서는  $p(w^{(t-d)}, \dots, w^{(t+d)} | w^{(t)})$   
그 주변에  $y, y_{(1)}, \dots, y_{(2d)}$ 가 있을 확률  
주변 단어들이 서로 확률적으로 독립이라고 추가적으로 가정



Corpus의 총 토큰 수

$$\frac{1}{T} \sum_{t=1}^T \sum_{-d \leq j \leq d, j \neq 0} \log P_{\text{skip}}(w^{(t+j)} | w^{(t)})$$

위 Log likelihood를 최대화하는 모델이 문맥을 제대로 보존한다고 가정!

## Skip-gram의 학습

단어 집합의  $i$  번째 단어  $w_i$ 에 대해

$v_i : w_i$  가 중심 단어일 때의 임베딩 벡터

$u_i : w_i$  가 주변 단어일 때의 임베딩 벡터

( $o$ : 주변 단어의 인덱스,  $c$ : 중심 단어의 인덱스)

$$P_{skip}(w_o | w_c) := \frac{\exp(u_o^T v_c)}{\sum_{i \in V} \exp(u_i^T v_c)}$$



$$NLL(U, V) = -\frac{1}{T} \sum_{t=1}^T \sum_{-d \leq j \leq d, j \neq 0} \log P_{skip}(w^{(t+j)} | w^{(t)})$$

Negative Log likelihood를 최소화하도록 경사하강법을 통해 학습하는 모델

## Skip-gram의 학습

단어 집합의  $i$  번째 단어  $w_i$ 에 대해

$v_i : w_i$  가 중심 단어일 때의 임베딩 벡터

$u_i : w_i$  가 주변 단어일 때의 임베딩 벡터

( $o$ : 주변 단어의 인덱스,  $c$ : 중심 단어의 인덱스)



(학습이 진행됨에 따라

중심 단어와 주변 단어의 임베딩 벡터 간 내적 값은 커지고,  
주변 단어가 아닌 단어와의 내적 값은 작아질 것!

$$NLL(U, V) = -\frac{1}{T} \sum_{t=1}^T \sum_{-d \leq j \leq d, j \neq 0} \log P_{skip}(w^{(t+j)} | w^{(t)})$$

Negative Log likelihood를 최소화하도록 경사하강법을 통해 학습하는 모델

## Skip-gram의 학습

단어 집합의  $i$  번째 단어  $w_i$ 에 대해

$v_i : w_i$  가 중심 단어일 때의 임베딩 벡터

$u_i : w_i$  가 주변 단어일 때의 임베딩 벡터

이때 전체 파라미터의 개수는

$U = [u_1 \ u_2 \ \dots \ u_{|V|}], V = [v_1 \ v_2 \ \dots \ v_{|V|}]$ 에서 각각  $d|V|$ 씩 총  $2d|V|$  개!

보통 학습이 끝난 후 행렬  $V = [v_1 \ v_2 \ \dots \ v_{|V|}]$  을 단어의 임베딩 행렬로 최종 선택

$$NLL(U, V) = -\frac{1}{T} \sum_{t=1}^T \sum_{-d \leq j \leq d, j \neq 0} \log P_{skip}(w^{(t+j)} | w^{(t)})$$

Negative Log likelihood를 최소화하도록 경사하강법을 통해 학습하는 모델

# 3

## 언어 모델링



## 언어 모델링

### 언어 모델링

언어라는 현상을 모델링하고자 **문장(단어 시퀀스)에 확률을 할당**하는 모델

이전 단어들을 입력으로 **조건부 확률**을 생성

$$P(Y) = P(y_1, y_2, \dots, y_n)$$

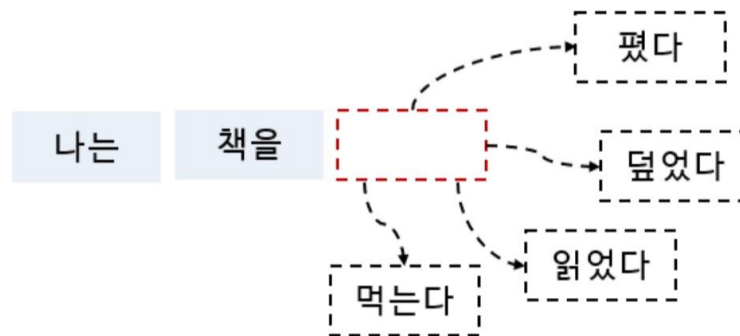


$$P(Y) = P(y_1, y_2, \dots, y_n) = \prod_{t=1}^n P(y_t \mid y_{t-1}, y_{t-2}, \dots, y_1)$$

## 언어 모델링

### 언어 모델링

언어라는 현상을 모델링하고자 **문장(단어 시퀀스)에 확률을 할당**하는 모델



Next word prediction

단순히 확률을 할당하는 것이 아닌 다음 시점에 올 단어를 예측하는 것으로 볼 수 있음

## 언어 모델링

### 언어 모델링

언어라는 현상을 모델링하고자 **문장(단어 시퀀스)에 확률을 할당**하는 모델

### 언어 모델을 만드는 방법

통계적 언어 모델(SLM)

인공 신경망 언어 모델

카운트 기반 모델(Unigram), N-gram

## 통계적 언어 모델(SLM)

$y_t$  가  $y_{t-1}, \dots, y_{t-n+1}$ 에만 확률적으로 종속된다는 가정을 기반으로 함



### Unigram

**N=1일 때** 각 단어가 이전 단어나  
다른 단어와 무관하게  
**독립적으로 발생**한다고 보는 방법

### N-gram

문맥을 고려하기 위해 **이전의  
(N-1)개의 단어만 보는 방법**

## 통계적 언어 모델(SLM)

### 카운트 기반 접근

전체 말뭉치 데이터로부터의 빈도수를 확률 계산에 사용

$$P(\text{I love deep learning}) = P(\text{I})P(\text{love} | \text{I})P(\text{deep} | \text{I love})P(\text{learning} | \text{I love deep})$$



$n=1$ 일 때의 가정을 적용

### Unigram 모델

$$P(\text{I love deep learning}) \approx P(\text{I})P(\text{love})P(\text{deep})P(\text{learning})$$

$$\text{각 단어의 확률: } P(I) = \frac{\text{Count}_{\text{corpus}}(I)}{\text{corpus size}}$$

## 통계적 언어 모델(SLM)

### 카운트 기반 접근

전체 말뭉치 데이터로부터의 빈도수를 확률 계산에 사용

$$P(\text{I love deep learning}) = P(\text{I})P(\text{love} | \text{I})P(\text{deep} | \text{I love})P(\text{learning} | \text{I love deep})$$



n=2일 때의 가정을 적용

### 2-gram 모델

$$P(\text{I love deep learning}) = P(\text{I})P(\text{love} | \text{I})P(\text{deep} | \text{love})P(\text{learning} | \text{deep})$$

love가 I 다음에 오는 빈도:  $P(\text{love} | \text{I}) = \frac{\text{Count}_{\text{corpus}}(\text{I love})}{\text{Count}_{\text{corpus}}(\text{I})}$

## 통계적 언어 모델(SLM)



## 카운트 기반 모델의 한계점

$y_t$  가  $y_{t-1}, \dots, y_{t-n+1}$ 에만 확률적으로 종속된다는 가정을 기반으로 함



유의어 간 연관성을 고려하지 못함



Unigram



중간 단어가 끼어드는 경우 관계 학습이 어려움

$N=1$ 일 때



장기 의존성을 고려하기 어려움

문맥을 고려하기 위해 이전의

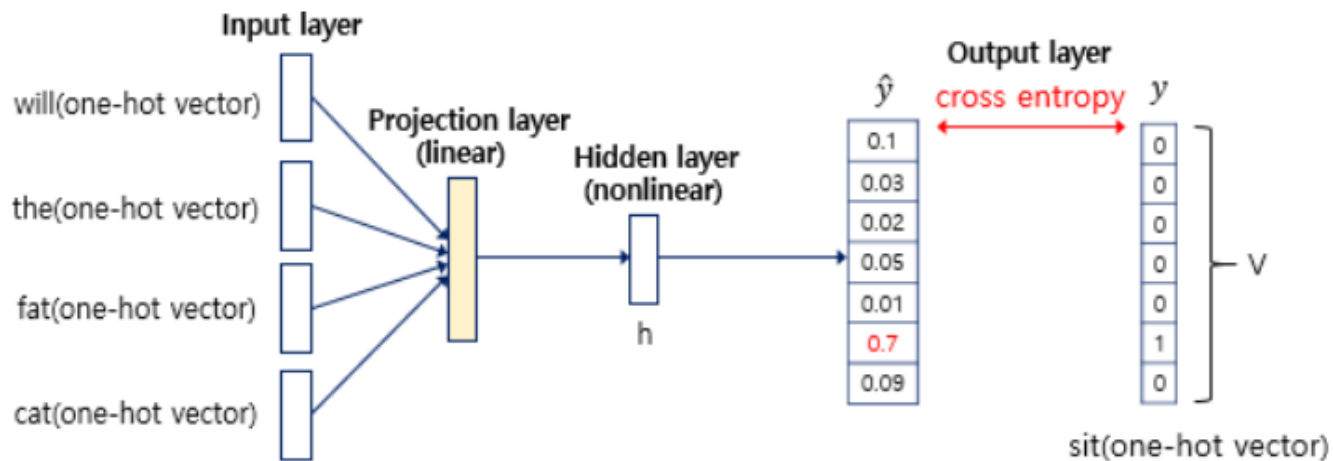
다른 단어와 무관하게

( $N-1$ )개의 단어만 보는 방법

독립적으로 발생한다고 보는 방법

## 인공 신경망 언어 모델

피드 포워드 신경망으로 밀집 표현인 임베딩을 학습해  
n-gram의 문제들을 일부 극복





## 인공 신경망 언어 모델



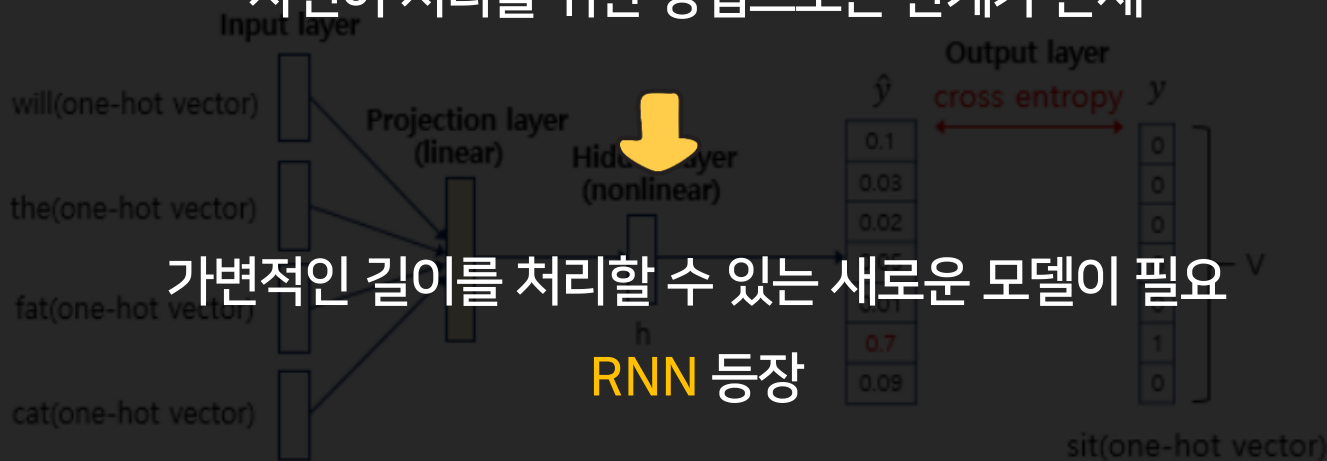
## 피드 포워드 신경망의 문제점

피드 포워드 신경망(Feed Forward Neural Network)

