밑바닥부터 시작하는 딥러닝 2

3장. word2vec

박채원

목차

3.1 추론 기반 기법과 신경망

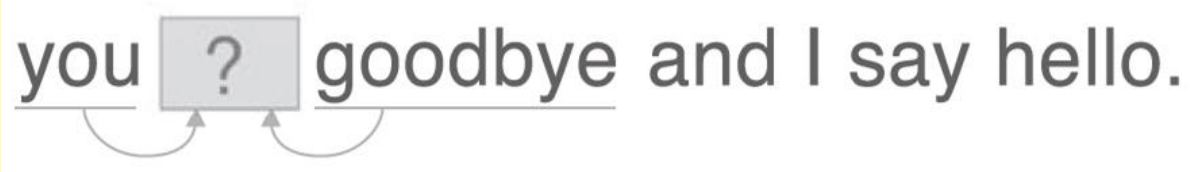
단어를 벡터로 표현하는 방법 <

통계 기반 기법 : 주변 단어의 빈도를 기초로 함. 말뭉치를 다룰때 문제 발생.
추론 기반 기법 : 미니배치로 학습. 소량의 학습 샘플씩을 반복해서 학습.

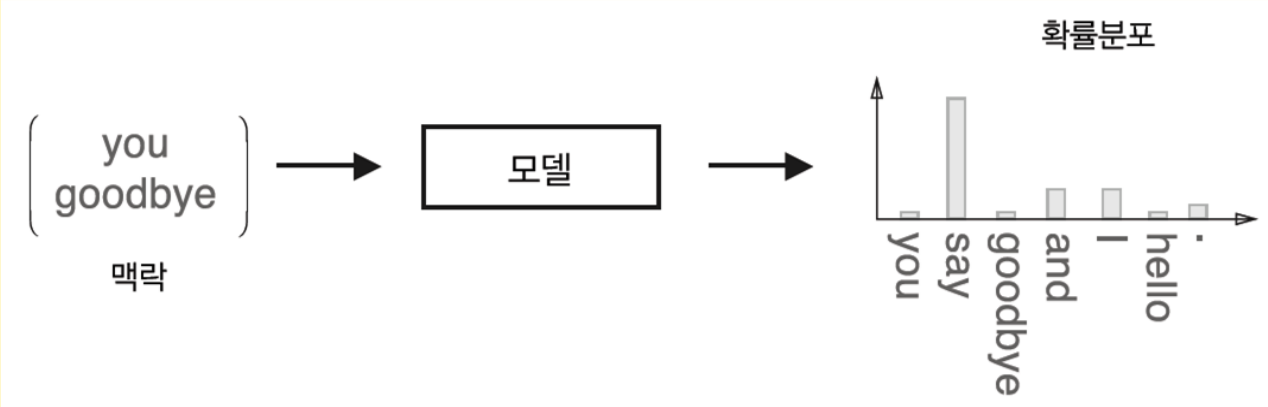
- 통계 기반 기법의 문제점
  - 어휘의 수가 많아지면 통계기반기법에서는 100만x100만 이라는 거대한 행렬을 만들게 된다. 이 행렬에 SVD를 적용하는 일은 현실적이지 않다.
  - SVD를 nxn 행렬에 적용하는 비용은 O(n^3) 이다. 즉 어휘가 많아질수록 비용도 커진다.

3.1 추론 기반 기법과 신경망

- 추론 기반 기법 개요



you와 goodbye 사이 들어갈 단어를 추측



모델 관점에서 본 추론 문제

즉, 단어의 출현 패턴을 학습

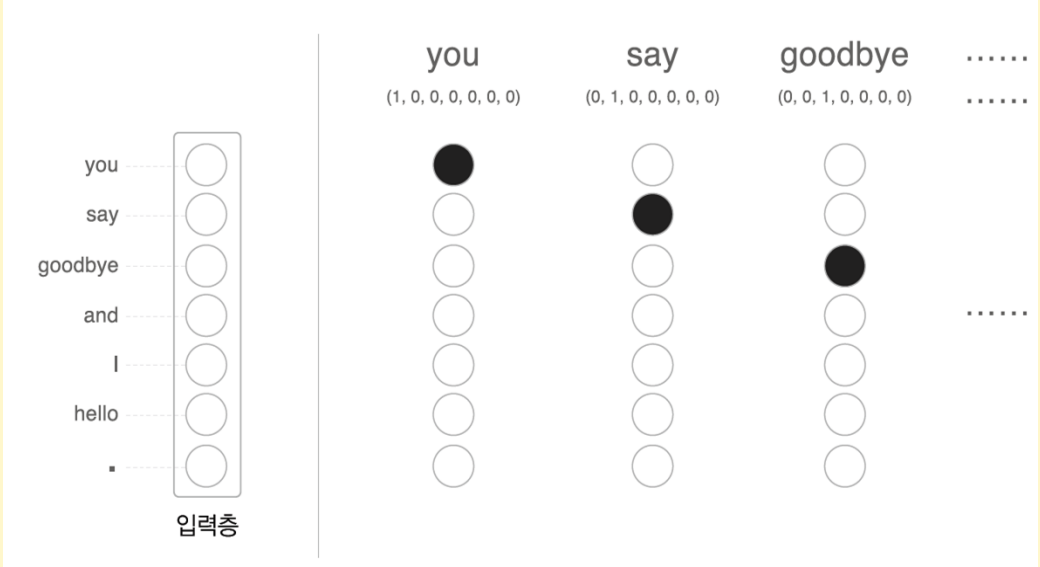
3.1 추론 기반 기법과 신경망

- 신경망에서의 단어 처리

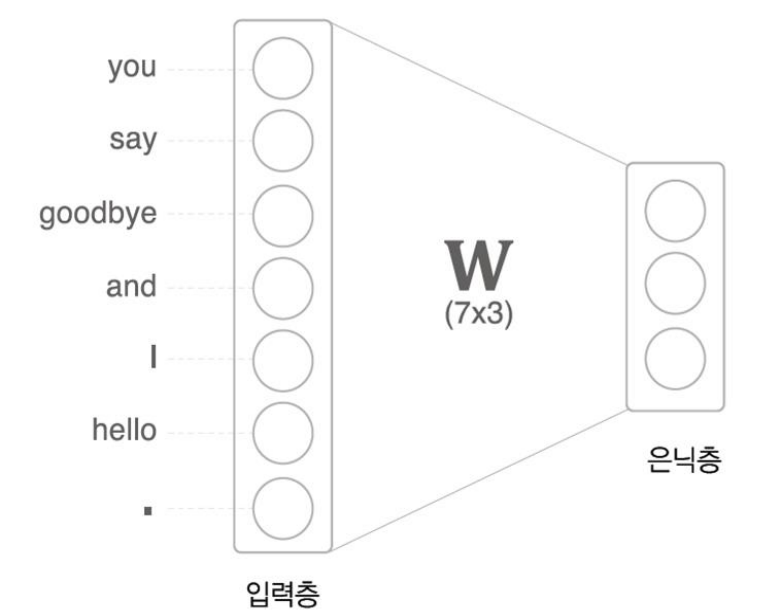
  단어를 있는 그대로 처리할 수 없으니 단어를 '고정 길이의 벡터'로 변환해야 함 -> **원핫 표현**

