chapter 1은 이미 1편에서 충분히 공부했으므로 생략

2,3,4장의 언어처리는 5장 이후에 예제로 쓰이므로 가볍게 공부

RNN이 자연어 처리만을 위한 신경망 구조는 아니고 언어처리는 하나의 예에 불과함, 본 강좌는 신경망 구현에 집중

chapter 2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

우리의 말을 컴퓨터에게 이해시키기 위한 기술

2.1.1 단어의 의미 : 의미의 최소단위

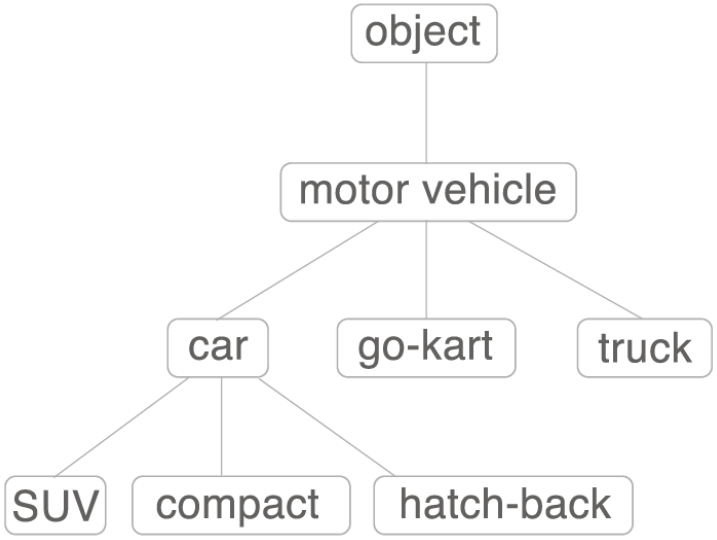
- 시소러스를 활용한 기법
- 통계 기반 기법
- 추론 기반 기법 (word2vec) : 다음 장

2.2 시소러스: 유의어 사전, 뜻이 같거나(동의어), 비슷한(유의어)가 한 그룹으로 분류

사람이 직접 단어의 의미를 정의

그림 2-1 동의어의 예: “car”, “auto”, “automobile” 등은 “자동차”를 뜻하는 동의어다.

그림 2-2 단어들을 의미의 상 · 하위 관계에 기초해 그래프로 표현한다(문헌 [14]를 참고하여 그림).



모든 단어에 대한 유의어 집합과 단어들의 관계를 그래프로 표현한 단어 네트워크를 이용하여 자연어 처리

note: car에 대한 검색을 automobile의 검색결과도 포함

2.2.1 WordNet : 프린스턴 대학에서 1985년부터 구축한 시소러스

2.2.2 시소러스의 문제점

- 시대 변화에 대응하기 어려움: 신조어, 의미변화
- 작업 인건비 높음
- 단어의 미묘한 차이 표현 안됨: retro, vintage

→ 사람의 개입을 줄이는 방향으로 자연언어 처리 개발

2.3 통계 기반 기법 (***)

RNN과 직접적인 관련 적으므로, 빠른 수업 진행을 위해 생략

chapter 3 word2vec

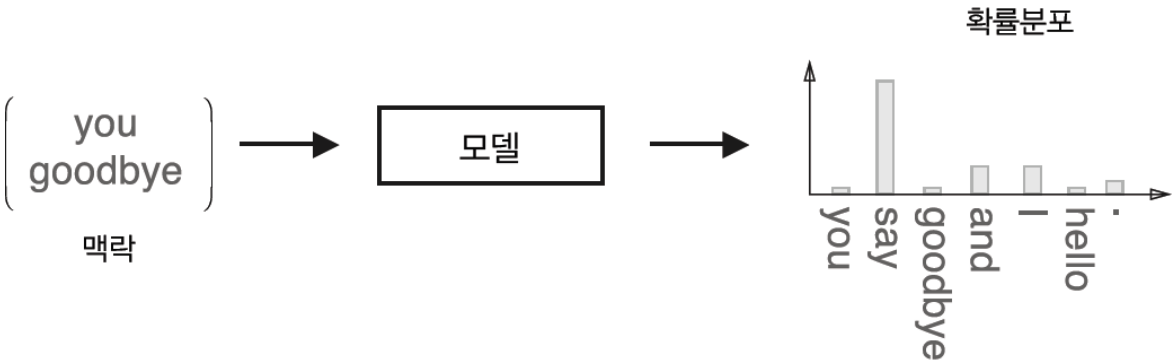
단어의 분산표현 → word를 vector로 표현

3.1.2 추론 기반 기법 개요

그림 3-2 주변 단어들을 맥락으로 사용해 "?"에 들어갈 단어를 추측한다.

이러한 과정 통하여 각 단어에 적절한 벡터 표현을 찾는 것이 목적

그림 3-3 추론 기반 기법: 맥락을 입력하면 모델은 각 단어의 출현 확률을 출력한다.



