3. word2vec

단어를 벡터로 표현하는 방법은 크게 두 부분이 있다.
1. 통계 기반 기법
2. 추론 기반 기법

단어의 의미를 얻는 방식은 서로 크게 다르지만,
그 배경에는 모두 분포 가설이 있다.

이번 절에서는 통계 기반 기법의 문제를 지적하고,
그 대안인 추론 기반 기법의 이점을 거시적 관점에서 설명한다.

통계 기반 기법의 문제점

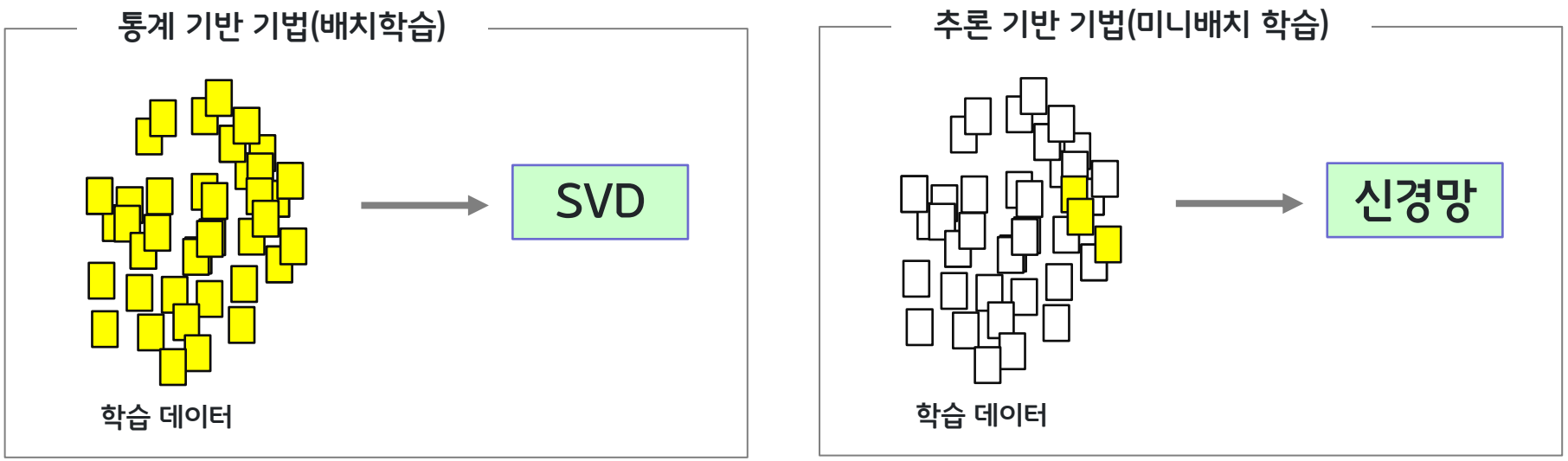
지금까지 본 것처럼 통계 기반 기법에서는 주변 단어의 빈도를 기초로 단어를 표현했다.
구체적으로는 단어의 동시발생 행렬을 만들고 그 행렬에 SVD를 적용하여
밀집벡터:단어의 분산 표현을 얻었다.
그러나 이 방식은 대규모 말뭉치를 다룰 때 문제가 발생한다.

현업에서 다루는 말뭉치의 어휘 수는 어마어마하다.
이런 거대 행렬에 SVD를 적용하는 일은 현실적이지 않다.

> SVD를 n*n 행렬에 적용하는 비용은 $O(n^3)$ 이다.

통계 기반 기법의 문제점

통계 기반 기법과 추론 기반 기법 비교



통계 기반 기법은 학습 데이터를 한꺼번에 처리한다.

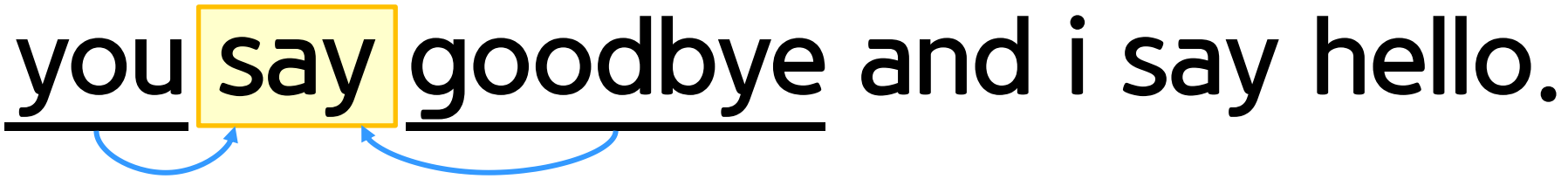
배치학습 추론 기반 기법은 학습 데이터의 일부를 사용하여 순차적으로 학습한다.

미니배치 학습 :
말뭉치의 어휘 수가 많아 SVD 등 계산량이 큰 작업을 처리하기
어려운 경우에도 신경망을 학습시킬 수 있다는 의미이다.
데이터를 작게 나눠 학습하기 때문이다.

추론 기반 기법 개요

추론 기반 기법에서는 추론이 주된 작업이다.
추론이란, 주변 단어:맥락이 주어졌을 때,
? 에 무슨 단어가 들어가는지를 추측하는 작업이다.

주변 단어들의 맥락으로 사용해 "?"에 들어갈 단어를 추측한다.

