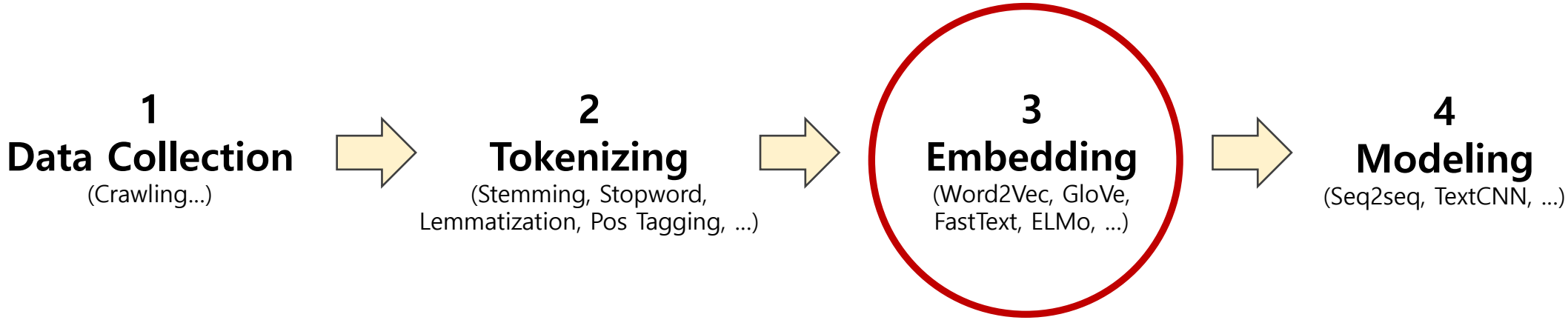
통계학과 202STG18 이재빈

Word Embeddings in 2020

NLP Process


1
Data Collection
(Crawling…)
2
Tokenizing
(Stemming, Stopword,
Lemmatization, Pos Tagging, …)
3
Embedding
(Word2Vec, GloVe,
FastText, ELMo, …)
4
Modeling
(Seq2seq, TextCNN, …)

00
01
02
03
04
05
06
07

Why Word Embedding?

Q. 컴퓨터는 각 의미 단위를 어떻게 이해할 수 있을까?

A. 컴퓨터가 처리할 수 있는 것은 **수치** 뿐
컴퓨터가 언어의 특성을 이해할 수 있도록 각 token 마다 **수치를 부여**!

Embedding 자연어를 숫자의 나열인 **벡터**로 변환

Word Embeddings

1. CountVectorizer
2. TF-IDF
3. Word2Vec
4. GloVe
5. FastText
6. ELMo
7. Transformers

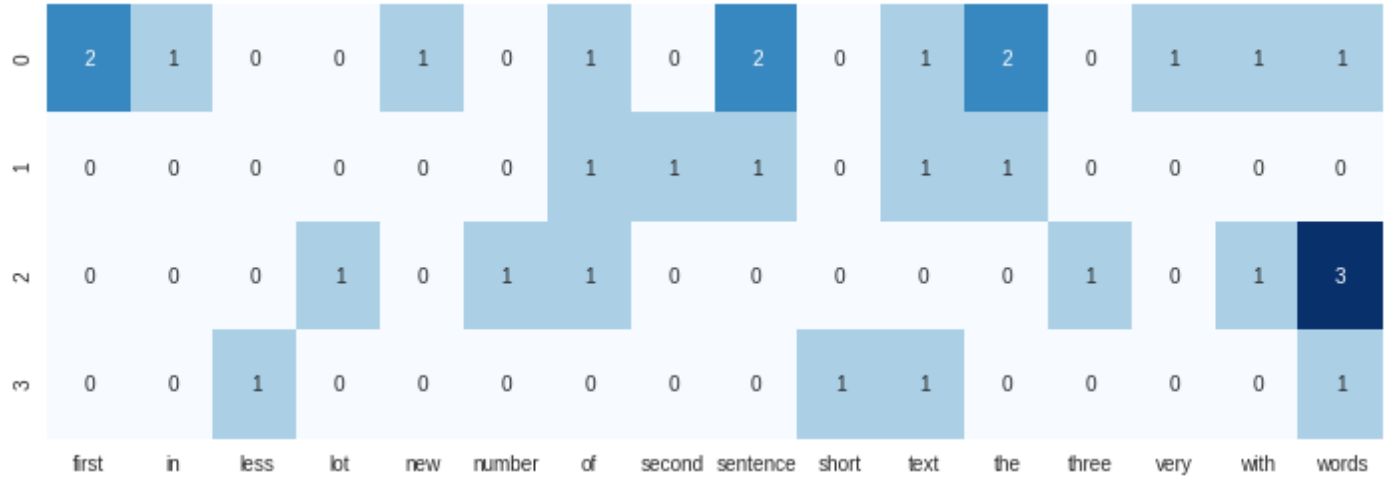
00
01
02
03
04
05
06
07

CountVectorizer

단어(**Token**)들의 카운트(출현 빈도(**frequency**))로 여러 문서(**Corpus**)들을 벡터화

```
Corpus = [ 0 'Text of the very first new sentence with the first words in sentence.',
           1 'Text of the second sentence.',
           2 'Number three with lot of words words words.',
           3 'Short text, less words.']

vocab = ['first', 'in', 'less', 'lot', 'new', 'number', 'of', 'second', 'sentence', 'short',
'text', 'the', 'three', 'very', 'with', 'words']
```