밀집 표현인 임베딩을 학습함으로써  $n$ -gram의 문제들을 일부 극복

고정된 길이의 단어 시퀀스만을 입력으로 받기 때문에

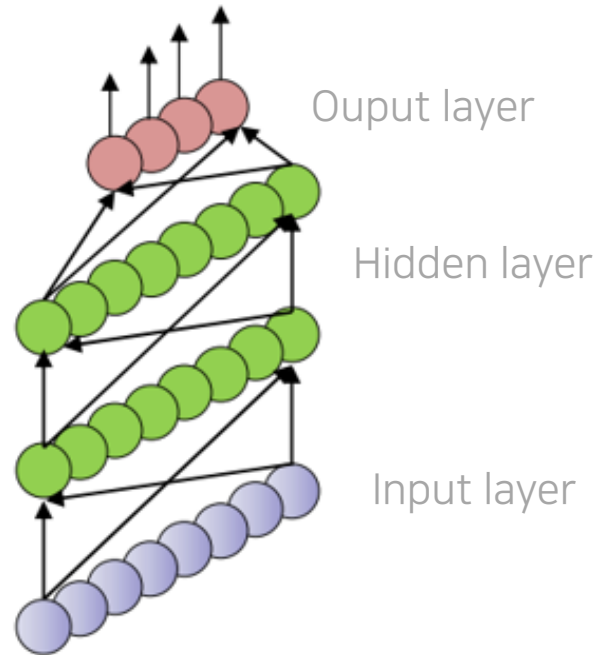
자연어 처리를 위한 방법으로는 한계가 존재



가변적인 길이를 처리할 수 있는 새로운 모델이 필요

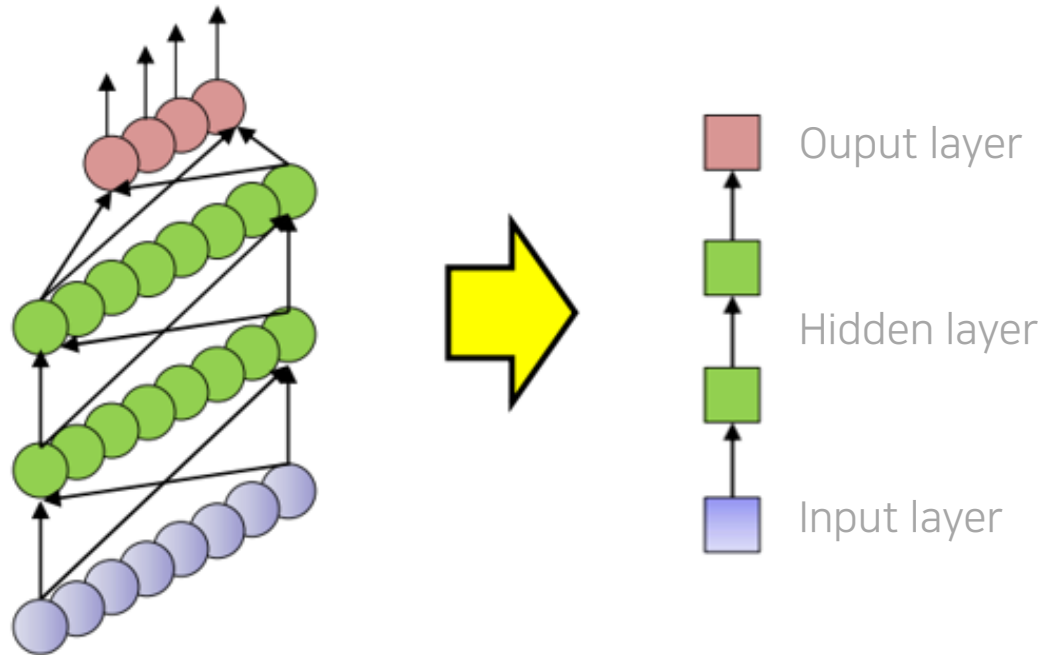
RNN 등장

## Vanilla RNN



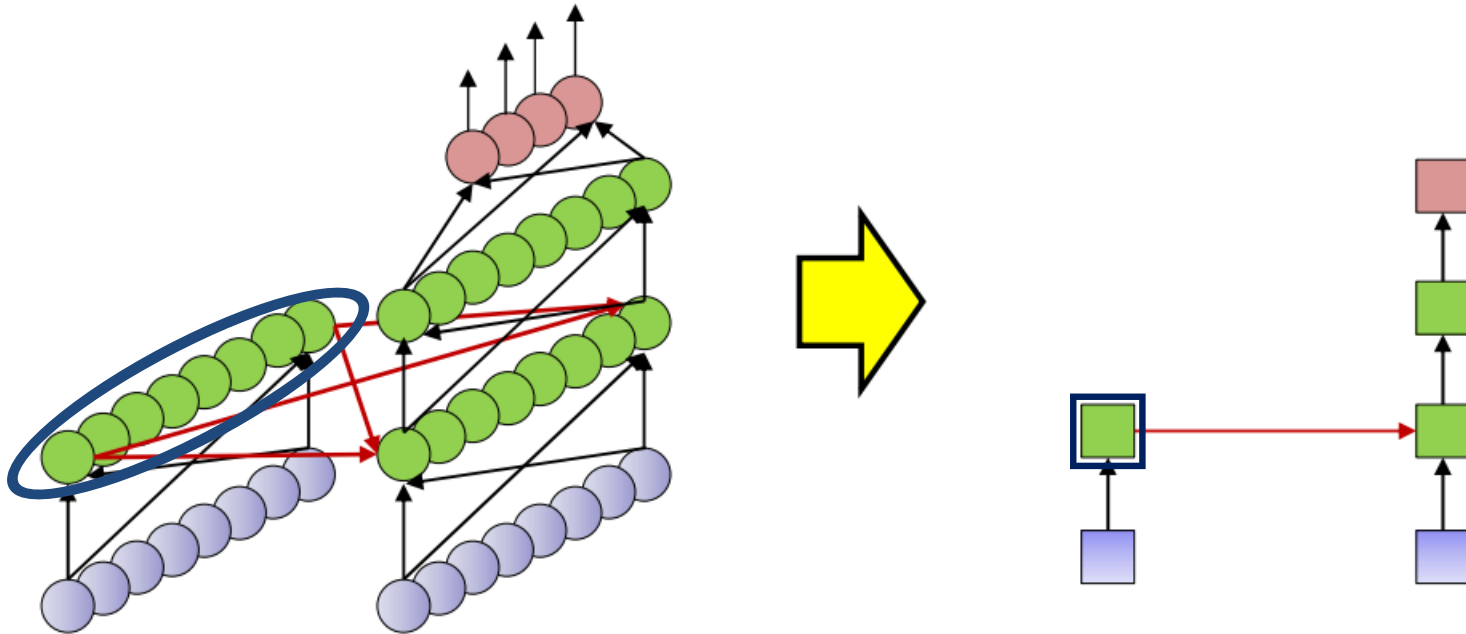
RNN 구조 설명을 위해 우선 기존 네트워크 모델의 표현 방법을 간소화

## Vanilla RNN



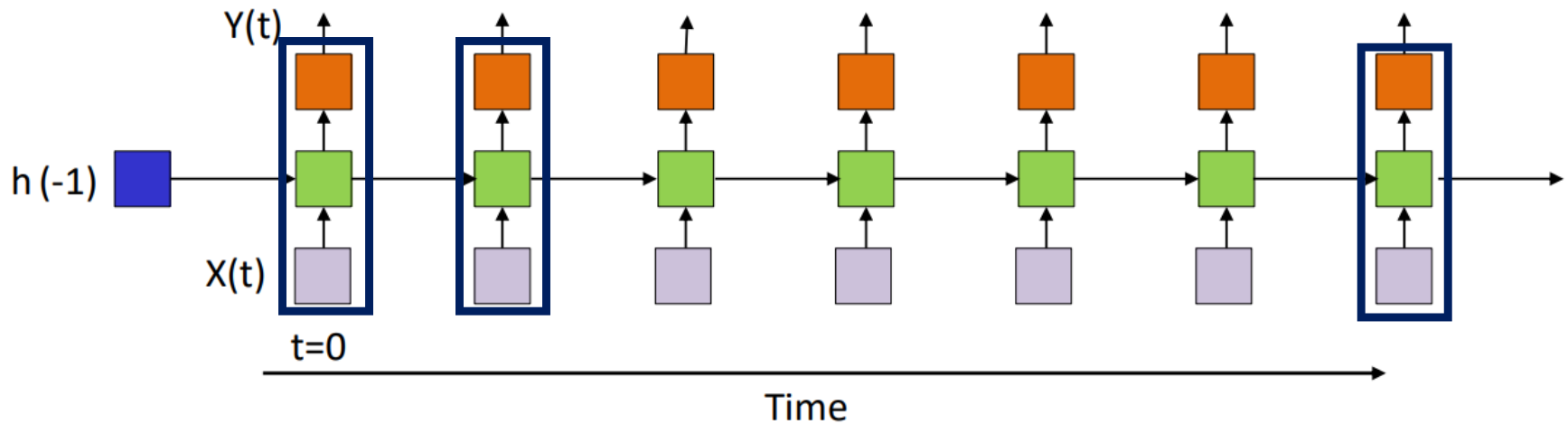
MLP(Multi Layer Perceptron)의 각 Layer를 color box로 표현  
화살표로 연결된 경우, 완전 연결을 의미

## Vanilla RNN



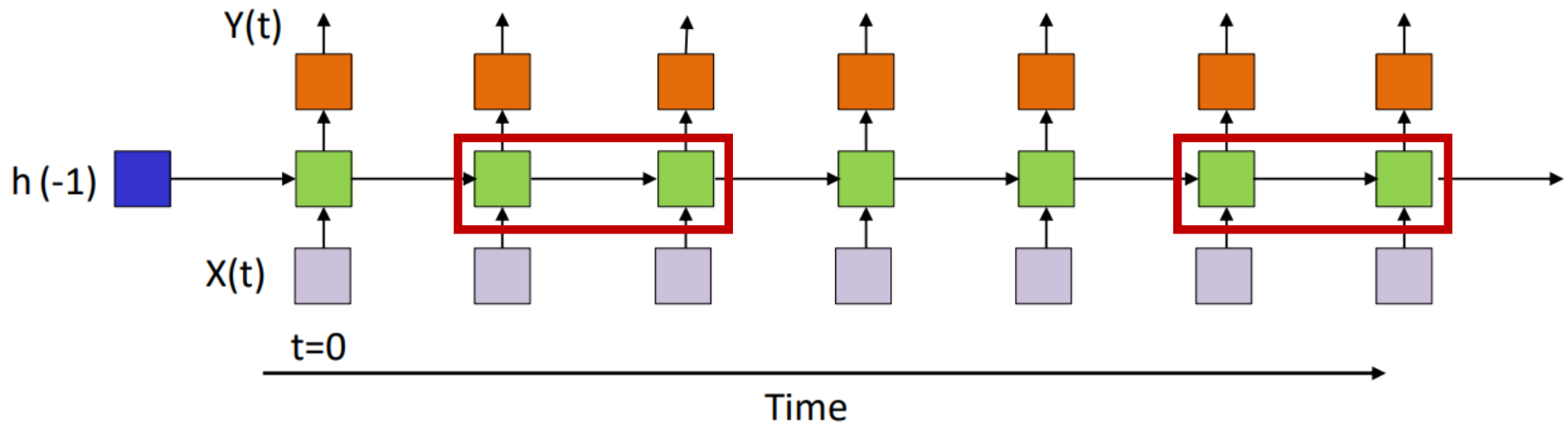
신경망이 Vector들의 sequence를 처리할 수 있도록 구조를 변형  
네모 박스는 각 layer 속 여러 node들의 묶음을 의미하고,  
각 layer의 output 벡터 하나와 대응됨

## Vanilla RNN – RNN의 구조



RNN의 구조는 여러 개의 MLP가 시간 축 방향으로 놓여 있는 형태

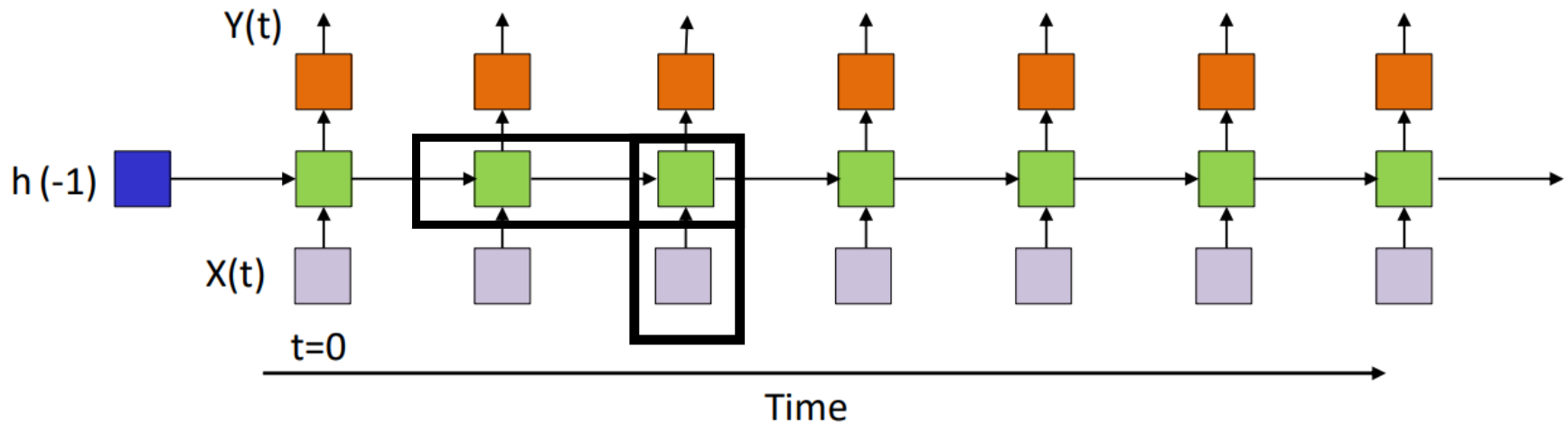
## Vanilla RNN – RNN의 구조



인접한 두 열에 놓인 MLP들은 hidden layer가 시간 축과 같은 방향으로 연결됨

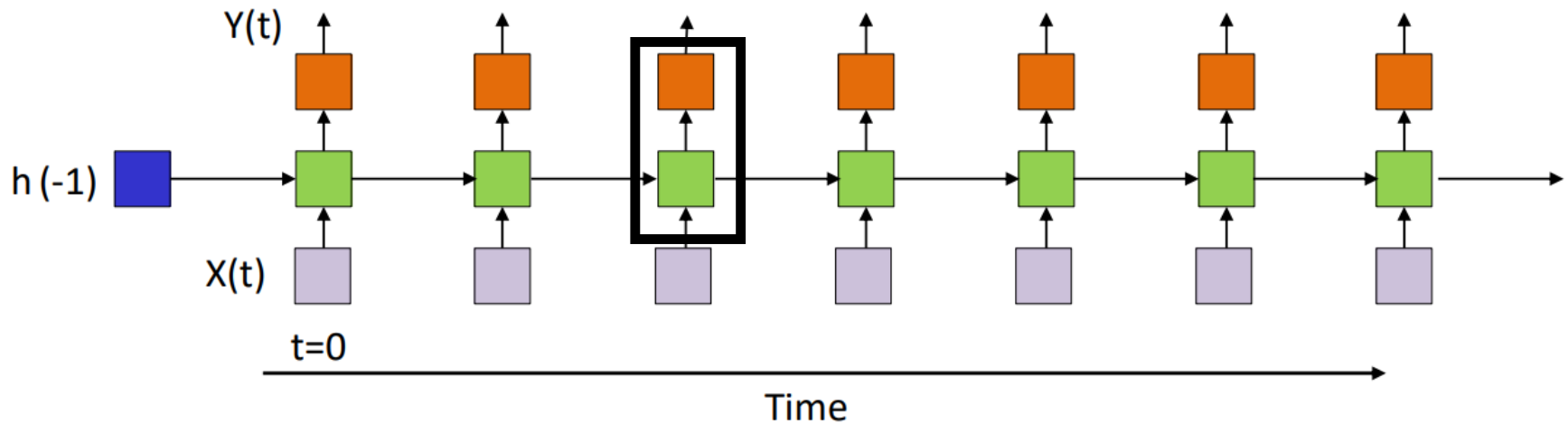
Hidden layer의 output은 **Hidden state**라고 불림

