

딥러닝팀

1 팀

김재희
유경민
김주연
문서영
이수경

INDEX

1. 임베딩 (Embedding)
2. 기본적인 형태의 임베딩
3. 언어 모델을 활용한 임베딩

1

임베딩(Embedding)

1

임베딩 (Embedding)

- 컴퓨터가 문자를 인식하는 방식

ex. ASCII

A: 65	E: 69	I: 73	
B: 66	F: 70	J: 74	
C: 67	G: 71	K: 75	...
D: 68	H: 72	L: 76	

→ UTF-8, EUC-KR ...

사전에 약속된 형식에 따라 컴퓨터는 문자를 숫자로 인식

1

임베딩 (Embedding)

- 컴퓨터가 문자를 인식하는 방식

ex. ASCII

A: 65

B: 66

C: 67

D: 68

E: 69

I: 73

F: 70

J: 74

...

G: 71

H: 72

I: 76

이러한 숫자들은 단순 **범주형** 변수

B가 A보다 1만큼 크다는 의미를 갖지 않음

→ UTF-8, EUC-KR ...

사전에 약속된 형식에 따라 컴퓨터는 문자를 숫자로 인식

1

임베딩(Embedding)

- 컴퓨터가 문자를 인식하는 방식

ex. ASCII

A: 65 E: 69 I: 73

의미를 파악하게 하기 위해서는

C: 67 G: 71 K: 75

D: 68 H: 72 L: 76

문자를 숫자로 바꿀 다른 방식이 필요!

→ UTF-8, EUC-KR ...

사전에 약속된 형식에 따라 컴퓨터는 문자를 숫자로 인식

1

임베딩(Embedding)

- 임베딩이란?

임베딩

사람이 쓰는 자연어를 기계가 이해할 수 있는 벡터로 변환한 결과



호양이 (재희네 고양이)

ex . 호양이



[1, -324, 25, 43]

사람이 이 벡터의 의미를 이해할 수는 X

BUT “고양이”와 같은 다른 단어와의 상대적인 관계를 학습할 수 있도록 함

2

기본적인 형태의 임베딩

- Document-Term Matrix (DTM)

Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

문서 1: 먹고 싶은 사과

문서 2: 먹고 싶은 바나나

문서 3: 길고 노란 바나나 바나나

문서 4: 저는 과일이 좋아요

변수: 단어

관측치:
문서

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

• Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

문서 1: 먹고 싶은 사과

문서 2: 먹고 싶은 바나나

문서 3: 길고 노란 바나나 바나나

문서 4: 저는 과일이 좋아요



문서 5: 사과 같은 내 얼굴은 예쁘다

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요	같은	내	얼굴은	예쁘다
문서1	0	0	0	1	0	1	1	0	0	0	0	0	0
문서2	0	0	0	1	1	0	1	0	0	0	0	0	0
문서3	0	1	1	0	2	0	0	0	0	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1	0	0	0	0
문서5	0	0	0	0	0	1	0	0	0	1	1	1	1

• Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요	같은	내	얼굴은	예쁘다
문서1	0	0	0	1	0	1	1	0	0	0	0	0	0
문서2	0	0	0	1	1	0	1	0	0	0	0	0	0
문서3	0	1	1	0	2	0	0	0	0	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1	0	0	0	0
문서5	0	0	0	0	0	1	0	0	0	1	1	1	1

문서 수 ↑ → 단어 수 ↑ → 변수 多

∴ 빈도수를 기준으로 빈도수가 높은 단어만 사용하는 방법 존재

- Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

But,

빈도로 설명하는 접근법에는 문제가 존재!

문제 1. Sparse matrix

문제 2. 중요도 파악 불가

문서 수 ↑ ➡ 단어 수 ↑ ➡ 변수 多

∴ 빈도수를 기준으로 빈도수가 높은 단어만 사용하는 방법 존재

	과일이	바나나	먹고	바나나	사과	싫은	저는	좋아요	같은	내	얼굴은	예쁘다
문서1	0	0	1	0	1	0	0	0	0	0	0	0
문서2	0	0	0	1	1	1	0	0	0	0	0	0
문서3	0	1	1	0	2	0	0	0	0	0	0	0
문서4	1	0	0	0	0	0	1	1	0	0	0	0
문서5	0	0	0	0	0	1	0	0	1	1	1	1

- Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

문제1 대부분의 값이 0인 Sparse Matrix가 생성됨

	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

➡ 비효율적인 데이터 형태로, 튜닝을 제대로 하기 어려움

- Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

문제1 대부분의 값이 0인 Sparse Matrix가 생성됨

Sparse matrix가 왜 비효율적인가요?

0 이 많아서 정보에 비해 데이터 사이즈가 큼

→ 학습 시간 ↑

→ 그리드 서치 시간 ↑

→ 제대로 튜닝하기 어려움

	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	1	0	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

- Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

문제2 어떤 단어가 중요한 변수인지 알기 힘들

문서1 : 나는 XGBoost 모델 좋아해

문서2 : 나는 탐앤탐스 성균관대점 좋아해

문서3 : 나는 XGBoost 모델 극혐해

	나	모델	좋아해	XGBoost	극혐해	탐앤탐스	성균관대점
문서1	1	1	1	1	0	0	0
문서2	1	0	1	0	0	1	1
문서3	1	1	0	0	1	0	0
문서총 빈도	3	2	2	1	1	1	1

• Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

문제2 어떤 단어가 중요한 변수인지 알기 힘들

문서1 : 나는 XGBoost 모델 좋아해

문서2 : 나는 탐앤탐스 성균관대점 좋아해

문서3 : 나는 XGBoost 모델 극혐해

	나	모델	좋아해	XGBoost	극혐해	탐앤탐스	성균관대점
문서1	1	1	1	1	0	0	0
문서2	1	0	1	0	1	1	1
문서3	1	1	0	0	1	0	0
문서총 빈도	3	2	2	1	1	1	1

총 등장 빈도로
상위 3개만 골랐을 때 사라짐

• Document-Term Matrix (DTM)

: 문서의 특징을 단어들의 출현 빈도를 통해 나타내는 방법

문제2 어떤 단어가 중요한 변수인지 알기 힘들

	나	모델	좋아해
문서1	1	1	1
문서2	1	0	1
문서3	1	1	0
문서총빈도	3	2	2

- 문서1 : 나는 XGBoost 모델 좋아해
- 문서2 : 나는 탐앤탐스 성균관대점 좋아해
- 문서3 : 나는 XGBoost 모델 극혐해

XGBoost

탐앤탐스

극혐

고려 X



문서1: Xgboost ♡

문서2: 탐앤탐스 ♡

문서3: Xgboost ♡

주제를
알 수 없음

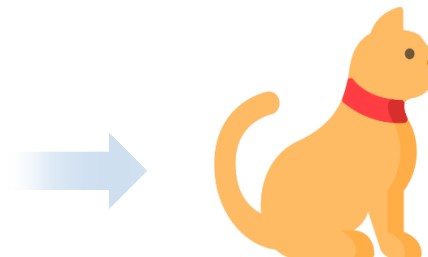
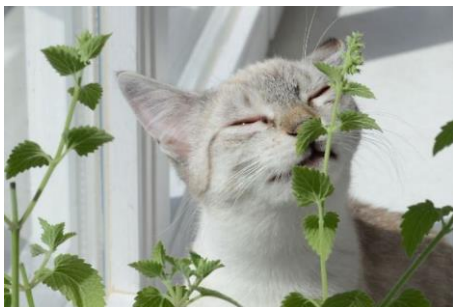
- TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency)

: 중요한 단어를 판단하는 기준을 마련해 임베딩 하는 방법

↪ 다른 문서와 구별 짓는데 효과적인 단어

Ex - 주제: 고양이, 선우정아, 커피, 이불



“캣닙”, “모래”와 같은 단어는 **고양이** 주제를 판별하는데 유용

- TF-IDF

: 중요한 단어를 판단하는 기준을 마련해 임베딩 하는 방법

TF

각 문서 별 단어 출현 빈도

$$tf(d, t) = count(d, t)$$

IDF

DF(문서 개수)의 역수

$$idf(t, D) = \log\left(\frac{n}{1 + df(t)}\right)$$

- TF-IDF

: 중요한 단어를 판단하는 기준을 마련해 임베딩 하는 방법

$$idf(t, D) = \log\left(\frac{n}{1 + df(t)}\right)$$

Handwritten annotations:
- Red arrow pointing to n : 전체 문서
- Orange arrow pointing to $df(t)$: t 가 등장한 문서

t 가 등장한 문서의 수 \downarrow \longrightarrow $idf(t, D) \uparrow$

\therefore 특정 문서에만 등장하는 단어에 가중치 주는 역할

- TF-IDF

: 중요한 단어를 판단하는 기준을 마련해 임베딩 하는 방법

$$idf(t, D) = \log\left(\frac{n}{1 + df(t)}\right)$$

df 값이 작은 단어들은 로그를 취하지 않을 경우 너무 큰 값이 나옴

\therefore log 스케일링

- TF-IDF

: 중요한 단어를 판단하는 기준을 마련해 임베딩 하는 방법

$$tf-idf(t, d, D) = tf(t, d) * idf(t, D)$$

	먹고	사과
문서1	1	1
문서2	1	0
문서3	0	0
문서4	0	0



	먹고	사과
문서1	0.2876	0.6931
문서2	0.2876	0
문서3	0	0
문서4	0	0

- TF-IDF

DTM TF-IDF

: 중요한 단어를 판단하는 기준을 마련해 임베딩하는 방법

왜 같은 빈도 1인데 값이 다를까요?

