

Word2Vec - 주론 기반 기법

(기존) 통계 기반 기법의 문제점

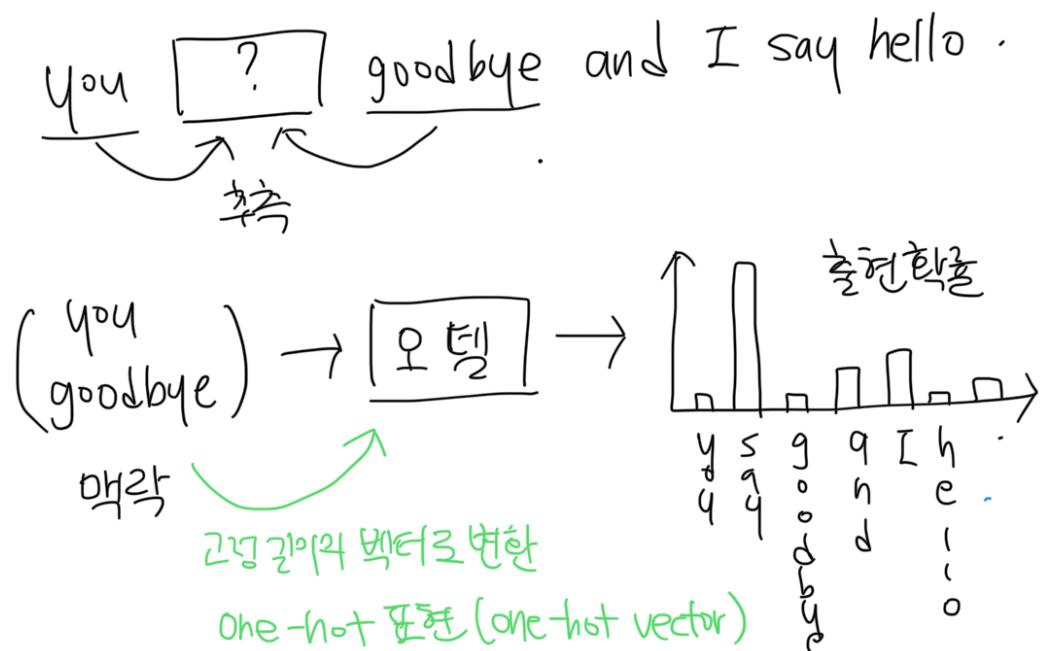
- 주변 단어의 빈도를 기준으로 단어를 표현함

$C \rightarrow SVD \rightarrow$ 일집벡터 (단어의 분산 표현)

$n \times n$ 행렬에 적용하는 비용 $O(n^3)$

주론 기반 기법

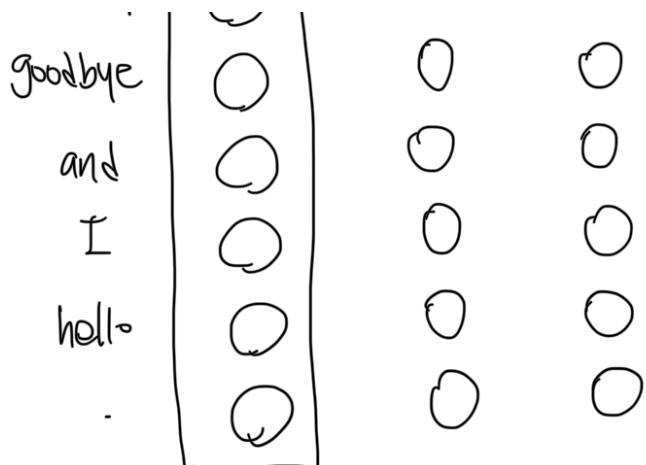
- 신경망에서 이니배치를 이용 (소량의 샘플을 갖고 가중치 갱신)



$$\begin{pmatrix} \text{you} \\ \text{goodbye} \end{pmatrix} \quad \begin{pmatrix} 0 \\ 2 \end{pmatrix} \quad \begin{pmatrix} (1, 0, 0, 0, 0, 0, 0) \\ (0, 0, 1, 0, 0, 0, 0) \end{pmatrix}$$