## Vanilla RNN – RNN의 구조



이전 시점의 **Hidden state**  $h_{t-1}$ 와 현재 시점 입력 값  $x_t$ 으로  
현재시점의  $h_t = f(h_{t-1}, x_t)$ 를 계산

## Vanilla RNN – RNN의 구조



$h_t$ 를 이용하여 시점  $t$ 의 output  $y_t = g(h_t)$ 을 계산함



## Vanilla RNN

### RNN을 통한 언어 모델링

현재 시점까지의 단어 정보  $x_t, x_{t-1}, \dots, x_1$ 를 하나의 잠재변수인 hidden state  $h_t$ 로 압축하고 이를 활용해 다음 시점에 올 단어를 예측

$$P(x_{t+1} | x_t, x_{t-1}, \dots, x_1) \approx P(x_{t+1} | h_t) \text{ 라고 가정}$$

$h_t$ : Hidden state



활성화함수로 softmax() 사용

$y_t$ 라는 **확률 벡터**를 얻고 이를  $P(x_{t+1} | x_t, x_{t-1}, \dots, x_1)$ 의 예측 값으로 활용

## Vanilla RNN

### RNN을 통한 언어 모델링

현재 시점까지의 단어 정보  $x_t, x_{t-1}, \dots, x_1$ 를 하나의 잠재변수인 hidden state  $h_t$ 로 압축하고 이를 활용해 다음 시점에 올 단어를 예측

$$P(x_{t+1} \mid x_t, x_{t-1}, \dots, x_1) \approx P(x_{t+1} \mid h_t) \text{ 라고 가정}$$

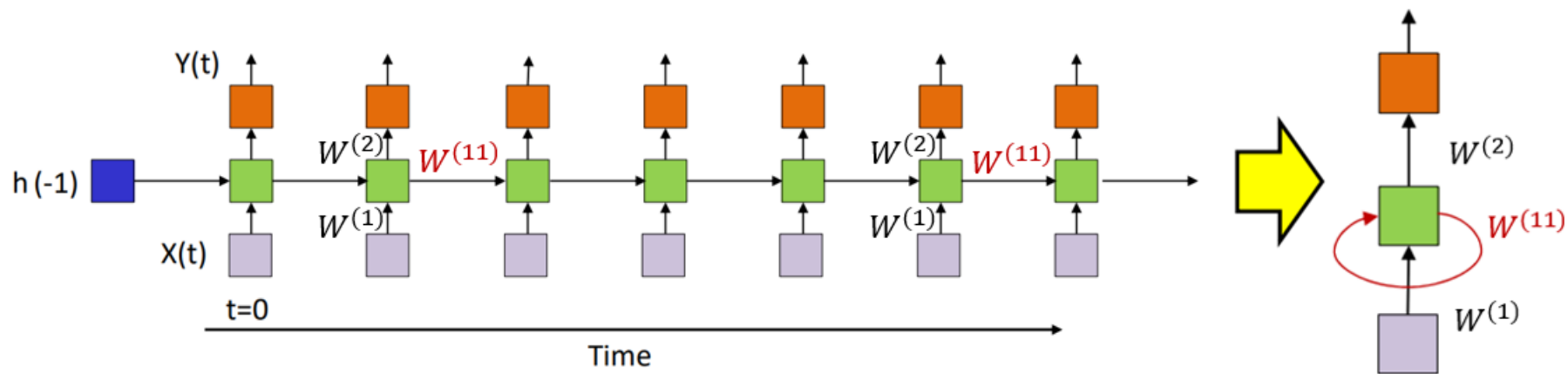
$h_t$ : Hidden state



$$e.g. \ y_1 = \begin{cases} 0.12 & (x_{t+1} \text{가 단어 집합의 1번째 단어일 확률}) \\ 0.04 & (x_{t+1} \text{가 단어 집합의 2번째 단어일 확률}) \\ \vdots & \\ 0.67 & (x_{t+1} \text{가 단어 집합의 } |V| \text{번째 단어일 확률}) \end{cases}$$

차원의 수 = 단어 집합의 크기

## RNN의 연산



시점  $t$ 의 Input vector :  $X(t)$ , Hidden state vector :  $h^{(1)}(t)$ , Output layer vector:  $Y(t)$

$$h^{(1)}(t) = f_1(W^{(1)}x(t) + W^{(11)}h^{(1)}(t-1) + b^{(1)})$$

$$Y(t) = f_2(W^{(2)}h^{(1)}(t) + b^{(2)})$$

$f_1$ 은 주로  $\tanh()$ 를 쓰며, 확률 모델링의 경우  $f_2$ 로  $\text{softmax}()$ 를 사용

## RNN에 사용되는 가중치 행렬



$W^{(1)}$ : Input-to-hidden 가중치로, 매 시점 Hidden layer로 들어온 입력들은 가중치 행렬  $W^{(1)}$  로 선형변환



$W^{(11)}$ : Hidden-to-hidden 가중치로, Hidden layer 사이 재귀적 선형변환 연산을 의미



$W^{(2)}$ : Hidden-to-output 가중치로, 매 시점 Output layer에 입력된 값들은 가중치 행렬  $W^{(2)}$  로 선형변환



parameter들은 시점  $t$ 에 의존하지 않음  
parameter는 모든 시점에서 동일한 값으로 공유됨



## RNN에 사용되는 가중치 행렬



parameter가 모든 시점에서 동일한 값으로  
공유되면 어떤 점이 좋을까?

## RNN의 연산 – parameter sharing

(예시) 길이가 3인 시퀀셜 데이터를 학습

MLP

가중치 행렬의 shape이 그에 맞춰 학습이 진행됨  
길이가 3보다 큰 새로운 데이터에 대한 Task에 적용 불가



가변 길이에 대한 유연성이 떨어짐

## RNN의 연산 – parameter sharing

(예시) 길이가 3인 시퀀셜 데이터를 학습

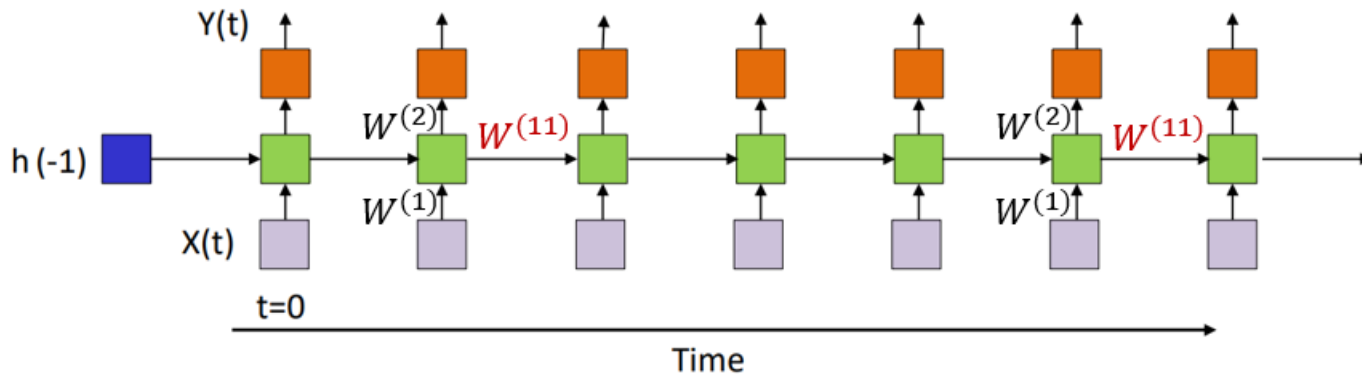
RNN

가중치 행렬의 shape은 시퀀스 길이와 무관하고  
가변 길이의 시퀀셜 데이터가 들어오면 그에 맞게 재귀 연산을 반복 수행함



가변 길이의 시퀀셜 데이터에 유연하게 대처 가능

## BPTT (Back-Propagation Through Time)



RNN을 구성하는 세 가지 가중치 행렬은 모든 시점에 공유됨



가중치 행렬  $W$ 의 변화는 **모든 시점의 hidden state  $h_t$ 를 거쳐** Loss  $L$ 에 영향을 줌

(Hidden-to-output의 경우 모든 시점의 output을 거쳐  $L$ 에 영향을 줌)

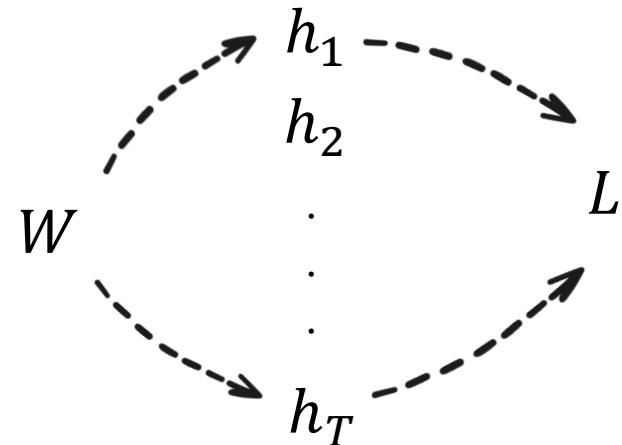


## BPTT (Back-Propagation Through Time)

전미분 (Total Differential)

다수의 변수의 변화에 따른  
다변수 함수의 변화를 근사하기 위한 양

$$dL = \sum_{t=1}^T \frac{\partial L}{\partial h_t} dh_t$$

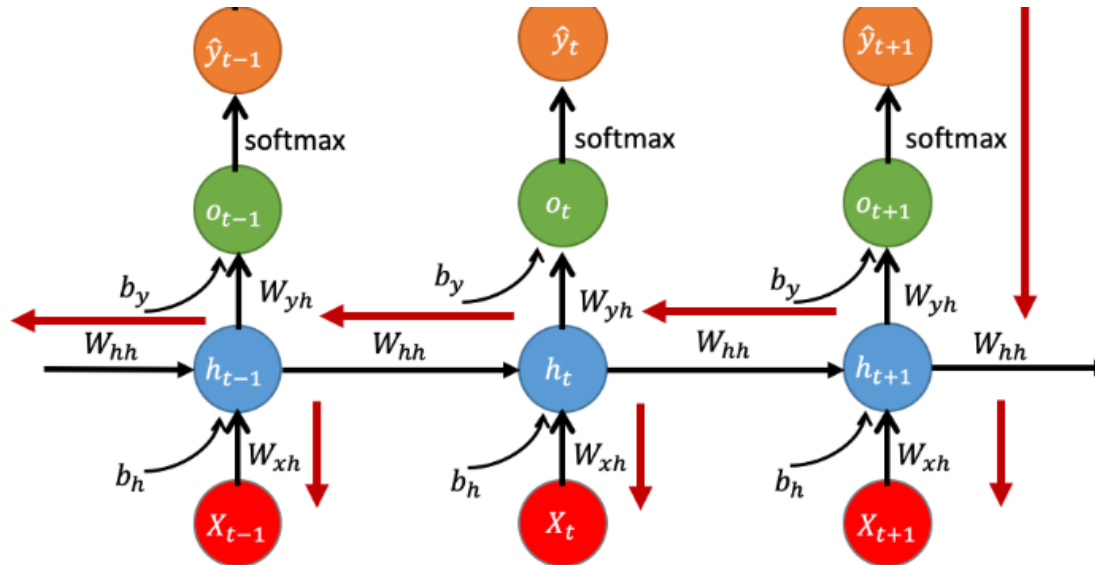


Chain Rule과 결합하여 **가중치에 대한  
Loss의 Gradient를 계산**할 수 있음



$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial W}$$

## BPTT (Back-Propagation Through Time)



Simple RNN의 gradient (편의상 비출력층의 활성화 함수를 항등 함수로 가정, 그림에 맞게 첨자 변경)

$$\frac{\partial L}{\partial W_{hx}} = \sum_{t=1}^T \frac{\partial L}{\partial h_t} x_t^\top$$

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial L}{\partial h_t} h_{t-1}^\top$$

$$\frac{\partial L}{\partial W_{yh}} = \sum_{t=1}^T \frac{\partial L}{\partial o_t} h_t^\top,$$

## BPTT (Back-Propagation Through Time)

Hidden state의 Loss Gradient 계산 결과 (편의상 비출력층의 활성화 함수를 항등 함수로 가정)

$$\frac{\partial L}{\partial h_t} = \sum_{i=t}^T (W_{hh}^\top)^{T-i} W_{qh}^\top \frac{\partial L}{\partial o_{T+t-i}}$$