	먹고	사과
문서1	1	1
문서2	1	0
문서3	0	0
문서4	0	0



	먹고	사과
문서1	0.2876	0.6931
문서2	0.2876	0
문서3	0	0
문서4	0	0

● TF-IDF

DTM TF-IDF

: 중요한 단어를 판단하는 기준을 마련해 임베딩하는 방법

왜 같은 빈도 1인데 값이 다를까요?

	먹고	사과
문서1	1	1
문서2	1	0
문서3	0	0
문서4	0	0



	먹고	사과
문서1	0.2876	0.6931
문서2	0.2876	0
문서3	0	0
문서4	0	0

‘먹고’는 전체 문서에서의 빈도가 2회, ‘사과’는 1회

∴ ‘사과’가 문서 1을 다른 문서와 구별 짓는 정도가 더 높음

- TF-IDF

: 중요한 단어를 판단하는 기준을 마련해 임베딩 하는 방법



문서의 특징을 반영하여 숫자를 바꾸어줌
스팸 메일 판별 문제, 토픽모델링 등에서 사용

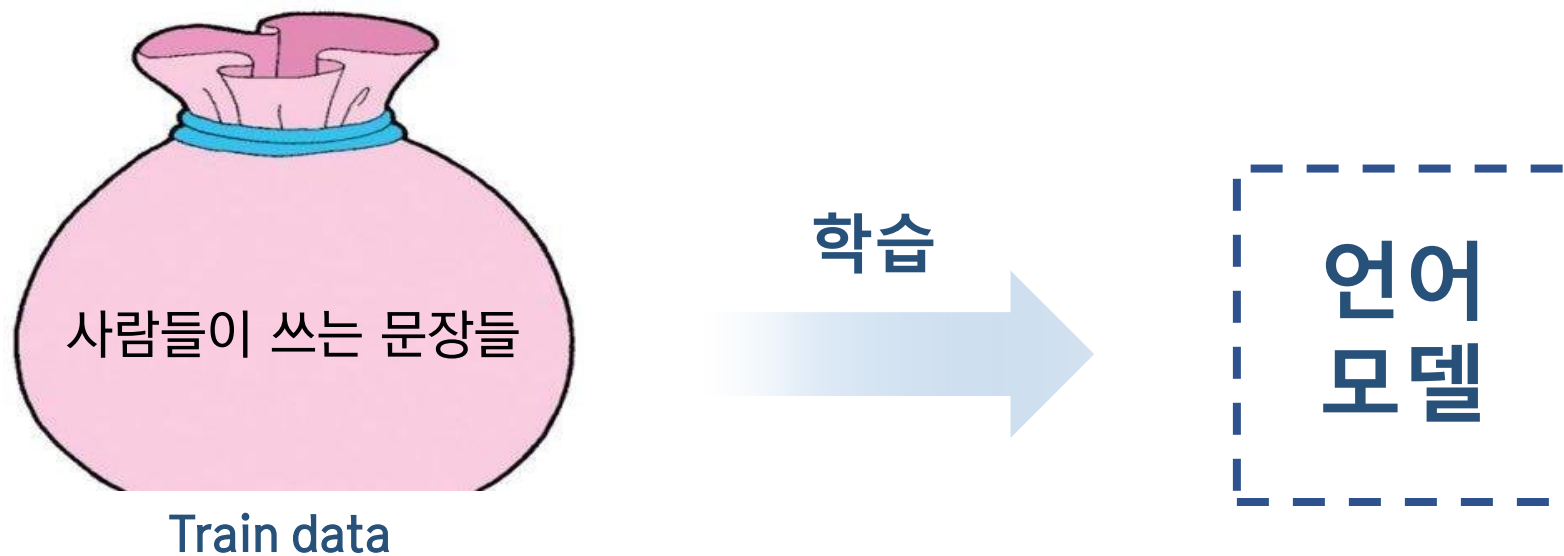


문서 단위의 임베딩 방식으로 단어들의 관계를 분석하지 못함
감성분석, 개체명 인식 문제에서 사용 불가

3

언어 모델을 활용한 임베딩

- 언어 모델이란?



이때, 단어의 **배열**과 **순서**가 중요

- 등장 배경

문장

진이는 이 책을 세 번 읽었다.

이 책이 진이한테 세 번을 읽혔다.

세 번이 진이한테 이 책을 읽혔다.

- 등장 배경

문장

진이는 이 책을 세 번 읽었다.

애만 말이 됨!

~~이 책이 진이한테 세 번을 읽혔다.~~

~~세 번이 진이한테 이 책을 읽혔다.~~

같은 단어들이라도 조합에 따라 첫번째 문장만 말이 됨

- 등장 배경

문장

진이는 이 책을 세번 읽었다.

야만 말이 됨!

문법 구조를 익히는 것이 아니라
단어들의 **쓰임새**를 배우는 것!

같은 단어들이라도 조합에 따라 1번 문장만 말이 됨

- 통계 기반 언어 모델(N-gram)

확률적 계산



아니 내가 어제 말이야, 친구랑 신촌을 갔거든?
일단 밥부터 먹고 카페 가서 p-sa 얘기하고
있었는데, 거기서 개 있잖아, 우리 엘씨 친구
OO이를 엄청 오랜만에 만난 거야!! 진짜 너무
반가웠어!

말을 순서대로 하기 때문에 반가웠어 를 예측하기 위해서
앞의 모든 단어를 조건으로 생각해야 됨.

- 통계 기반 언어 모델(N-gram)

확률적 계산

$$\frac{\text{Freq}(\text{아니, 내가, 어제, 말아야, } \dots, \text{진짜, 너무, 반가웠어})}{\text{Freq}(\text{아니, 내가, 어제, 말아야, } \dots, \text{진짜, 너무})} \doteq 0$$

동일한 어순으로 문장이 여러 개 작성될 확률은 0에 수렴함

∴ 문장이 길어질수록, 조건부 확률값이 의미가 없게 됨

- 통계 기반 언어 모델(N-gram)

확률적 계산

그렇다면 0이 아닌 확률 값은 어떻게 구할까?

$$\frac{\text{Freq}(\text{아니, 내가, 어제, 말이야, ... , 진짜, 너무})}{\text{Freq}(\text{아니, 내가, 어제, 말이야, ... , 진짜, 너무})} \doteq 0$$

동일한 어순으로 문장이 작성될 확률은 0에 수렴함

∴ 문장이 길어질수록, 조건부 확률값이 의미가 없게 됨

- 통계 기반 언어 모델(N-gram)

마코프 가정

Markov Assumption

T 시점의 행동은
“ 그 직전 n개 시점의 영향만을 받는다 ”

진이는 어제 이 책을 세번 **읽었다**.

- 통계 기반 언어 모델(N-gram)

마코프 가정

T 시점의 행동은
“ 그 직전 n개 시점의 영향만을 받는다 ”

어떤 단어를 예측할 때
직전 n개 단어만 보면 된다!

- 통계 기반 언어 모델(N-gram)

단어 예측 (N=1)

$$\frac{Freq(\text{즐겁게, 읽었다})}{Freq(\text{즐겁게})}$$

진이는 어제 책을 즐겁게 읽었다.

~~진이~~ ~~어제~~ ~~책~~ 즐겁게 읽었다.

무시됨!

1개

1-gram은 직전 한 개의 단어 “즐겁게”를 조건으로 생각한다.