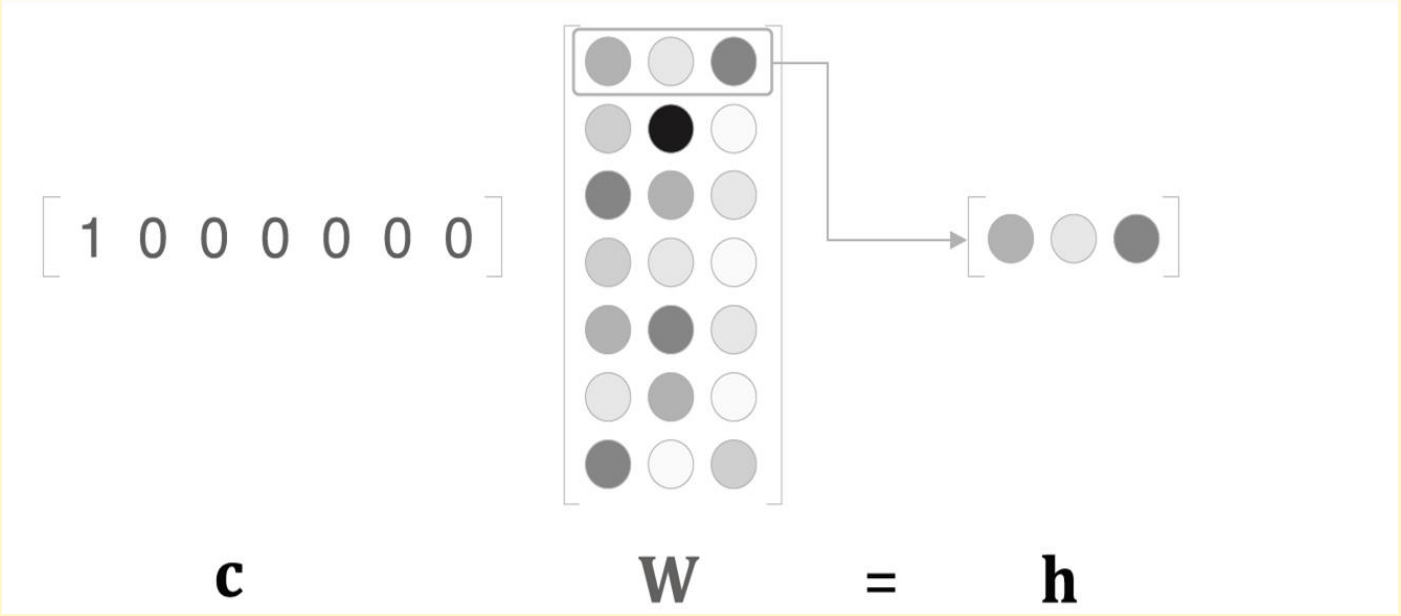
예시 : you say goodbye and I say hello. -> 말뭉치
0 1 2 3 4 5 6 -> 단어 ID

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



3.1 추론 기반 기법과 신경망



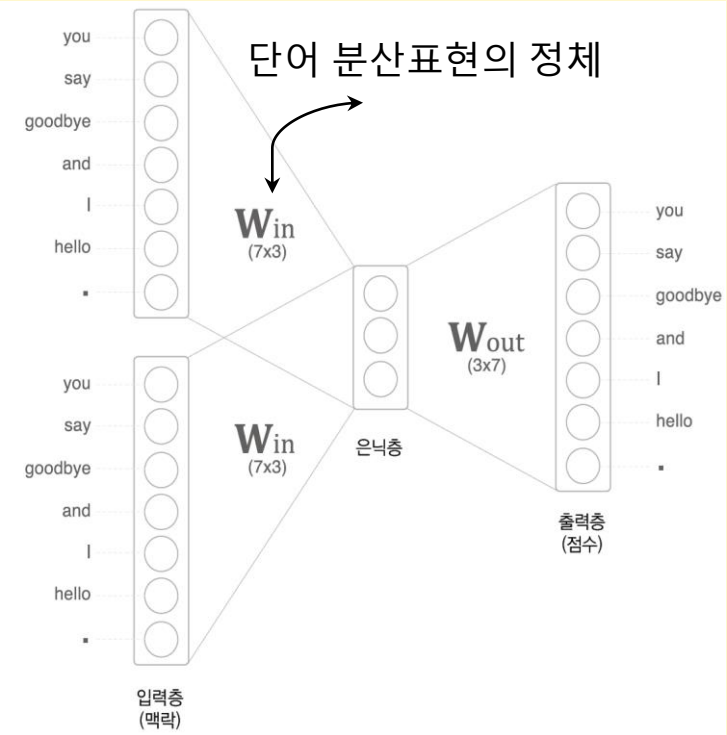


- 입력층 뉴런 총 7개
- 단어를 원핫벡터로 나타낼 수 있다.
- 계층들을 벡터로 처리 가능
- 즉, 단어를 신경망으로 처리가능

즉, 가중치의 행 벡터 하나를 뽑아내는 것과 같다.