Hidden-to-hidden 가중치 행렬이 **반복적으로** 곱해지는 형태  
초기 hidden stage일수록 더 자주 곱해짐



결과값의 수치적 안정성은 행렬의 고윳값과 관련이 있음

## RNN의 한계점

### 장기 의존성 (Long-term dependency)

멀리 떨어져 있는 토큰 사이에 의존성이 존재하는 경우

Ex) 코드 생성 언어 모델

```
for (i = 0; i < blocks; i++) {  
    seq = buf[i++];  
    bpf = bd->bd.next + i * search;  
    if (fd) {current = blocked; } }
```

첫 번째 중괄호와 마지막 중괄호 사이에는  
임의의 길이를 갖는 시퀀스가 들어올 수 있음



두 토큰 사이의 관계를 학습하기 위해  
첫 중괄호의 존재를 모델이 기억할 수 있어야 함

## RNN의 한계점

### 장기 의존성 (Long-term dependency)

멀리 떨어져 있는 토큰 사이에 의존성이 존재하는 경우

Ex) 코드 생성 언어 모델

```
for (i = 0; i < blocks; i++) {  
    seq = buf[i++];  
    bpf = bd->bd.next + i * search;  
    if (fd) {current = blocked; } }
```

마지막 중괄호가 입력된 이후엔  
기억할 필요가 없음



특정 시점의 입력 값이 이전 정보를 기억할지  
말지를 결정하는 중요한 신호로 작용 가능

## RNN의 한계점

### Vanishing / Exploding Gradient

역전파를 사용하는 대부분의 심층 신경망이 가지고 있는 문제

Standard MLP의 구조를 가정

$F$ : 선택 가능한 손실함수,  $f_k$ :  $K$ 번째 Hidden Layer의 Output vector

$$Loss = F \left( f_N \left( W_N f_{N-1} \left( W_{N-1} f_{N-2} \left( \dots W_1 X \right) \right) \right), Y_{True} \right)$$

$$\nabla_{f_k} Loss = \nabla F * \nabla f_N * W_N * \nabla f_{N-1} * W_{N-1} \dots \nabla f_{k+1} W_{k+1}$$

$\nabla_{f_k}$ : Jacobian 행렬,  $W_k$ : 가중치 행렬

## 3

## 언어 모델링



두 행렬이 Gradient가 초기 층으로 갈수록 불안정해지게 만드는 요인

Vanishing / Exploding Gradient

Jacobian 행렬

가중치 행렬

활성화 함수의 기울기

값이 담긴 대각행렬

RNN의 경우 정방행렬

$F$ : 선택 가능한 손실함수,  $f_k$ : K번째 Hidden Layer의 Output vector

RNN의 경우  $\tanh$ 를 사용하여

값이 대부분 1 이하

고유벡터의 방향과 나란한

벡터들을 고윳값의 크기만큼

반복적으로 스케일링

$$\nabla_{f_k} \text{Loss} = \nabla F * \nabla f_N * W_N * \nabla f_{N-1} * W_{N-1} \dots \nabla f_{k+1} * W_{k+1}$$

$\nabla_{f_k}$ : Jacobian 행렬,  $W_k$ : K번째 Hidden Layer의 Output vector

이 행렬들이 계속 곱해지는 과정에서 Gradient가 폭발하거나 소멸

## 3

## 언어 모델링

RNN의 한계점 이론적으로 RNN은 첫 시점의 Input이

마지막 시점의 Output까지 영향을 줄 수 있도록 설계

Vanishing / Exploding Gradient

그러나 실제로는 가까운 시점의 데이터 위주로 학습이 진행

역전파를 사용하는 대부분의 심층 신경망이 가지고 있는 문제

Standard MLP의 구조를 가정

$F$ : 선택 가능한 손실함수,  $f_k$ : K번째 Hidden Layer의 Output vector

RNN의 과거의 정보를 기억하는 능력이

$$Loss = F(f_N(W_N f_{N-1}(W_{N-1} f_{N-2}(\dots W_1 X))), Y_{True})$$

활성화 함수와 가중치 행렬에만 강하게 의존

$$\nabla_{f_k} Loss = \nabla F * \nabla f_N * W_N * \nabla f_{N-1} * W_{N-1} \dots \nabla f_{k+1} W_{k+1}$$

$\nabla_{f_k}$ : Jacobian 행렬,  $W_k$ : K번째 Hidden Layer의 Output vector

가까이 위치한 토큰들의 관계만 제대로 학습하게 되고

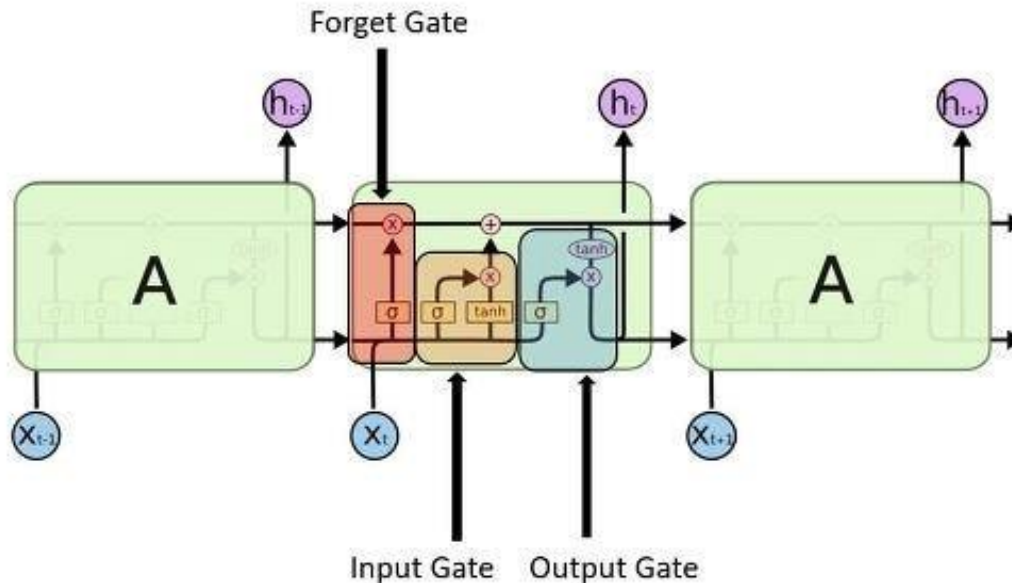
장기 의존성 문제를 해결할 수 없음



## LSTM (Long Short-Term Memory)

### LSTM

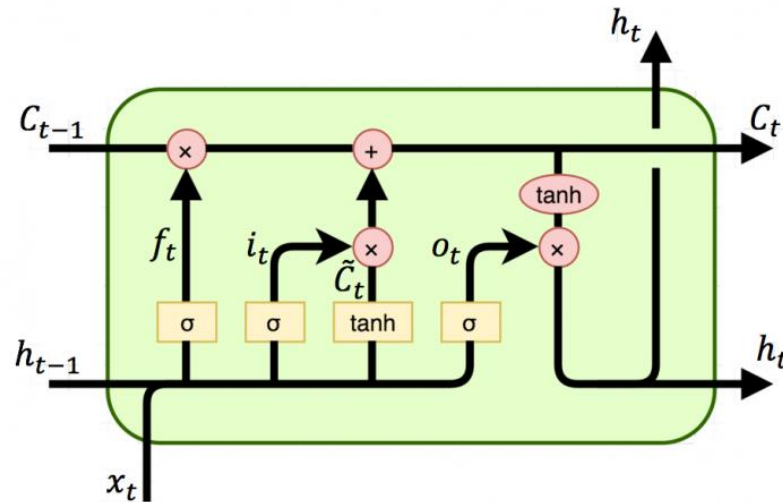
단기 기억을  $h_t$  (hidden state), 장기 기억을  $c_t$  (cell state)가  
각각 담당해 **두 개의 hidden vector**를 갖는 RNN 계열 모델



## LSTM (Long Short-Term Memory)

### LSTM

단기 기억을  $h_t$  (hidden state), 장기 기억을  $c_t$  (cell state)가  
각각 담당해 **두 개의 hidden vector**를 갖는 RNN 계열 모델

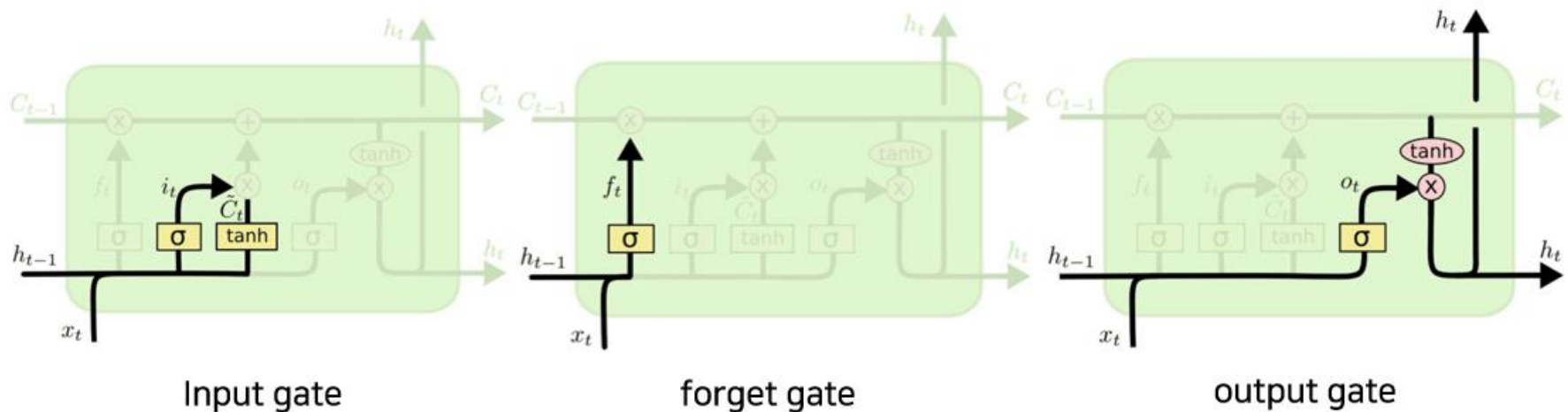


LSTM의 memory cell

## LSTM (Long Short-Term Memory)

### LSTM

단기 기억을  $h_t$  (hidden state), 장기 기억을  $C_t$  (cell state)가  
각각 담당해 **두 개의 hidden vector**를 갖는 RNN 계열 모델



Input gate, Forget gate, Output gate에서 각각

**어떤 정보를 읽고, 지우고, 쓸 것인지 결정**

## LSTM (Long Short-Term Memory)

### 게이트 연산

$$i_t = \sigma(W_{hi}h_{t-1} + W_{xi}x_t + b_i)$$

$$f_t = \sigma(W_{hf}h_{t-1} + W_{xf}x_t + b_f)$$

$$o_t = \sigma(W_{ho}h_{t-1} + W_{xo}x_t + b_o)$$



RNN의 연산과 동일하며 가중치는 시점과 무관하게 존재



0과 1 사이의 값을 가지는 시그모이드 함수를 적용하여  
정보를 얼마나 반영할지에 관한 가중치로 만드는 것



이 때, 현재 시점의 입력값이 유용한 정보로 사용될 수 있음

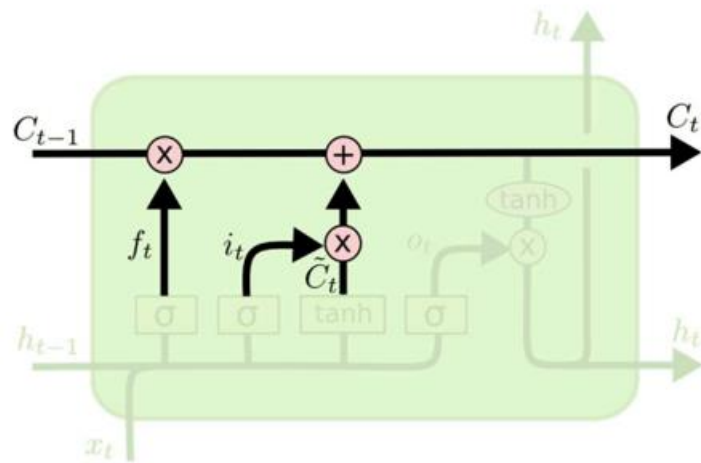
## LSTM의 연산

### 셀 상태 & 은닉 상태 연산

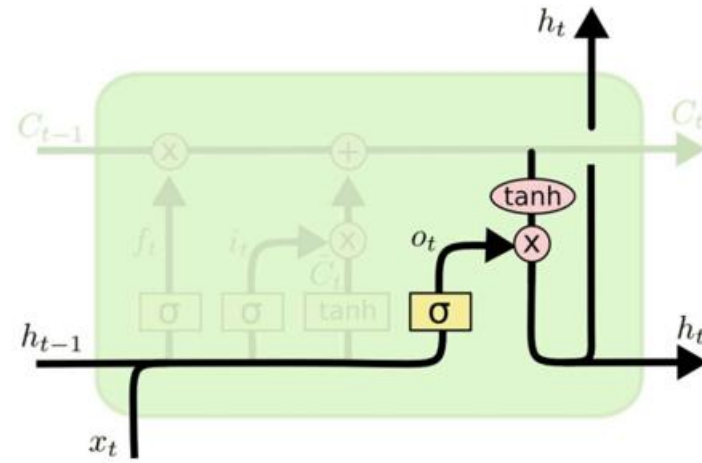
$$\tilde{C}_t = \tanh(W_{hc}h_{t-1} + W_{xc}x_t + b_c)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

