Word2Vec

단어의 분산 표현

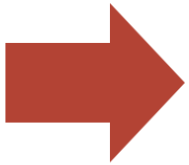
Feat. ELMo, BERT, ERNIE

컴퓨터에게 ‘단어’의 의미를 어떻게 알려줄까?

단어 표현의 고전 방식

- **One-hot encoding**

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch



	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Look-up table

Each word gets a 1x9 vector representation

문서 표현의 고전 방식

- **BOW(Bag-of-Words)**
  - 단어가 문서에 존재한다 or 안한다

$$d^1_{binary} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

  - 단어가 문서에 n번 존재한다
    - TF(Term Frequency)

$$d^1_{tf} = \begin{bmatrix} 3 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

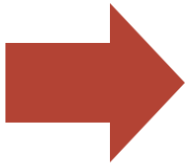
  - 단어에 중요도를 할당하고 문서에 등장한 단어 중요도의 가중합을 구한다
    - TF-IDF(Term Frequency Inverse Document Frequency)

고전 방식의 한계점

- Sparse vector여서 생기는 한계

$\overrightarrow{woman} \cdot \overrightarrow{man} = 0$

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch



	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

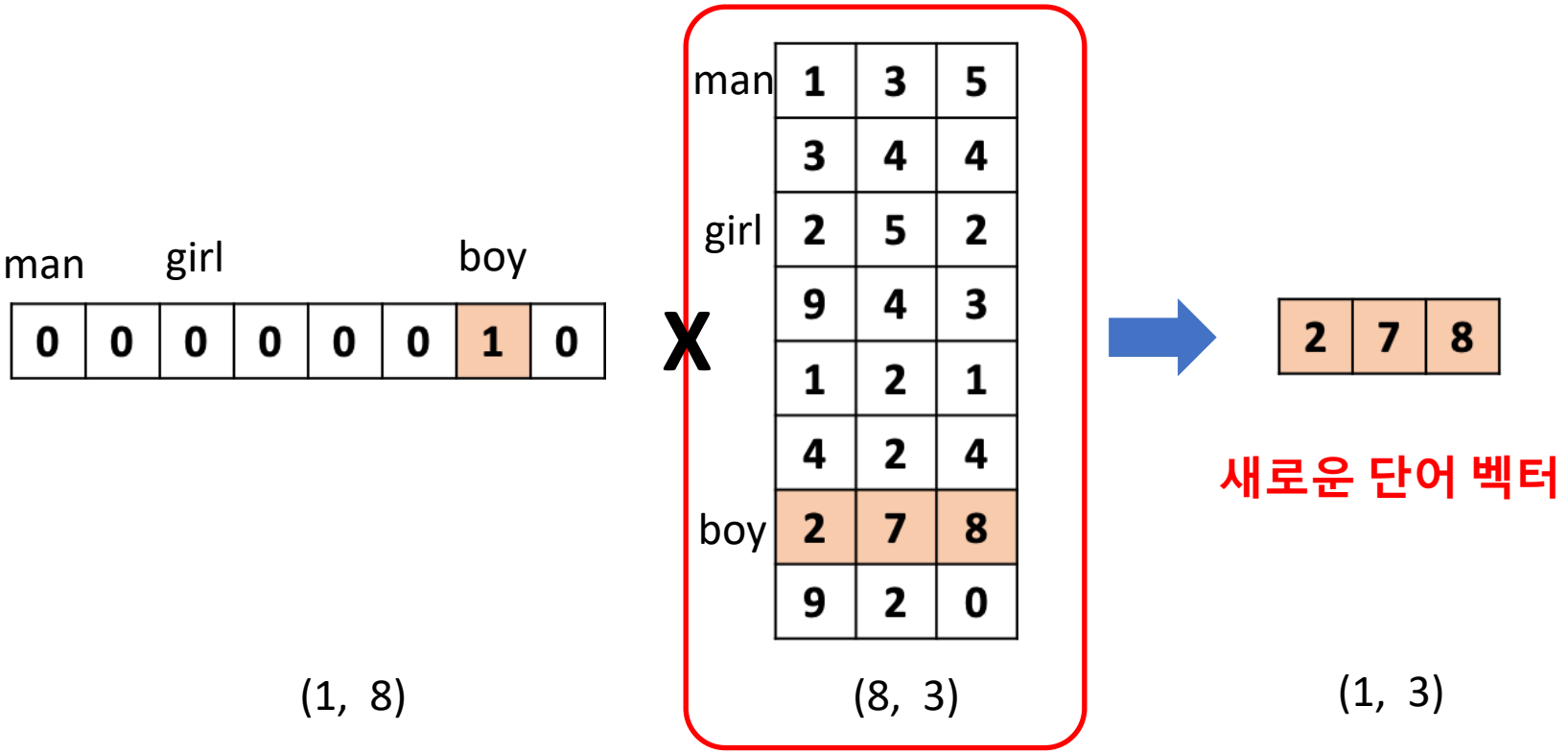
Each word gets a 1x9 vector representation

1. Vocab 수가 많아지면 벡터값이 너무 커져서 메모리면에서 비효율적 !

2. 두 단어의 벡터끼리 내적을 하게 되면 0이 되어서 유사도를 구할 수 없다!