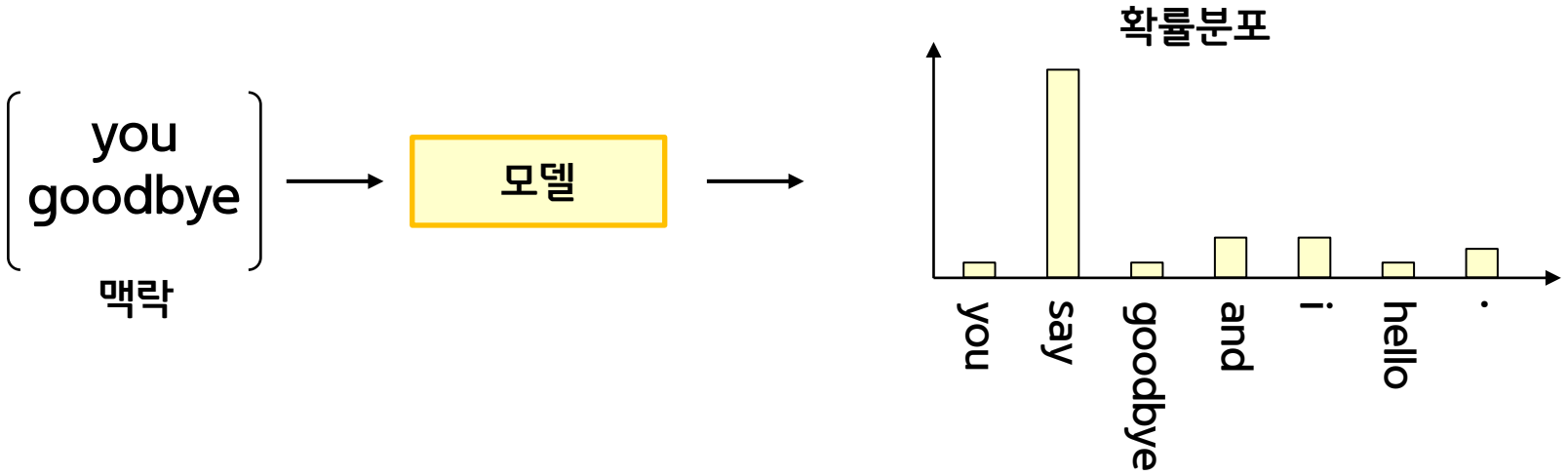


추론 문제를 풀고 학습하는 것이 추론 기반 기법이 다루는 문제이다.
이러한 추론 문제를 반복해서 풀면서 단어의 출현 패턴을 학습하는 것이다.

추론 기반 기법 개요

모델 관점에서 보면, 추론 문제는 다음과 같다.

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



추론 기반 기법에는 어떠한 모델이 등장한다.
우리는 이 모델로 신경망을 사용한다.
모델은 맥락 정보를 입력 받아 출현할 수 있는 각 단어의 출현 확률을 출력한다.
이러한 틀 안에서 말뭉치를 사용해 모델이 올바른 추측을 내놓도록 학습시킨다.
그리고 그 학습의 결과로 단어의 분산 표현을 얻는 것이 추론 기반 기법의 전체 그림이다.

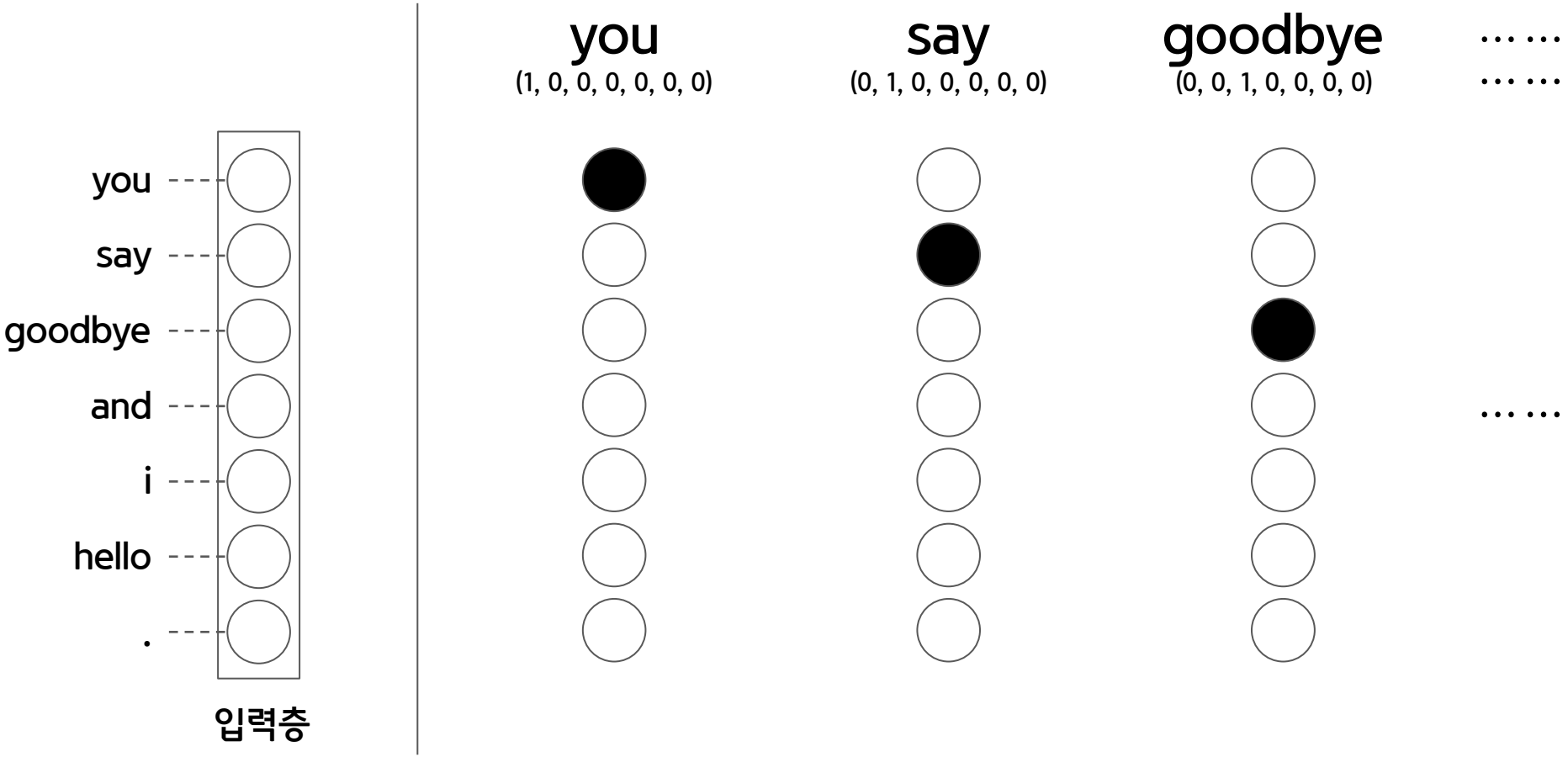
지금부터 신경망을 이용해 단어를 처리해보자.
단어를 있는 그대로 처리할 수 없으니 고정 길이의 벡터로 변환해야 한다.
이때 사용하는 대표적인 방법이 단어를 원 핫 표현으로 변환하는 것이다.
원 핫 표현이란, 벡터의 원소 중 하나만 1이고, 나머지는 모두 0인 벡터를 말한다.