3.1.3 신경망에서의 단어 처리

one-hot vector → distributed representation

그림 3-4 단어, 단어 ID, 원핫 표현

단어(텍스트)	단어 ID	원핫 표현
you goodbye	0 2	(1, 0, 0, 0, 0, 0, 0) (0, 0, 1, 0, 0, 0, 0)

그림 3-5 입력층의 뉴런: 각 뉴런이 각 단어에 대응(해당 뉴런이 1이면 검은색, 0이면 흰색)

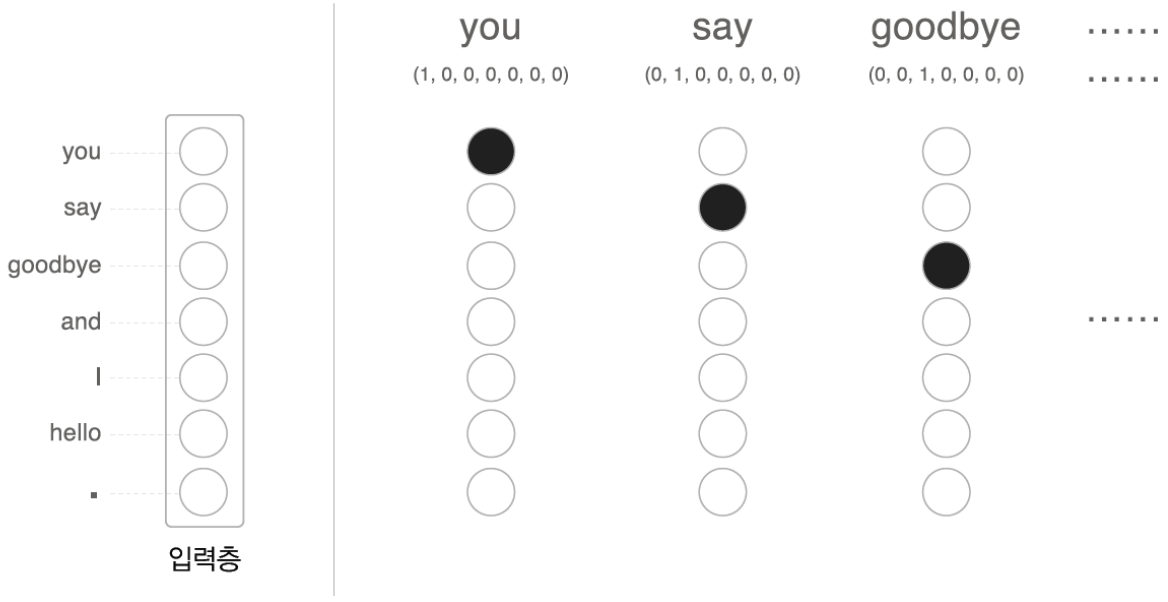
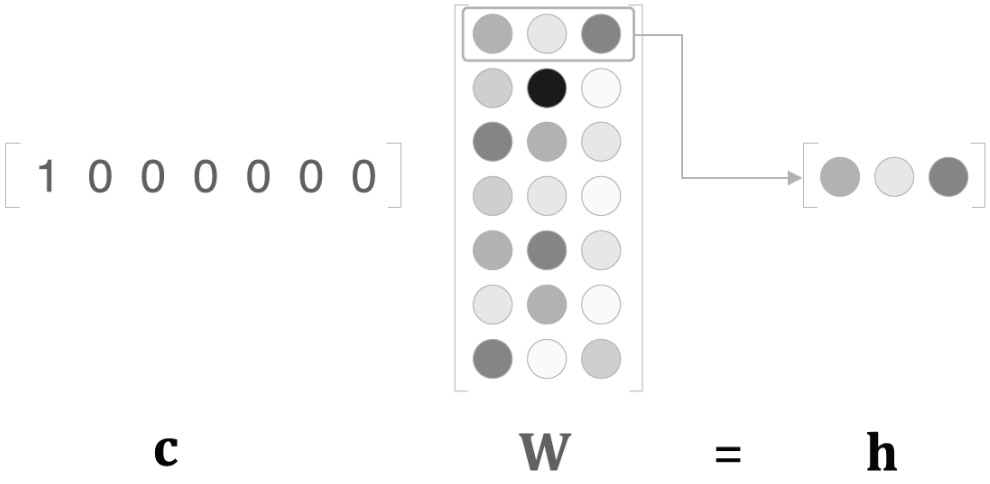