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



Cell state

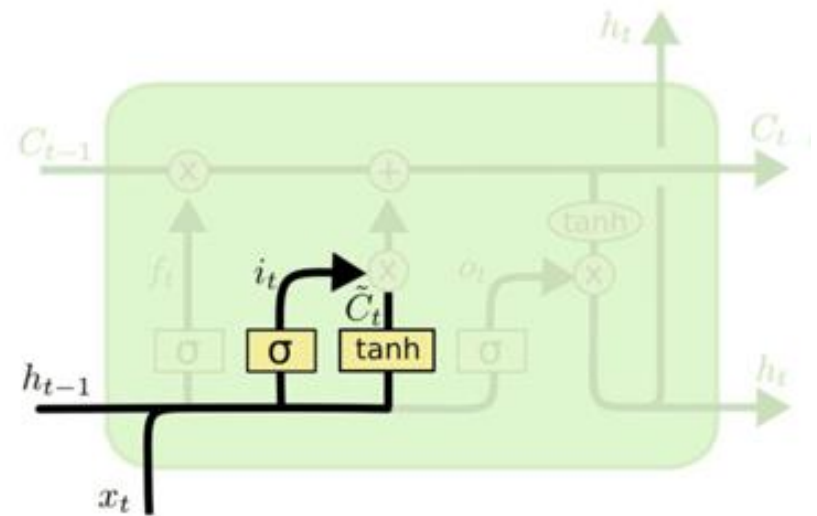


Hidden state

## LSTM의 연산

새로운 cell content를 만드는 과정

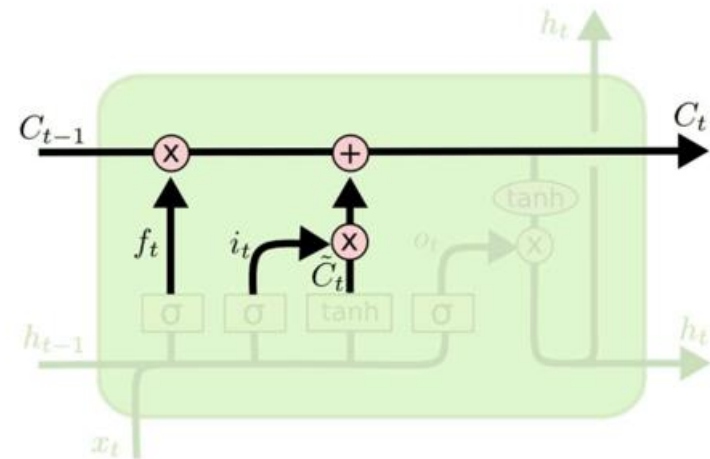
$$\tilde{C}_t = \tanh(W_{hc}h_{t-1} + W_{xc}x_t + b_c)$$



## LSTM의 연산

현재 시점의 cell state를 만드는 과정

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$



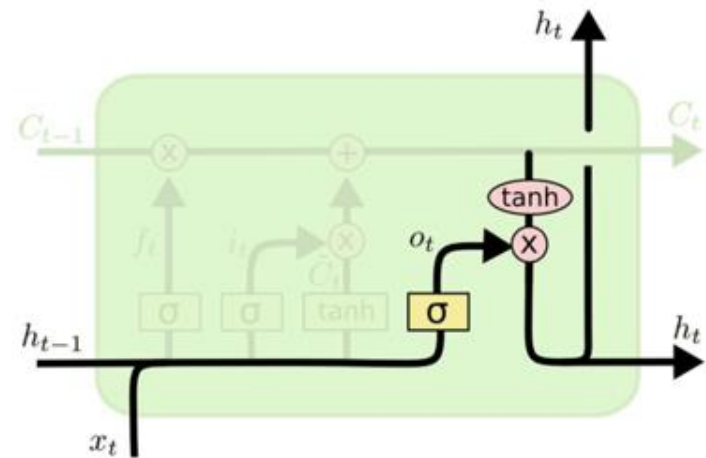
Cell state

이전 상태의 정보를 얼마나 잊고, 현재 상태의 정보를 얼마나 반영할지 결정

## LSTM의 연산

새로운 hidden state를 만드는 과정

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



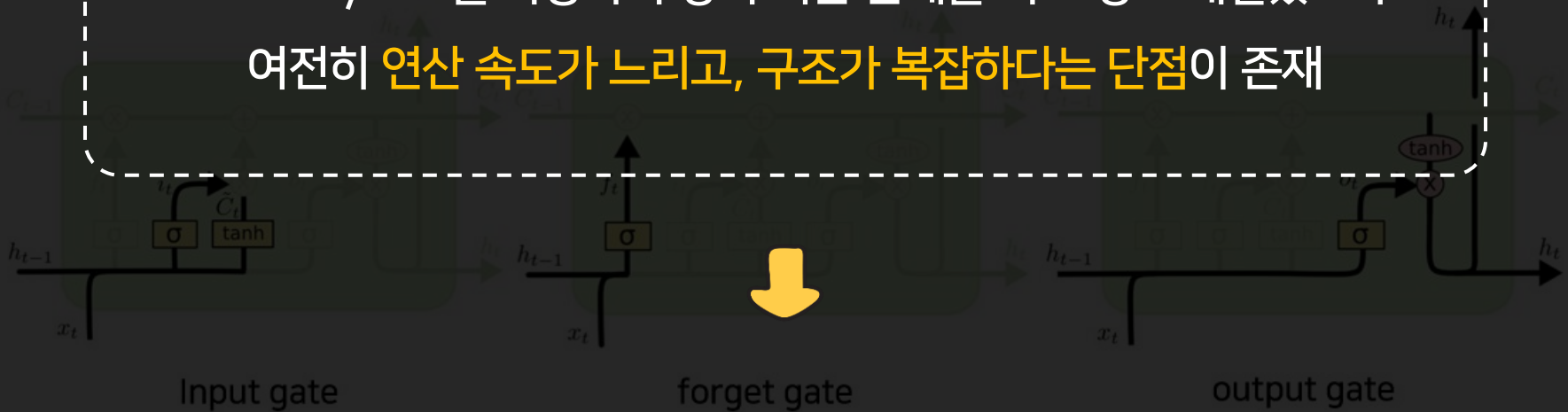


## LSTM의 연산



## LSTM의 문제점

Memory cell을 사용하여 장기 의존 문제를 어느 정도 해결했으나  
여전히 연산 속도가 느리고, 구조가 복잡하다는 단점이 존재

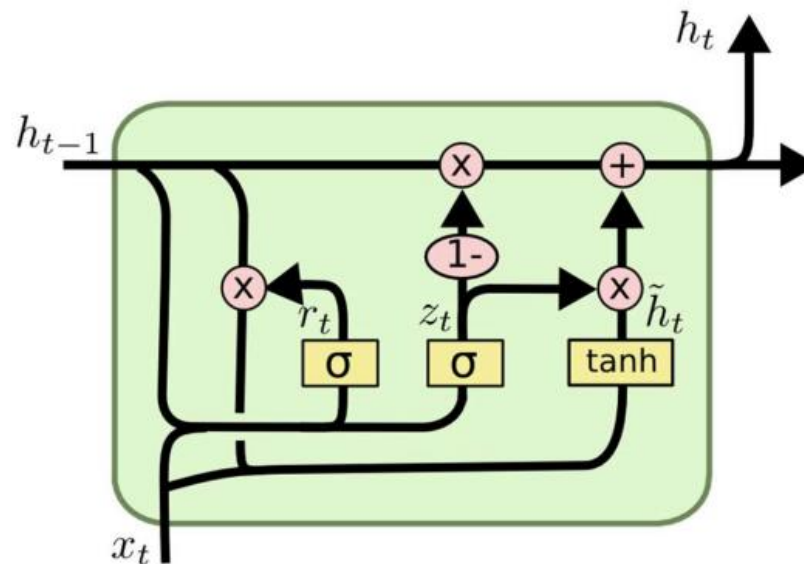


GRU의 등장

## GRU (Gated Recurrent Unit)

### GRU

LSTM의 구조에서 3개의 게이트(input, forget, output)를  
2개(update, reset)로 줄여 **더 간단하게** 만든 모델



LSTM보다 학습 속도가 빠르다고 알려져 있으나 성능은 비슷함

## RNN Variations

### Multi-layer RNN

RNN을 여러 층으로 쌓아 올려  
더 복잡한 구조를  
학습할 수 있도록 만든 모델

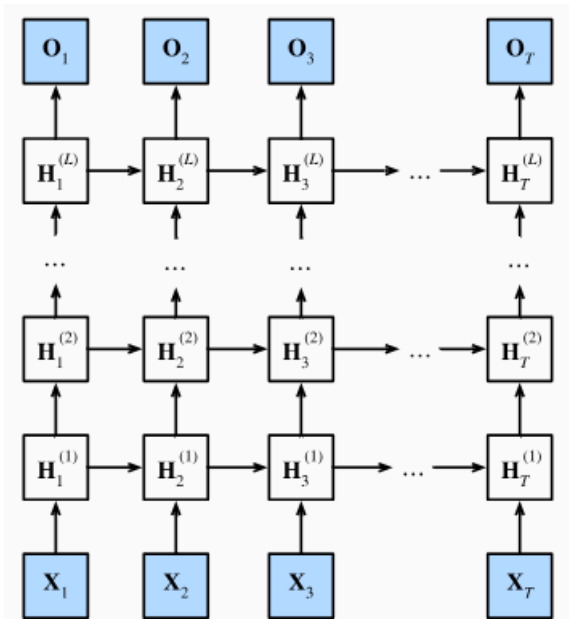
### Bi-directional RNN

이전 시점의 정보만을 활용하는  
Vanilla RNN과 달리 양방향으로  
RNN을 학습시키는 구조

## RNN Variations

### Multi-layer RNN

RNN을 여러 층으로 쌓아 올려 더 복잡한 구조를  
학습할 수 있도록 만든 모델



RNN 대신 LSTM이나 GRU로  
대체하여 동일한 구조 생성 가능

## RNN Variations


### Bi-directional RNN

이전 시점의 정보만을 활용하는 Vanilla RNN과 달리  
양방향으로 RNN을 학습시키는 구조

*The movie was **terribly** exciting!*

단어 *exciting*은 **terribly**를 부정적인 의미에서 긍정적인 의미로 바꾸지만  
Vanilla RNN은 이를 포착하지 못함

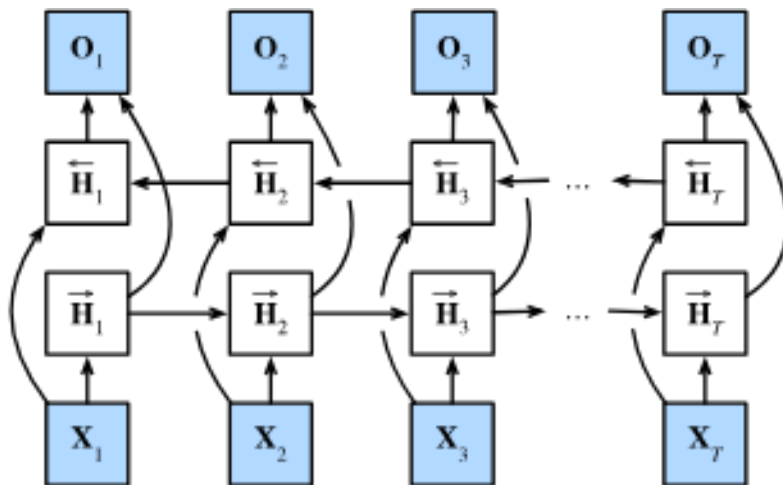


Bidirectional RNN 구조를 사용하여 해결 

## RNN Variations

### Bi-directional RNN

이전 시점의 정보만을 활용하는 Vanilla RNN과 달리  
양방향으로 RNN을 학습시키는 구조



양방향으로 RNN을 각각 정의  
Input sequence는 공유  
parameter는 다르게 하고 각각 학습

## Perplexity (PPL)

### Perplexity

언어 모델링에서 모델 자체적으로 자신의 성능을 수치화하는 지표

$$\text{Perplexity} = P(w_1, w_2, \dots, w_n)^{-\frac{1}{n}}$$

학습이 끝난 언어 모델에 corpus를 넣으면 확률이 할당됨  
좋은 언어 모델이라면, 이 corpus에 높은 확률을 할당할 것!



실제로 존재하고 말이 되는 문장을 넣어주는 것이기 때문!