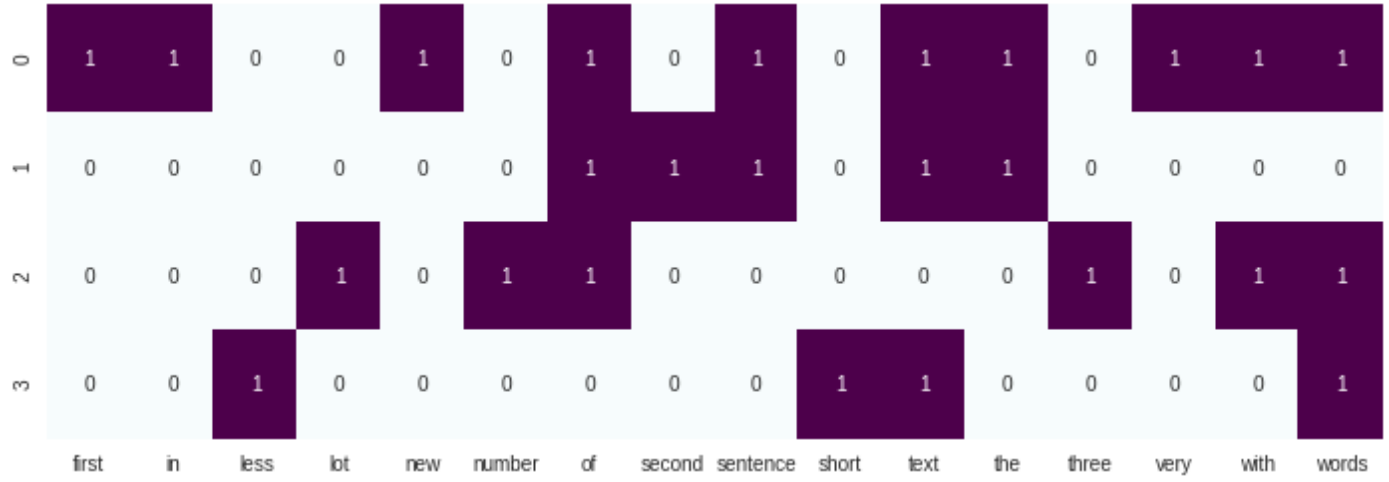
00
01
02
03
04
05
06
07

One-hot Vector

단어(**Token**)들의 등장 여부(**0 / 1**)로 여러 문서(**Corpus**)들을 벡터화

```
Corpus = [ 0 'Text of the very first new sentence with the first words in sentence.',
           1 'Text of the second sentence.',
           2 'Number three with lot of words words words.',
           3 'Short text, less words.']

vocab = ['first', 'in', 'less', 'lot', 'new', 'number', 'of', 'second', 'sentence', 'short',
'text', 'the', 'three', 'very', 'with', 'words']
```



TF-IDF

단어의 빈도(**Term Frequency**)와 역 문서 빈도(**Inverse Document Frequency**)를 사용하여 각 단어들마다 **중요한 정도**를 가중치로 주는 방법

$$W_{t,d} = tf_{t,d} \times \log_{10}\left(\frac{n}{1+df(t)}\right)$$

단어 빈도 (TF)

역문서 빈도 (IDF)

- 단어 빈도 (Term Frequency) = tf(t,d) : 특정 문서 d에서 특정 단어 t의 등장 횟수
- 문서 빈도 (Document Frequency) = df(t) : 특정 단어 t가 등장한 문서의 수
- 역문서 빈도 (Inverse Document Frequency) = idf(t) : df(t)에 반비례하는 수

- 모든 문서에 많이 나오는 단어 (ex. a, the, is, …) 굳이 가중치를 높게 줄 필요 없다!

00
01
02
03
04
05
06
07

TF-IDF

```
Corpus = [ 0 'Text of the very first new sentence with the first words in sentence.',
           1 'Text of the second sentence.',
           2 'Number three with lot of words words words.',
           3 'Short text, less words.']
```

```
tf                              # CountVectorizer
```

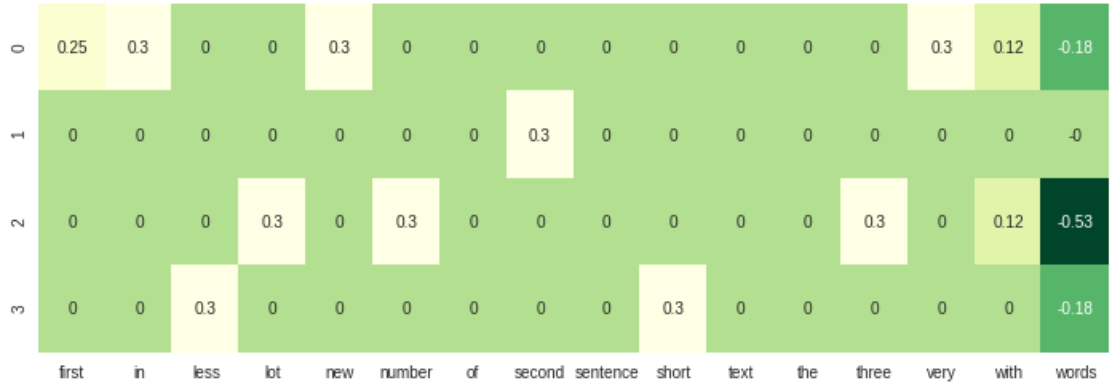
```
array([[2, 1, 0, 0, 1, 0, 1, 0, 2, 0, 1, 2, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 3],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1]])
```

```
np.sum(tf, axis=0)                          # df
```

```
array([2, 1, 1, 1, 1, 1, 3, 1, 3, 1, 3, 3, 1, 1, 2, 5])
```

```
np.log10(4/(1+np.sum(tf, axis=0)))          # idf
```