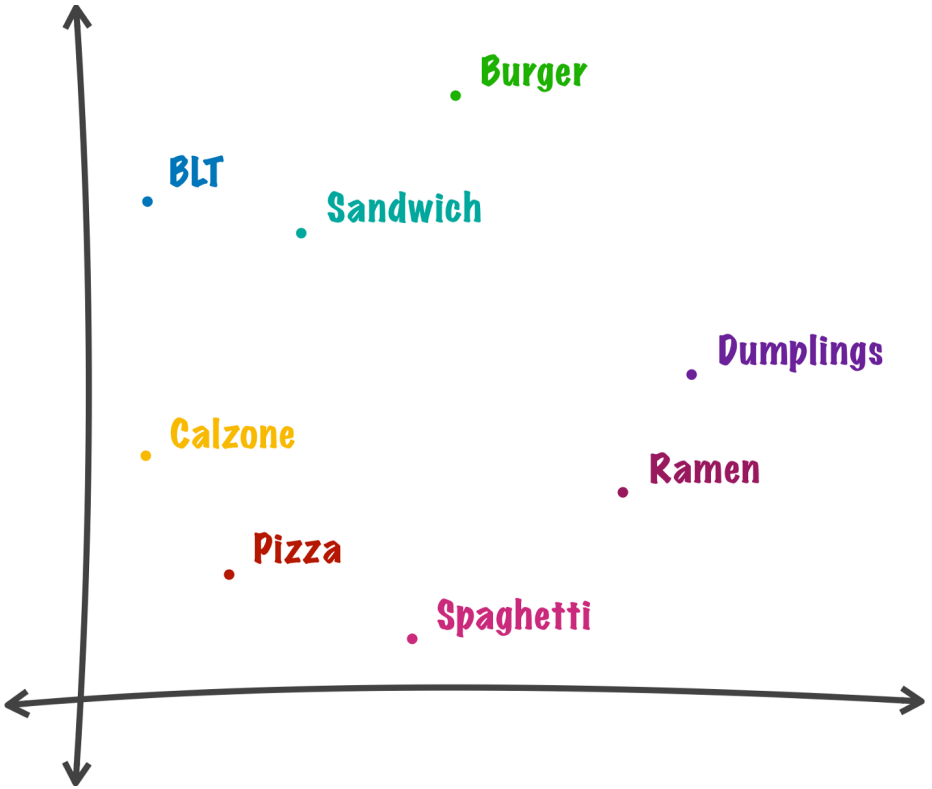
행렬곱을 통한 차원 축소

- 차원축소 목적으로 **행렬을 "학습" 시키자**



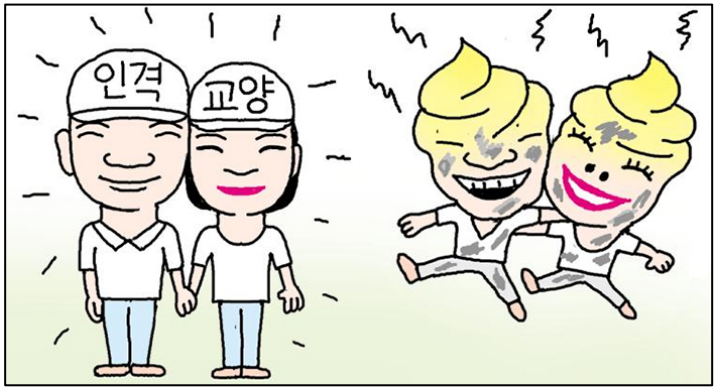
단어의 벡터 공간에서의 표현

- 비슷한 의미의 단어에게 비슷한 벡터값을 부여하자!
- 비슷한 의미의 단어들끼리 뭉치게!



단어의 벡터 공간에서의 표현

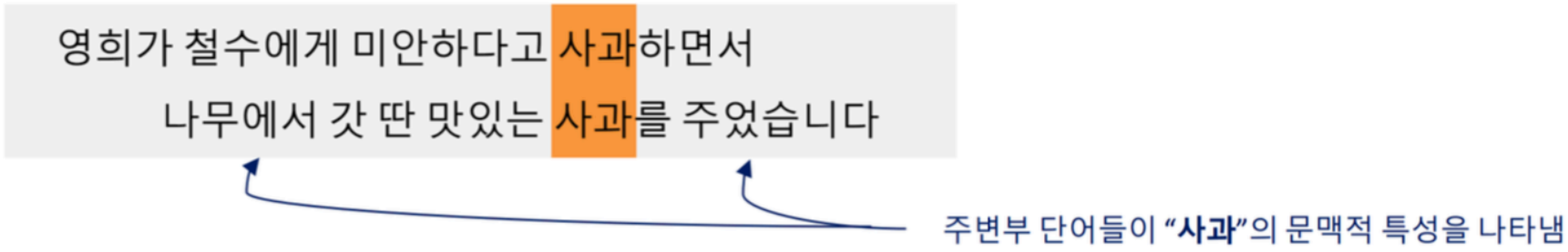
- 친구를 보면 그 사람을 알 수 있다



- 단어의 주변을 보면 그 단어를 안다
  - 새로운 디자인의 선이 __ 하다.
  - 야, 부엌 좀 __히 하자.
  - 깜빡하고 책상 위를 __하게 치우지 않았다.