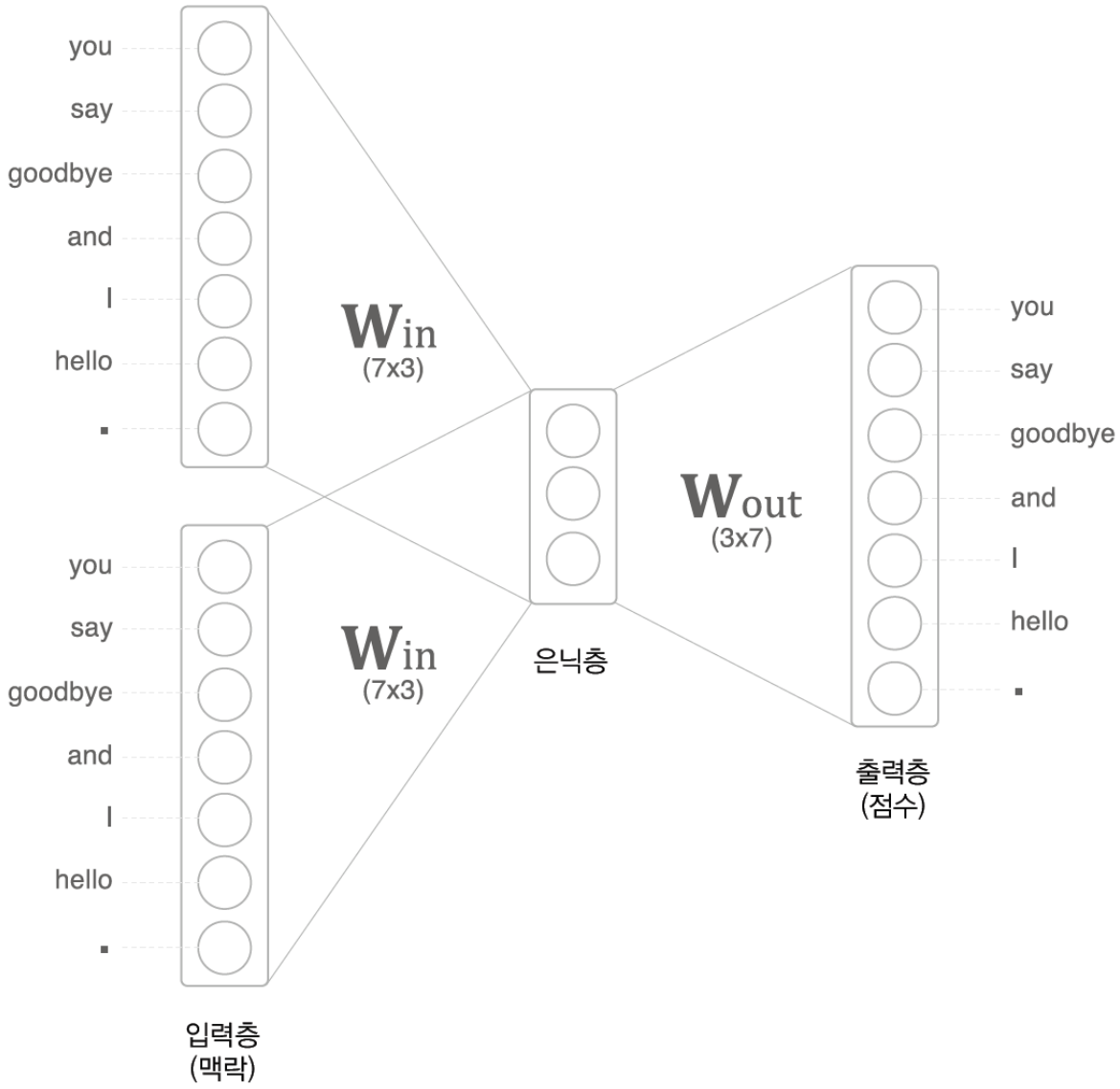
그림 3-8 맥락 **c**와 가중치 **W**의 곱으로 해당 위치의 행벡터가 추출된다(각 요소의 가중치 크기는 흑백의 진하기로 표현).