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

단어를 원 핫 표현으로 변환하는 방법

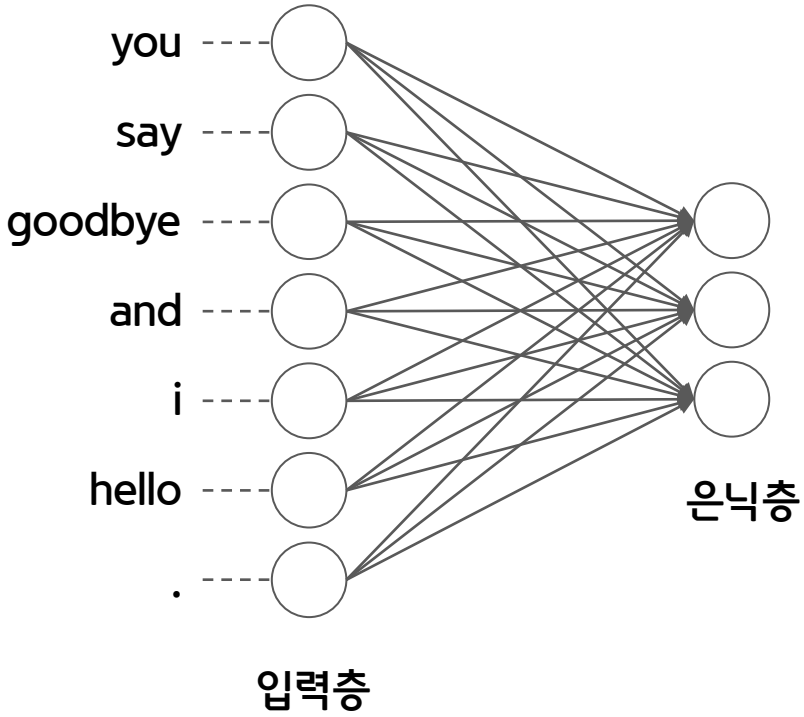
- 먼저 총 어휘 수만큼의 원소를 갖는 벡터를 준비하고,
- 인덱스가 단어 ID 와 같은 원소를 1로, 나머지는 모두 0으로 설정한다.
  이처럼 단어를 고적 길이 벡터로 변환하면, 신경망의 입력층은 뉴런의 수를 고정할 수 있다.

입력층의 뉴런: 각 뉴런이 각 단어에 대응한다.(해당 뉴런이 1이면 검은색, 0이면 흰색)



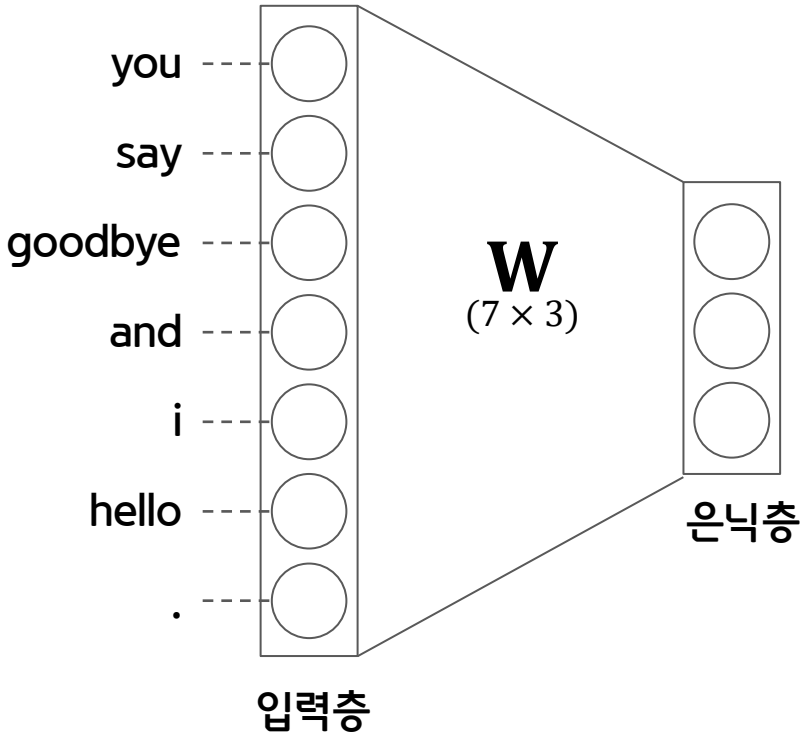
단어를 벡터로 나타낼 수 있고, 신경망을 구성하는 계층들은 벡터를 처리할 수 있다.
다시 말해, 단어를 신경망으로 처리할 수 있다는 뜻이다.

완전연결층에 의한 변환 : 입력층의 각 뉴런은 7개의 단어 각각에 대응(은닉층의 뉴런은 3개를 준비함)