단어의 벡터 공간에서의 표현

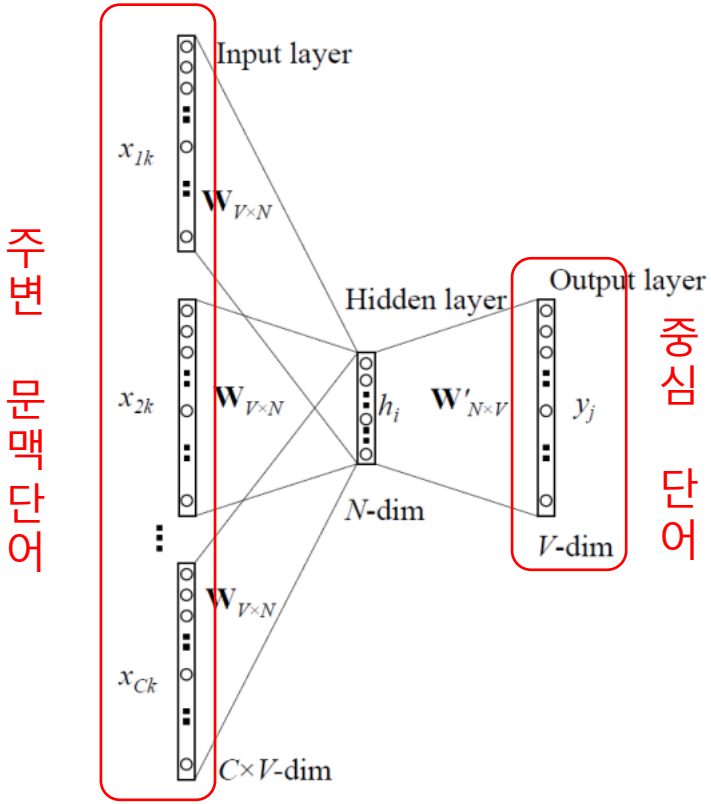
- 단어의 의미는 해당 단어의 주위 **문맥**이 담고 있다



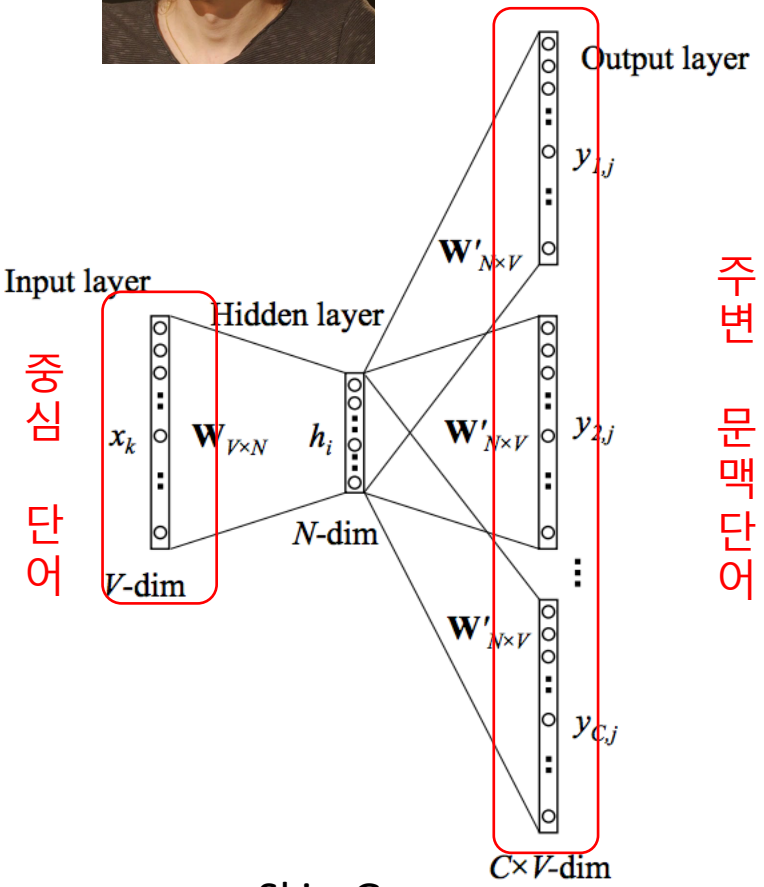
  - 두 단어의 문맥이 비슷하면 "의미적"으로 유사한 단어
  - 문맥은 정해진 구간(window)내의 단어 즉, 좌우 1~8단어들

Word2vec

- Mikolov et al. 2013 (논문 발표)