3.2 단순한 word2vec

3.2.1 CBOW(continuous back-of-words) 모델의 추론 처리

그림 3-9 CBOW 모델의 신경망 구조

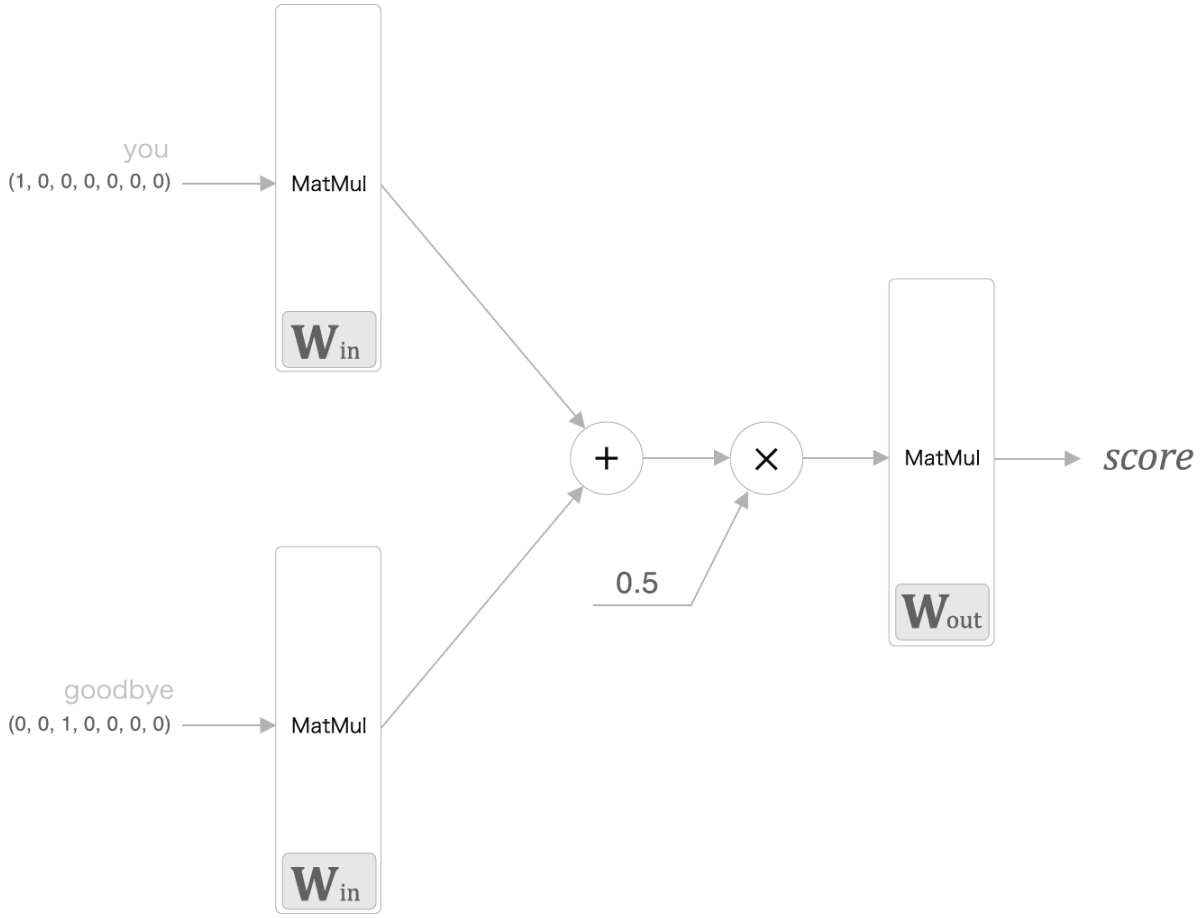


가중치 W_in의 각행이 해당 단어의 분산 표현

W_in은 학습을 통해 맥락에서 출현하는 단어를 잘 추측하도록 갱신됨

one-hot vector의 차원은 전체 단어수 → 분산 표현으로 차원 감소

그림 3-11 계층 관점에서 본 CBOW 모델의 신경망 구성: MatMul 계층에서 사용하는 가중치($\mathbf{W}_{in}$, $\mathbf{W}_{out}$)는 해당 계층 안으로 넣었음

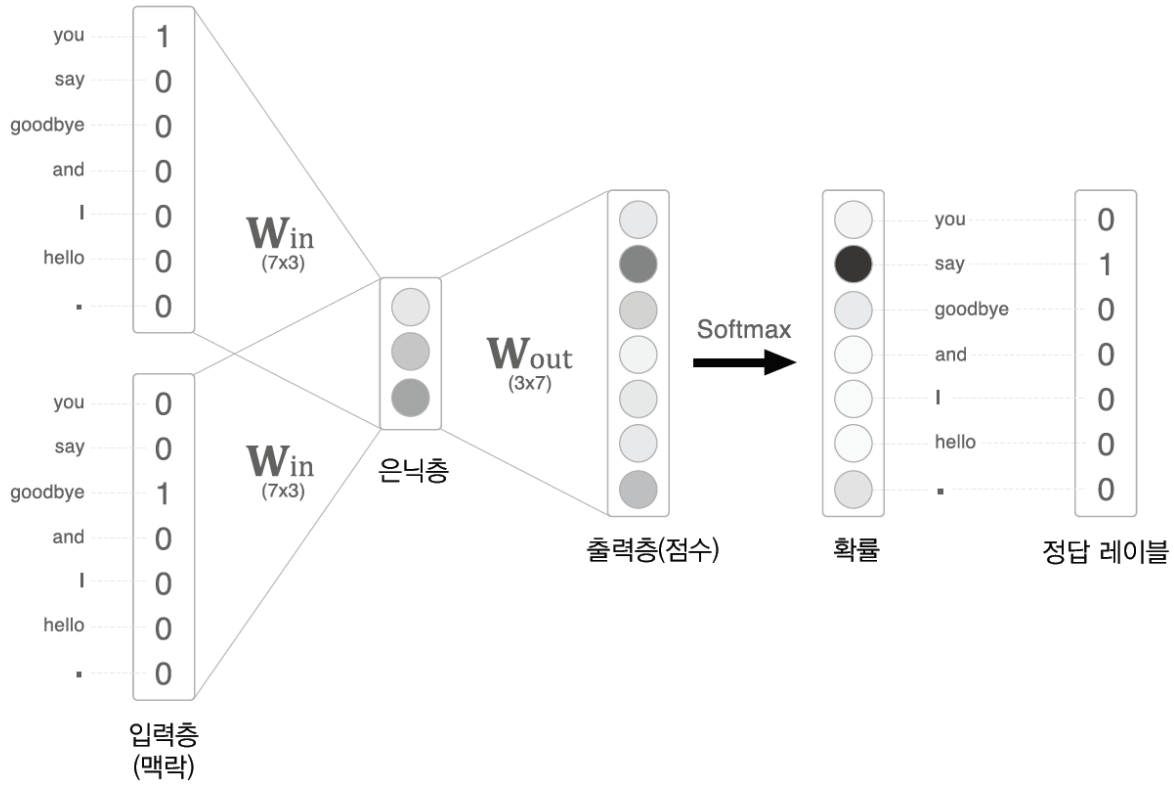


ch03/cbow_predict.py 보라

활성화함수를 사용하지 않는 간단한 구조

가중치 Win을 공유한다는 점이 기존과는 다른 점

그림 3-12 CBOW 모델의 구체적인 예(노드 값의 크기를 흑백의 진하기로 나타냄)