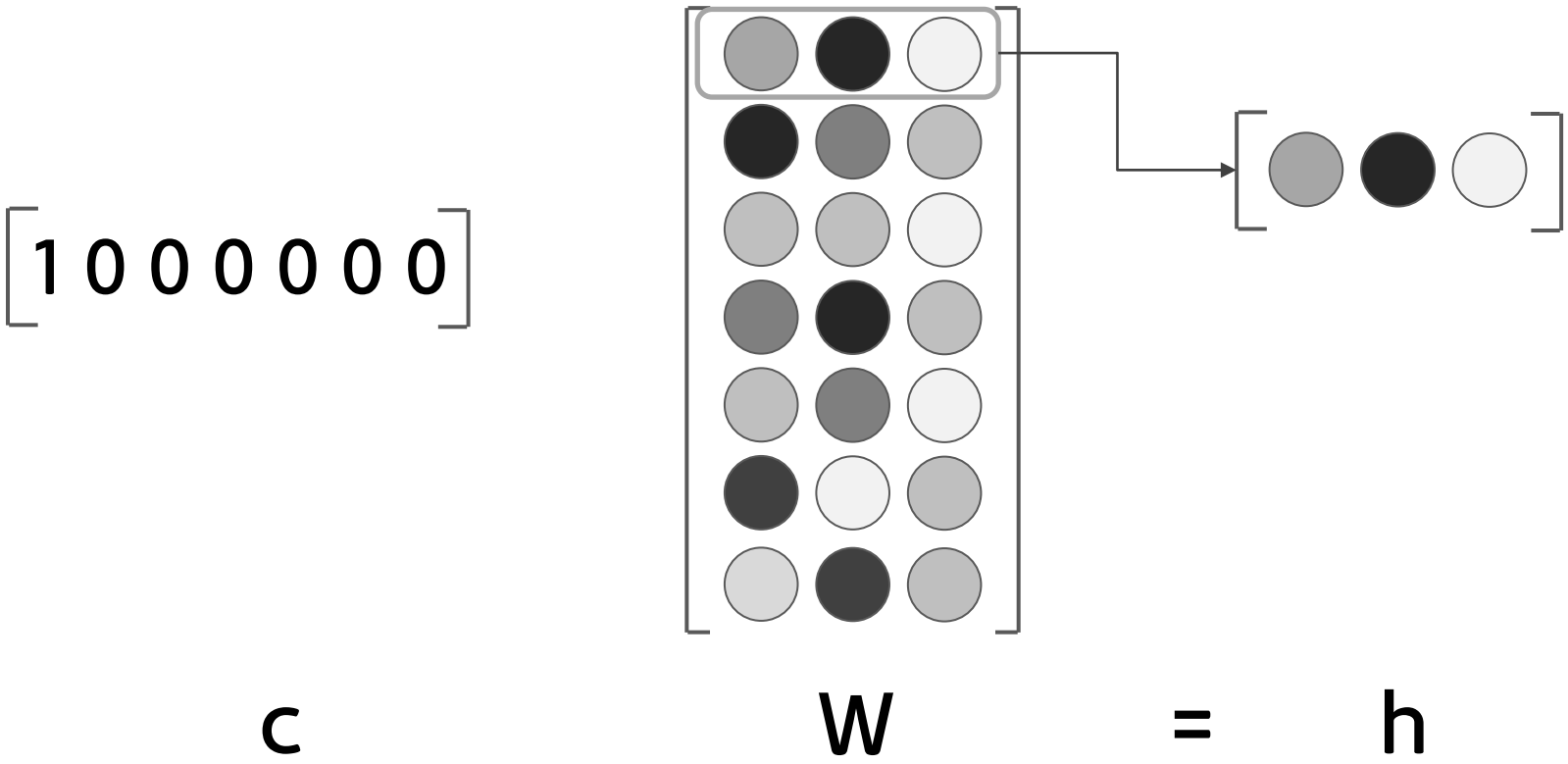


화살표에는 가중치:매개변수가 존재하여, 입력층 뉴런과의 가중합이 은닉층 뉴런이 된다.
간단한 설명을 위해 완전연결계층에서는 편향을 생략했다.
편향을 이용하지 않은 완전연결계층은 행렬 곱 계산에 해당한다.

완전연결층에 의한 변환을 단순화한 그림(완전연결계층의 가중치를 7x3 크기의 W라는 행렬로 표현)



맥락 c와 가중치 W의 곱으로 해당 위치의 행벡터가 추출된다.
(각 요소의 가중치 크기는 흑백의 진하기로 표현)



3. word2vec

앞 절에서 추론 기반 기법을 배우고, 신경망으로 단어를 처리하는 방법을 코드로 살펴보았다.
이제 word2vec 을 구현할 차례이다.

지금부터 할 일은 모델을 신경망으로 구축하는 것이다.
이번 절에서 사용할 신경망은

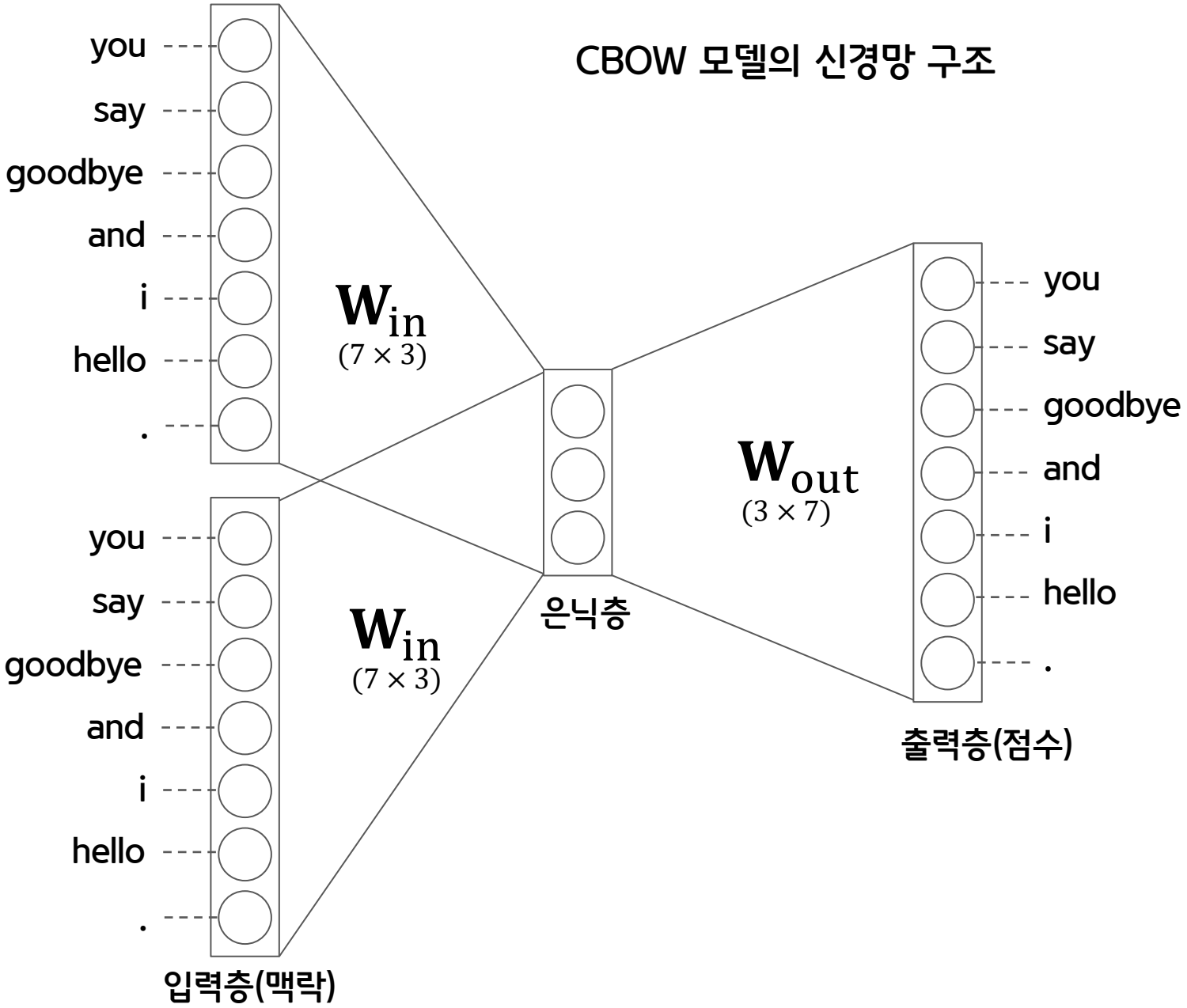
word2vec 에서 제안하는 CBOW, continuous bag-of-words 모델이다.