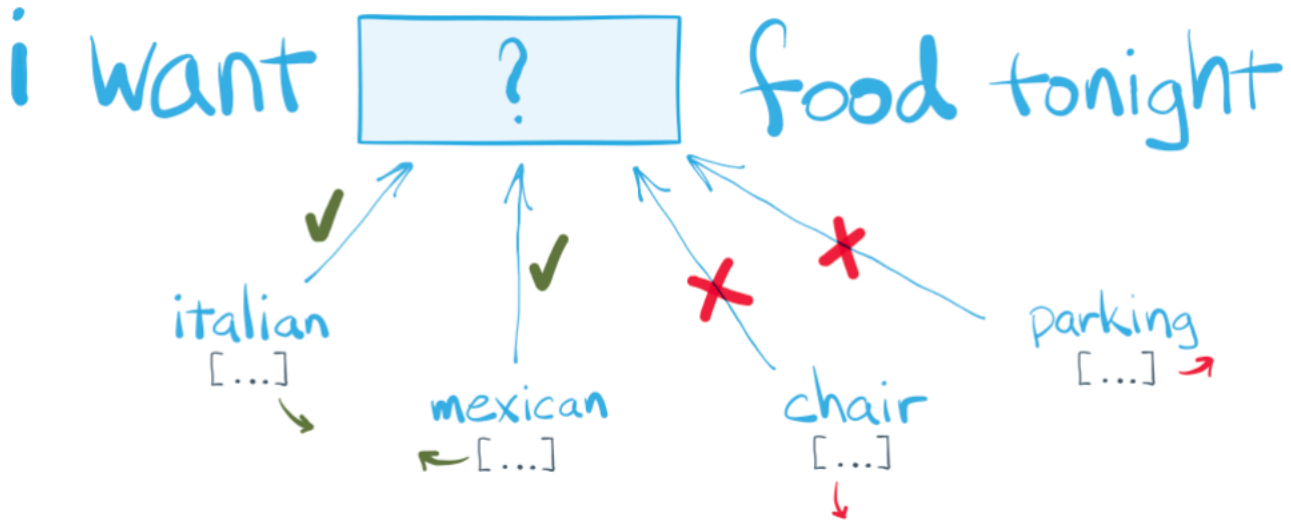
```
array([ 0.12493874,  0.30103   ,  0.30103   ,  0.30103   ,  0.30103   ,
        0.30103   ,  0.        ,  0.30103   ,  0.        ,  0.30103   ,
        0.        ,  0.        ,  0.30103   ,  0.30103   ,  0.12493874,
       -0.17609126])
```



np.multiply(tf, idf)