CBOW



Skip-Gram

Word2vec

- Skip-Gram
  - window size : 2

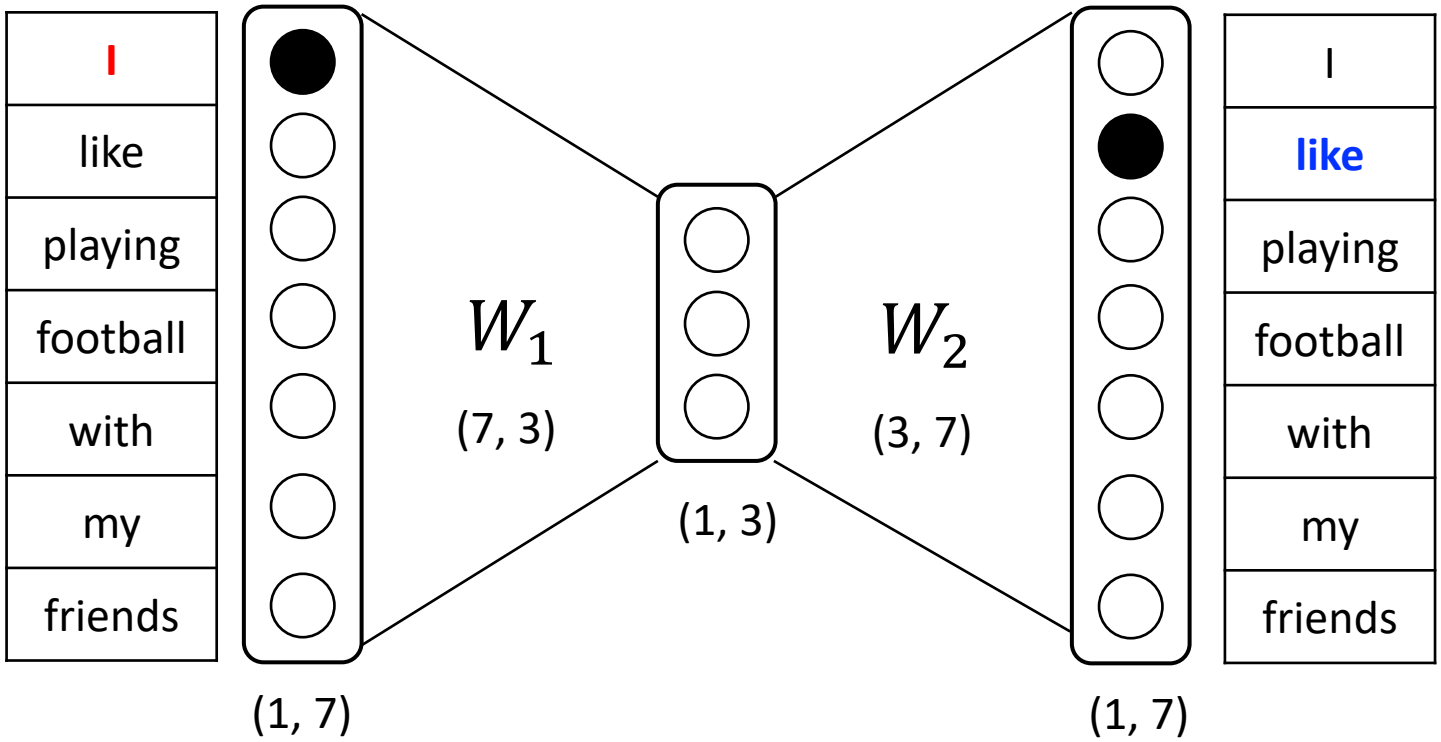
Input: center word	Output: context words
[1,0,0,0,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0]
[0,1,0,0,0,0,0]	[1,0,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,1,0,0,0]
[0,0,1,0,0,0,0]	[1,0,0,0,0,0,0] [0,1,0,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,1,0,0]
[0,0,0,1,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,1,0]
[0,0,0,0,1,0,0]	[0,0,1,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,0,1,0] [0,0,0,0,0,0,1]
[0,0,0,0,0,1,0]	[1,0,0,1,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,0,1]
[0,0,0,0,0,0,1]	[0,0,0,0,1,0,0] [0,0,0,0,0,1,0]

Word2vec

- Skip-Gram (iter-1)
  - window size : 2

INPUT : “I”

OUTPUT : “like”