CBOW 모델의 추론 처리

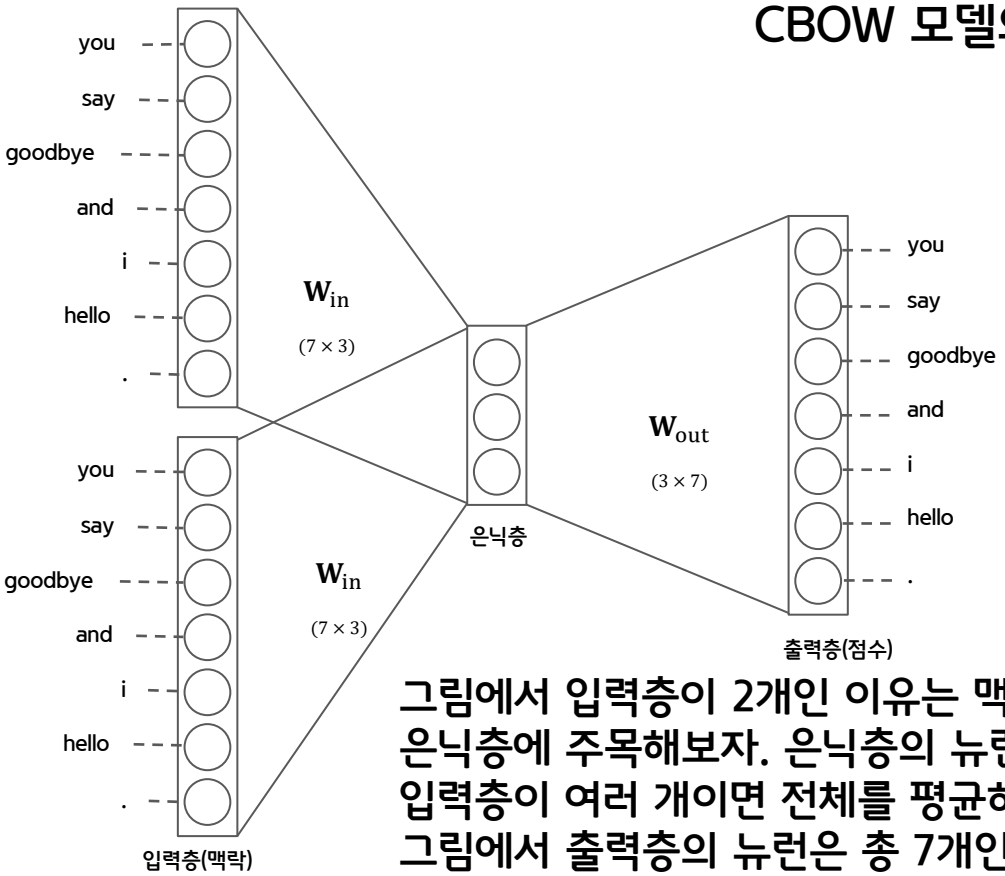
CBOW 모델은 맥락으로부터 타깃을 추측하는 용도의 신경망이다.
타깃은 중앙 단어이고, 그 주변 단어들이 맥락이다.
우리는 이 CBOW 모델이 가능한 한 정확하게 추론하도록 훈련시켜서 단어의 분산 표현을 얻어낼 것이다.

CBOW 모델의 입력은 맥락이다.
가장 먼저, 이 맥락을 원핫 표현으로 변환하여 CBOW 모델이 처리할 수 있도록 준비한다.