맥락 you, goodbye 일때 say로 예측되야 함

그림 3-13 CBOW 모델의 학습 시 신경망 구성

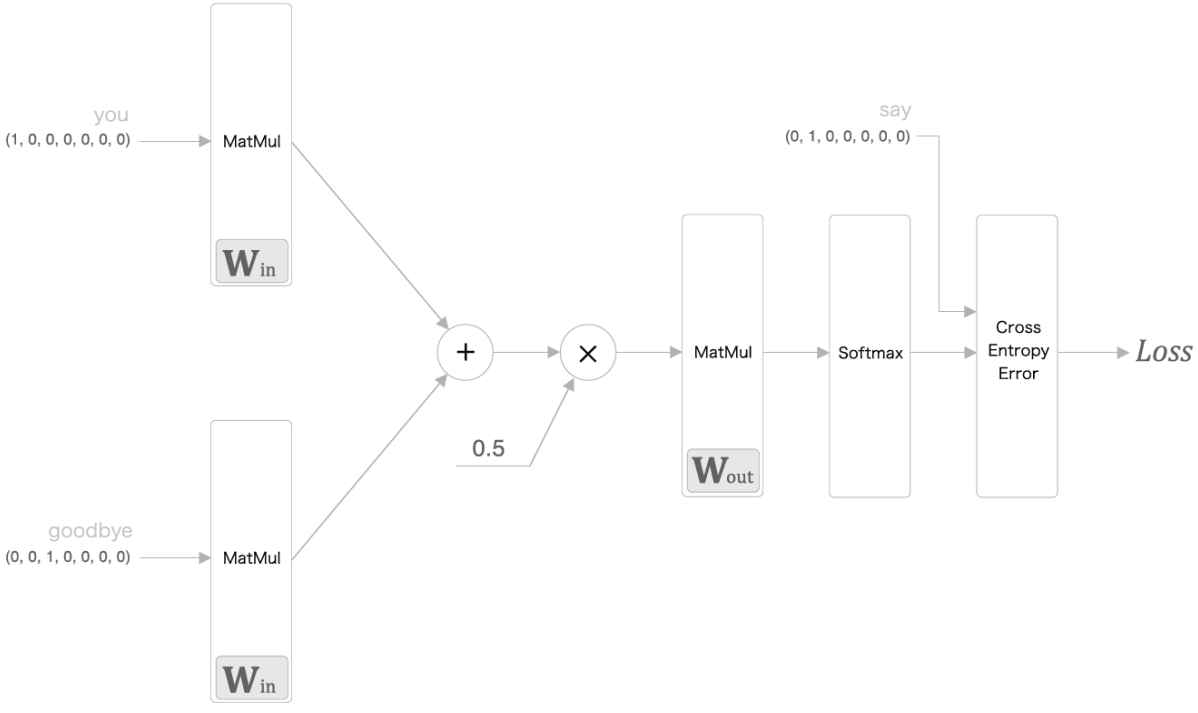
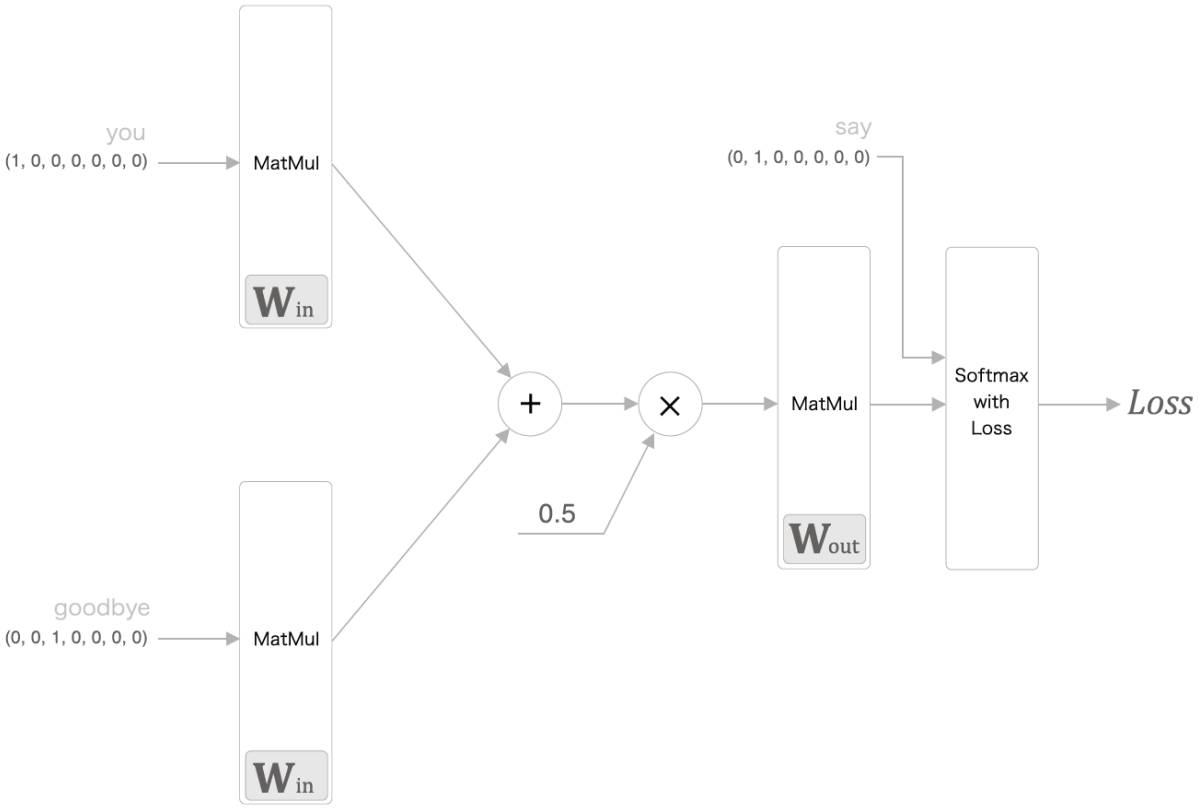