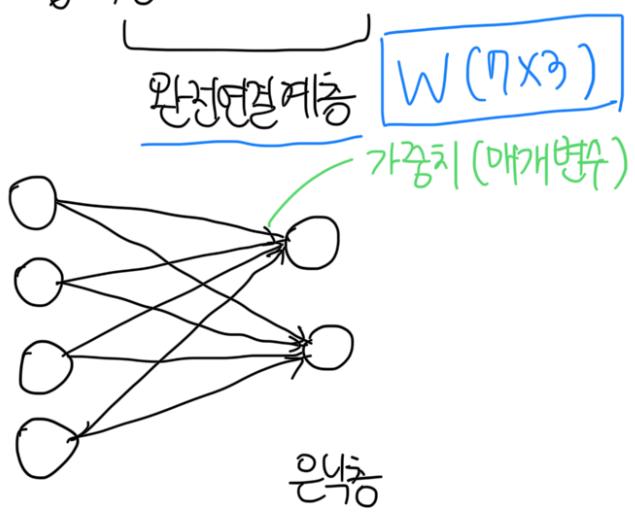
단어(텍스트) 단어 ID one-hot vector
뉴런수를 고정할수 있게됨





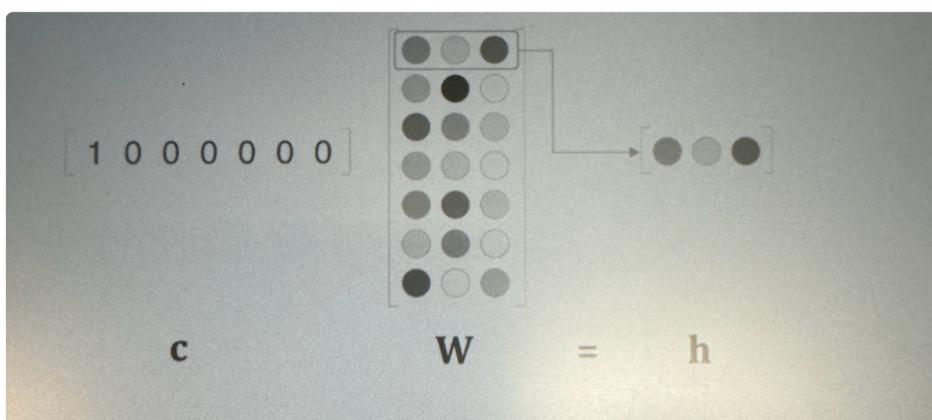
입력층

여기서는 입력층 7개, 은닉층 3개로 준비.