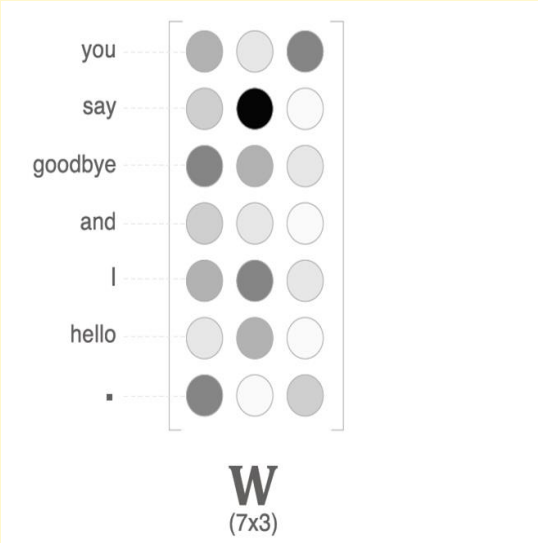
3.2 단순한 word2vec

- CBOW 모델의 추론 처리
  : 맥락(주변 단어)으로부터 타깃(중앙 단어)을 추측하는 용도의 신경망



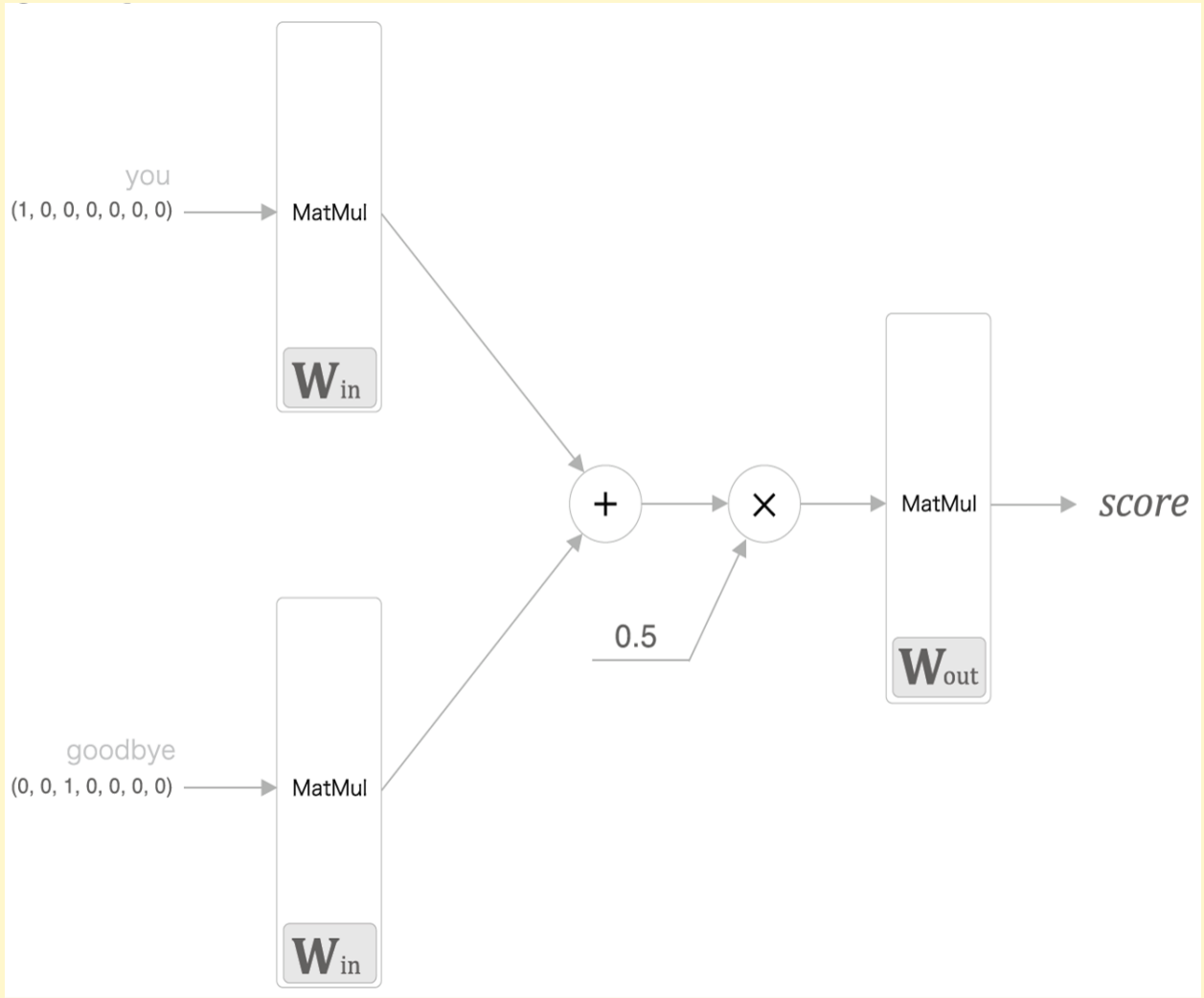
완전연결계층에 의해 첫 번째 입력층이 h_1 으로 변환되고, 두번째 입력층이 h_2 로 변환되었다고 하면 은닉층 뉴런은 $\frac{1}{2}(h_1 + h_2)$이 된다.

가중치
: 각 행에는 해당 단어의 분산 표현이 담겨있다고 볼 수 있다.



여기서 포인트는 은닉층의 뉴런 수를 입력층의 뉴런 수보다 적게 하는 것이 중요한 핵심.

3.2 단순한 word2vec



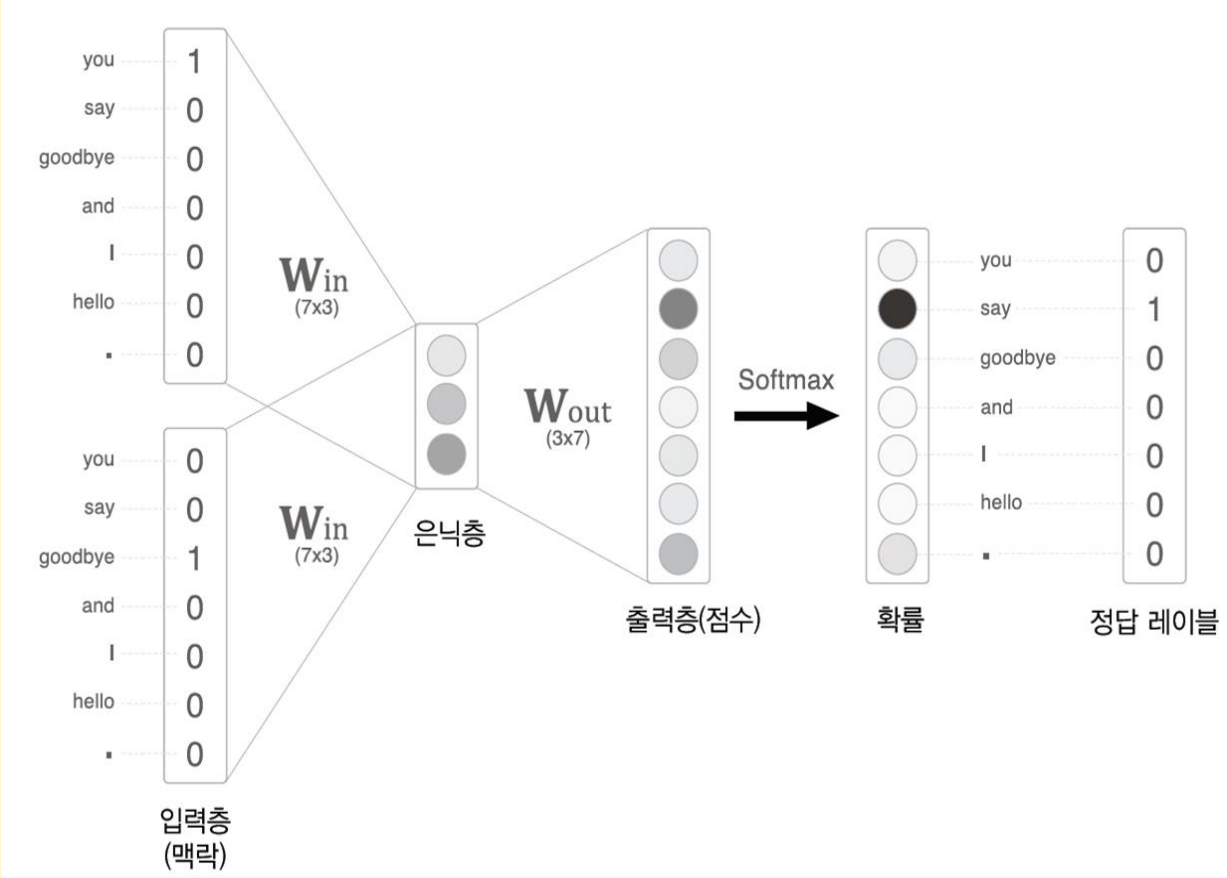
계층 관점에서 본 CBOW 모델의 신경망 구성

```python
import sys
sys.path.append('..')
import numpy as np
from common.layers import MatMul

c0 = np.array([[1, 0, 0, 0, 0, 0, 0]])
c1 = np.array([[0, 0, 1, 0, 0, 0, 0]])

W_in = np.random.randn(7, 3)
W_out = np.random.randn(3, 7)


in_layer0 = MatMul(W_in)
in_layer1 = MatMul(W_in)
out_layer = MatMul(W_out)


h0 = in_layer0.forward(c0)
h1 = in_layer1.forward(c1)
h = 0.5 * (h0 + h1)
s = out_layer.forward(h)
print(s)
```