Word2Vec

비슷한 맥락을 갖는 단어들은 **비슷한 의미**(semantic similarity)를 가진다

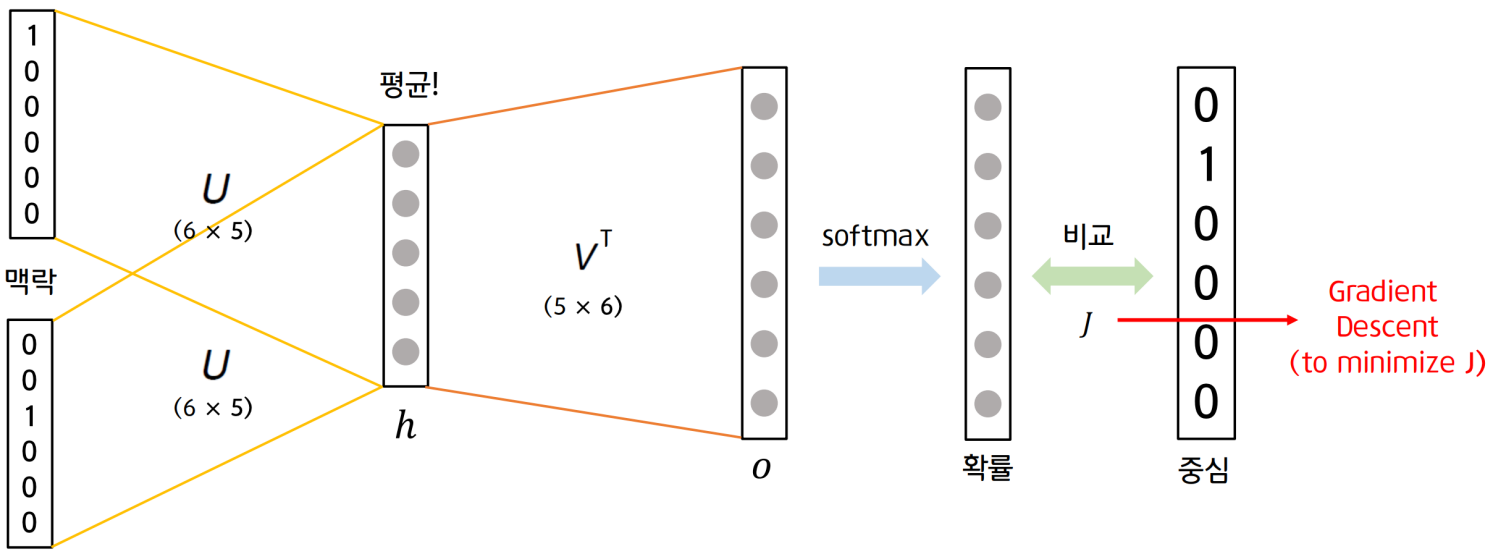


비슷한 맥락을 갖는 단어에 **비슷한 벡터**를 준다!

(중심단어와 주변단어 벡터의 내적이 코사인 유사도가 되도록 단어벡터를 벡터공간에 임베딩한다)

Word2Vec : CBOW

주변에 있는 단어들을 통해 **중심**에 있는 단어를 예측하는 방법

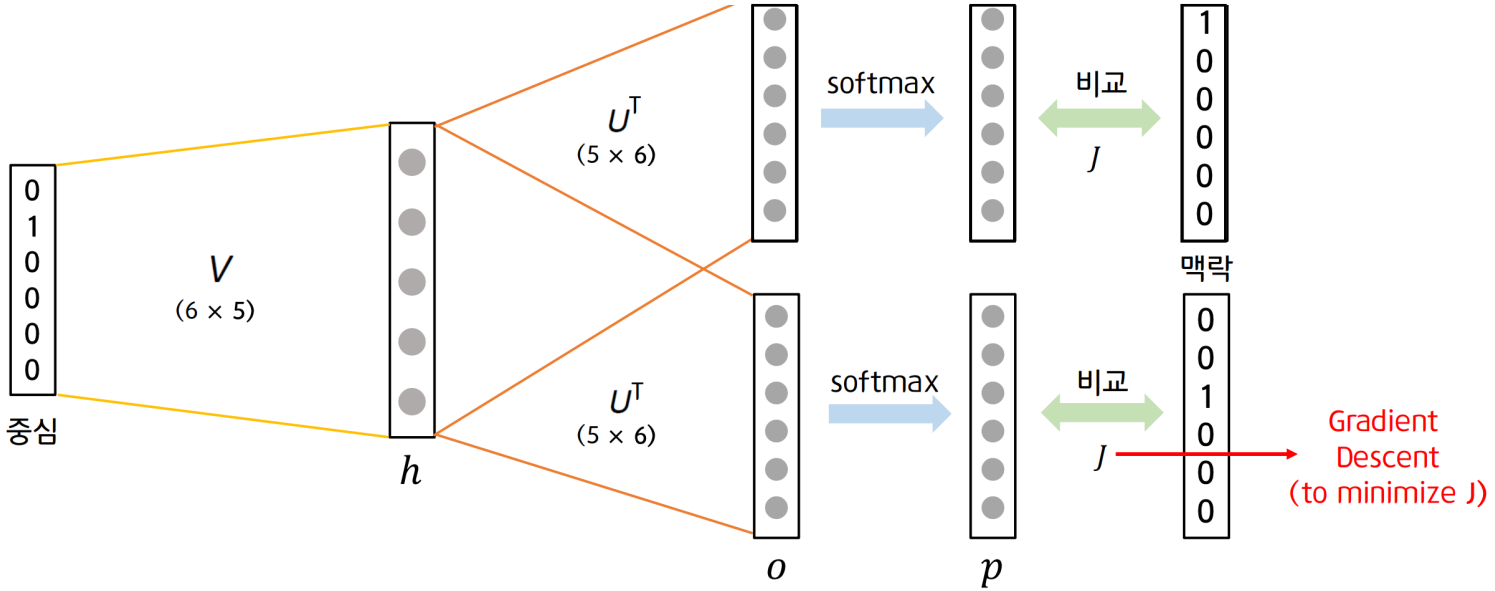


아침을 **안 먹었더니 __ 너무 고프다**

Word2Vec : Skip-gram

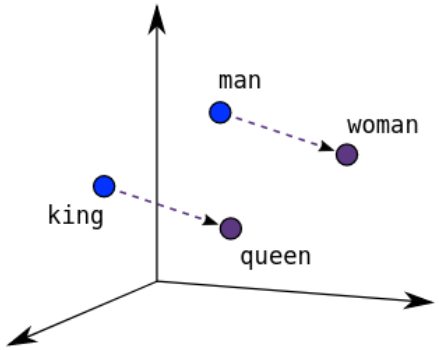
중심에 있는 단어를 통해 **주변**에 있는 단어들을 예측하는 방법

(CBOW 모델보다 update 기회가 더 많기 때문에 좀 더 많이 사용하는 추세)