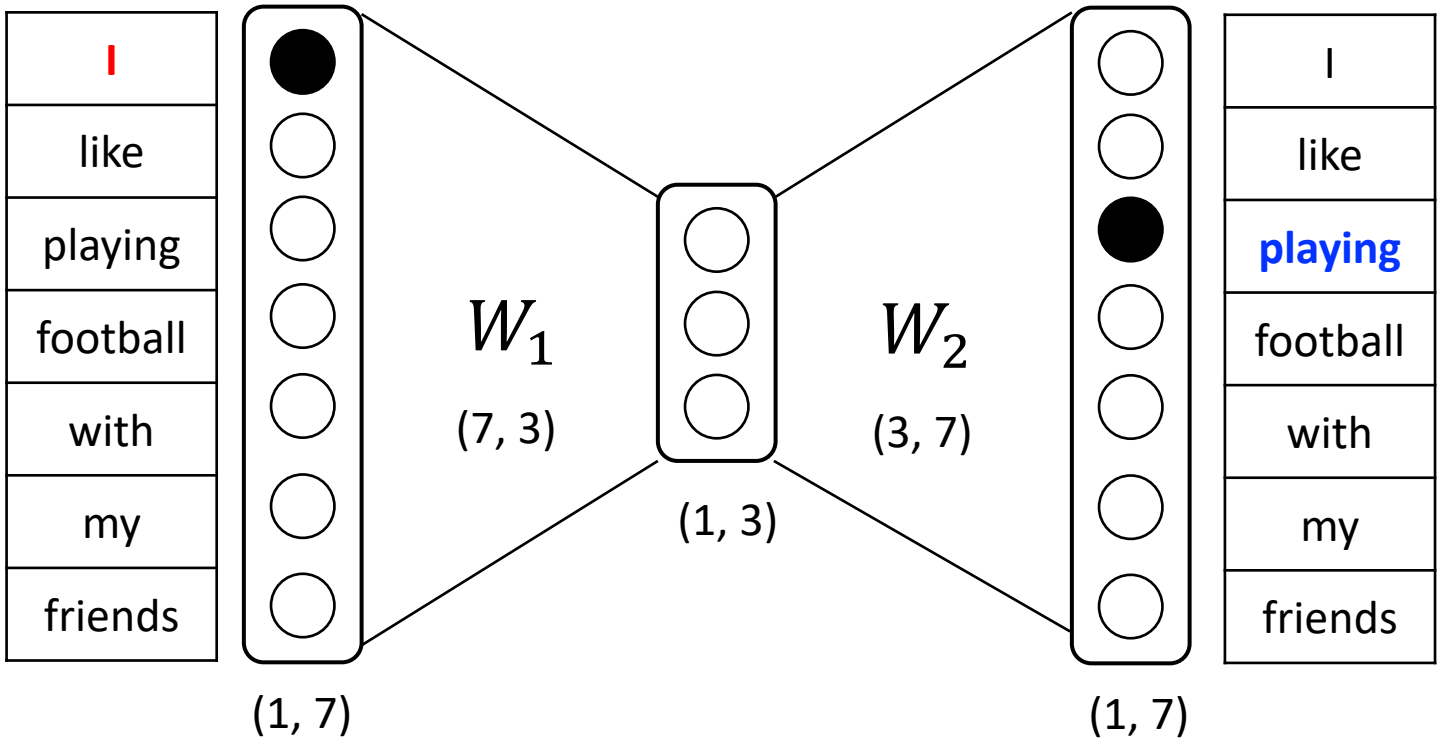


Word2vec

- Skip-Gram (iter-2)
  - window size : 2

INPUT : “I”

OUTPUT : “playing”

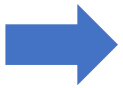


INPUT

man	girl					boy	
0	0	0	0	0	0	1	0

X

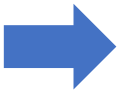
man	1	3	5
	3	4	4
girl	2	5	2
	9	4	3
	1	2	1
	4	2	4
boy	2	7	8
	9	2	0



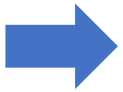
2	7	8

INPUT

Index = 7



man	1	3	5
	3	4	4
girl	2	5	2
	9	4	3
	1	2	1
	4	2	4
boy	2	7	8
	9	2	0



2	7	8

tf.nn.embedding_lookup(W, index)

이렇게 하면 왜..?

왜 비슷한 문맥의 단어끼리

비슷한 벡터를 갖는 걸까?

Word2vec

데이터

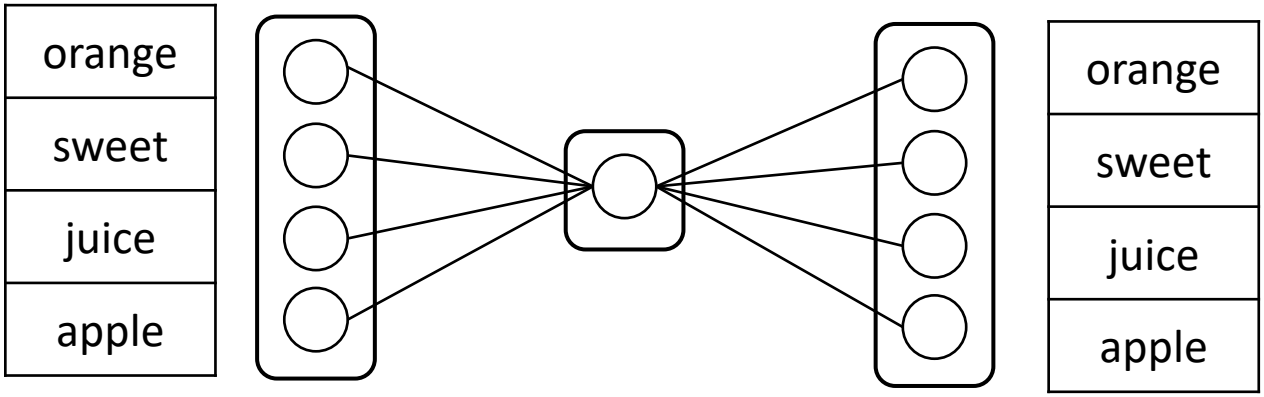
sweet, **orange**, juice

sweet, **orange**, juice

sweet, **apple**, juice

sweet, **orange**, juice

sweet, **apple**, juice

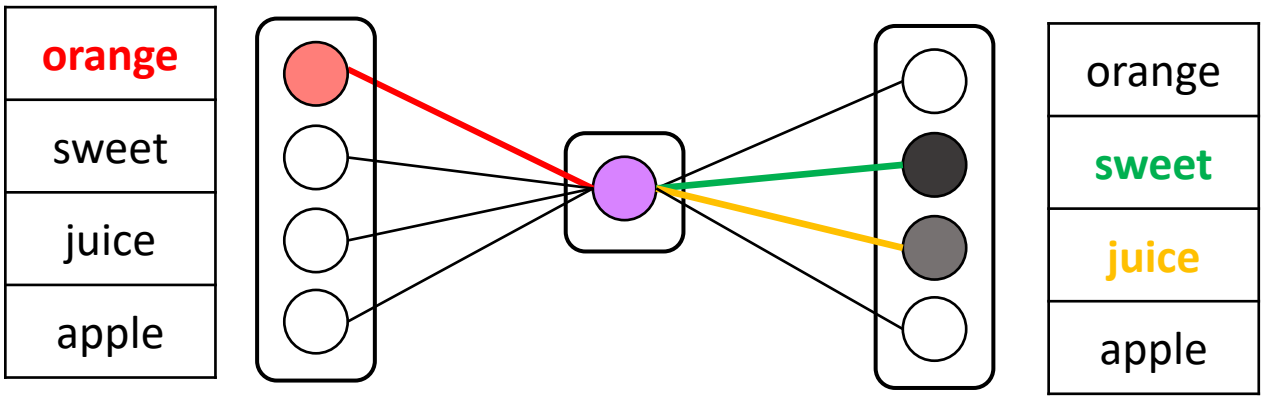


Word2vec

데이터

sweet, **orange**, juice
sweet, **orange**, juice
sweet, apple, juice
sweet, **orange**, juice
sweet, apple, juice