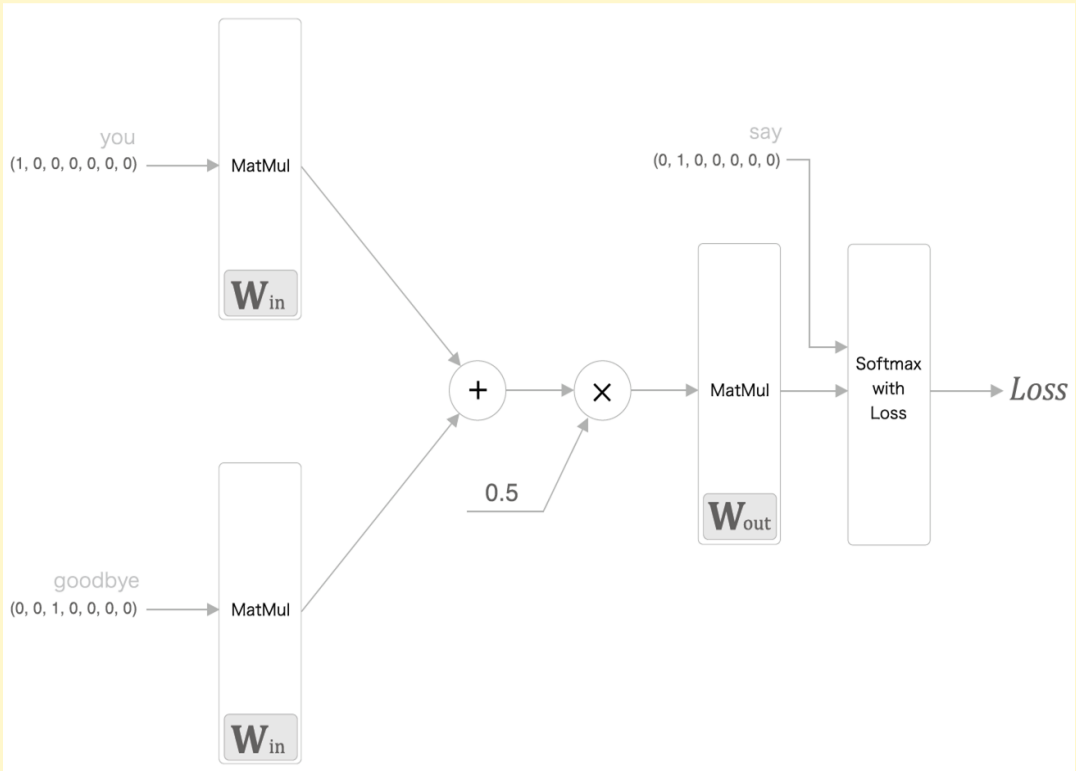
CBOW모델의 추론 처리

3.2 단순한 word2vec

- CBOW 모델의 학습



CBOW 모델의 학습에선
올바른 예측을 할 수 있도록 가중치를
조정하는 일을 한다.



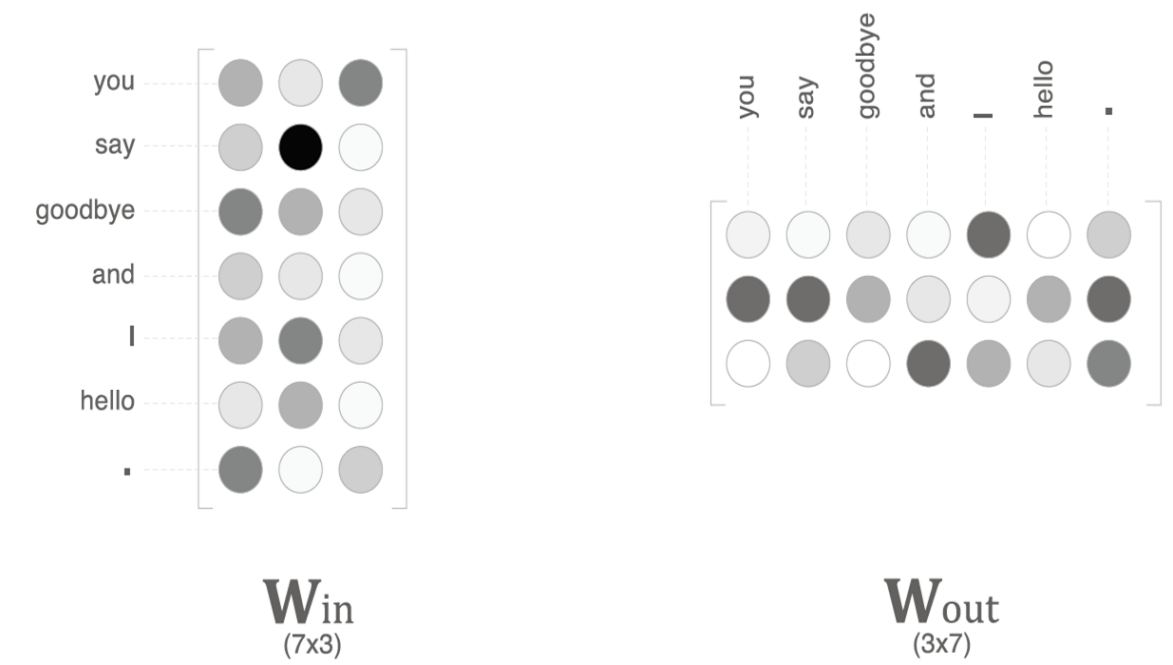
3.2 단순한 word2vec

- word2vec의 가중치와 분산 표현

word2vec에서 사용되는 가중치는 두 가지가 있다.