- 통계 기반 언어 모델(N-gram)

단어 예측 (N=2)

$$\frac{Freq(\text{책, 즐겁게, 읽었다})}{Freq(\text{책, 즐겁게})}$$

진이는 어제 책을 즐겁게 읽었다.

~~진이~~ ~~어제~~ 책 즐겁게 읽었다.

무시됨!

2개

2-gram은 직전 두 개의 단어 “책” “즐겁게” 를 조건으로 생각한다.

- 통계 기반 언어 모델(N-gram)

단어 예측 (N=3)

$$\frac{Freq(\text{어제, 책, 즐겁게, 읽었다})}{Freq(\text{어제, 책, 즐겁게})}$$

진이는 어제 책을 즐겁게 읽었다.

~~진이~~ 어제 책을 즐겁게 읽었다.

무시됨!

37%

3-gram은 직전 세 개의 단어 “어제” “책” “즐겁게” 를 조건으로 생각한다.

- 통계 기반 언어 모델(N-gram)

단어 예측

1-gram

$$\frac{\text{Freq (즐겁게, 읽었다)}}{\text{Freq (즐겁게)}}$$

$$= 0.5$$

2-gram

$$\frac{\text{Freq (책, 즐겁게, 읽었다)}}{\text{Freq (책, 즐겁게)}}$$

$$= 0.3$$

3-gram

$$\frac{\text{Freq (어제, 책, 즐겁게, 읽었다)}}{\text{Freq (어제, 책, 즐겁게)}}$$

$$= 0.001$$

N이 커짐에 따라 예측값이 극도로 작아지는 것을 확인할 수 있음

✓ N을 증가시키는데 한계가 있다!

- 통계 기반 언어 모델(N-gram)

단어 예측

1-gram

N을 증가시키는데 한계가 있다

2-gram

3-gram

실제로는 인근 단어만이 서로 종속 관계 x

$Freq(\text{즐겁게, 읽었다})$

$Freq(\text{책, 즐겁게, 읽었다})$

$Freq(\text{어제, 책, 즐겁게, 읽었다})$

$Freq(\text{즐겁게})$

$Freq(\text{책, 즐겁게})$

$Freq(\text{어제, 책, 즐겁게})$

→ N-gram 기본 개념만 익혀 두자!

= 0.5

마코프 가정을 통해
= 0.3

= 0.001

인근 단어만 이용해서 현재 단어를 파악한다

N이 커짐에 따라 예측값이 극도로 작아지는 것을 확인할 수 있음

∴ N을 증가시키는데 한계가 있다!

- 통계 기반 언어 모델(N-gram)

But,

N-gram에서는 임베딩 방법을 알려주지 않음

N-gram 기반으로 어떻게 하면

벡터 형태의 입력값을 만들 수 있을까?

- NPLM

NPLM

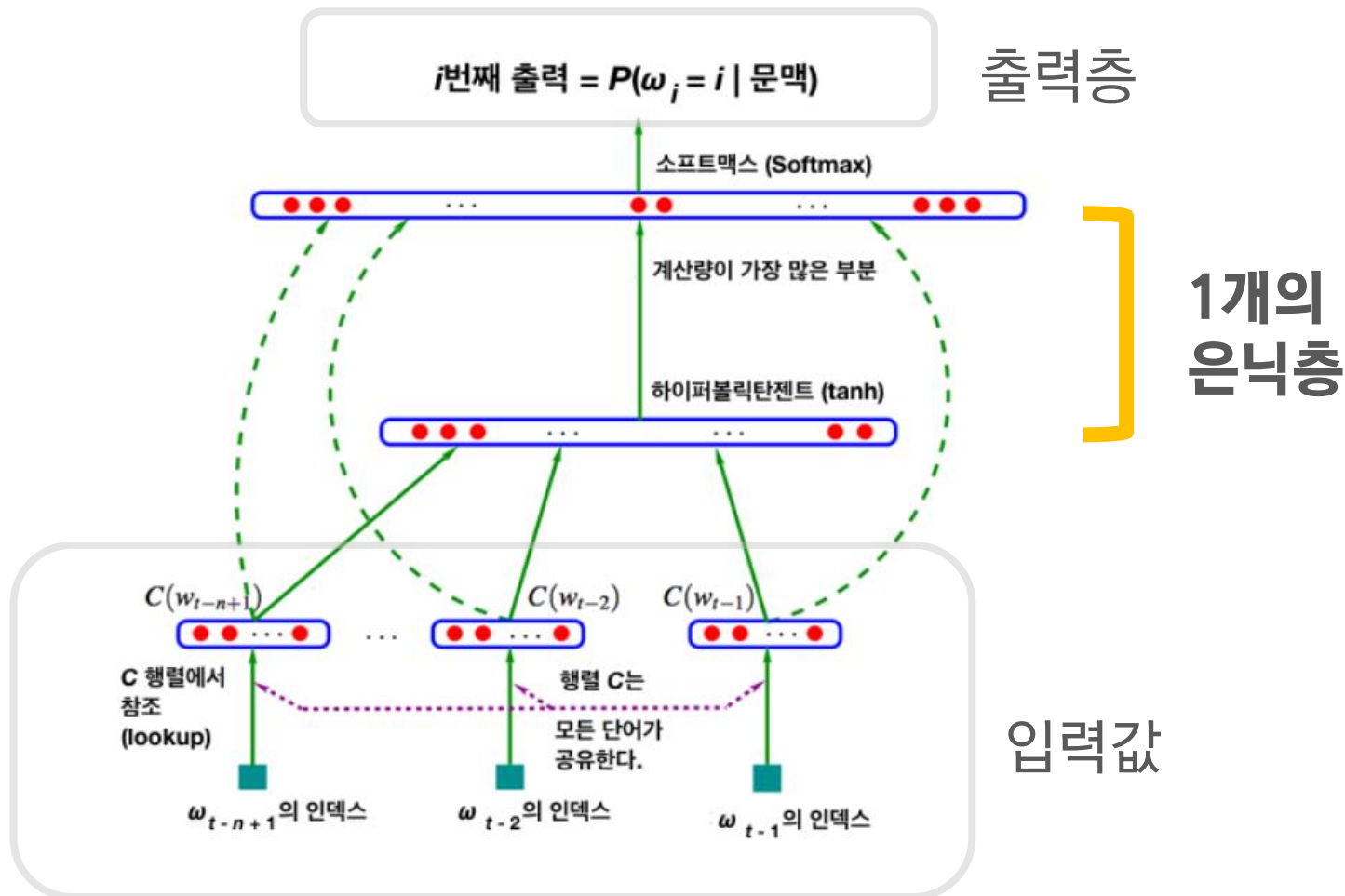
(Neural Probabilistic Language Model)

: 은닉층이 1개인 인공신경망

3

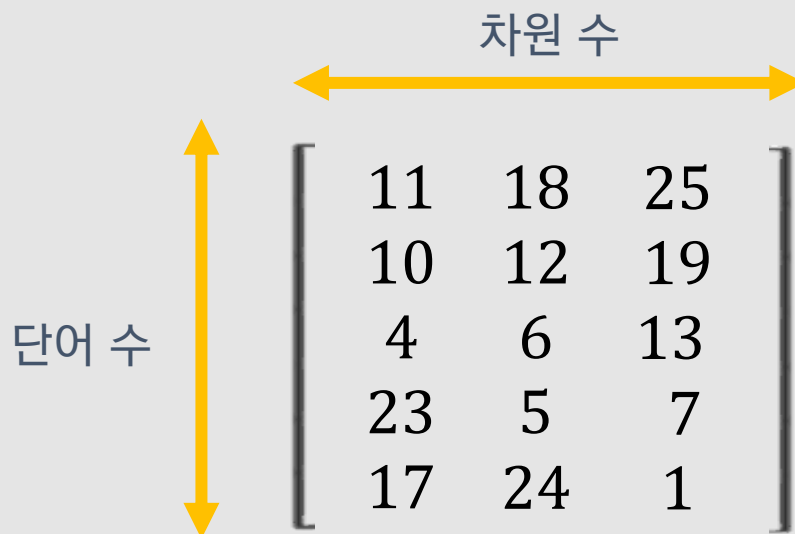
언어 모델을 활용한 임베딩

• NPLM



- NPLM

행렬 C



A diagram showing a 5x3 matrix C. To the left of the matrix is a vertical yellow double-headed arrow labeled '단어 수' (Number of words). Above the matrix is a horizontal yellow double-headed arrow labeled '차원 수' (Number of dimensions).