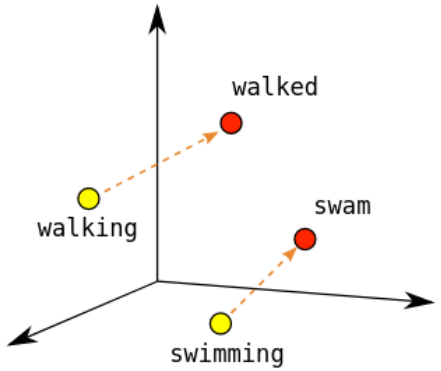


아침을 __ __ **배가** __ __

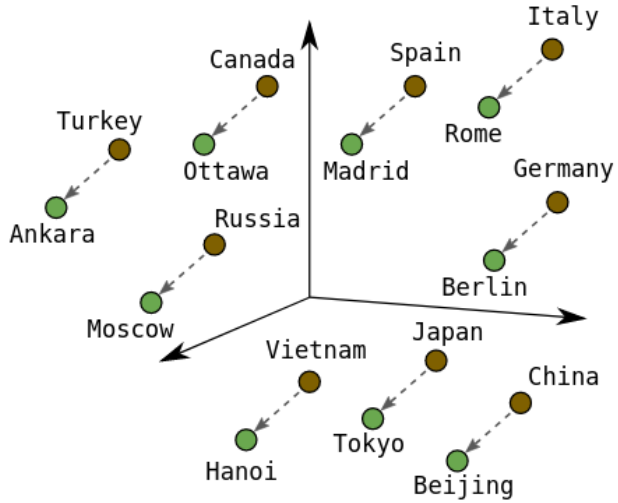
Word2Vec



Male-Female



Verb Tense



Country-Capital

- **Distance** between vectors for words with close meaning (ex. king – queen)
- Allow **mathematical operations** on vectors (ex. king – man + woman = queen)

GloVe

전체 문서 정보를 이용, 동시 등장 확률(**co-occurrence information**)을 고려해 임베딩하는 방법

LSA, TF-IDF

장점
- 빠른 훈련 속도
- 통계 정보 사용

단점
- 단어 간 관계는 파악 불가능
- 주로 단어 유사성 여부만을 파악하는 데에 사용

Count-based

+

Skip-gram, CBOW

장점
- 단어 유사성 이상의 복잡한 패턴을 파악
- 높은 수준의 성능

단점
- 임베딩 벡터가 윈도우 크기 내에서만 주변 단어를 고려하기 때문에, Corpus의 전체적인 통계 정보 반영 불가능

Direct Prediction

GloVe

통계 정보를 포함한
Direct Prediction Embedding

GloVe

임베딩 된 단어 벡터 간 **유사도** 측정을 수월하게 하면서
말뭉치 전체의 **통계 정보**를 반영하자

word2vec 장점
co-occurrence matrix 장점

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

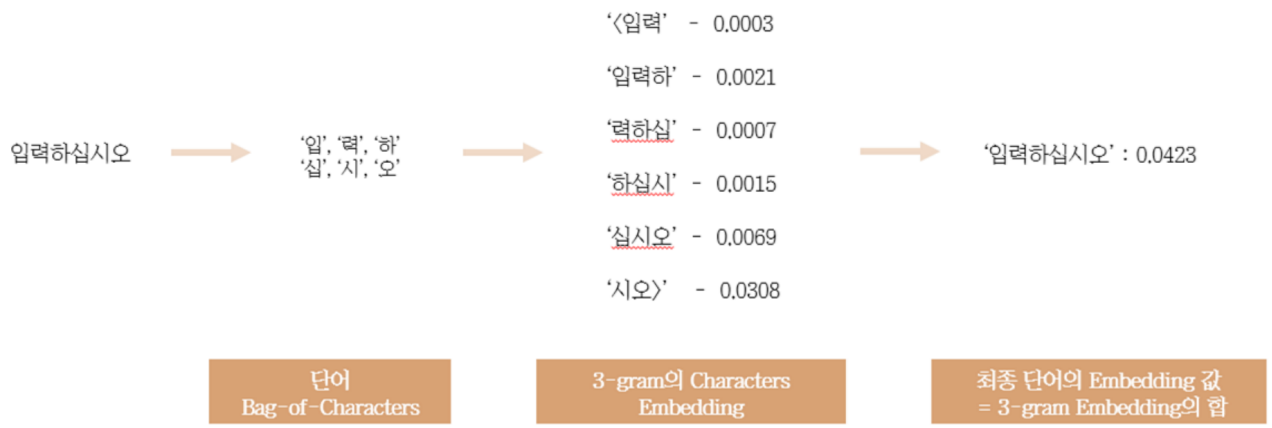
ice가 주어졌을 때, solid 등장할 확률 > steam 등장할 확률

$$\frac{P(solid|ice)}{P(solid|steam)} > 1 \Leftrightarrow \frac{P(gas|ice)}{P(gas|steam)} < 1$$

(임베딩 된 **두 단어벡터의 내적**이 말뭉치 전체에서의 **동시 등장확률 로그값**이 되도록 목적함수를 정의)