W_{in} : 입력 측 완전연결계층의 가중치 -> 이 가중치의 각 행이 각 단어의 분산 표현에 해당.
W_{out}: 출력 측 완전연결계층의 가중치 -> 이 가중치에도 단어의 의미가 인코딩된 벡터가 저장되고 있다
.

행 방향



열 방향

*최종적으로 이용하는 단어분산표현은 입력 측 가중치만을 사용한다.

3.3 학습 데이터 준비

- 맥락과 타깃

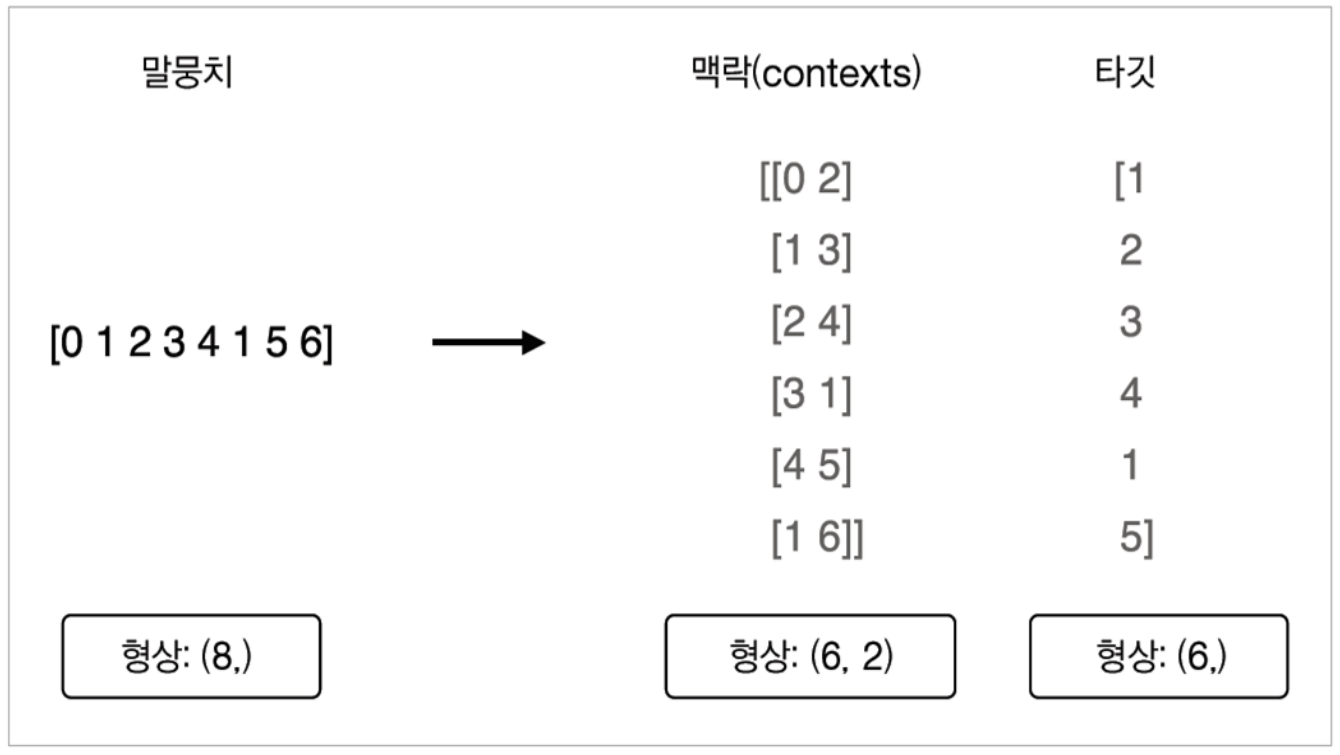
말뭉치 : You say goodbye and I say hello.

타깃 : 말뭉치로부터 목표로 하는 단어
맥락 : 타깃의 주변 단어

양 끝을 제외한 모든 단어에 대해 타깃과 맥락을 뽑아냄

3.3 학습 데이터 준비

- 맥락과 타깃

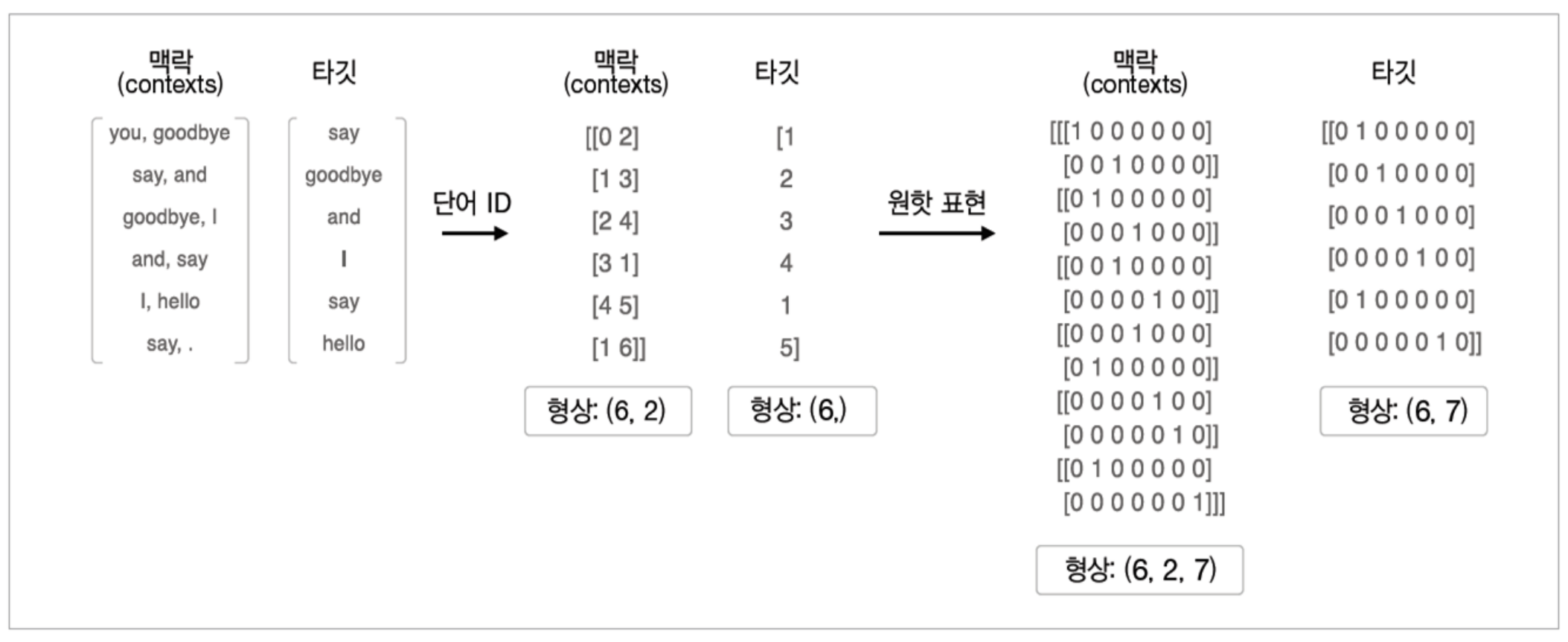


2장에서 구현했던 preprocess() 함수를 이용해서 말뭉치 텍스트를 단어 ID로 변환한다.