INPUT	OUTPUT
orange	sweet
orange	juice
orange	sweet
orange	juice
orange	sweet
orange	juice



Word2vec

데이터

sweet, orange, juice

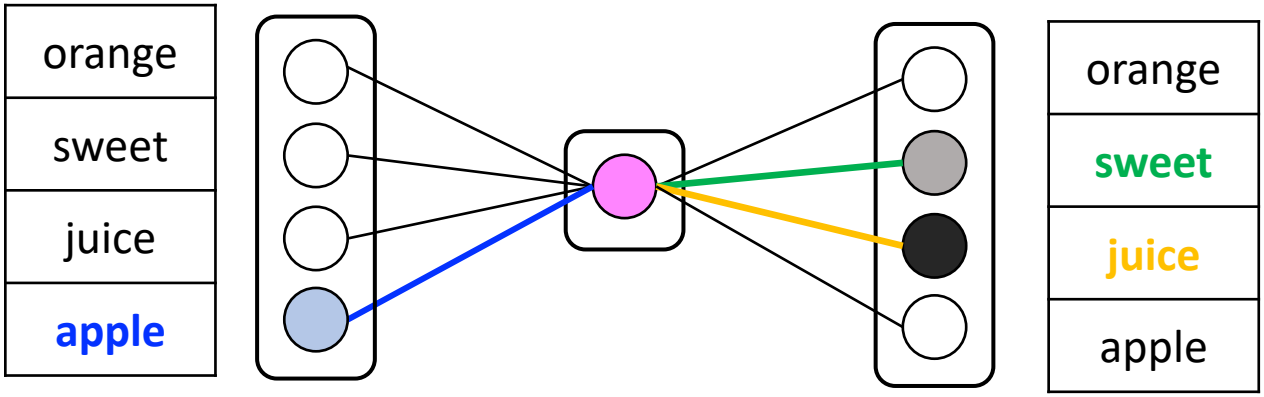
sweet, orange, juice

sweet, **apple**, juice

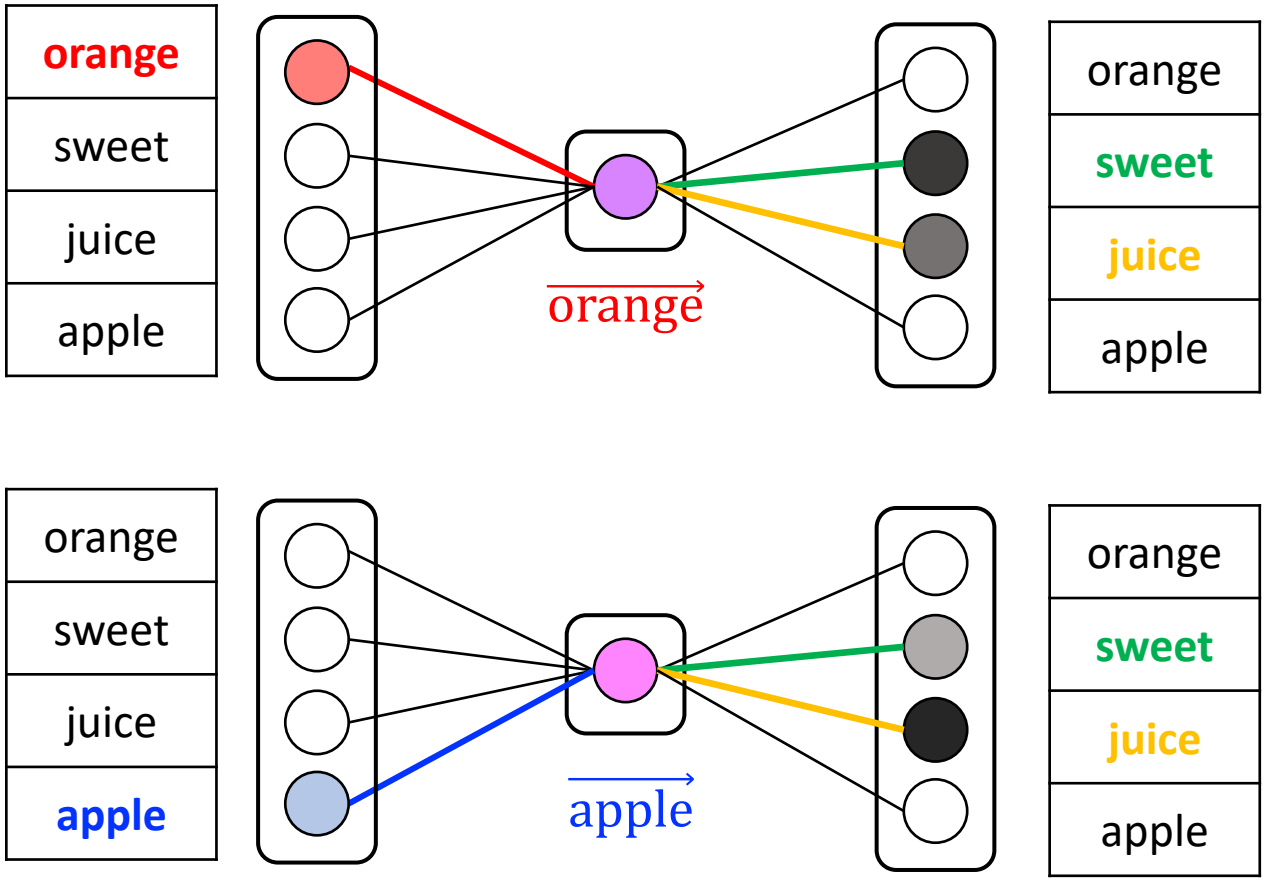
sweet, orange, juice

sweet, **apple**, juice

INPUT	OUTPUT
apple	sweet
apple	juice
apple	sweet
apple	juice



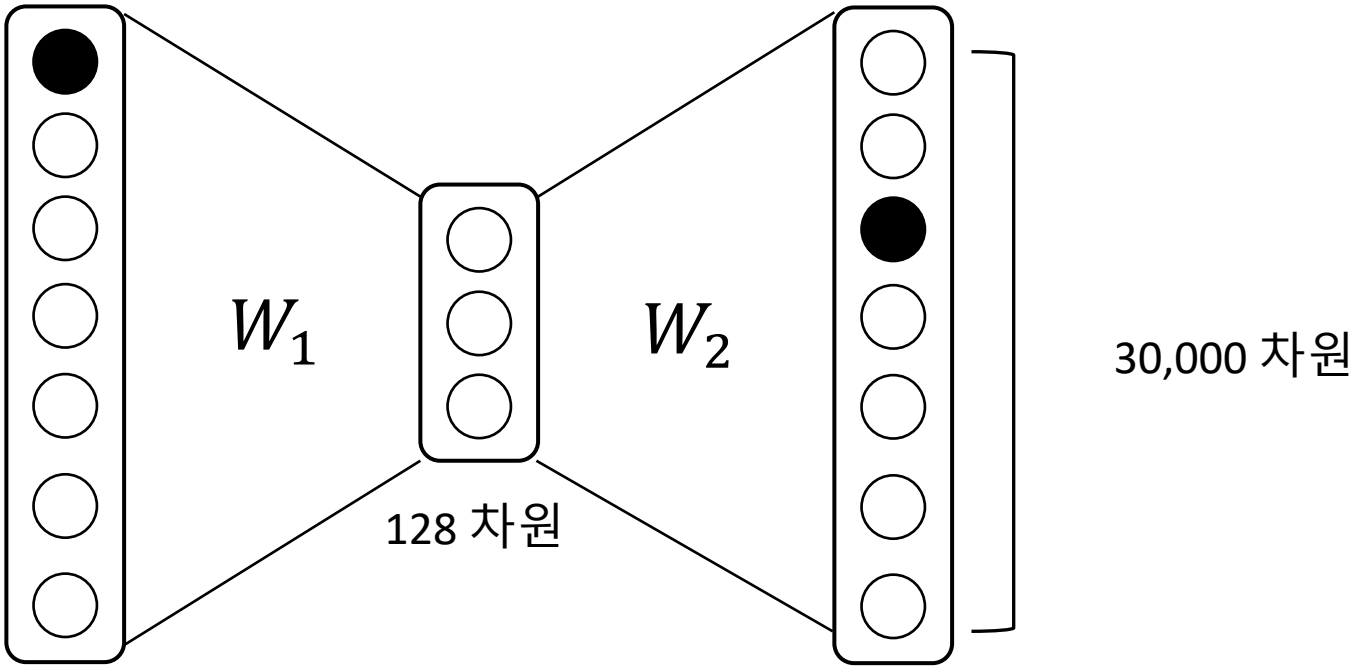
Word2vec



$$\overrightarrow{\text{orange}} \approx \overrightarrow{\text{apple}}$$