그림 3-14 Softmax 계층과 Cross Entropy Error 계층을 Softmax with Loss 계층으로 합침



3.2.3 word2vec의 가중치와 분산 표현

W_in 만 사용 : word2vec

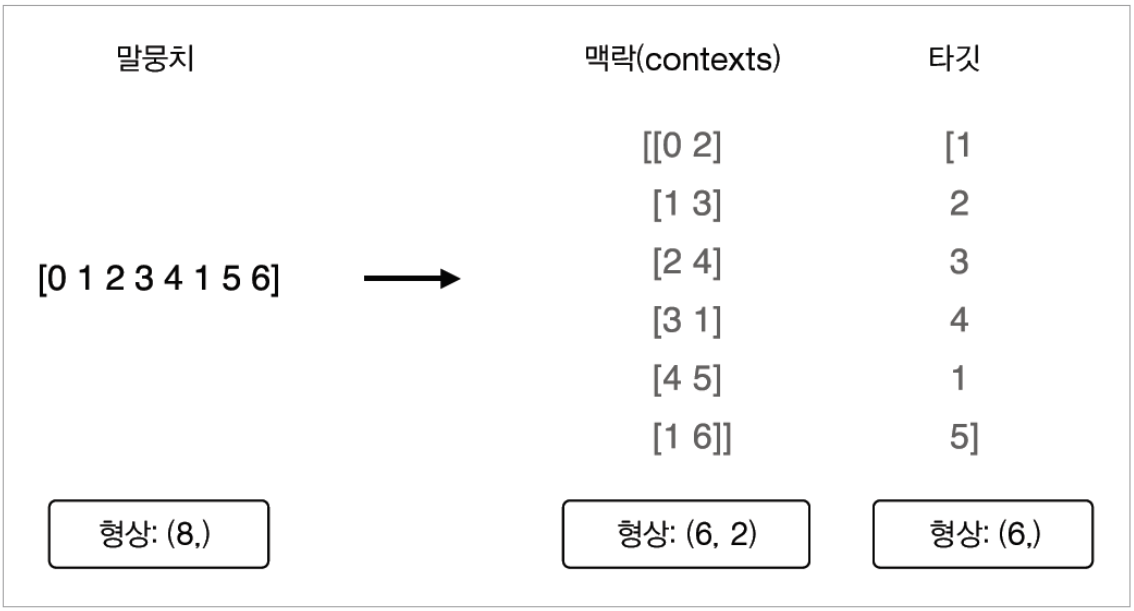
W_in, W_out : GloVe

3.3 학습 데이터 준비

그림 3-16 말뭉치에서 맥락과 타깃을 만드는 예

common/util.py 의 preprocess 읽기

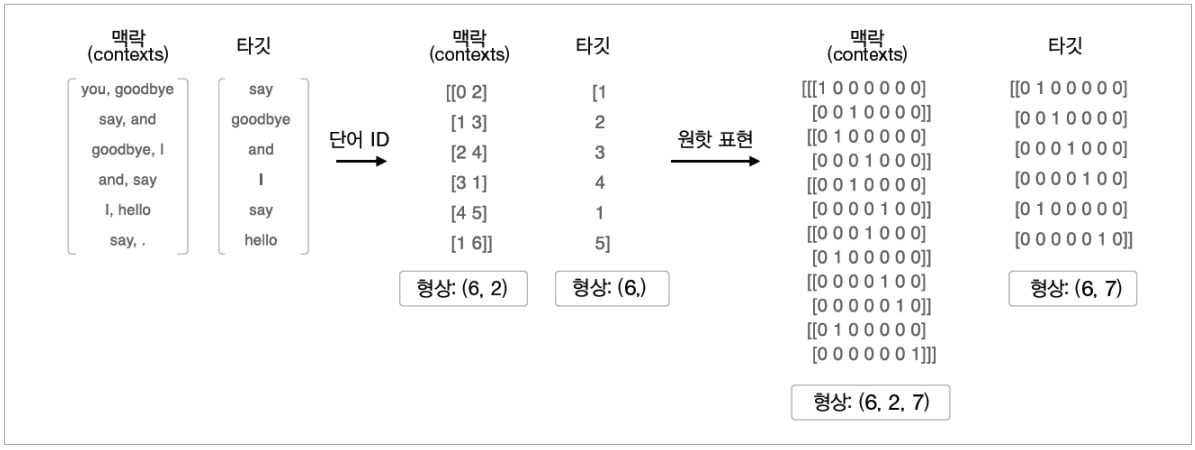
그림 3-17 단어 ID의 배열인 corpus로부터 맥락과 타깃을 작성하는 예(맥락의 윈도우 크기는 1)