그 다음 단어 ID의 배열인 corpus로부터 맥락과 타깃을 만들어냄.

3.3 학습 데이터 준비

- 맥락과 타깃



3.3 학습 데이터 준비

```
import sys
sys.path.append('..')
from common.util import preprocess

text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
print(corpus)

print(id_to_word)
```

▲코드 1. 텍스트 -> ID

```
import sys
sys.path.append('..')
from common.util import preprocess, create_contexts_target, convert_one_hot

text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)

contexts, target = create_contexts_target(corpus, window_size=1)

vocab_size = len(word_to_id)
target = convert_one_hot(target, vocab_size)
contexts = convert_one_hot(contexts, vocab_size)
```

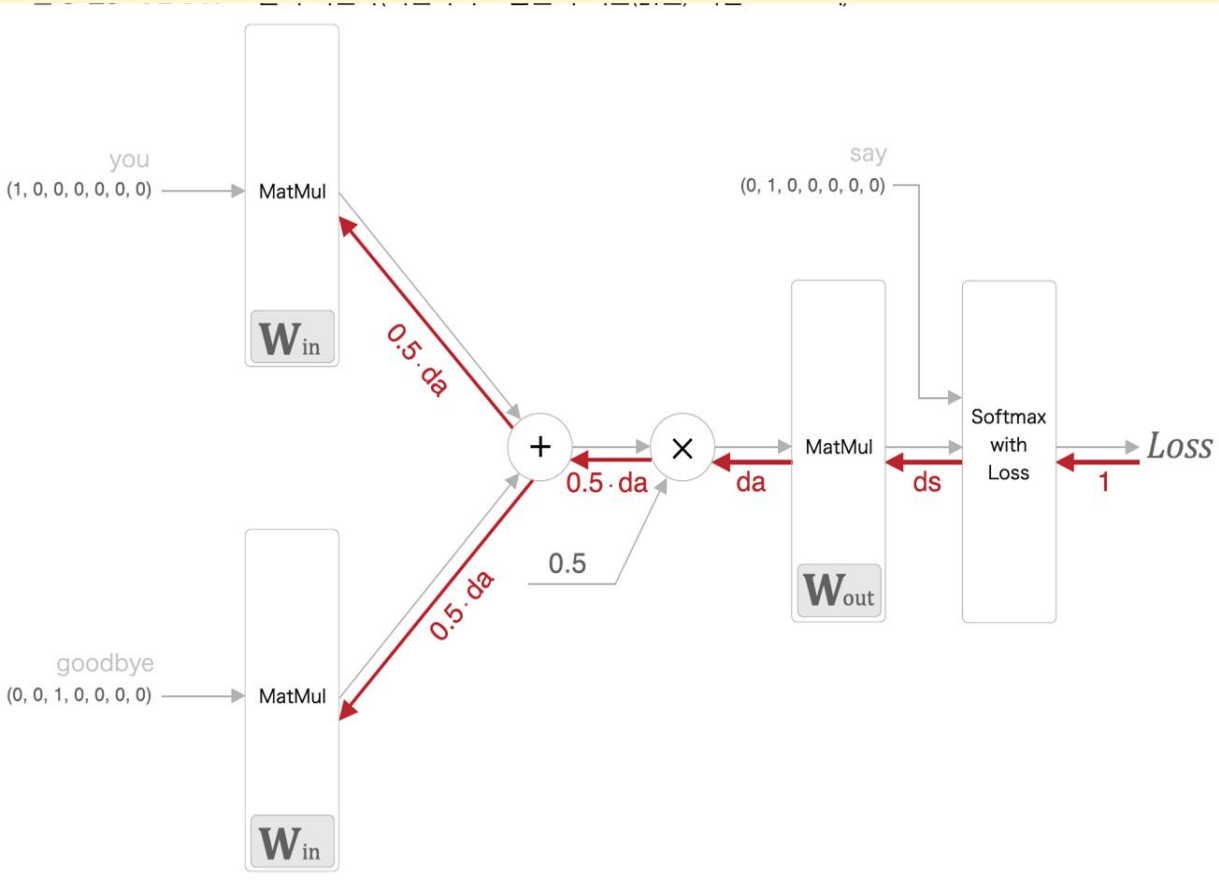
◀ 코드 3. 데이터 준비 전체 코드

```
def create_co_matrix(corpus, vocab_size, window_size=1):
    corpus_size = len(corpus)
    co_matrix = np.zeros((vocab_size, vocab_size), dtype=np.int32)

    for idx, word_id in enumerate(corpus):
        for i in range(1, window_size + 1):
            left_idx = idx - i
            right_idx = idx + i

            if left_idx >= 0:
                left_word_id = corpus[left_idx]
                co_matrix[word_id, left_word_id] += 1

            if right_idx < corpus_size:
                right_word_id = corpus[right_idx]
                co_matrix[word_id, right_word_id] += 1

    return co_matrix
```

▲ 코드 2. 맥락&타깃 생성 함수

3.4 CBOW 모델 구현

- CBOW 모델 구현



초기화 메서드는 인수로 어휘수와 은닉층의 뉴런 수를 받는다.
가중치 초기화에선 가중치 2개 생성. 각각 작은 무작위 값으로 초기화.

필요한 계층 생성.
Matmul 계층 3개(입력 2, 출력1)
softmax with Loss 계층 하나 생성.

forward 메서드 구현.
인수로 맥락과 타깃을 받아 손실을 반환.