## Perplexity (PPL)

### Perplexity

언어 모델링에서 모델 자체적으로 자신의 성능을 수치화하는 지표

$$\text{Perplexity} = P(w_1, w_2, \dots, w_n)^{-\frac{1}{n}}$$



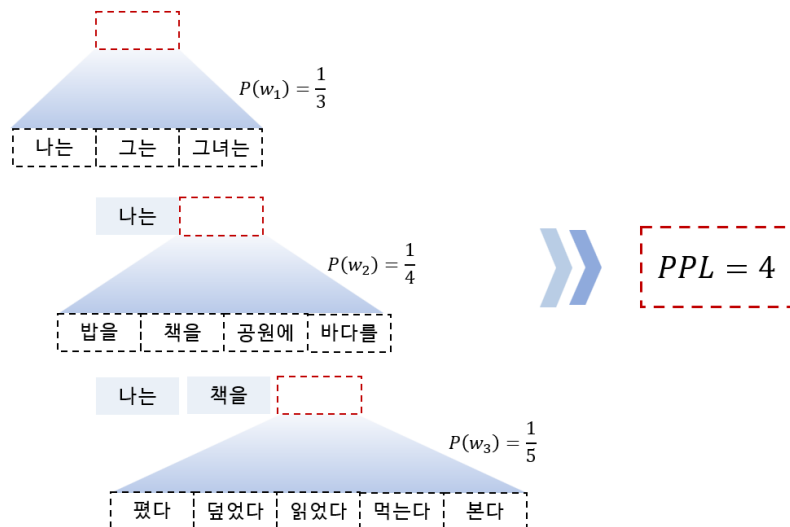
한 모델의 성능을 서로 다른 데이터셋에 대해 측정 하는 경우

corpus의 길이가 길수록 확률 값은 작아지는 경향이  
있기 때문에 거듭제곱으로 이를 정규화



## Perplexity (PPL) | 계산 예시

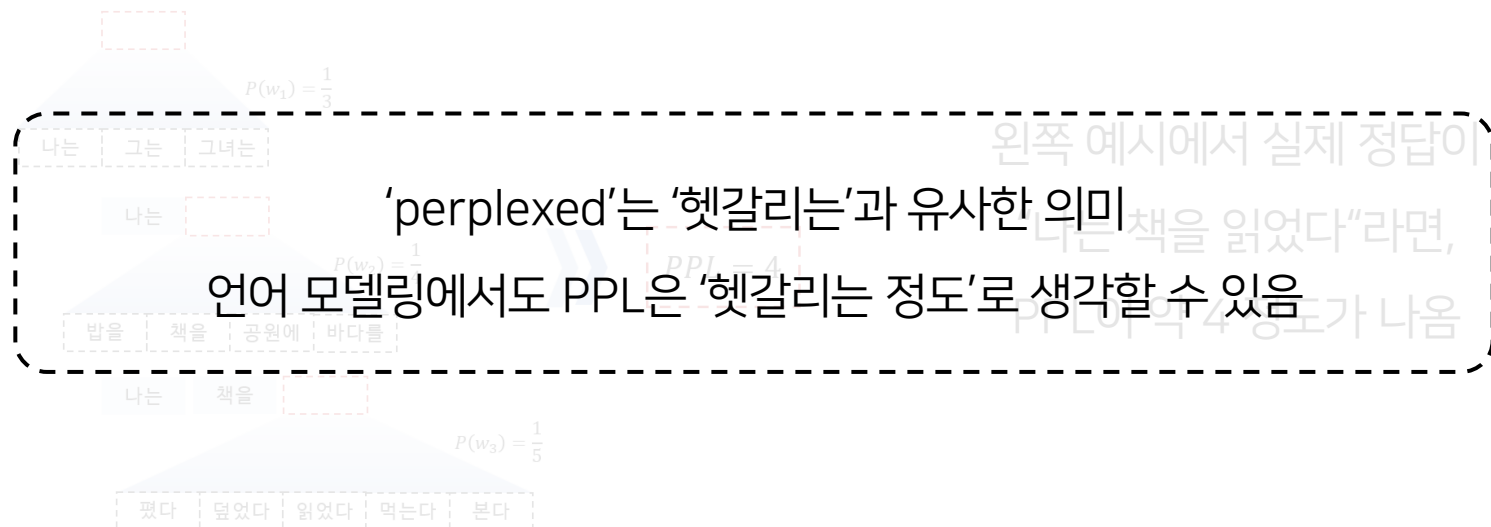
이 수식은 이 모델이 실제 정답 문장을 맞히기까지  
 평균적으로 몇 개의 선택지를 고민하는지를 의미  
 따라서 **PPL이 작을수록 좋음**



왼쪽 예시에서 실제 정답이  
 “나는 책을 읽었다”라면,  
 PPL이 약 4 정도가 나옴

## Perplexity (PPL) | 계산 예시

이 수식은 이 모델이 실제 정답 문장을 맞히기까지  
 평균적으로 몇 개의 선택지를 고민하는지를 의미  
 따라서 **PPL이 작을수록 좋음**



# 4

기계번역

## 기계 번역 (Machine Translation)

기계 번역 *Machine Translation*

자동화된 도구를 사용하여 주어진 문장을 다른 언어로 번역하는 작업

X: 나는 NLP 팀원들이 있어서 안심이 돼



Y: I am relieved to have NLP team members

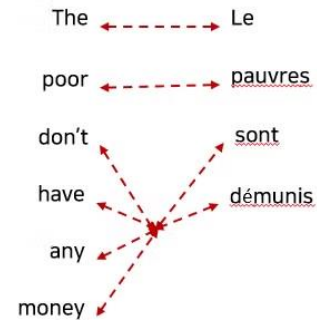
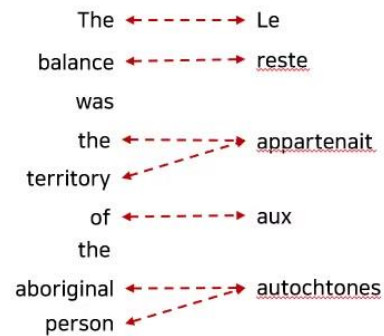
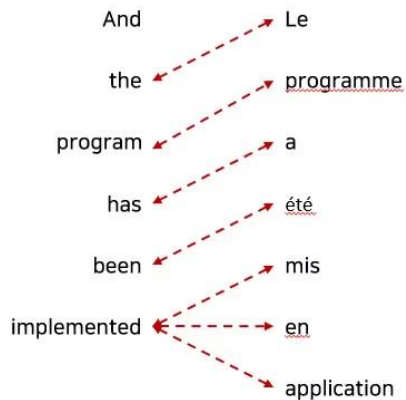
## 기계 번역 | 주요 이슈

대응되는 단어의 부재

대응되는 단어가 **없는** 경우가 있음

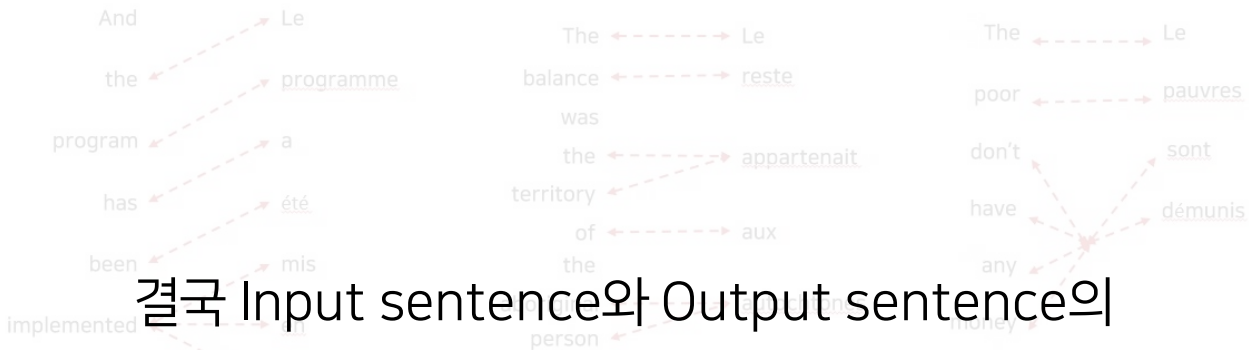
정렬(Alignment)의 문제

**복수**의 단어들끼리 대응될 수 있음



## 기계 번역 | 주요 이슈

대응5



결국 Input sentence와 Output sentence의  
단어 간 일대일 대응 관계가 성립하지 않는다는 것을 고려해

정렬(Alignment)의 문제 모델을 설계해야 함!

어떤 단어는 하나의 단어에만 대응하지만,  
어떤 단어는 여러 개에 대응하고,  
또 여러 개의 단어가 여러 개에 대응하기도 함

## Seq2Seq

과거 (~2014년)에는 데이터로부터 **확률적 모델을 학습하는**  
**SMT(Statistical Machine Translation)** 모델이 사용됨  
그러나 SMT는 매우 복잡했고, 큰 성과를 거두지 못함



2014년, 하나의 **end-to-end 모델로 기계 번역을 할 수 있는**  
**NMT(Neural Machine Translation)**가 등장하였고,  
자연어 처리 분야에서 급진적인 진보를 이룸

구글 번역의 경우, 2016년 SMT에서 NMT로 전환했다고 알려짐

## Seq2Seq

과거 (~2014년)에는 데이터로부터 확률적 모델을 학습하는  
SMT(Statistical Machine Translation)이 NMT에서 사용되는 모델이 사용됨  
Sequence-to-Sequence model(Seq2seq)



2014년, 하... 을 할 수 있는  
NMT(Neural Machine Translation)이 등장하였고,  
자연... 이름

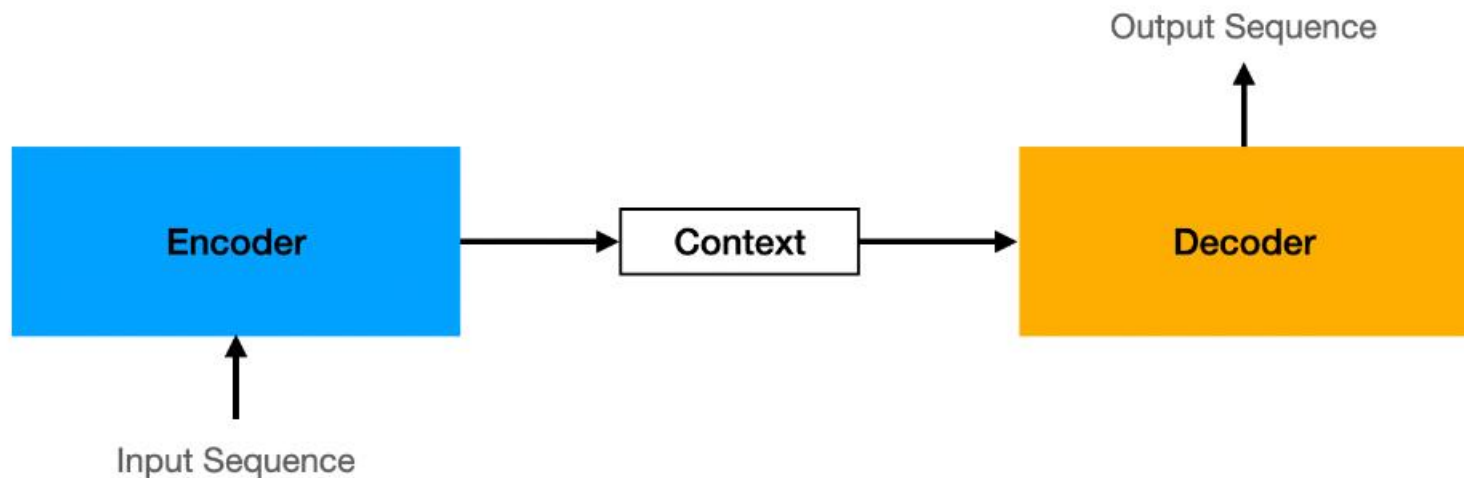
구글 번역의 경우, 2016년 SMT에서 NMT로 전환했다고 알려짐



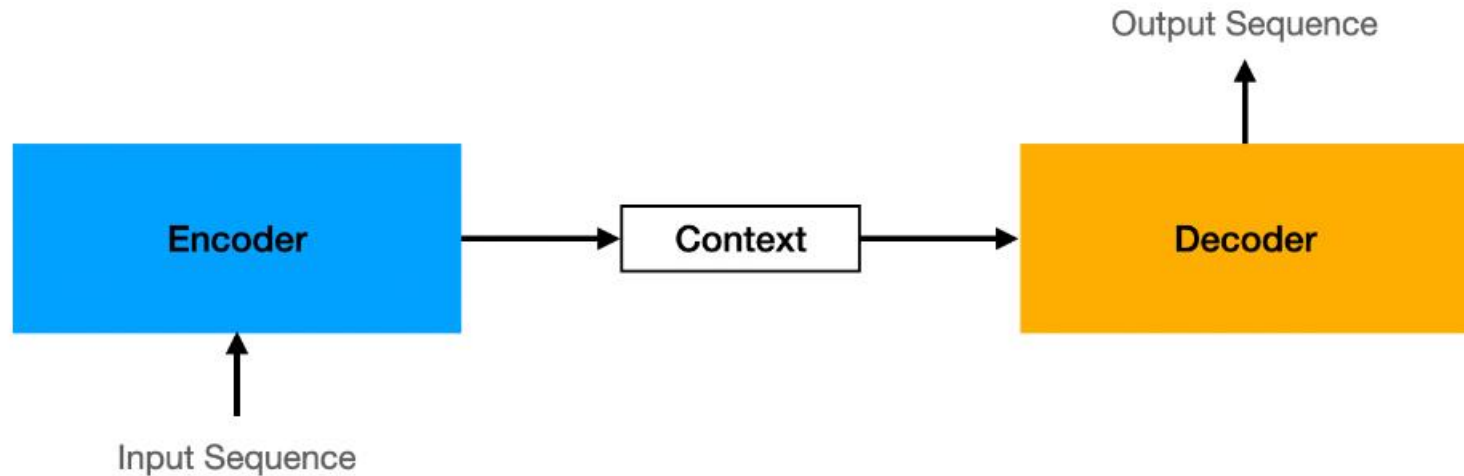
## Seq2Seq

### Seq2Seq

Sequence를 입력 받고 Sequence를 출력하는 모델로,  
Encoder-Decoder 구조로 되어 있음

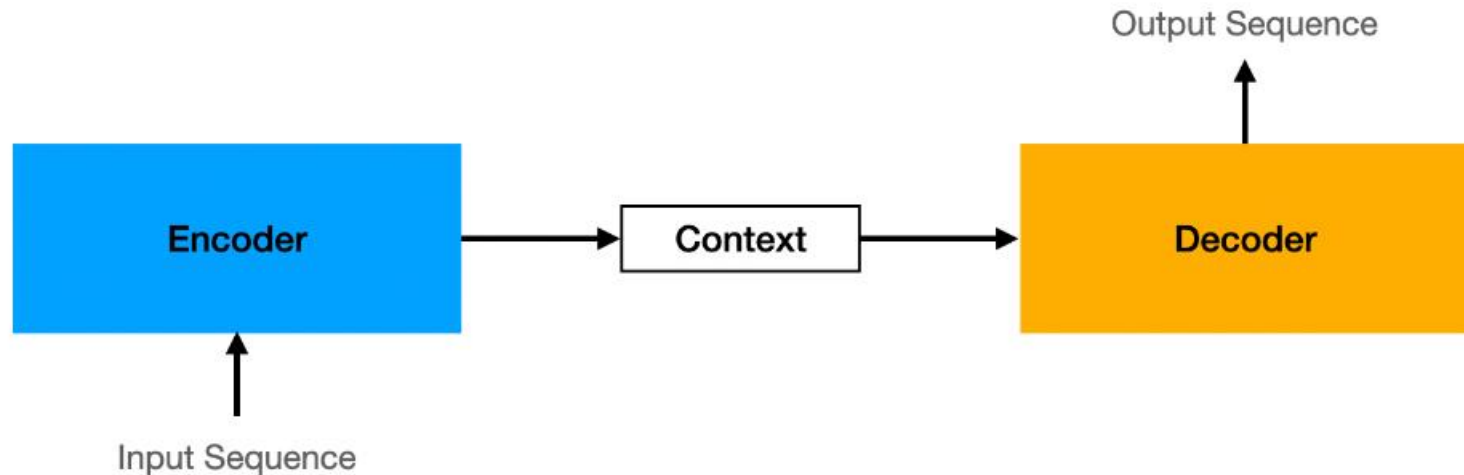


## Seq2Seq | Encoder-Decoder



Encoder는 원래의 문장을 압축하여 하나의 벡터로 만들고,  
Decoder는 만들어진 벡터를 바탕으로 새로운 문장을 만들어 냄

## Seq2Seq | Encoder-Decoder



이때 만들어지는 벡터를 **context vector**라고 함  
즉, Encoder의 마지막 hidden state 값이 context vector가 되고,  
Decoder는 이 context vector를 받아서 첫 번째 hidden state로 사용