	11	18	25
	10	12	19
	4	6	13
	23	5	7
	17	24	1

$$C_{\text{행렬}} = V(\text{단어}) * m(\text{차원})$$

초기 C는 무작위 배정

- NPLM

행렬 C

단어1	11	18	25
단어2	10	12	19
단어3	4	6	13
단어4	23	5	7
단어5	17	24	1

각 행의 벡터 = 하나의 단어

- NPLM

입력값

진이 어제 책 즐겁게 읽었다.

$$\begin{array}{ccccc}
 [& 1 & 0 & 0 & 0 & 0] \\
 & \text{진이} & \text{어제} & \text{책} & \text{즐겁게} & \text{읽었다} \\
 & & & & & *
 \end{array}
 \begin{bmatrix}
 11 & 18 & 25 \\
 10 & 12 & 19 \\
 4 & 6 & 13 \\
 23 & 5 & 7 \\
 17 & 24 & 1
 \end{bmatrix}
 = [11 \quad 18 \quad 25]$$

One-hot encoding 벡터

C행렬

단어에 해당하는 Imbedded vector

One-hot encoding 된 벡터와 C행렬을 내적한다.

- NPLM

입력값

진이 어제 책 즐겁게 읽었다.

$$\begin{array}{ccccc}
 [0 & 1 & 0 & 0 & 0] \\
 \text{진} & \text{어} & \text{책} & \text{즐} & \text{읽} \\
 \text{이} & \text{제} & & \text{겁} & \text{었} \\
 & & & \text{게} & \text{다}
 \end{array}
 *
 \begin{bmatrix}
 11 & 18 & 25 \\
 10 & 12 & 19 \\
 4 & 6 & 13 \\
 23 & 5 & 7 \\
 17 & 24 & 1
 \end{bmatrix}
 = [10 \quad 12 \quad 19]$$

One-hot encoding 벡터

C행렬

단어에 해당하는 Imbedded vector

One-hot encoding 된 벡터와 C행렬을 내적한다.

- NPLM

입력값

진이 어제 책 즐겁게 읽었다.

$$\begin{array}{ccccc}
 [& 0 & 0 & 1 & 0 & 0] \\
 & \text{진} & \text{어} & \text{책} & \text{즐} & \text{읽} \\
 & \text{이} & \text{제} & & \text{겁} & \text{었} \\
 & & & & \text{게} & \text{다}
 \end{array}
 * \begin{bmatrix} 11 & 18 & 25 \\ 10 & 12 & 19 \\ 4 & 6 & 13 \\ 23 & 5 & 7 \\ 17 & 24 & 1 \end{bmatrix} = [4 \quad 6 \quad 13]$$

One-hot encoding 벡터

C행렬

단어에 해당하는 Imbedded vector

One-hot encoding 된 벡터와 C행렬을 내적한다.

- NPLM

입력값

진이 어제 책 **즐겁게** 읽었다.

$$\begin{array}{ccccc}
 [& 0 & 0 & 0 & 1 & 0] \\
 & \text{진} & \text{어} & \text{책} & \text{즐} & \text{읽} \\
 & \text{이} & \text{제} & & \text{겁} & \text{었} \\
 & & & & \text{게} & \text{다}
 \end{array}
 *
 \begin{bmatrix}
 11 & 18 & 25 \\
 10 & 12 & 19 \\
 4 & 6 & 13 \\
 23 & 5 & 7 \\
 17 & 24 & 1
 \end{bmatrix}
 = [23 \quad 5 \quad 7]$$

One-hot encoding 벡터

C행렬

단어에 해당하는 Imbedded vector

One-hot encoding 된 벡터와 C행렬을 내적한다.

- NPLM

입력값 : 3-gram일 경우

진이 어제 책 즐겁게 읽었다.

$x = [10 \quad 12 \quad 19 \quad 4 \quad 6 \quad 13 \quad 23 \quad 5 \quad 7]$

어제 책 즐겁게

하나의 벡터로 Concatenate

• NPLM

입력값 : 3-gram일 경우

진이 **어제** **책** **즐겁게** 읽었다.

$x = [10 \quad 12 \quad 19 \quad 4 \quad 6 \quad 13 \quad 23 \quad 5 \quad 7]$

어제

책

즐겁게

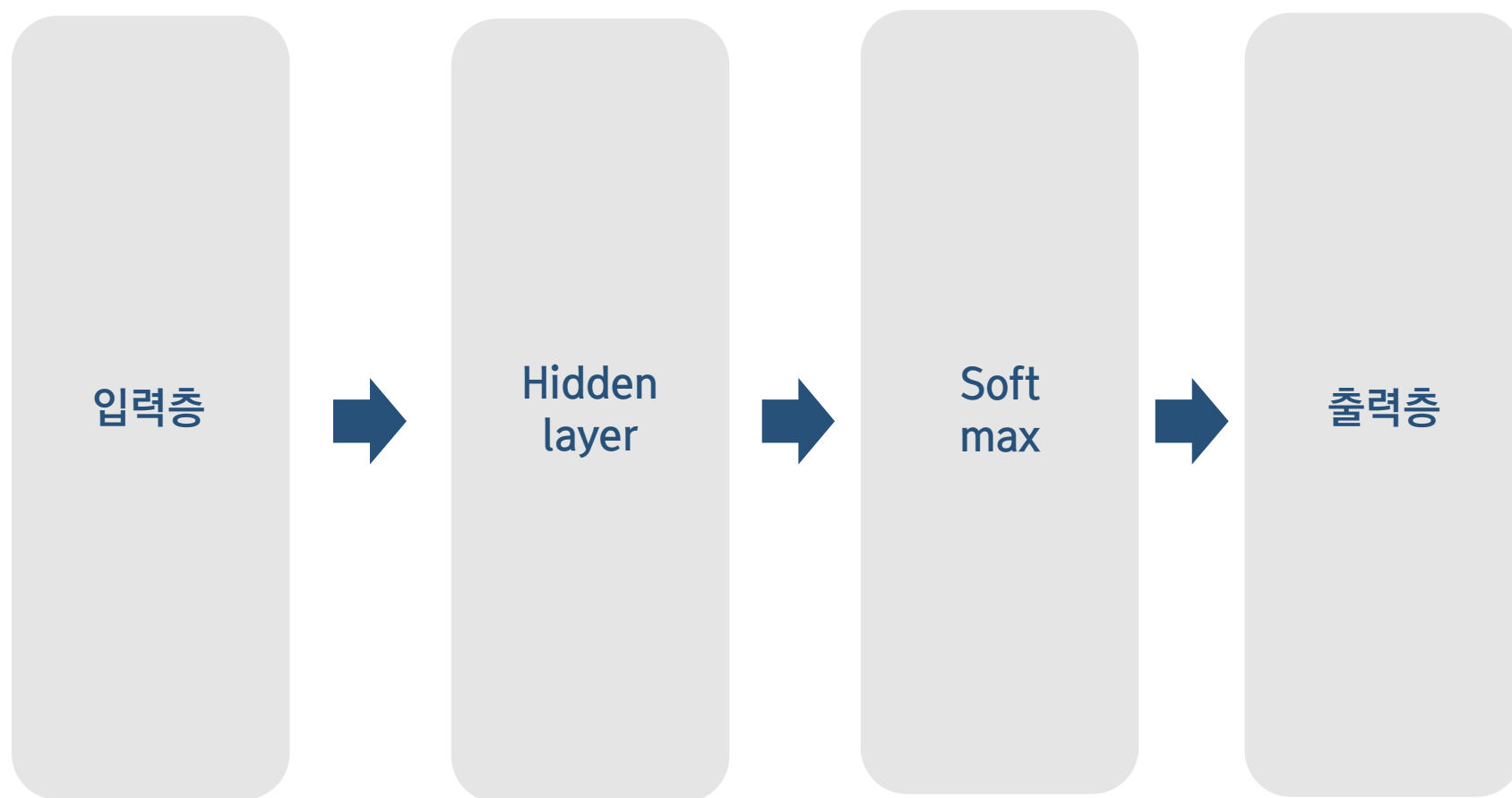
“어제” “책” “즐겁게”는

“읽었다”를 예측하는데 **동일하게** 반영됨.

하나의 벡터로 Concatenate

- NPLM

전체 구조



- NPLM

전체 구조

NPLM에서는 입력값이

입력층 → Hidden layer → Soft max → 출력층

Concatenate된 x벡터라는 것이 중요!