common/util.py 의 create_contexts_target 읽기

3.3.2 원핫 표현으로 변환

그림 3-18 '맥락'과 '타깃'을 원핫 표현으로 변환하는 예



common/util.py 의 convert_one_hot

3.4 CBOW 모델 구현

그림 3-19 CBOW 모델의 신경망 구성

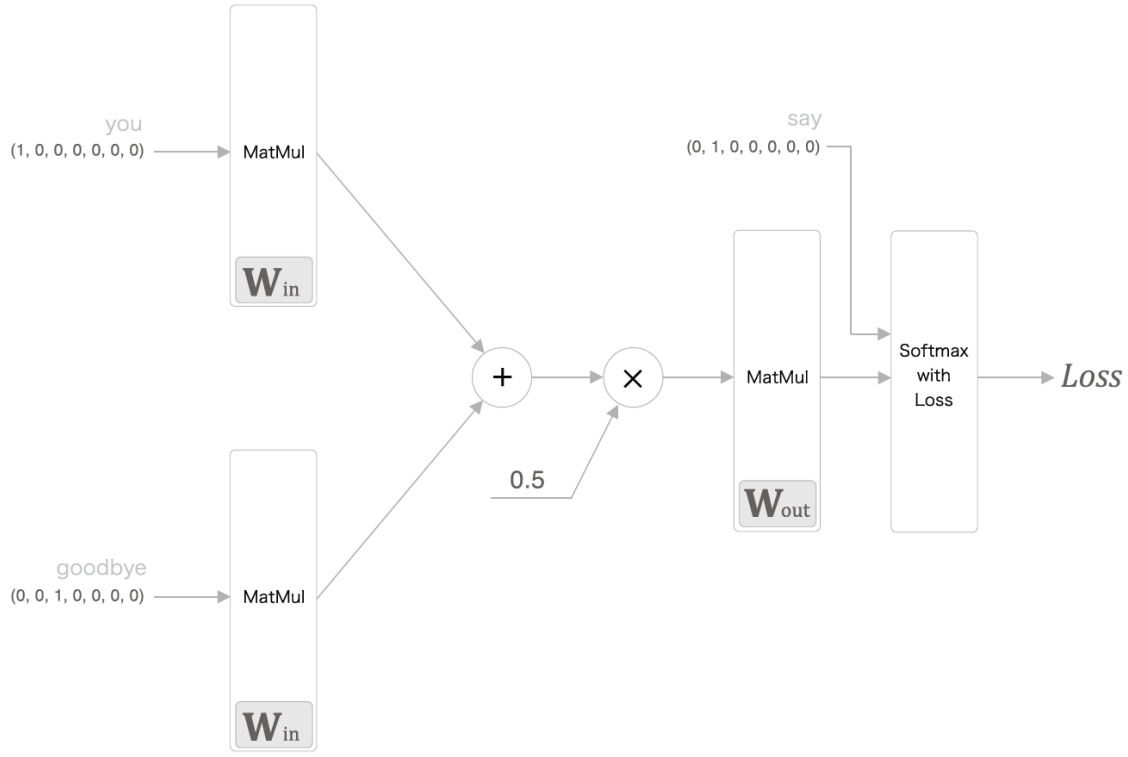
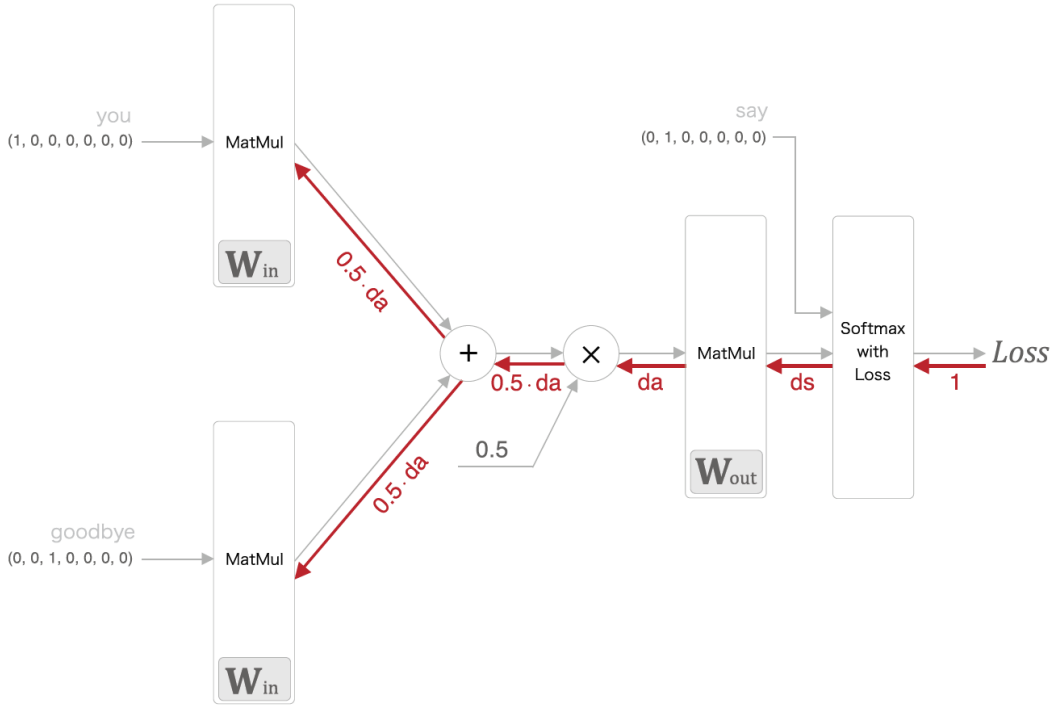


그림 3-20 CBOW 모델의 역전파(역전파의 흐름은 두꺼운(붉은) 화살표로 표시)