입력층

입력층 뉴런과의 가중합 = 은닉층 뉴런



Word2vec - CBOW (continuous bag-of-words) 모델

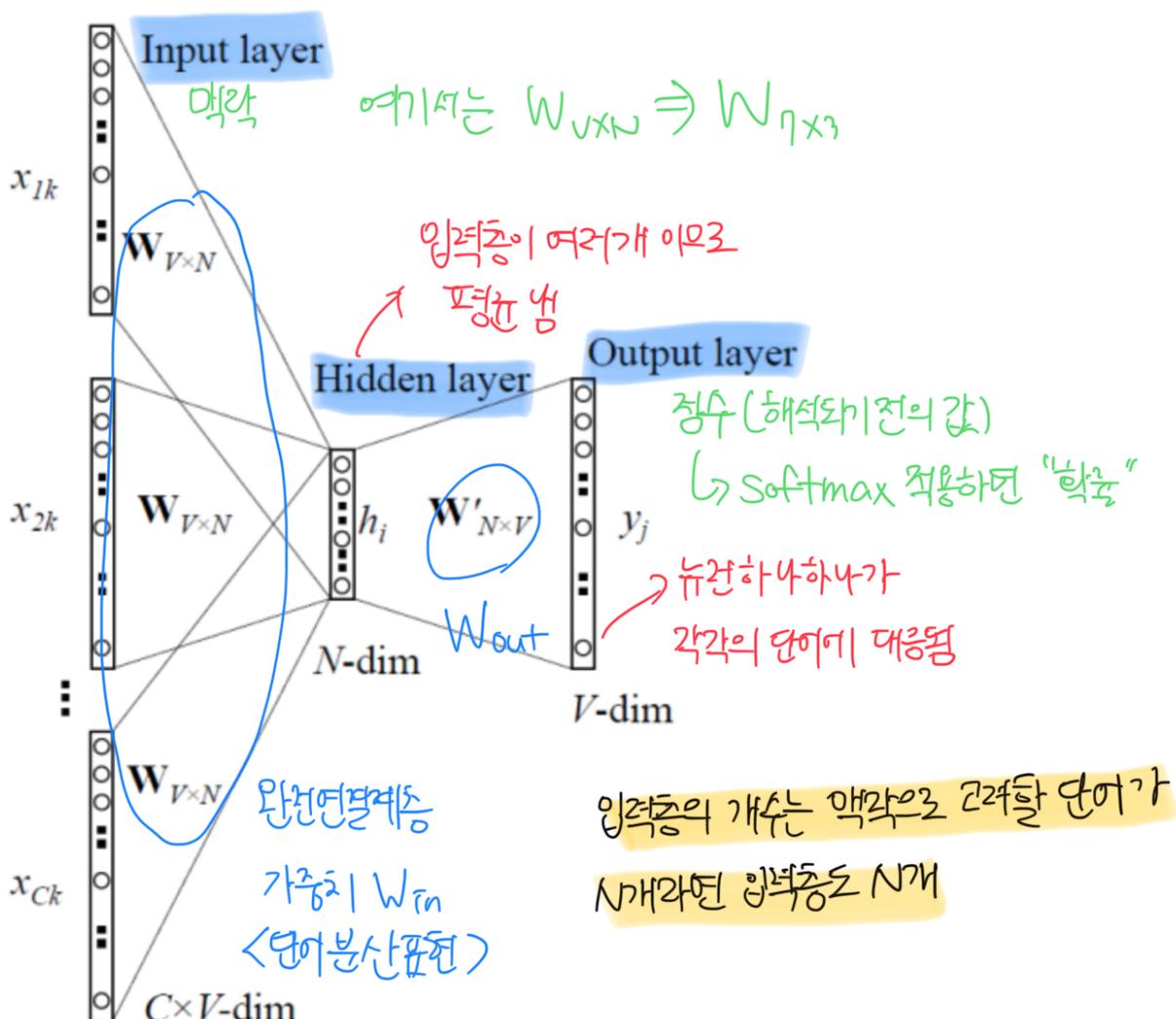
CBOW 모델

- 맥락으로부터 타깃을 추측하는 용도의 신경망

주변단어 중앙단어

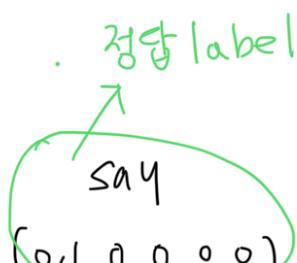
- Input: 맥락 ex) "you", "goodbye" 같은 단어리스트

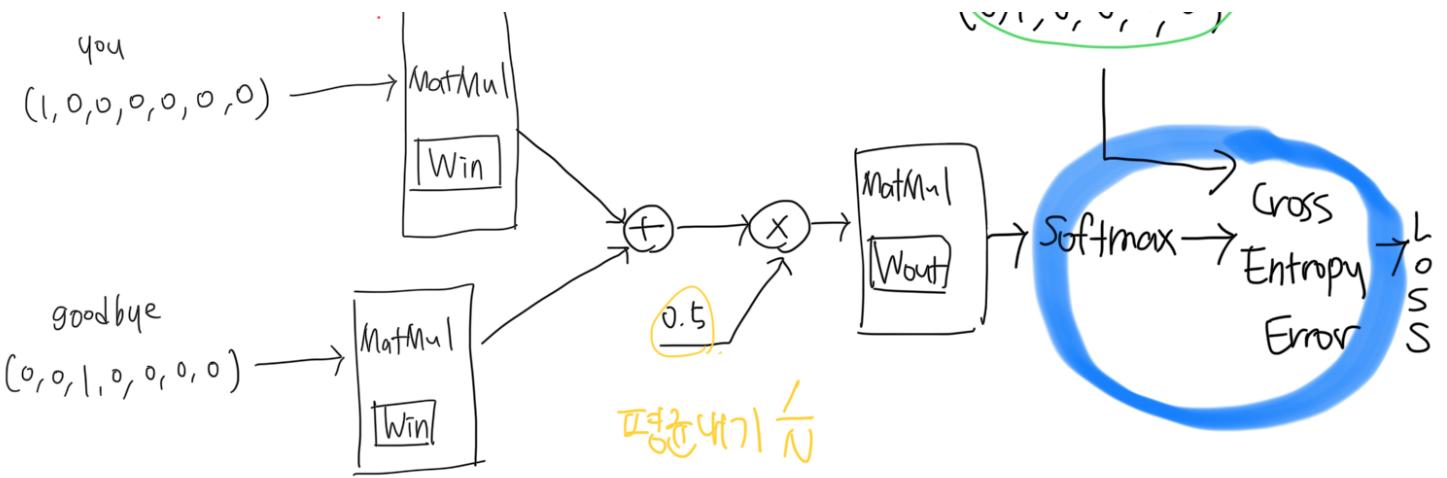
① 맥락을 one-hot 표현으로 변환해 CBOW 모델이 처리 가능하도록 함