- NPLM

NPLM의 의미정보

단어1	11	18	25
단어2	10	12	19
단어3	4	6	13
단어4	23	5	7
단어5	17	24	1

NPLM을 통해 최종적으로 구하고자 하는 것은 C행렬

NPLM을 학습한 뒤의 C행렬의 각 행 벡터는 각 단어의 임베딩 값

C행렬에서 필요한 행벡터를 골라 RNN LSTM 모델의 입력 값으로 사용

- NPLM

NPLM의 의미정보

〈3-gram, **walking** 예측〉

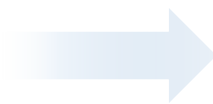
The cat is **walking** in the bedroom

A dog is **walking** in a bedroom

〈 입력값 〉

cat $\begin{bmatrix} 8 & 3 & 1 \end{bmatrix}$

dog $\begin{bmatrix} 1 & 2 & 1 & 9 & 2 & 5 \end{bmatrix}$



〈 출력값 〉

walking

• NPLM

NPLM의 의미정보

〈3-gram, **walking** 예측〉The cat is **walking** in the bedroomA dog is **walking** in a bedroom

〈 입력값 〉

cat $\begin{bmatrix} 8 & 3 & 1 \end{bmatrix}$ dog $\begin{bmatrix} 1 & 2 & 1 & 9 & 2 & 5 \end{bmatrix}$

〈 출력값 〉

walking

역전파를 통해 임베딩 벡터 최적화

- NPLM

NPLM의 의미정보

〈3-gram, walking 예측〉

The cat is **walking** in the bedroom

A dog is **walking** in a bedroom

The dog was **walking** in the room

동일한 출력값을 내는 단어의 벡터는
비슷해질 수 밖에 없다! 〈출력값〉

<u>cat</u>	1 3	2 0	2 5
<u>dog</u>	1 2	1 9	2 5

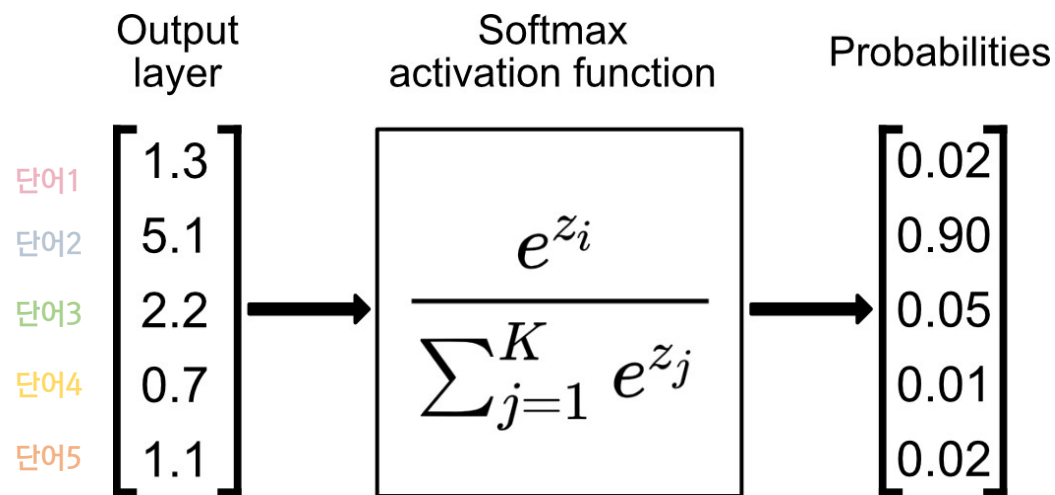
walking

역전파를 통해 임베딩 벡터 최적화

● NPLM

NPLM의 한계

: 소프트 맥스 계층의 계산량이 너무 많음



소프트 맥스 계층에서 단어의 개수만큼 단어에 대한 각각의 확률을 구해야 함
하지만 단어의 개수는 3만개를 넘는 경우가 흔하기에 계산량이 너무 많고 비효율적

- Word2Vec

Skip-gram

임베딩 하고자 하는 단어(t)를 중심으로, 다른 단어들이 n -gram에 해당하는
단어들인지(= t 의 주변에 있는 단어인지) 예측하는 모델
 n 번째 단어(중심단어)로 앞 뒤의 n 개의 주변단어를 맞추고자 함

• Word2Vec

모델 구조

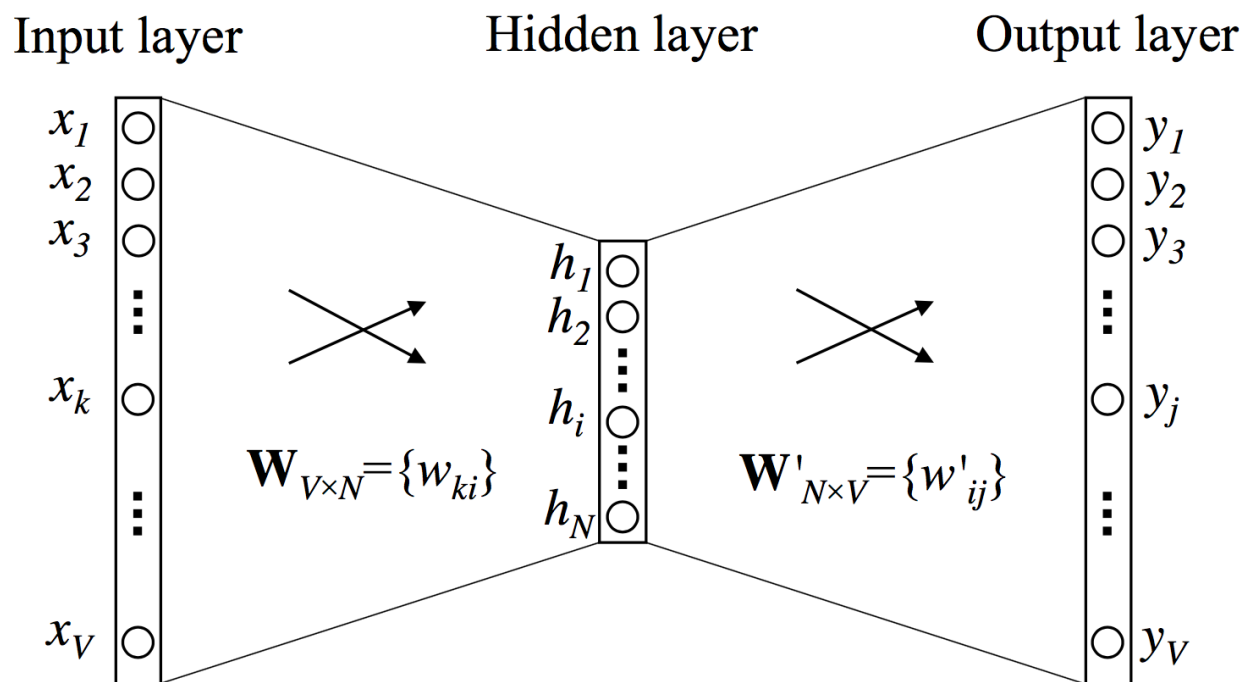


Figure 1: A simple CBOW model with only one word in the context

- Word2Vec

Ex 2-gram

The quick brown fox ^{중심단어} jumps over the lazy dog
↖ 주변단어 ↗

중심단어를 가지고 주변 단어 예측

- 중심단어 → 입력 값
- 주변단어 → 출력 값

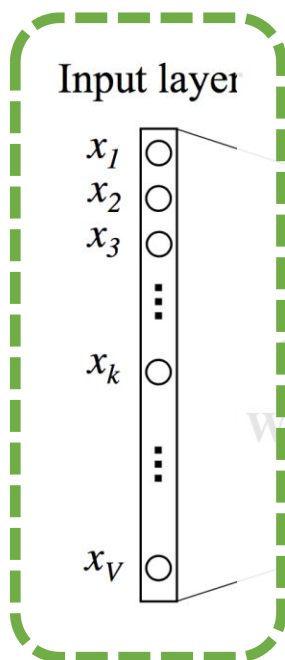
- Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox jumps over the lazy dog

중심단어
주변단어



Input layer

: one hot encoding된 단어들이 벡터 형태로 들어 감

(the, quick, brown, fox, jumps, over, the, lazy, dog)

(0 0 0 0 1 0 0 0 0)