FastText

Subword Embedding을 통해 언어의 형태학적 (**Morpological**) 특성을 반영하는 방법

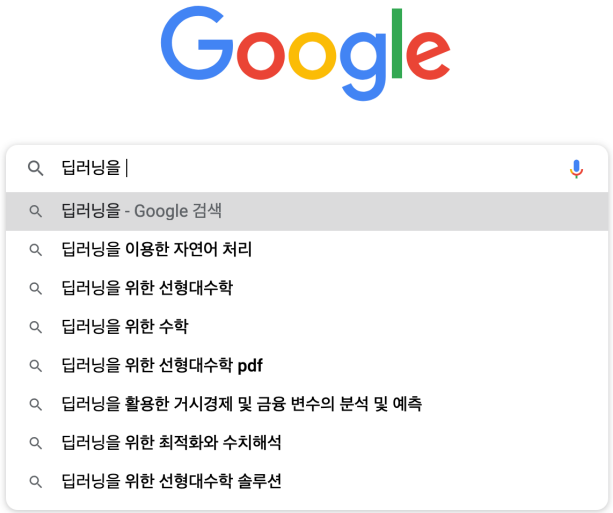


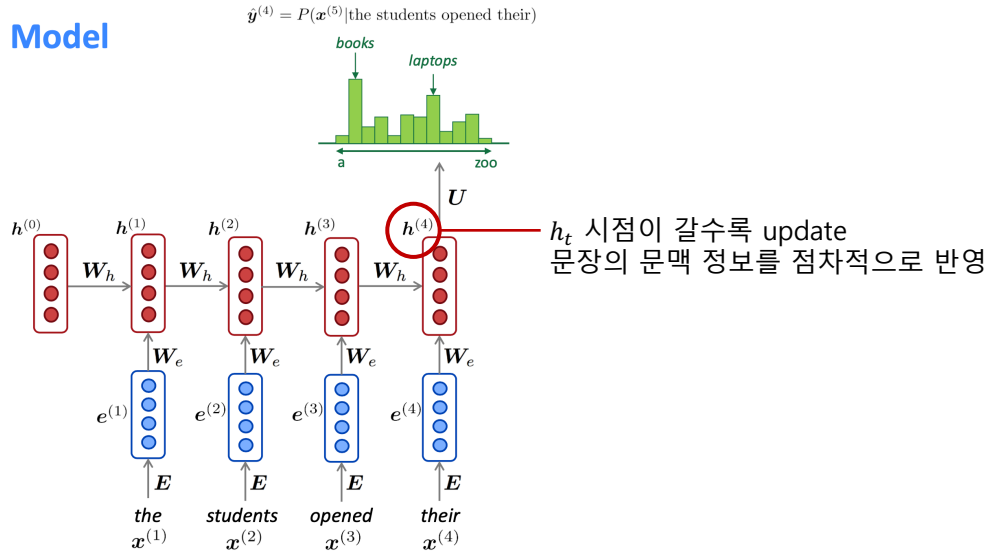
- **OOV (Out-of-Vocabulary)** : Word2Vec의 한계점, 새로운 단어가 들어오면 embedding 불가능

- 단어를 **Bag-of-Characters**로 보고, 개별 단어가 아닌 **n-gram의 Characters**를 embedding
- 희소한 단어에 대해 더 좋은 word embedding 가능
- 어휘사전(vocabulary words)으로부터 훈련된 말뭉치에 존재하지 않았던 단어 벡터를 만들어 낼 수 있음
  ex) disaster, disastrous

ELMo

Embeddings from Language Models, 문맥을 반영한 워드 임베딩(**Contextualized** Word Embedding)