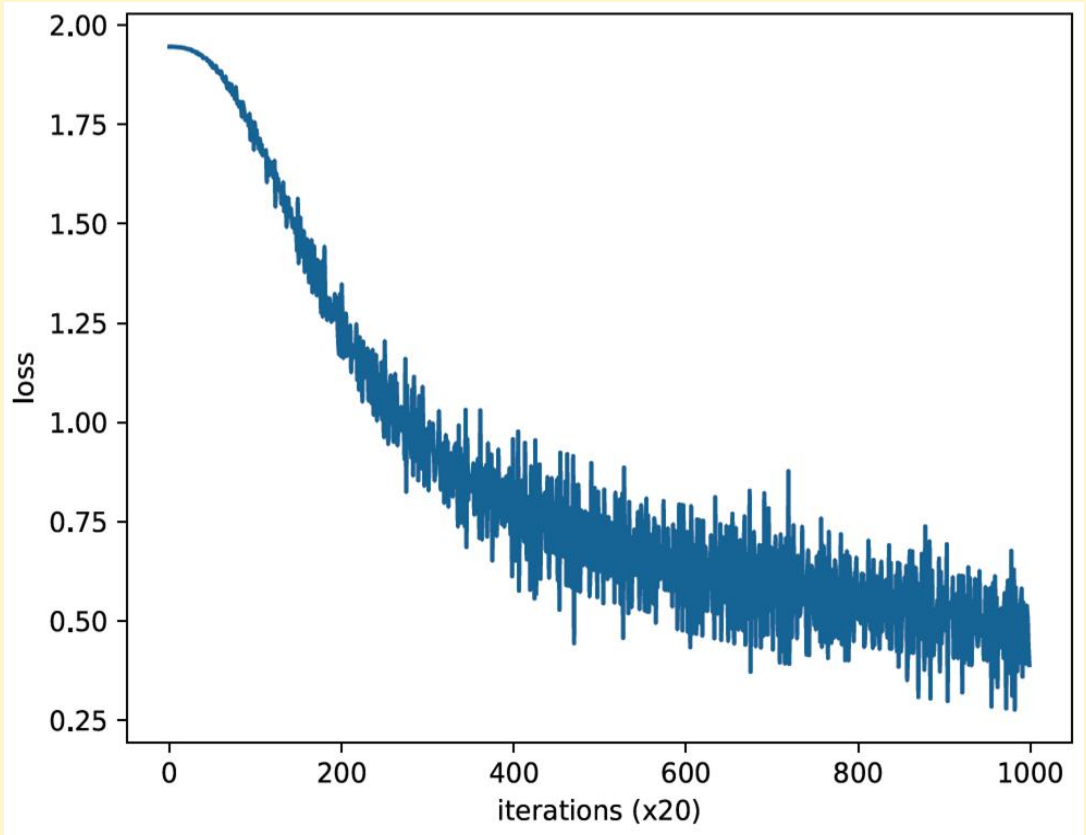
backward 메서드 구현.
순전파 때와는 반대 방향으로 전파.

각 매개변수의 기울기를 grads에 모아둠.
forward 호출 후 backward 메서드를 실행하는 것 만으로 grads 리스트에 기울기가 갱신된다.

3.4 CBOW 모델 구현

- 학습 코드 구현

```
import sys
sys.path.append('..')
from common.trainer import Trainer
from common.optimizer import Adam
from simple_cbow import SimpleCBOW
from common.util import preprocess, create_contexts_target, convert_one_hot


window_size = 1
hidden_size = 5
batch_size = 3
max_epoch = 1000

text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)

vocab_size = len(word_to_id)
contexts, target = create_contexts_target(corpus, window_size)
target = convert_one_hot(target, vocab_size)
contexts = convert_one_hot(contexts, vocab_size)

model = SimpleCBOW(vocab_size, hidden_size)
optimizer = Adam()
trainer = Trainer(model, optimizer)

trainer.fit(contexts, target, max_epoch, batch_size)
trainer.plot()
```



<학습 경과 그래프>

3.4 CBOW 모델 구현

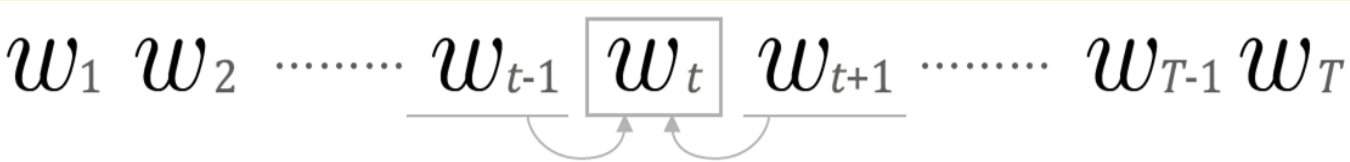
- 입력 측 MatMul 계층의 가중치

```
you [ 0.01178248  0.01342613 -0.01825819 -0.00531672  0.01964338]
say [ 0.63442814 -1.1782721   1.0292728  -0.9097165   1.0255805 ]
goodbye [ 1.7009413  1.3123868 -1.3366075 -1.3643217  1.1284081]
and [ 0.46462712 -1.1723162   1.0998902  -1.0015242   0.9973545 ]
i [-1.9195325   0.7654874  -0.65731263  1.9749272   0.95143664]
hello [ 2.2331247  1.2272426 -1.2585099 -2.007749   1.0424646]
. [ 1.4546813  1.6860805  1.3951857  1.5522774 -1.4960616]
```

단어를 밀집벡터로 나타낸 것 -> 단어의 분산 표현 (단어의 의미를 잘 파악한 벡터 표현)

3.5 word2vec 보충

- CBOW 모델과 확률



맥락이 주어졌을때 타깃이 될 확률
: 사후 확률

$P(w_t \mid w_{t-1}, w_{t+1})$ w_{t-1}과w_{t+1}이 주어졌을 때 타깃이 w_t가 될 확률

$E = -\sum_k t_k \log y_k$

* 교차 엔트로피 오차 식

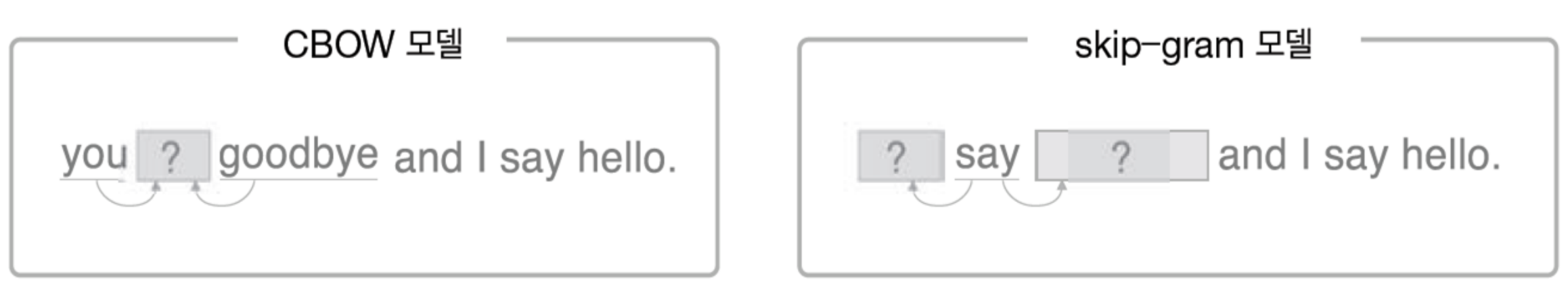
t_k:정답 레이블(원핫 벡터), y_k: k번째에 해당하는 사건이 일어날 확률