Output layer

y_1
 y_2
 y_3
 \vdots
 y_V

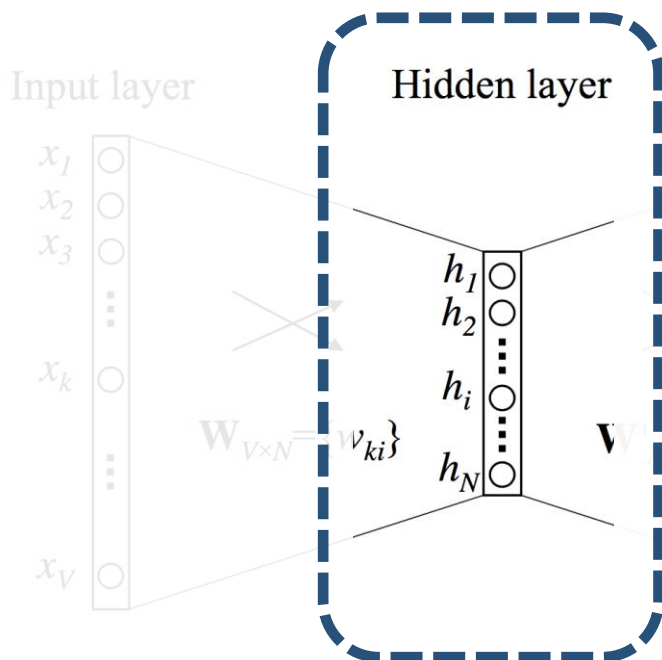
- Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox **jumps** over the lazy dog

중심단어
주변단어



Hidden Layer

- 각 단어의 임베딩 벡터의 차원 수
= Hidden Layer의 Node 수
- Word2Vec은
하나의 Hidden Layer을 가지는 인공신경망 모델

3

언어 모델을 활용한 임베딩

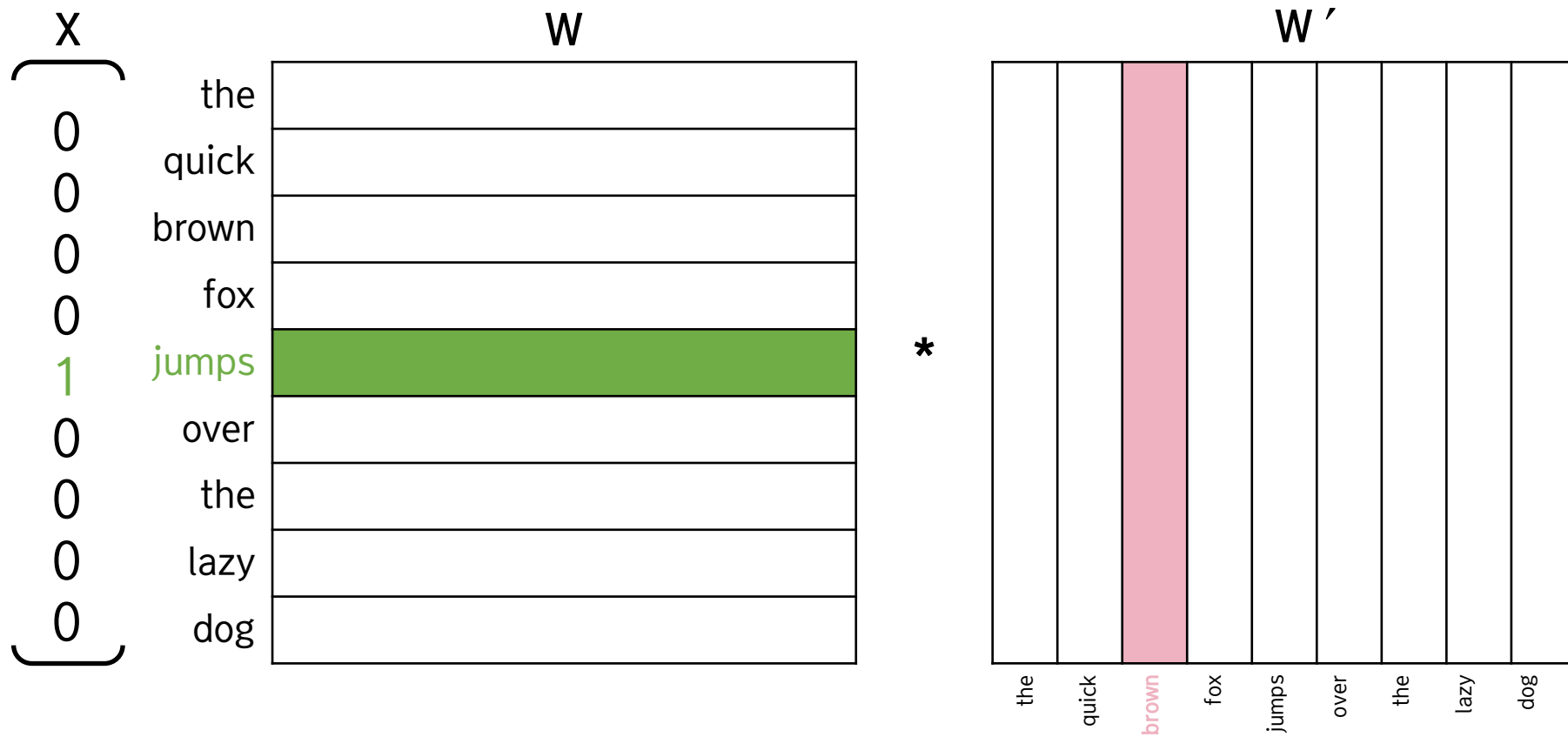
• Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox jumps over the lazy dog

중심단어
주변단어



3

언어 모델을 활용한 임베딩

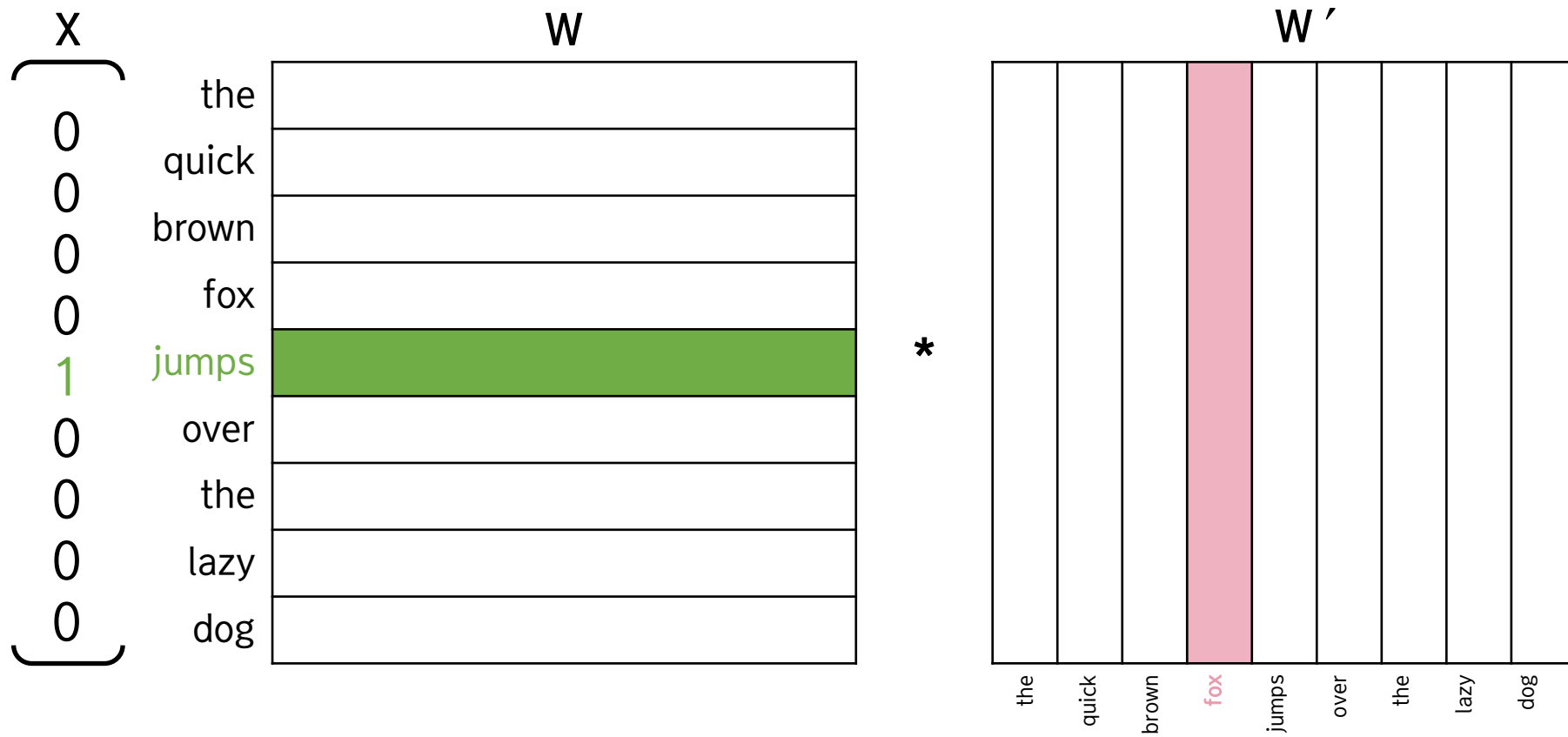
• Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox jumps over the lazy dog