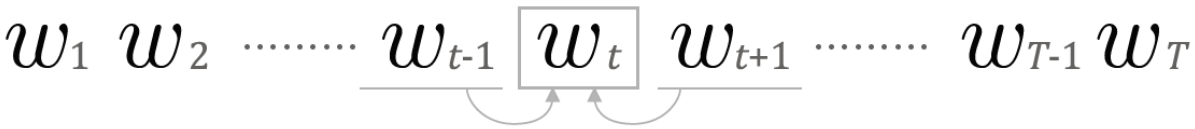
ch03/simple_cbow.py읽기

3.4.1 학습 코드 구현

ch03/train.py 실행 후 읽기 common/trainer.py remove_duplicate 참조 (더함)

3.5 word2vec 보충

그림 3-22 word2vec의 CBOW 모델(맥락의 단어로부터 타깃 단어를 추측)



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

손실함수

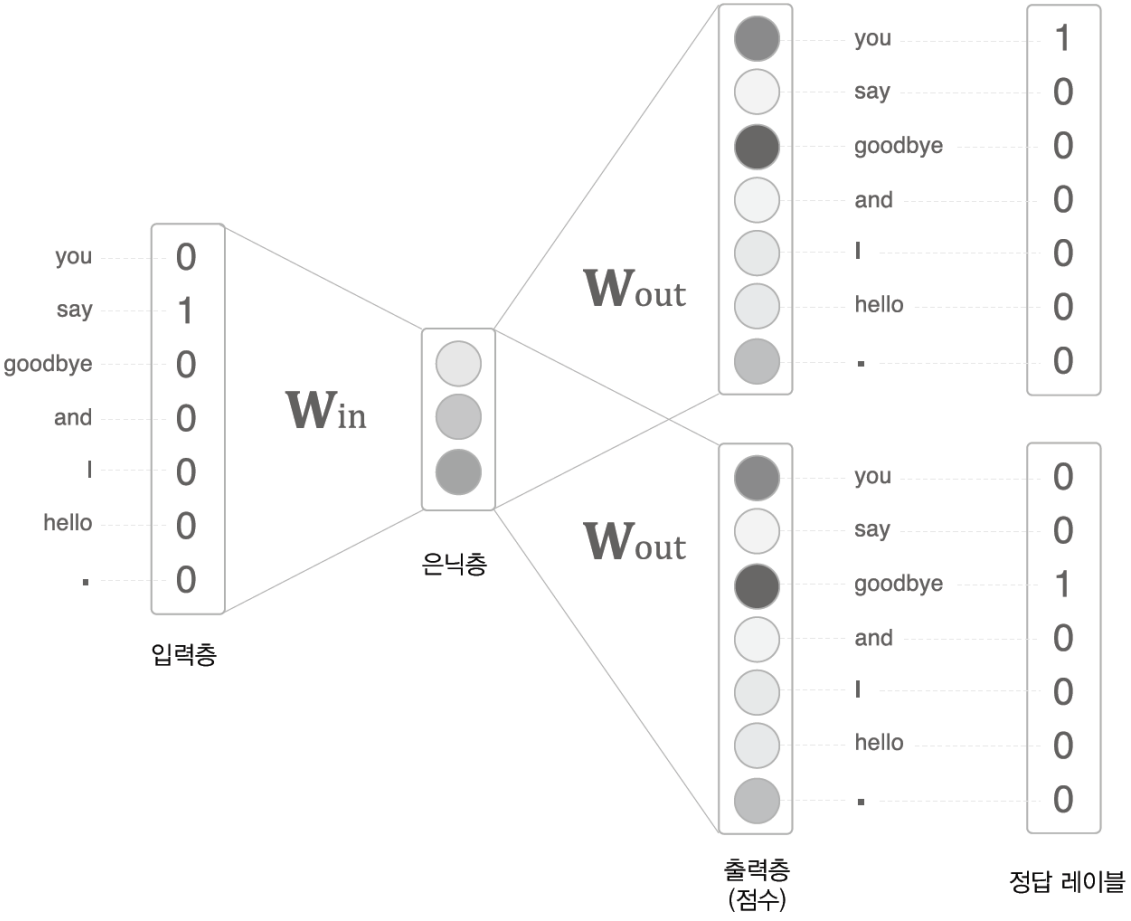
$$L = -\log P(w_t \mid w_{t-1}, w_{t+1})$$

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

3.5.2 skip-gram 모델

그림 3-23 CBOW 모델과 skip-gram 모델이 다루는 문제

그림 3-24 skip-gram 모델의 신경망 구성 예



$P(w_{t-1}, w_{t+1} \mid w_t)$ → 조건부 독립 가정하면

$$P(w_{t-1}, w_{t+1} \mid w_t) = P(w_{t-1} \mid w_t) P(w_{t+1} \mid w_t)$$

$$\begin{aligned} L &= -\log P(w_{t-1}, w_{t+1} \mid w_t) \\ &= -\log P(w_{t-1} \mid w_t) P(w_{t+1} \mid w_t) \\ &= -(\log P(w_{t-1} \mid w_t) + \log P(w_{t+1} \mid w_t)) \end{aligned}$$

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

학습은 CBOW가 빠르나, 성능은 skip-gram 이 우수

chapter 4 생략하나, word2vec의 고속구현 시 필요 다음장들이 더 중요