* 은닉층의 뉴런 수를 입력층보다 적게 해야 함

단어여록에 필요한 정보만 담아야 밀집벡터 표현을 얻음





Win과 Wout에 단어의 출현 패턴을 파악한 벡터가 학습됨

다중클래스 분류를 수행하는 신경망 - Softmax, 교차엔트로피 → 손실계산.

점수 → 학률 학률
정답레이블)

※ 과정 ※

1. 입력 단어들의 one-hot vector

'you', 'goodbye' → one-hot vector 변환 → Win 행렬곱 → 임베딩 벡터

2. 입력 단어들의 평균 계산

두 단어의 임베딩 벡터를 더하고 $\frac{1}{2} = 0.5$ 를 곱해 하나의 벡터로 변환

3. 출력 벡터 계산

평균 벡터 Wout 행렬과 곱하고 나서 Softmax 함수 적용해서

단어 학률 분포로 변환

4. 정답과 비교해서 손실 계산

실제 정답 say의 one-hot vector (0, 1, 0, 0, 0, 0) 를

교차엔트로피 오차 함수에 입력해 예측 학률과 비교후

손실 계산

$$\Gamma \text{Win의 크기} = 7 \times 3$$

W_{out} 의 크기 = 3×7

* 입격학의 W_{in} 만 사용하는 이유

- 단어의 진짜 의미를 갖고 있기 때문
- 단어의 의미가 압축된 형태로 표현 \Rightarrow Word Embedding 이 학습됨
- W_{out} 가 갖치는 단어 분포를 예측하는 용도
즉, 단어 임베딩이 아니라 Softmax를 위한 가중치 행렬로 작동

말뭉치	맥락(contexts)	타깃	..
(0 1 2 3 4 1 5 6) \rightarrow	$\begin{pmatrix} 0 & 2 \\ 1 & 3 \\ 2 & 4 \\ 3 & 1 \\ 4 & 5 \\ 1 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$	
형상: (8, 1)			
여기서는 windowsize = 2	형상: (6, 2)	형상: (6, 1)	