중심단어
주변단어



3

언어 모델을 활용한 임베딩

• Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox jumps over the lazy dog

중심단어
주변단어

$$X = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

	W
the	
quick	
brown	
fox	
jumps	
over	
the	
lazy	
dog	

*

	W'
the	
quick	
brown	
fox	
jumps	
over	
the	
lazy	
dog	

3

언어 모델을 활용한 임베딩

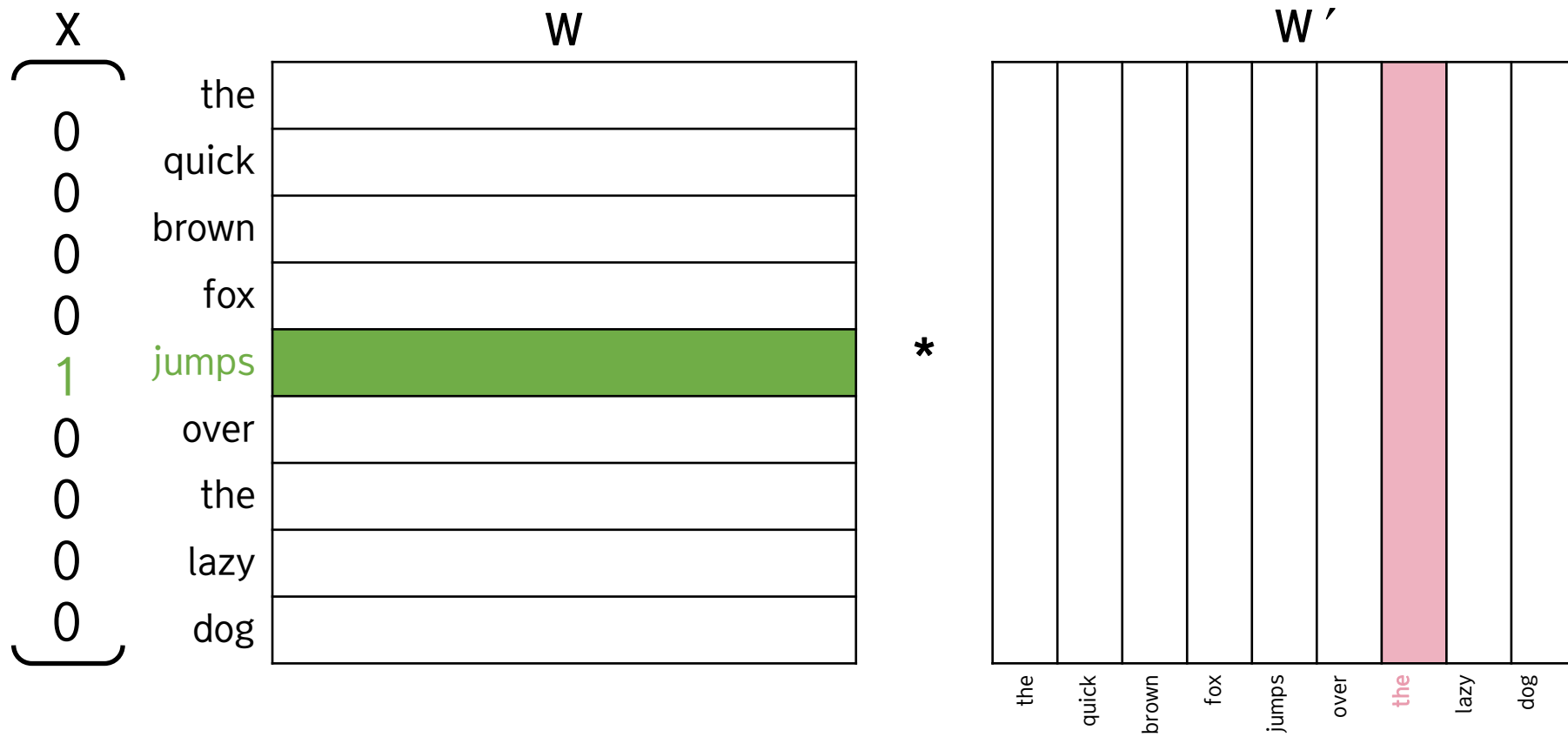
• Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox jumps over the lazy dog

중심단어
주변단어



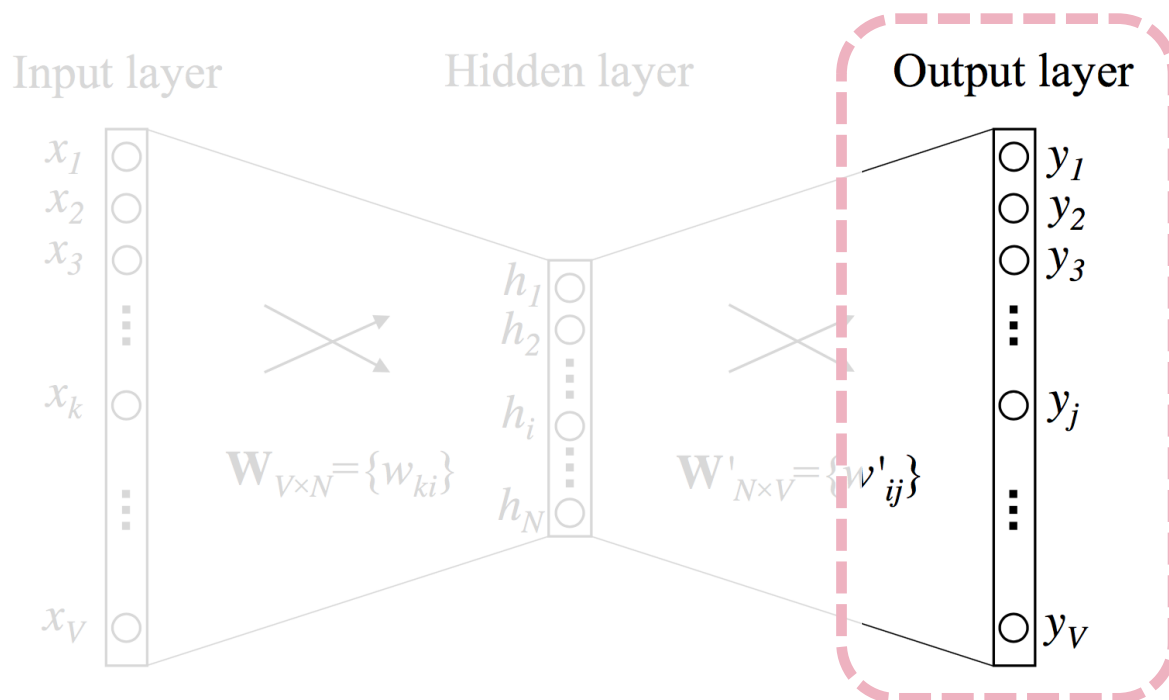
- Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox **jumps** over the lazy dog