## Seq2Seq | Encoder-Decoder의 학습

Source sentence  $X$ 는 "I ate an apple",  
실제 정답(Ground Truth)인 Target sentence  $D$ 는  
"Ich habe einen apfel gegessen"인 데이터 샘플이 주어짐

영어, 독일에 대해 각각 토큰 집합이 준비된 상태로 가정  
<eos>는 문장의 끝, <sos>는 문장의 시작을 알리는 토큰

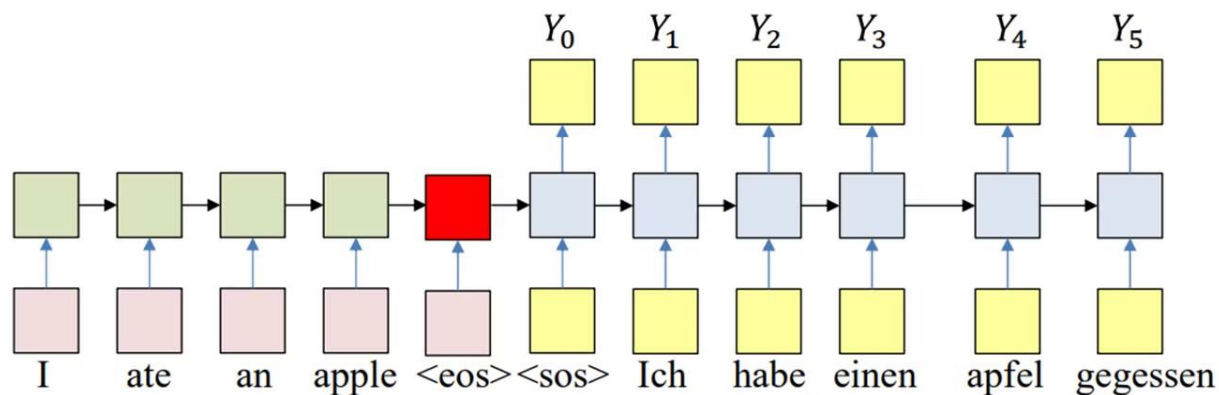
## Seq2Seq | Encoder-Decoder의 학습



학습 : 순전파 단계

원래 Ground Truth는 Loss를 계산할 때만 사용되고 모델 내부로 입력되지 않음

Seq2Seq 모델의 순전파는 **Input과 Target sentence가 같이 (순차적으로) 입력된다는 점**에서 기존의 순전파와 다름



Decoder가 RNN의 방식 그대로 output vector를 순차적으로 계산하는 모습

## Seq2Seq | Encoder-Decoder의 학습

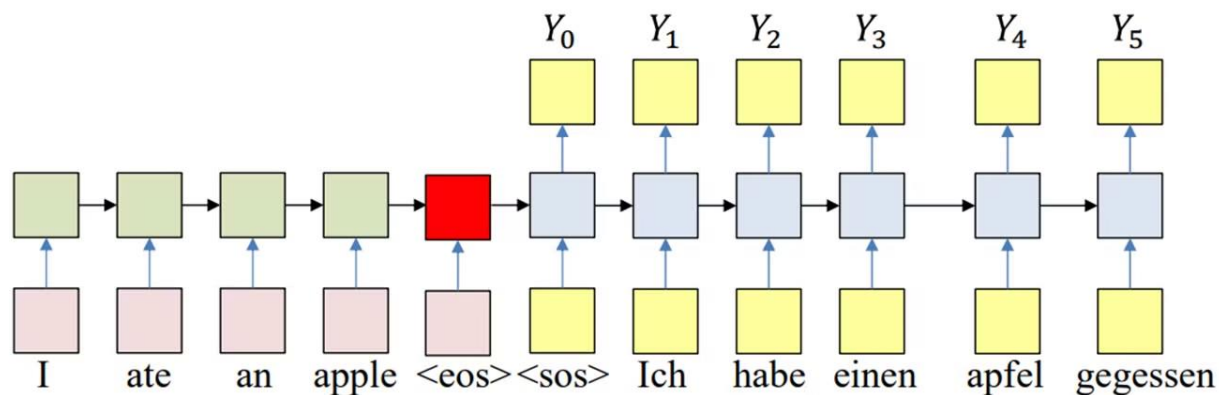
학습 : 순전파 단계

여기서 Output vector는

원래 **토큰 집합 속 각 토큰에 확률을 할당한 (조건부) 확률 벡터**를 의미

Seq2Seq 모델의 순예를 들어 벡터  $Y_2$ 는

$P(d_2 | d_0 = Ich, d_1 = habe, X = "I ate an apple")$ 의 예측값이 됨



Decoder가 RNN의 방식 그대로 output vector를 순차적으로 계산하는 모습

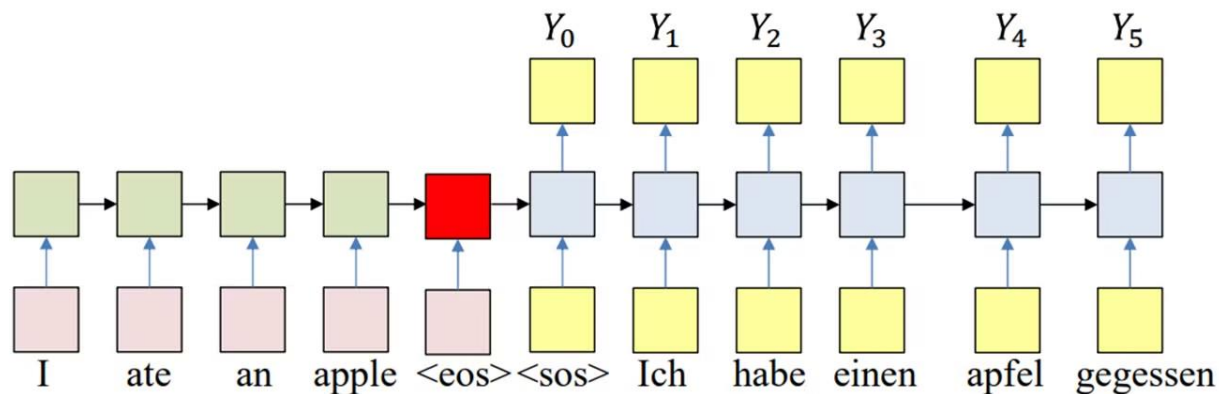
## Seq2Seq | Encoder-Decoder의 학습

학습 : 순전파 단계  
이때 Decoder의 입력으로는 Target sentence

원래 Ground Truth는 Loss를 계산할 때는 사용되지만 모델 내부로 입력되지 않음  
"Ich habe einen apfel gegessen"가 순차적으로 들어감

이처럼 실제 정답(Ground Truth)을 모델에 직접 활용하는 것을

같이 (순차적) 교사 강습(Teacher Forcing)이라고 함



Decoder가 RNN의 방식 그대로 output vector를 순차적으로 계산하는 모습

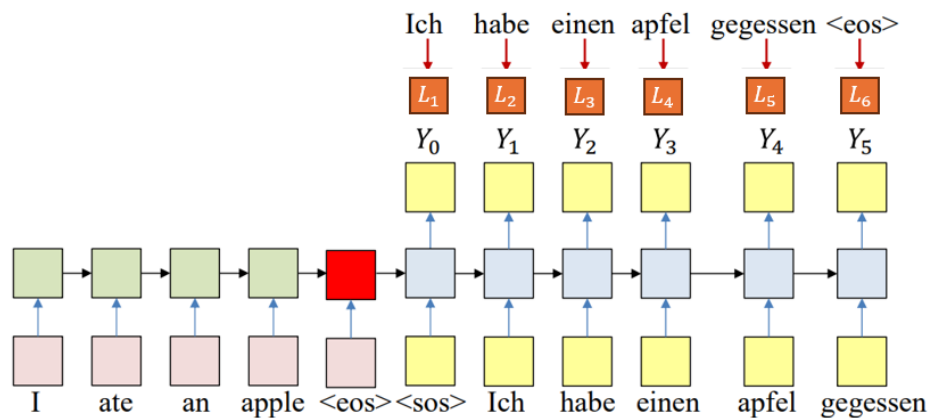
## Seq2Seq | Encoder-Decoder의 학습



학습 : 역전파 단계

각 Time step에서 발생하는 토큰 예측은 classification에 속함  
따라서 이에 맞는 손실함수인 Cross entropy Loss를 사용

Decoder의 Output vector 각각에 대해 손실함수 값  $L_i$ 이 계산되고,  
BPTT를 통해 역전파가 진행됨





## Seq2Seq | Encoder-Decoder의 학습



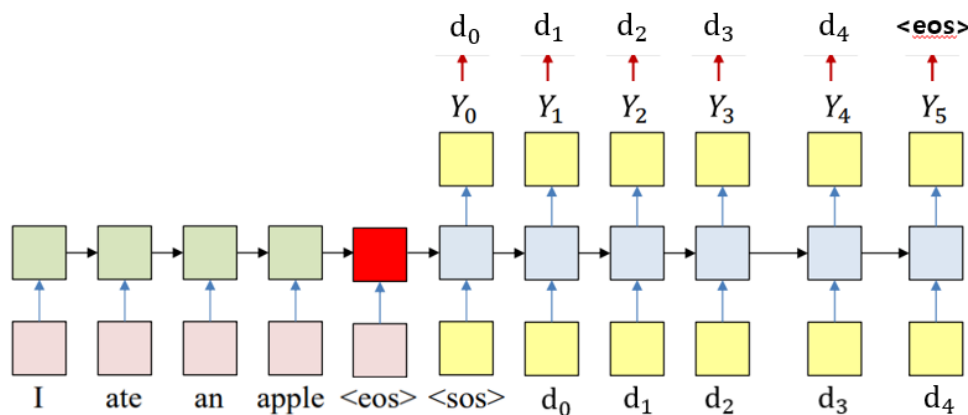
추론 : Test data에 대한 예측 단계

확률 벡터를, 이산형 확률 변수의 확률질량함수가

주어진 것처럼 생각할 수 있음

각 시점 확률 분포를 이용해 토큰  $d_i$ 을 추출(draw)할 수 있고  
그 토큰을 다음 시점의 RNN 입력값으로 넣어주는 방식으로 진행

이전 시점에 추출된 토큰이 달라지면  
그 이후에 나올 토큰들도 달라질 가능성이 있음



## Seq2Seq | Encoder-Decoder의 학습



추론 : Test data에 대한 예측 단계

확률 벡터를, 이산형 확률 변수의 확률질량함수가

주어진 것처럼 생각할 수 있음

각 시점 확률 분포를 이용해 토큰  $d_i$ 을 추출(draw)할 수 있고  
그 토큰을 다음 시점의 RNN 입력값으로 넣어주는 방식으로 진행

이전 시점에 추출된 토큰이 달라지면  
그 이후에 나올 토큰들도 달라질 가능성이 있음

새로운 Output vector의 계산은 <eos>가 나올 때까지 진행됨  
 $P(d_{t+1} | d_t, \dots, X = \text{Input sentence})$ 을 계산 후 <eos>가 추출됐다면  
계산은 거기서 마무리되고 하나의 Output sentence가 완성됨

## Seq2Seq | Encoder-Decoder의 학습



Decoding step : 토큰 추출

매 시점 모델은  $P(d_{t+1}|d_t, \dots, X)$ 에 대한 예측값을 출력하고,  
이 값에 기반해 토큰  $d_{t+1}$ 을 추출

추출의 궁극적인 목적은 **추출된 토큰 sequence  $d_0, d_1, \dots, d_{t+1}$ 의  
결합 확률이 최대가 되도록** 하는 것

즉  $P(d_0, \dots, d_{t+1}|X) = \prod_{i=0}^{t+1} P(d_i | d_{i-1}, \dots, X)$ 가 최대가 되도록  
 $d_0, d_1, \dots, d_{t+1}$ 을 잘 고르는 것이 목적

## Seq2Seq | Encoder-Decoder의 학습

언어 모델의 확률 벡터를 사용해 토큰을 추출하는 방법에는  
여러 가지가 있음  
모든 경우의 수를 고려해서 다 계산해보고 비교할 수 없기 때문에,  
일부 경우의 수만 골라 계산하는 방법을 택함