Skip-gram 모델

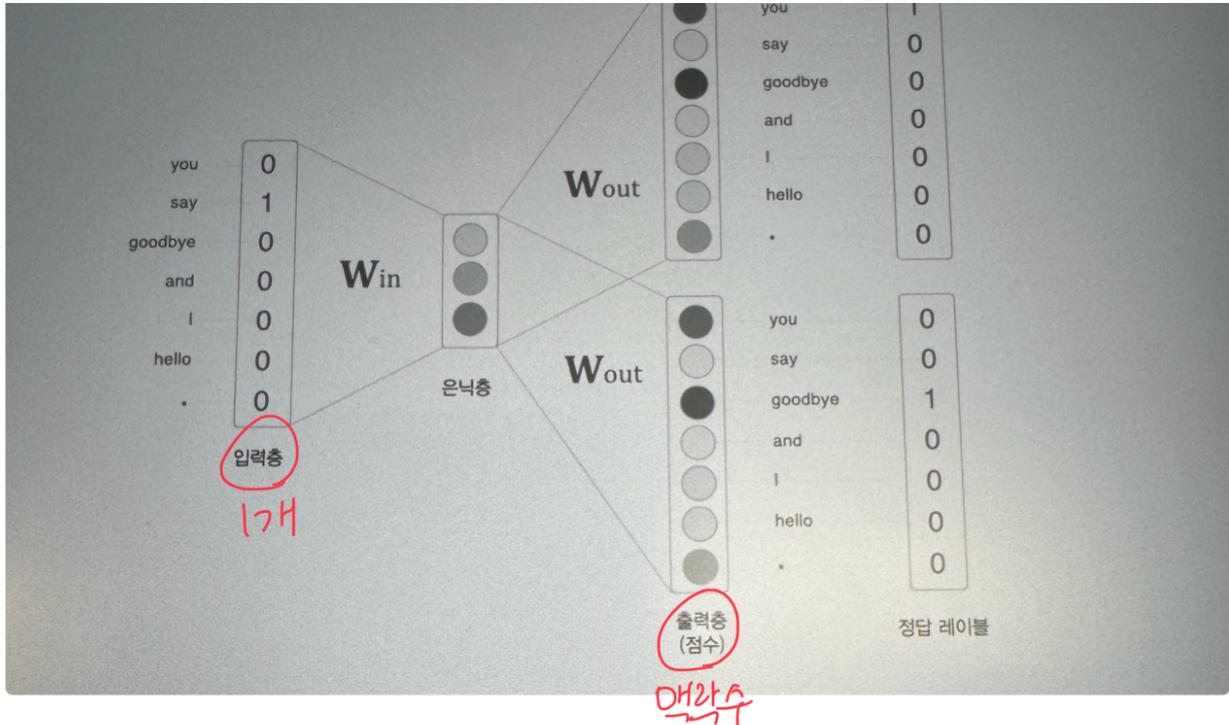
you ? goodbye and I say hello

- CBOW 모델 -

? say ? and I say hello

- skip-gram 모델 -

- 중앙의 단어(타깃)으로부터 주변 여러 단어(맥락) 추측



$$P(w_{t-1}, w_{t+1} | w_t)$$

w_t 가 주어졌을 때 w_t 와 w_{t+1} 이 동시에 일어날 확률

$$= P(w_{t-1} | w_t) P(w_{t+1} | w_t)$$

$$L = -\log P(w_{t-1}, w_{t+1} | w_t)$$

$$= -\log P(w_{t-1} | w_t) P(w_{t+1} | w_t)$$

$$= -(\log P(w_{t-1} | w_t) + \log P(w_{t+1} | w_t))$$

↓ 말뭉치 전체로 확장

$$L = -\frac{1}{T} \sum_{t=1}^T (\log P(w_{t-1} | w_t) + \log P(w_{t+1} | w_t))$$

Skip-gram 학습 과정

① 입력단어 (target) 를 원-핫 벡터로 변환

② 입력벡터를 W_{in} 와 곱해서 embedding vector 를 얻음

- ③ 임베딩 벡터를 W_{out} 과 곱해서 주변단어의 분포 예측
 ④ 예측된 분포와 실제 주변단어의 원-핫 벡터를 비교하여
 고차엔트로피 계산
 ⑤ 역전파를 통해 W_{in} 과 W_{out} 업데이트하여 학습

I
love
natural
 language processing.

Target : love

주변단어 : I, natural

$$I \rightarrow (1, 0, 0, 0, 0)$$

$$love \rightarrow (0, 1, 0, 0, 0)$$

$$natural \rightarrow (0, 0, 1, 0, 0)$$

$$W_{in} = \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 1.0 & 1.1 & 1.2 \\ 1.3 & 1.4 & 1.5 \end{pmatrix} \quad W_{out} = \begin{pmatrix} 0.2 & 0.3 & 0.5 & 0.7 & 0.9 \\ 0.1 & 0.4 & 0.6 & 0.8 & 1.0 \\ 0.3 & 0.5 & 0.7 & 0.9 & 1.1 \end{pmatrix}$$

크기: 5x3 크기: 3x5

$$V_{love} = (0 \ 1 \ 0 \ 0 \ 0) \cdot W_{in} = (0.4 \ 0.5 \ 0.6)$$

$$\text{Score} = V_{love} \cdot W_{out} = \begin{pmatrix} 0.2 & 0.3 & 0.5 & 0.7 & 0.9 \\ 0.1 & 0.4 & 0.6 & 0.8 & 1.0 \\ 0.3 & 0.5 & 0.7 & 0.9 & 1.1 \end{pmatrix}$$

$$\text{score} = (0.31 \ 0.62 \ 0.92 \ 1.22 \ 1.52)$$

Softmax 5%

$$e^{0.31} + e^{0.62} + e^{0.92} + e^{1.12} + e^{1.52} = 13.69 \rightarrow 13.69$$

$$\Rightarrow P_1 = \frac{1.31}{13.69} = 0.099 \quad \text{설명}$$

정답

$$P_2 = \frac{1.86}{13.69} = 0.136$$

$$P_3 = \frac{2.51}{13.69} = 0.183$$

$$P_4 = \frac{3.39}{13.69} = 0.247$$

$$P_5 = \frac{4.51}{13.69} = 0.335$$