중심단어
← 주변단어 →



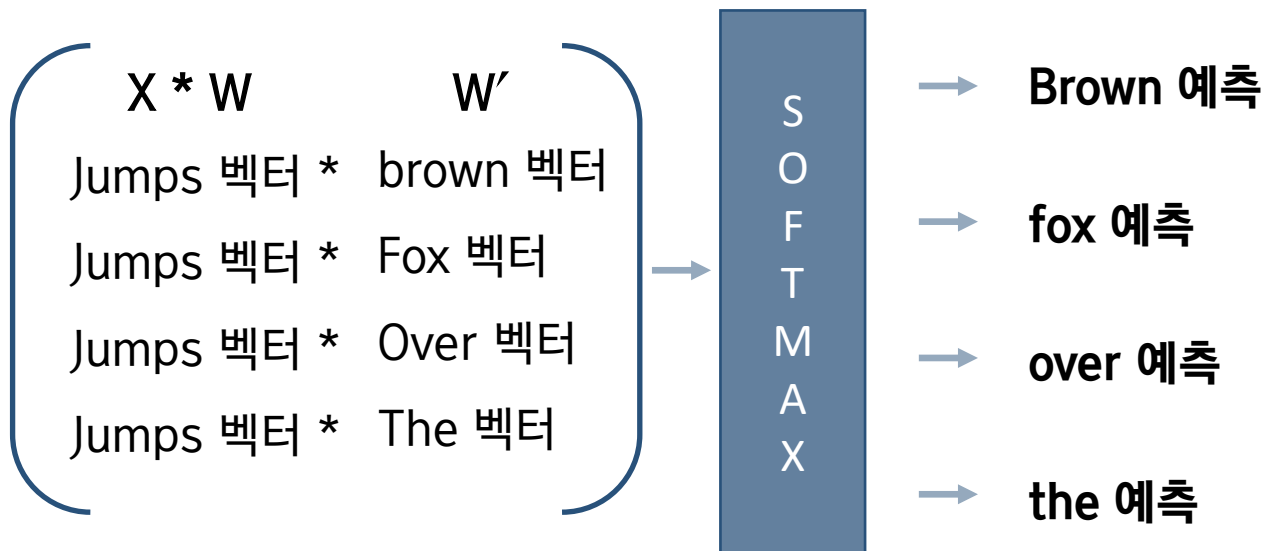
- Word2Vec

Input Hidden Output

EX 2-gram

The quick brown fox jumps over the lazy dog

중심단어
주변단어



Embedded vector와 가중치를 곱해 나온 예측 값을
'소프트맥스'에 통과시켜 주변 단어를 예측

- Word2Vec

Input Hidden Output

2-gram

The quick brown fox **jumps** over the lazy dog

중심단어
주변단어

→ 소프트맥스 계층에서 계산량이 너무 많아 비효율적

→ NPLM과 큰 차이 없음

Jumps 벡터 * brown 벡터

Jumps 벡터 * Fox 벡터

✓ Jumps 벡터 * Over 벡터

Jumps 벡터 * The 벡터

S
O
F
T
M
A
X

→ Brown 예측

→ fox 예측

→ the 예측

→ the 예측

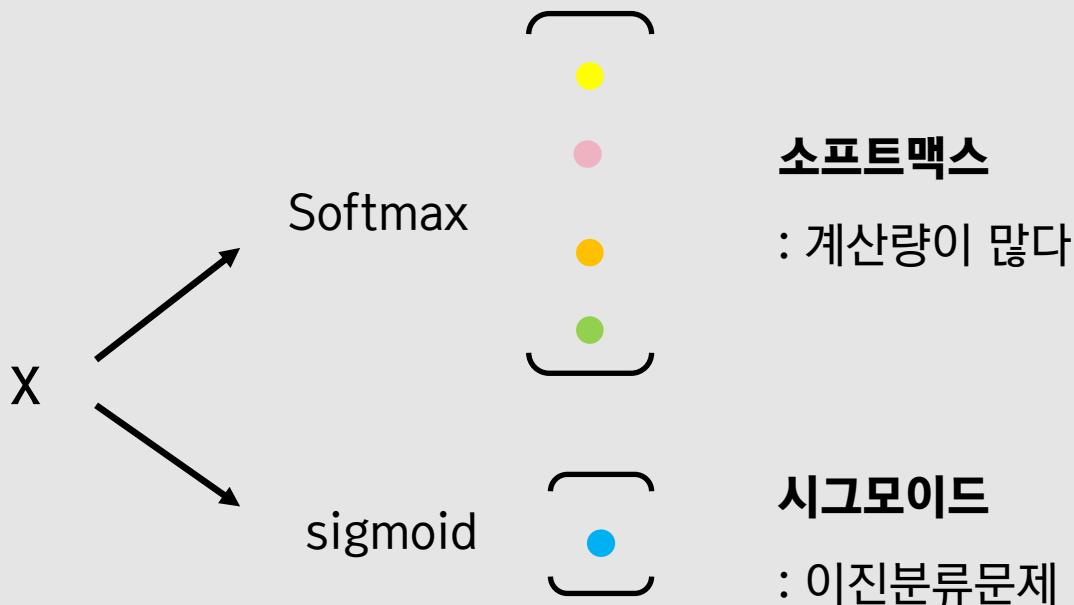
소프트맥스가 아니라 **시그모이드**를 쓰자!

Embedded vector와 가중치를 곱해 나온 예측 값을

‘소프트맥스’에 통과시켜 주변 단어를 예측

- Word2Vec

Q. 왜 시그모이드를 쓸까?



- Word2Vec

Q. 왜 시그모이드를 쓸까?



- Word2Vec

Q. 왜 시그모이드를 쓸까?

주변 단어가 아닌 단어(k 개)들도

데이터 셋으로 만들어 주어야 함

→ 네거티브 샘플링!

Softmax

소프트맥스

: 계산량이 많다

sigmoid

시그모이드

: 이진분류문제

...> 주변 단어인지 아닌지 판단

- Word2Vec

네거티브 샘플링

주변 단어가 아닌 k개의 단어를 뽑는데 사용되는 방법

$$P_{negative}(w_i) = \frac{U(w_i)^{\frac{3}{4}}}{\sum (U(w_j^{\frac{3}{4}}))}$$

K개의 단어 추출 확률

$U(w_i)$: w_i 단어의 등장 비율

➔ 빈도가 적은 단어들이 좀 더 많이 뽑히게 됨

- Word2Vec

〈 2-gram, 중심단어 The 〉

The brown fox

The red cat

$\left\{ \begin{array}{l} \text{The, brown} \\ \text{The, fox} \\ \text{The, red} \\ \text{The, cat} \end{array} \right\}$



자주 등장하는 단어(The)는 여러 번 업데이트 진행
하지만, 좋은 임베딩을 위해서는
여러 단어들이 균등하게 업데이트 되는 것이 중요

3

언어 모델을 활용한 임베딩

- Word2Vec

The brown fox

The red fox

→ 서브 샘플링!

자주 등장하는 단어들은

업데이트를 몇 번 스킵

자주 등장하는 단어(The)는 여러 번 업데이트 진행

하지만, 좋은 임베딩을 위해서는

여러 단어들이 균등하게 업데이트 되는 것이 중요

- Word2Vec

서브 샘플링

자주 등장하는 단어들은 몇 번은 업데이트하지 않고 그냥 지나가
단어들이 균일하게 업데이트 되도록 하는 방법

$$P_{\text{subsampling}}(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

$P_{\text{subsampling}}(w_i)$: 해당 단어를 학습에 이용할지, 그냥 지나갈지 결정하는 확률
 t 가 커질수록 서브 샘플링이 약하게 일어남

- Word2Vec

Summary

Word2Vec의 학습은 **행렬 연산**이 전부 !

두 단어 벡터의 내적을 Sigmoid를 통해 예측 값을 산출

- Word2Vec

Summary

Word2Vec의 학습은 행렬 연산이 전부 !

두 단어 벡터의 내적을 Sigmoid를 통해 예측 값을 산출

두 단어의 유사한 정도를 이용해

주변 단어인지 아닌지 예측하는 모델

• FastText

FastText

문자 단위의 N-gram

1

삼성전자



삼

성

전

자

2

중심단어

주변단어

삼성전자

〈삼성
삼성전
성전자
전자〉

• FastText

영어/중국어

단어 단위(공백)로 문장을 이룸
시제, 의지의 표현이 단어의 변용을 통해 이루어짐

I am eating dinner

I will eat dinner

한국어/일본어

단어 + 조사로 문장을 이룸
시제, 의지의 표현이 접두사, 어간, 접미사 등으로 변함

나는 저녁을 먹고 있어

나는 저녁을 먹을 거야



먹었다 = 먹- + -었 + 다

• FastText

영어/중국어

단어 단위(구백)로 문장을 이룸

I am eating dinner

시제, 의지의 표현이 단어의 변용을 통해 이루어짐

I will eat dinner

한국어 임베딩은 Word2Vec보다 FastText가 더 잘 된다!

한국어/일본어

단어 + 조사로 문장을 이룸

나는 저녁을 먹고 있어

시제, 의지의 표현이 접두사, 어간, 접미사 등으로 변함

나는 저녁을 먹을 거야

 **삼성전자 = 삼성 + 전자**