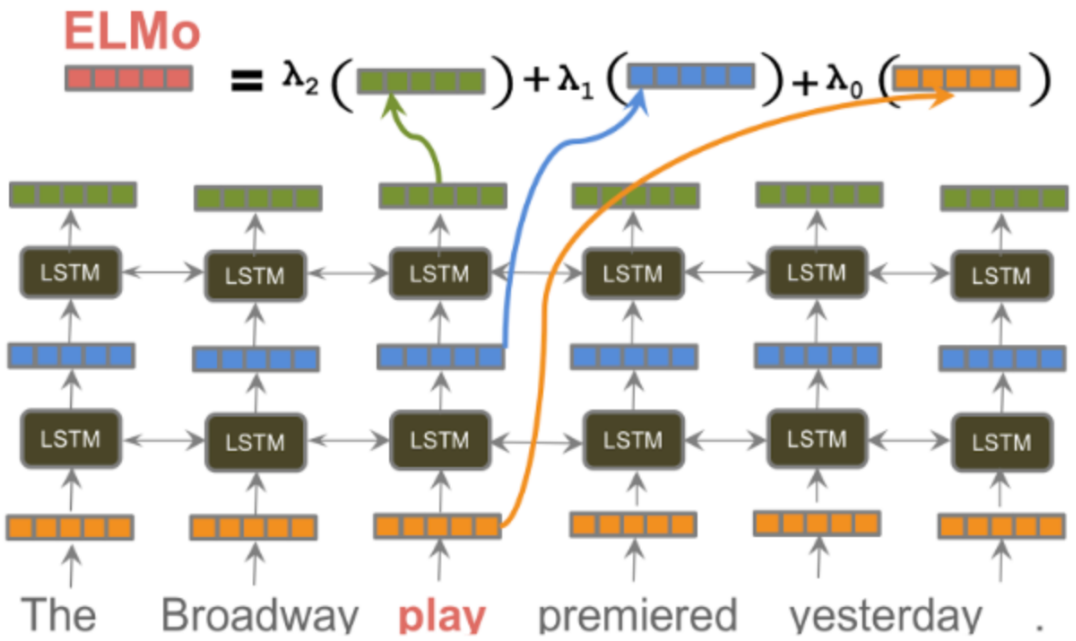
h_t 시점이 갈수록 update
문장의 문맥 정보를 점차적으로 반영

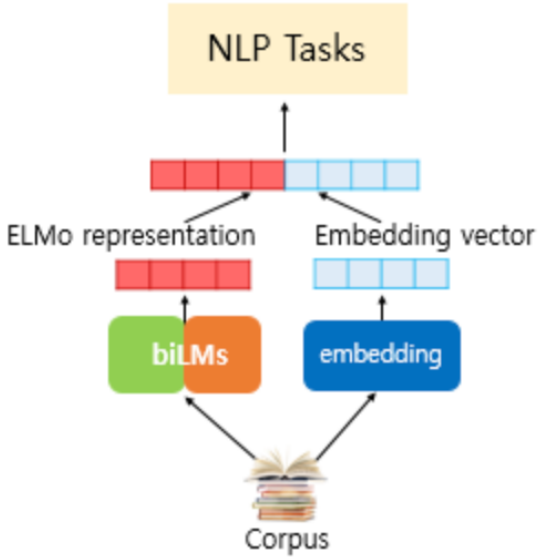
- **Language Modeling** : 현재까지 주어진 문장을 기반으로, 다음 단어를 예측하는 것

- $P(x^{(t+1)}|x^{(t)},\dots,x^{(1)})$, where $x^{(t+1)}$ can be word in the Vocabulary $V=\{w_1,\dots,w_{|v|}\}$
- $P(x^{(t)},\dots,x^{(1)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times \dots \times P(x^{(T)}|x^{(T-1)},\dots,x^{(1)}) = \prod_{t=1}^{T} P(x^{(t)}|x^{(t-1)},\dots,x^{(1)})$