CBOW 모델의 추론 처리

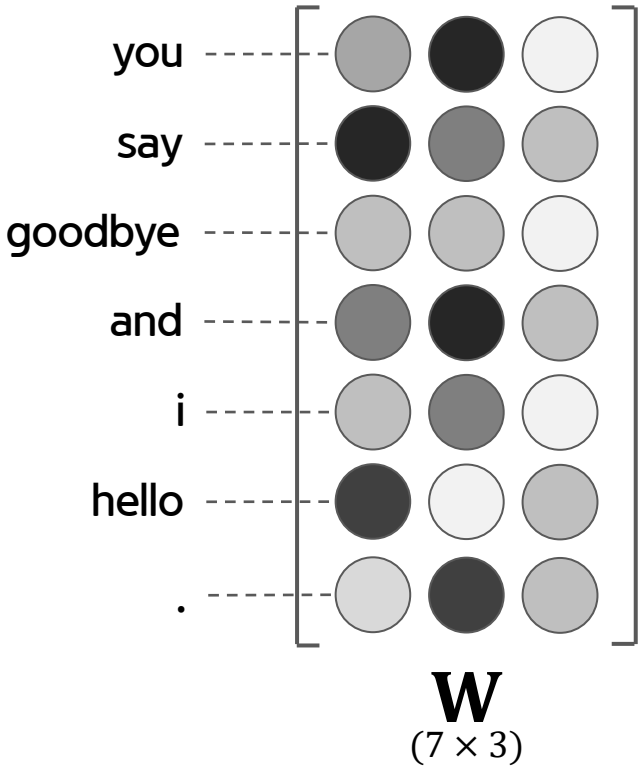


CBOW 모델의 신경망 구조



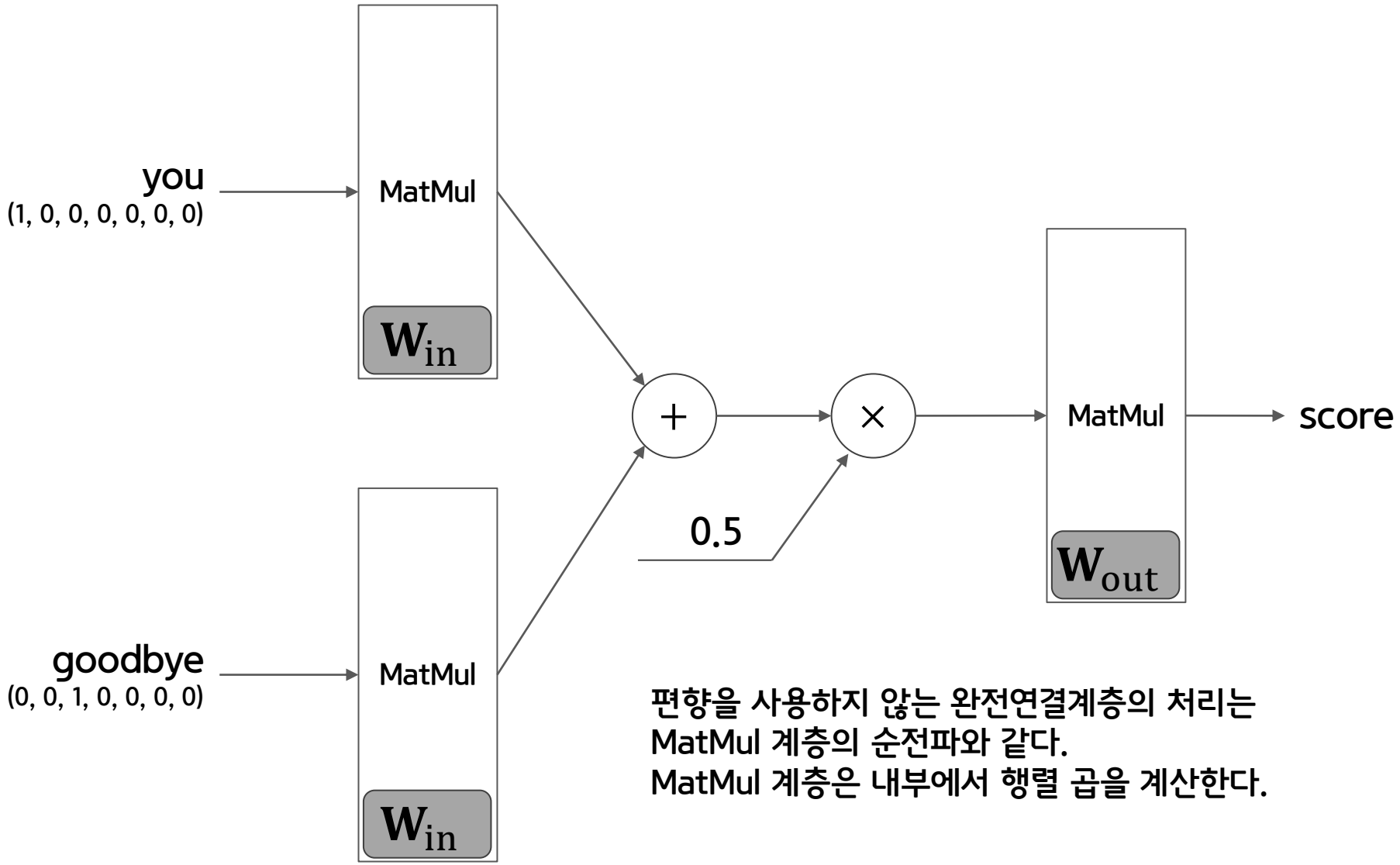
그림에서 입력층이 2개인 이유는 맥락으로 고려할 단어를 2개로 정했기 때문이다.
은닉층에 주목해보자. 은닉층의 뉴런은 입력층의 완전연결계층에 의해 변환된 값이 되는데,
입력층이 여러 개이면 전체를 평균하면 된다.
그림에서 출력층의 뉴런은 총 7개인데, 중요한 것은 이 뉴런 하나하나가
각각의 단어에 대응한다는 점이다.
그리고 출력층 뉴런은 각 단어의 점수를 뜻하며, 값이 높을수록 대응 단어의 출현 확률도 높아진다.
여기서 점수란, 확률로 해석되기 전의 값이고, 이 점수에 소프트맥스
함수를 적용해서 확률을 얻을 수 있다.
점수를 Softmax 계층에 통과시킨 후의 뉴런을 출력층이라고도 한다.

가중치의 각 행의 해당 단어의 분산 표현이다.

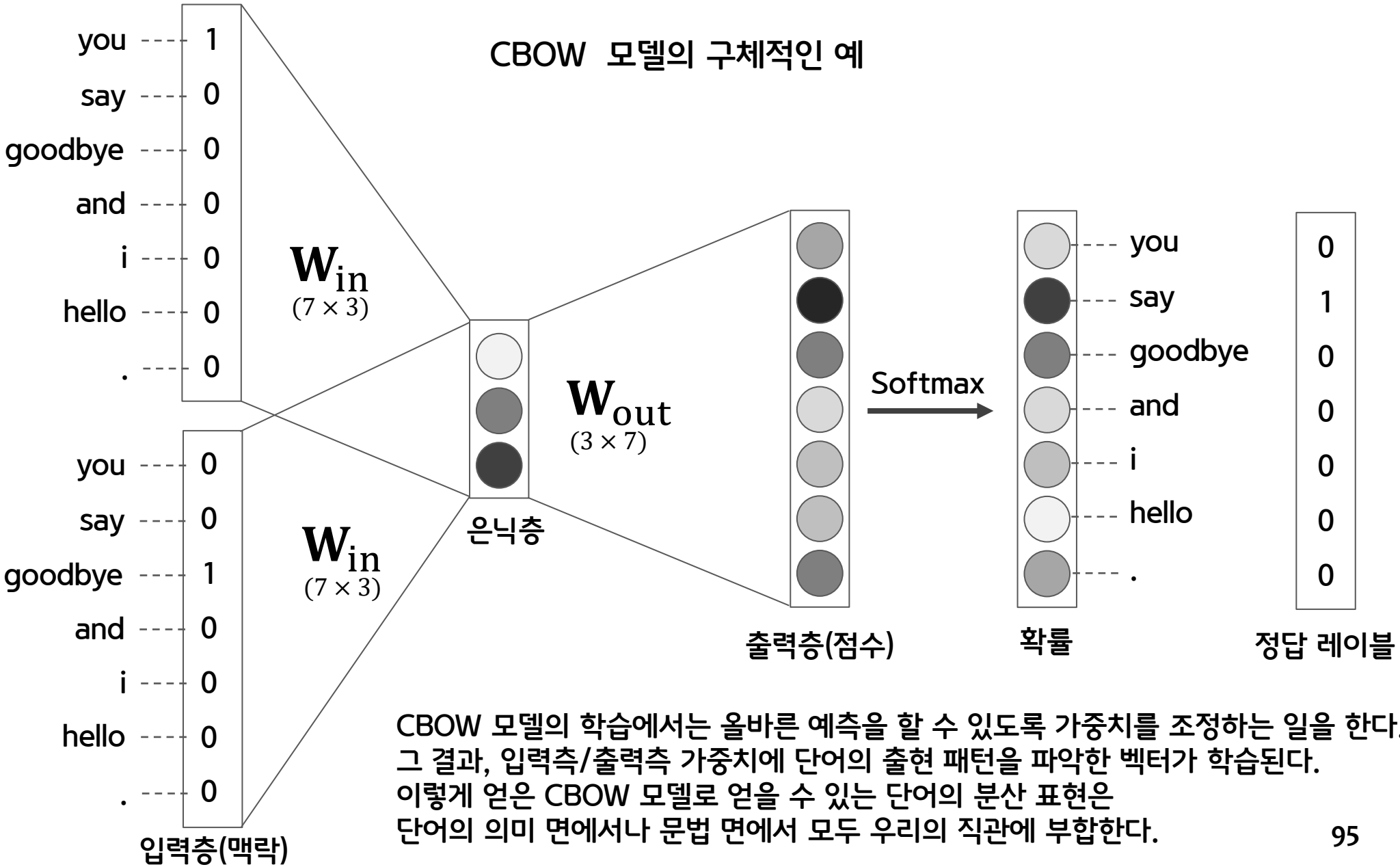


은닉층의 뉴런 수를 입력 층의 뉴런 수보다 적게 하는 것이 중요한 핵심이다.
이렇게 해야 은닉층에는 단어 예측에 필요한 정보를 간결하게 담게 되며,
결과적으로 밀집벡터 표현을 얻을 수 있다.
이때 은닉층 정보는 인간이 이해할 수 없는 코드로 쓰여 있다. (인코딩)
한편, 은닉층의 정보로부터 원하는 결과를 얻는 작업은 디코딩이라고 한다.
즉, 디코딩이란 인코딩된 정보를 인간이 이해할 수 있는 표현으로 복원하는 작업이다.