$L = -\log P(w_t \mid w_{t-1}, w_{t+1})$ 음의 로그 가능도/ 샘플 데이터 하나에 대한 손실함수

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

3.5 word2vec 보충

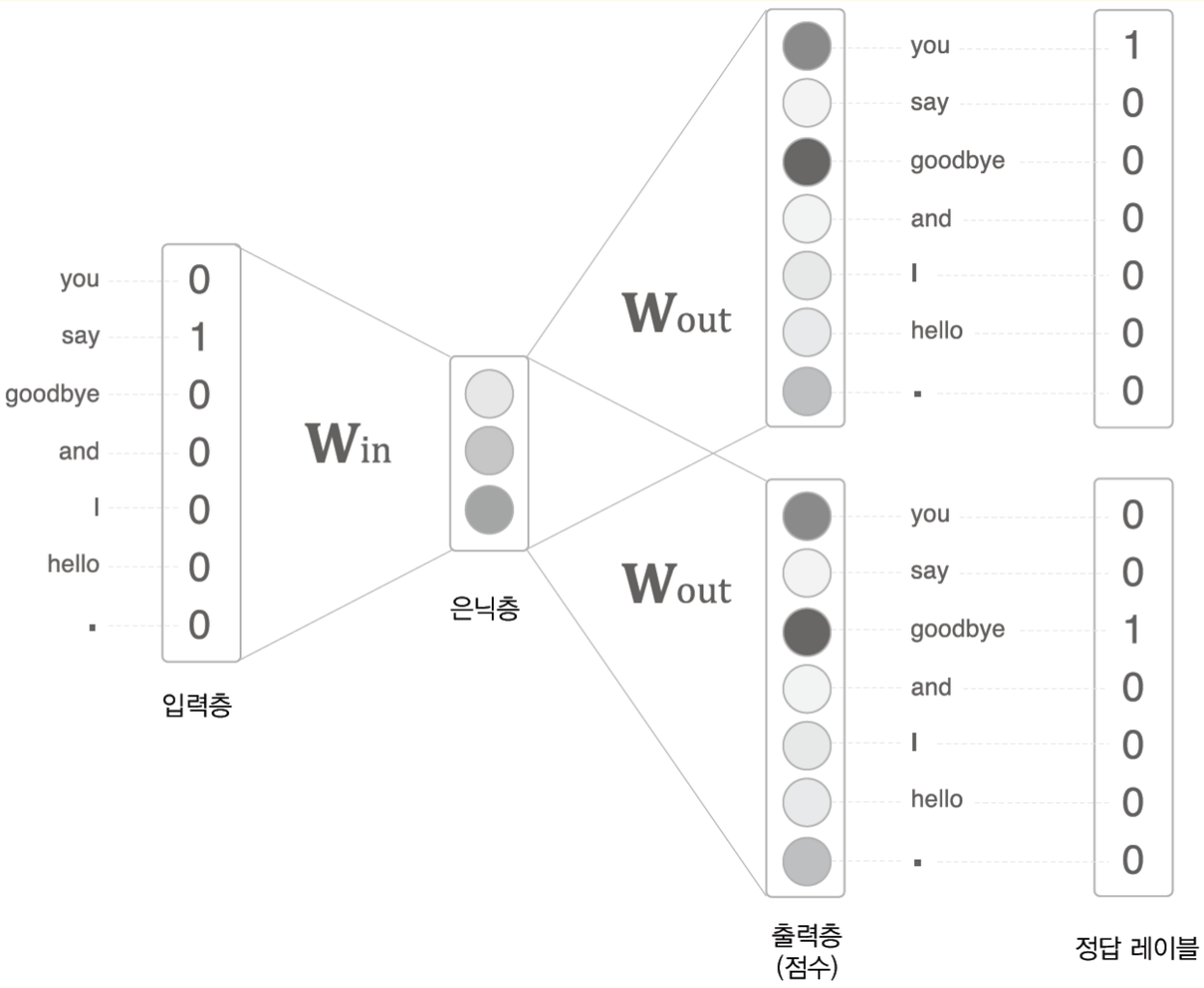
- skip-gram 모델



CBOW : 맥락으로부터 타깃을 추측
skip-gram : 타깃으로부터 맥락을 추측

3.5 word2vec 보충

- skip-gram 모델



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

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

*skip-gram은 맥락의 단어들 사이 관련성이 없다고 가정.
즉, 조건부 독립을 가정한다.

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

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

3.5 word2vec 보충

- 통계 기반 vs 추론 기반

통계 기반 : 말뭉치의 전체 통계로부터 1회 학습하여 단어의 분산 표현을 얻어냈다.
추론 기반 : 말뭉치를 일부분씩 여러 번 보면서 학습했다.(미니배치 학습)

- 어휘에 추가할 새 단어가 생겨 단어의 분산 표현을 갱신해야 한다면?
  통계 기반 : 계산을 처음부터 다시 해야한다. 동시발생 행렬을 다시 만들고 SVD를 수행. 비효율적
  추론 기반 : 학습한 가중치를 초깃값으로 사용해 다시 학습하면 된다. 효율적.

- 두 기법으로 얻는 단어의 분산표현의 성격이나 정밀도면에서 본다면?
  통계 기반 : 단어의 유사성이 인코딩된다.
  추론 기반 : 단어의 유사성은 물론 단어 사이의 패턴까지 파악되어 인코딩 된다.
  ex) king - man + woman = queen

이를 보면 추론 기반이 통계 기반보다 정확하다고 생각 할 수도 있지만, 실제로 단어의 유사성을 정량 평가해본 결과, 의외로 두 기법의 우열을 가릴 수 없었다.

3.6 정리

- 이번 장에서 배운 것

1. 추론 기반 기법은 추측하는 것이 목적이며, 그 부산물로 단어의 분산 표현을 얻을 수 있다.
2. word2vec은 추론 기반 기법이며, 단순한 2층 신경망이다.
3. word2vec은 skip-gram 모델과 CBOW 모델을 제공한다.
4. CBOW 모델은 여러 단어(맥락)로부터 하나의 단어(타깃)를 추측한다.
5. 반대로 skip-gram 모델은 하나의 단어(타깃)로부터 다수의 단어(맥락)을 추측한다.
6. word2vec은 가중치를 다시 학습할 수 있으므로, 단어의 분산 표현 갱신이나 새로운 단어 추가를 효율적으로 수행할 수 있다.

감사합니다
:)