ELMo

Bank Account (계좌) vs River Bank (강둑) : **문맥**에 따라 워드 임베딩을 다르게 수행



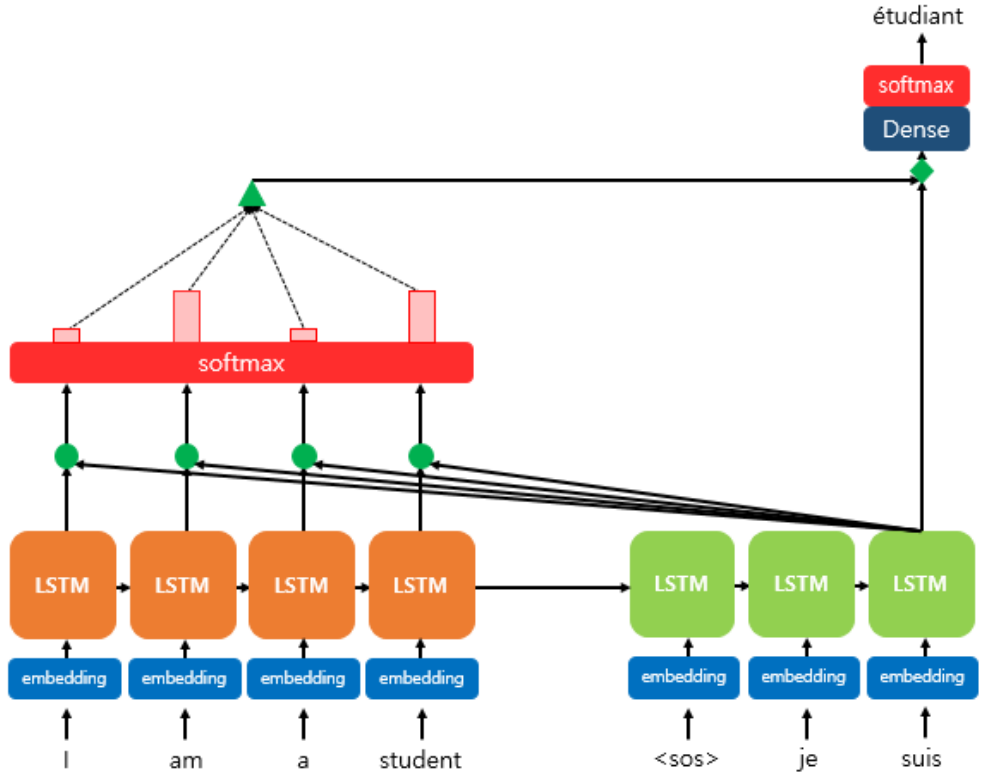


biLM (Bidirectional Language Model)
순방향 + 역방향 모두 고려

ELMo representation을 임베딩 벡터와 concatenate해서 입력으로 사용

Transformers

Attention Mechanism을 활용해, 단어 간 연관성(connections and dependencies)을 반영하는 방법



Attention Mechanism

RNN 기반 모델의 문제

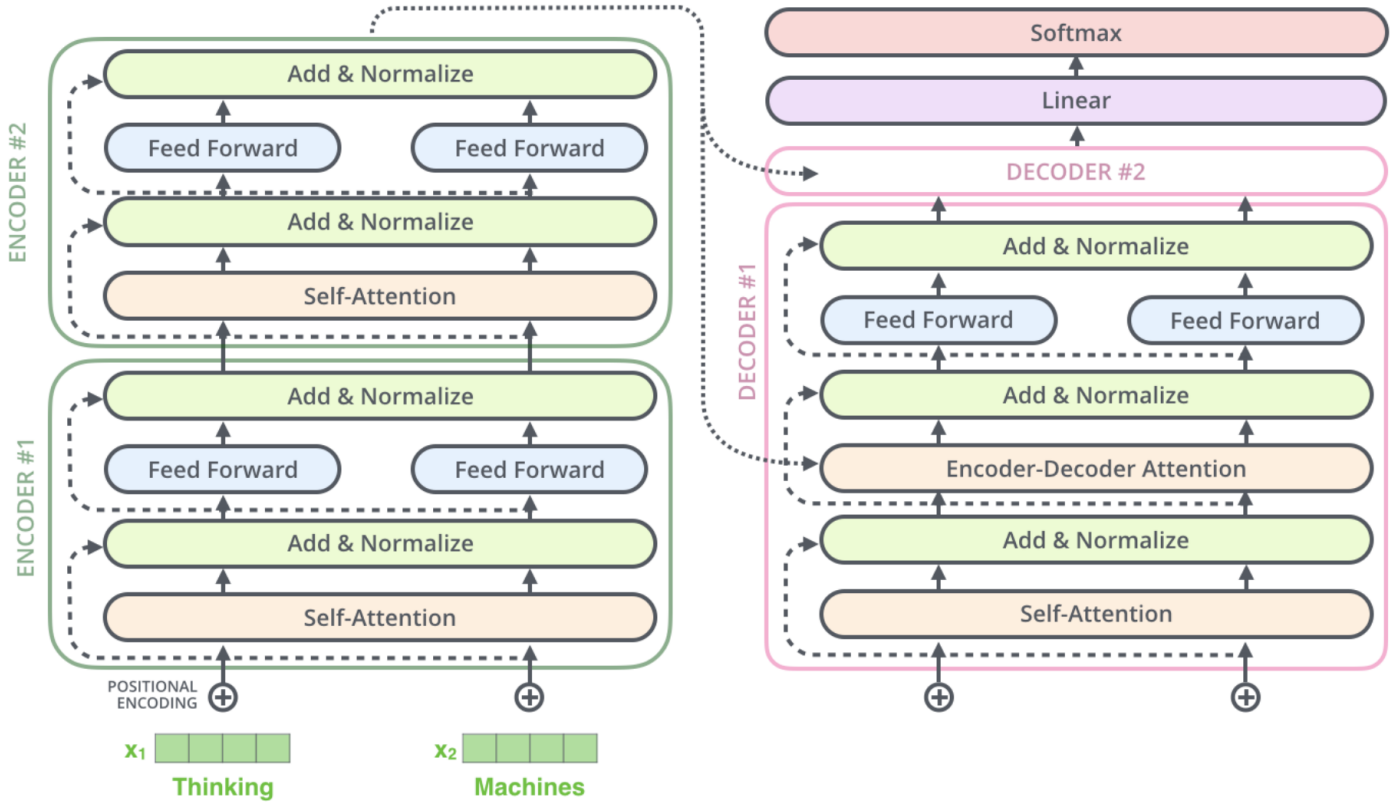
- Long Dependency & Vanishing Gradient

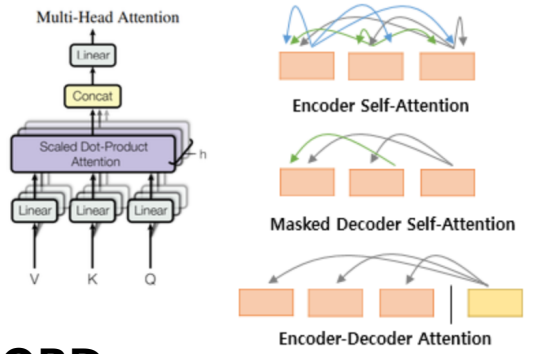
Attention?

- Decoder에서 출력 단어를 예측하는 매 시점마다, Encoder에서의 전체 입력 문장을 다시 한 번 참고
- 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야 할 단어와 **연관이 있는 입력 단어 부분**을 좀 더 집중(attention)해서 보게 됨

Transformers

Attention + Neural Network





KEYWORD

- Positional Encoding
  단어의 위치 정보 반영
- Multi-head Attention
  여러 head를 사용해, 다양한 특징 분석
- Self-Attention
  input 자신에 대한 attention
  (The animal is hungry, because **it** is tired)

00
01
02
03
04
05
06
07

Reference

- Word Embeddings in 2020. Review with code examples
  https://towardsdatascience.com/word-embeddings-in-2020-review-with-code-examples-11eb39a1ee6d

- Word2Vec Efficient Estimation of Word Representations in Vector Space (2013)
- GloVe Glove: Global vectors for word representation (2014)
- FastText Enriching word vectors with subword information (2016)
- ELMo Deep contextualized word representations (2018)
- Transformers Attention is all you need (2017)

감사합니다 😊