계층 관점에서 본 CBOW 모델의 신경망 구성



편향을 사용하지 않는 완전연결계층의 처리는
MatMul 계층의 순전파와 같다.
MatMul 계층은 내부에서 행렬 곱을 계산한다.

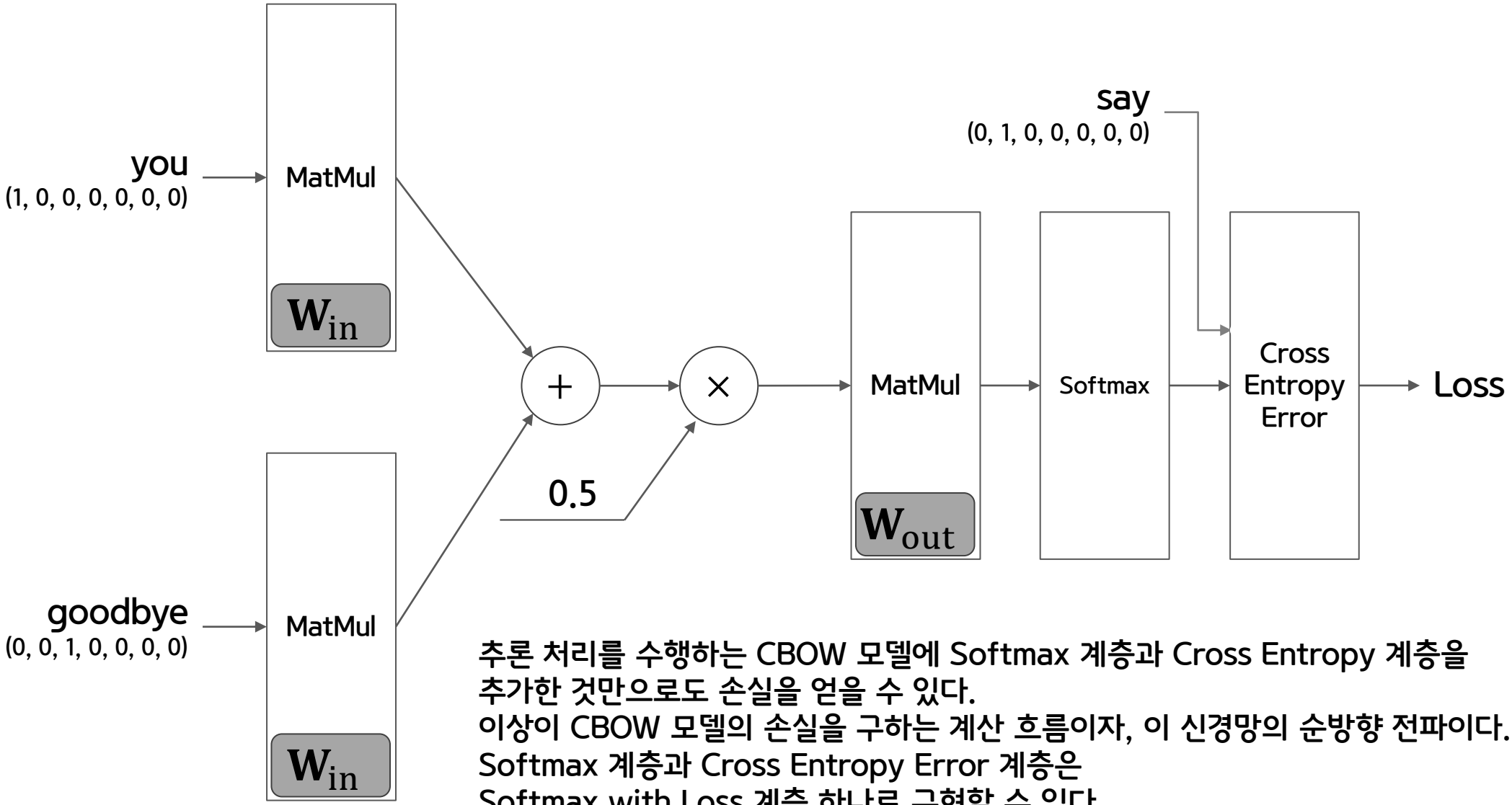


CBOW 모델의 학습에서는 올바른 예측을 할 수 있도록 가중치를 조정하는 일을 한다.
그 결과, 입력측/출력측 가중치에 단어의 출현 패턴을 파악한 벡터가 학습된다.
이렇게 얻은 CBOW 모델로 얻을 수 있는 단어의 분산 표현은
단어의 의미 면에서나 문법 면에서 모두 우리의 직관에 부합한다.

신경망의 학습에 대해 생각해보자.
우리가 다루는 모델은 다중 클래스 분류를 수행하는 신경망이다.
따라서 이 신경망을 학습하려면, 소프트맥스 함수와 교차 엔트로피 오차만 이용하면 된다.