학습 트릭

- Softmax를 모든 vocab에 대해서 계산하는건 비용이 큼

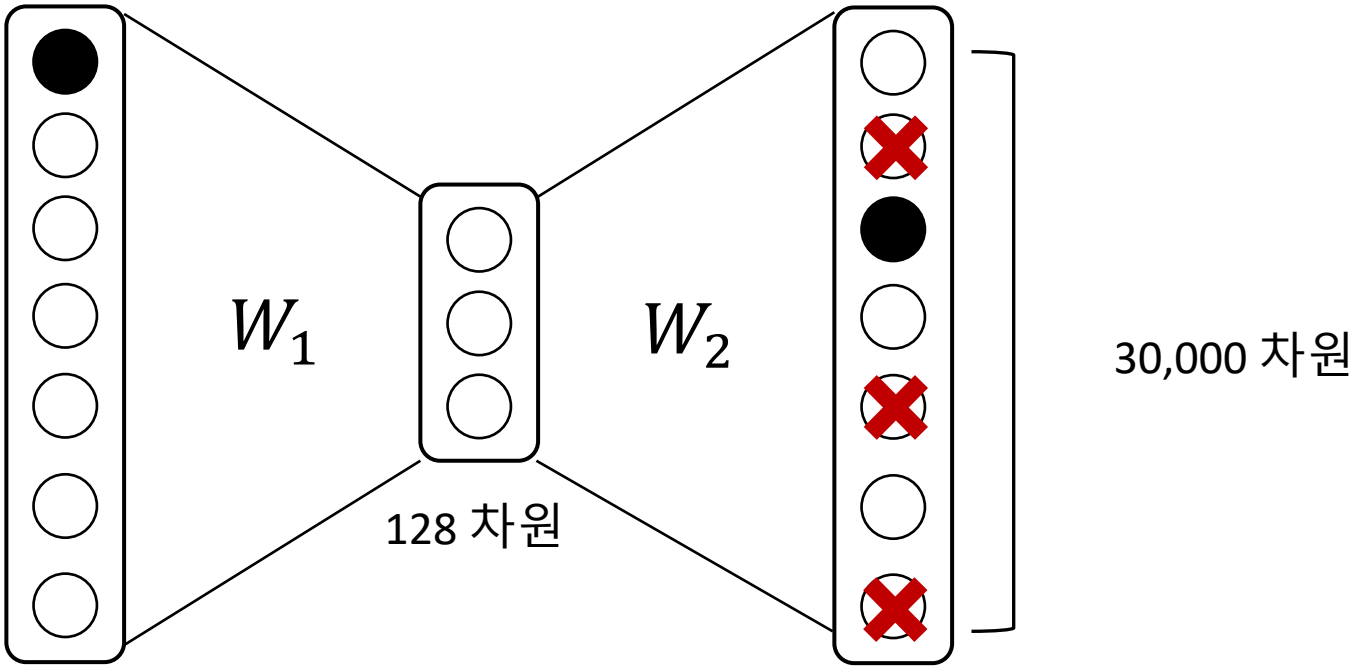


128 x 30,000 = 3,840,000 파라미터 계산 !!

학습 트릭

- Sampled Softmax

tf.nn.sampled_softmax_loss



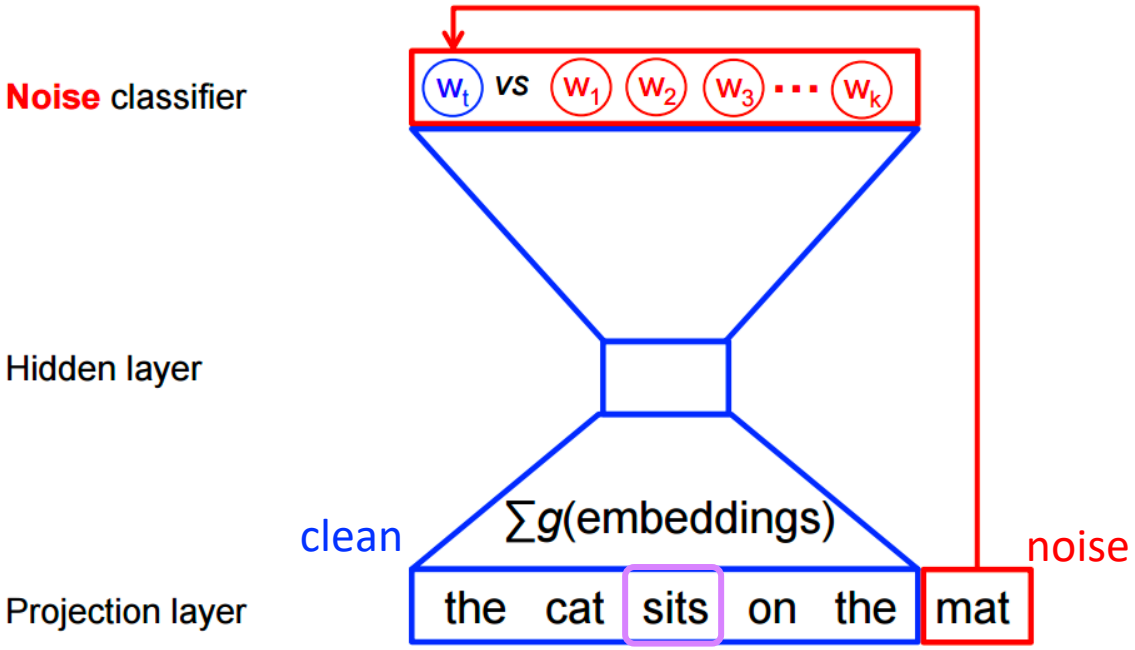
30,000 다 계산하지 말고 일부분만 계산!!

학습 트릭

- Negative Sampling

tf.nn.nce_loss

  - **문맥 안에** 있는 단어들은 **clean**
  - **문맥 밖에** 있는 단어들을 **noise**로 간주
  - 중심단어와 어떤 단어가 입력 받았을 때 그 단어가 **noise인지 clean인지** 분류하는 **이진분류방법**



여전한 한계점

- 신조어 및 학습에서 보지 못한 단어에 대한 처리 한계
  - <unk> 토큰으로 통일하기 때문에 서로 다른 단어가 동일한 벡터를 갖게 됨
  - 이를 해결하기 위해 **“FastText”, “Subword Tokenizer”**와 같은 기법이 등장

- 동음이의어에 대한 처리 한계
  - 과일에서의 “apple”과 회사에서의 “apple”에 대해서 구분을 못함
  - 이를 해결하기 위해 **“ELMo”, “BERT”, “ERNIE”, “XLNet”** .. 등등 요새 대세인 **주변 문맥에 따라 단어 벡터가 달라지는 “Contextualized Word Embedding”** 기법 등장

요즘 트렌드

- **“ELMo”, “BERT”, “ERNIE”** 의 등장으로 자연어처리 분야는 pre-trained language 모델의 시대가 열렸다
- 이 모델들(ELMo제외)은 **Transformer 모델**을 기반으로 한다