## Seq2Seq | Encoder-Decoder의 학습

### Greedy decoding

매 시점  $P(d_{t+1}|d_t, \dots, X)$ 가  
최대가 되도록 하는 토큰을 추출

### Sampling

매 시점 확률질량 함수  $P(d_{t+1}|d_t, \dots, X)$ 로부터  
샘플링을 해 토큰을 추출

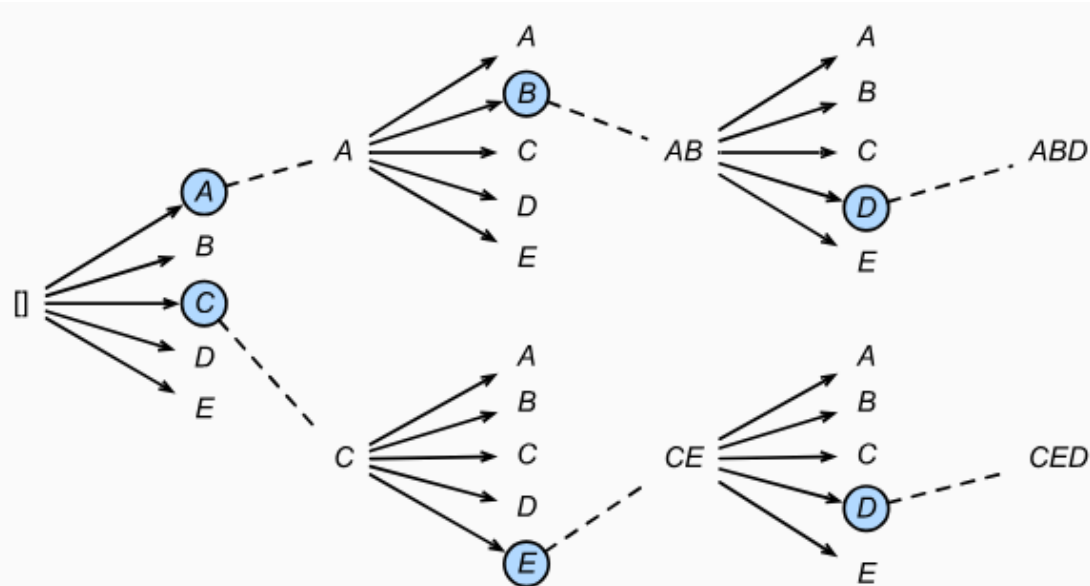
### Beam search

매 시점 결합확률 상위  $k$ 개의 토큰 조합만 남겨가며,  
가장 결합확률 값이 큰 최종 토큰 조합을 너비 우선 탐색의 방식으로 찾아감

## Seq2Seq | Encoder-Decoder의 학습

Greedy

Sampling



Beam search

매 시점 결합확률 상위 k개의 토큰 조합만 남겨가며,  
가장 결합확률 값이 큰 최종 토큰 조합을 너비 우선 탐색의 방식으로 찾아감

## Seq2Seq | Encoder-Decoder의 한계점

고정된 길이의 context vector

Seq2seq 모델은 하나의 고정된 길이의 context vector를 만들어 냄

입력 문장의 모든 정보를 하나의 벡터에 저장하기 때문에  
정보의 손실이 일어날 수밖에 없음

기울기 소실 문제

Encoder-Decoder 구조 또한 이전의 Hidden state와  
현재의 입력을 바탕으로 현재의 출력을 예측하는 구조

결국 시간의 흐름에 따라 연속적인 구조를 가지고 있기 때문에  
RNN과 마찬가지로 기울기 소실 문제를 극복하기 어려움

## BLEU score

BLEU(Bilingual Evaluation Understudy) score

기계 번역 결과와 사람이 직접 번역한 결과가  
**얼마나 유사한지 비교**하여 번역에 대한 성능을 측정하는 방법

“실제 전문가의 번역과 가까울수록 좋은 번역이다”라는  
함의를 담고 있음

‘헛갈리는 정도’를 나타냈던 PPL과는 달리  
높을수록 성능이 좋음을 의미



## BLEU score



### BLEU score의 장점

- ① 빠르게 점수를 구할 수 있고 계산 비용이 저렴함
- ② 비교적 결과를 이해하기 쉬움
- ③ 언어에 구애되지 않고 사용할 수 있음
- ④ 사람의 번역과 크게 연관시킬 수 있음
- ⑤ 폭 넓게 적용 가능함

## BLEU score



BLEU score 계산식

$$BLEU = BP \times \prod_{n=1}^N p_n^{w_n}$$

만들어낸 문장의 길이가 짧을 경우 페널티를 주는 BP(Brevity Penalty)와  
보정된 n-gram 정밀도의 기하평균으로 구성되어 있음

$p_n$ 은 보정된 n-gram 정밀도를 의미,  $n=1,2,3,4$ 를 사용

$w_n$ 은 각 n-gram에 대한 가중치, 보통은 다 같은 가중치  $\frac{1}{N}$ 을 부여

## BLEU score



BLEU score 계산식

$$BLEU = BP \times \prod_{n=1}^N p_n^{w_n}$$

Unigram 정밀도, 보정된 정밀도, 보정된 n-gram 정밀도 순으로

만들어낸 문장의 길이가 짧을 강 **알아보자!** 주는 BP(Brevity Penalty)와  
보정된 n-gram 정밀도의 기하평균으로 구성되어 있음

$p_n$ 은 보정된 n-gram 정밀도를 의미,  $n=1,2,3,4$ 를 사용

$w_n$ 은 각 n-gram에 대한 가중치, 보통은 다 같은 가중치  $\frac{1}{N}$ 을 부여

## BLEU score | Unigram 정밀도

### <Machine Translation>

**Candidate 1** : It is a guide to action which ensures that the military always obeys the commands of the party.

**Candidate 2** : It is to insure the troops forever hearing the activity guidebook that party direct.

### <Human Translation>

**Reference 1** : It is a guide to action that ensures that the military will forever heed Party commands.

**Reference 2** : It is the guiding principle which guarantees the military forces always being under the command of the Party.

**Reference 3** : It is the practical guide for the army always to heed the directions of the party.

n=1인 경우

Unigram에서는 Candidate Translation 문장의 각 단어 하나하나가

Reference Translation 문장 안에 들어 있기만 하면,

그리고 그 수가 많으면 정밀도가 높다고 봄

## BLEU score | Unigram 정밀도

Candidate 1 - ( $P = \frac{17}{18}$ )

C1	It	is	a	guide	to	action	which	ensures	that	the	military	always	obeys	the	commands	of	the	party
R1	0	0	0	0	0	0	X	0	0	0	0	X	X	0	0	X	0	0
R2	0	0	X	X	X	X	0	X	X	0	0	0	X	0	0	0	0	0
R3	0	0	X	0	0	X	X	X	X	0	X	0	X	0	X	0	0	0

Candidate 2 - ( $P = \frac{8}{14}$ )

C2	It	is	to	insure	the	troops	forever	hearing	the	activity	guidebook	that	party	direct
R1	0	0	0	X	0	X	0	X	0	X	X	0	0	X
R2	0	0	X	X	0	X	X	X	0	X	X	X	0	X
R3	0	0	0	X	0	X	X	X	0	X	X	X	0	X

이 예시에서 Candidate 1은 18개의 예측 토큰을 내놓았는데,  
그 중 17개가 참조 문장 안에 존재하기 때문에 정밀도가 17/18로 계산됨

## BLEU score | Unigram 정밀도의 문제점

중복된 단어를 처리하지 못함

<Machine Translation>

Candidate : the the the the the the the.

<Human Translation>

Reference 1 : The cat is on the mat.

Reference 2 : There is a cat on the mat.

이 예시에서 기계 번역의 경우 "the"라는 단어로만 이루어짐

이는 분명 좋은 번역이 아님

그러나 Unigram의 방법으로 계산하면  $\frac{7}{7} = 1$ 의 정밀도가 나옴

## BLEU score | Unigram 정밀도의 문제점

문장의 순서를 고려하지 않음

Unigram 방식은 단어 각각에 대해 독립적으로 카운트하는 방법

자연어는 기본적으로 순차 데이터이기 때문에  
순서를 고려해줄 방법 필요



보정된 n-gram 정밀도를 사용

## BLEU score | 보정된 정밀도

<Machine Translation>

Candidate : the the the the the the the.

<Human Translation>

Reference 1 : The cat is on the mat.

Reference 2 : There is a cat on the mat.

참조 문장에서는 the가 많이 나와봤자 2번이지만  
이를 고려하지 못함



## BLEU score | 보정된 정밀도

<Machine Translation>

Candidate : the the the the the the the.

<Human Translation>

Reference 1 : The cat is on the mat.

Reference 2 : There is a cat on the mat.

그래서 정밀도를 계산할 때 중복된 단어를 카운트할 때 **참조 문장에서의 최대 등장 횟수를 넘지 않도록 제한을 걸어두는 것이** 보정된 정밀도!

## BLEU score | 보정된 n-gram 정밀도

Candidate 1 - ( $P = \frac{10}{17}$ )

C1	It is	is a	a guide	guide to	to action	action which	which ensures	ensures that	that the
R1	0	0	0	0	0	X	X	0	0
R2	0	X	X	X	X	X	X	X	X
R3	0	X	X	X	X	X	X	X	X
C1	the military	military always	always obeys	obeys the	the commands	commands of	of the	the party	
R1	0	X	X	X	X	X	X	X	
R2	0	X	X	X	X	X	0	0	
R3	X	X	X	X	X	X	0	0	

같은 예시에 대해 보정된 Bigram 정밀도를 계산

## BLEU score | 보정된 n-gram 정밀도

Candidate 2 – ( $P = \frac{1}{13}$ )

C2	it is	is to	to insure	insure the	the troops	troops forever	forever hearing
R1	0	X	X	X	X	X	X
R2	0	X	X	X	X	X	X
R3	0	X	X	X	X	X	X
C2	hearing the	the activity	activity guidebook	guidebook that	that party	party direct	
R1	X	X	X	X	X	X	
R2	X	X	X	X	X	X	
R3	X	X	X	X	X	X	

번역 문장에서 연속적으로 등장한 두 단어 쌍이 참조 문장에서도 마찬가지로 등장해야  
정밀도에 기여할 수 있음

## BLEU score | Brevity Penalty (BP)

### Brevity Penalty (BP)

짧은 문장에 대해 페널티를 주는 것

<Machine Translation>

Candidate 1 : That cat sat on the soft mat.

Candidate2 : Black and white cat sat on mat in corner of room.

<Human Translation>

Reference : The black and white that cat sat peacefully on the  
soft mat in the corner of the room.

## BLEU score | Brevity Penalty (BP)

보정된 2-gram 정밀도로 계산한 결과

Candidate1 - ( $P = \frac{5}{6}$ )

C1	that cat	cat sat	sat on	on the	the soft	soft mat
R	0	0	X	0	0	0

Candidate 2 - ( $P = \frac{5}{10}$ )

C2	black and	and white	white cat	cat sat	sat on	on mat	mat in	in corner	corner of	of room
R	0	0	X	0	X	X	0	X	0	X

Candidate 1은 많은 부분이 생략되어 있어서 좋은 문장이 아님  
그러나 2-gram 정밀도로 계산하면 Candidate 2보다 훨씬 좋다고 판단함

## BLEU score | Brevity Penalty (BP)

보정된 2-gram 정밀도로 계산한 결과

Candidate1 - ( $P = \frac{5}{6}$ )

C1	that cat	cat sat	sat on	on the	the soft	soft mat
R	0	0	X	0	0	0

Candidate 2 - ( $P = \frac{5}{10}$ )

C2	black and	and white	white cat	cat sat	sat on	on mat	mat in	in corner	corner of	of room
R	0	0	X	0	X	X	0	X	0	X

이를 완화하기 위해 BP factor를 부여하여 짧은 문장에 대한 페널티를 부여

$$BP = \exp \left( \min \left( 0, 1 - \frac{\text{len}_{\text{refer}}}{\text{len}_{\text{pred}}} \right) \right)$$

즉, 번역 문장의 길이가 Reference 문장 이상이면 1, 그렇지 않으면 1보다 작은 값을 부여

## BLEU score | Brevity Penalty (BP)

보정된 2-gram 정밀도로 계산한 결과

Candidate1 - ( $P = \frac{5}{6}$ )

C1	that cat	cat sat	sat on	on the	the soft	soft mat
R	0	0	X	0	0	0

이 요소들을 다 종합하여

Candidate 2 - ( $P = \frac{5}{10}$ )

C2	$BLEU = BP \times \prod_{n=1}^N p_n^{w_n}$ 라는 식을 만들 수 있음!									of room
R	0	0	X	0	X	X	0	X	0	X

이를 완화하기 위해 BP factor를 부여하여 짧은 문장에 대한 페널티를 부여

$$BP = \exp \left( \min \left( 0, 1 - \frac{\text{len}_{\text{refer}}}{\text{len}_{\text{pred}}} \right) \right)$$

즉, 번역 문장의 길이가 Reference 문장 이상이면 1, 그렇지 않으면 1보다 작은 값을 부여

## 3주차 클린업 예고



### Attention & Transformer

앞서 다룬 Seq2Seq 모델은 고정된 context vector를  
가지며 기울기 소실 문제가 있다는 단점이 있었음

## 3주차 클린업 예고

### Attention & Transformer

앞서 다룬 Seq2Seq 모델은 고정된 context vector를  
가지며 기울기 소실 문제가 있다는 단점이 있었음



3주차에는 이를 극복하게 해준 Attention 매커니즘과  
이를 활용한 Transformer에 대해 살펴볼 예정!





THANK YOU