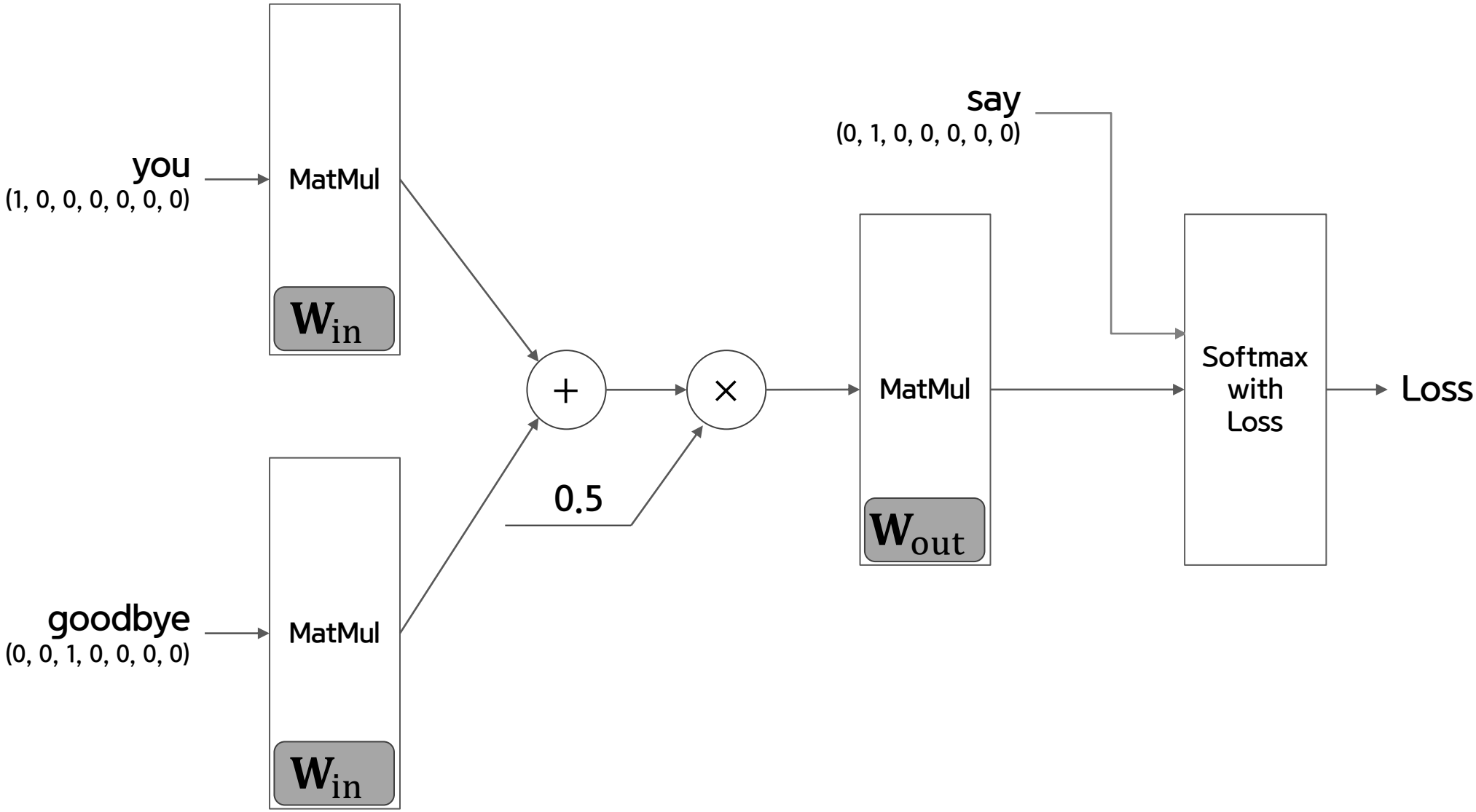
소프트맥스를 이용해 점수를 확률로 변환하고,
그 확률과 정답 레이블로부터 교차 엔트로피 오차를 구한 후,
그 값을 손실로 사용해 학습을 진행한다.

CBOW 모델의 학습 시 신경망 구성

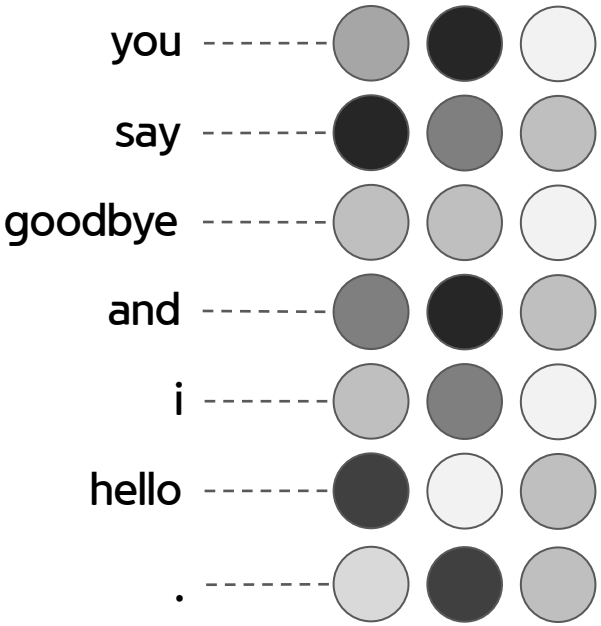


추론 처리를 수행하는 CBOW 모델에 Softmax 계층과 Cross Entropy 계층을 추가한 것만으로도 손실을 얻을 수 있다.
이상이 CBOW 모델의 손실을 구하는 계산 흐름이자, 이 신경망의 순방향 전파이다.
Softmax 계층과 Cross Entropy Error 계층은
Softmax with Loss 계층 하나로 구현할 수 있다.

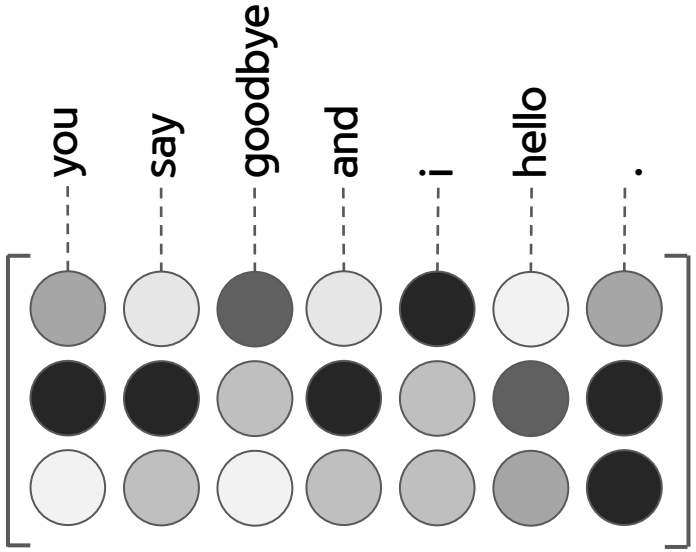
CBOW 모델의 학습 시 신경망 구성



각 단어의 분산 표현은 입력 측과 출력 측 모두의 가중치에서 확인할 수 있다.



$\mathbf{W}_{\text{in}}$
(7×3)



$\mathbf{W}_{\text{out}}$
(3×7)

그러면 최종적으로 이용하는 단어의 분산 표현으로는 어느 쪽 가중치를 사용하면 좋을까?
선택지는 3가지.

1. 입력 측의 가중치만 이용
2. 출력 측의 가중치만 이용
3. 양쪽 가중치를 모두 이용

word2vec, 특히 skip-gram 모델에서는 입력 측 가중치만 이용하는 것이 가장 대중적이다.

3. word2vec

3.1 추론 기반 기법과 신경망

3.2 단순한 word2vec

3.3 학습 데이터 준비

3.4 CBOW 모델 구현

3.5 word2vec 보충

word2vec에서 이용하는 신경망의 입력은 맥락이다.

그리고 정답 레이블은 맥락에 둘러싸인 중앙의 단어, 즉 타깃이다.

우리가 해야 할 일은 신경망에 맥락을 입력했을 때 타깃이 출현할 확률을 높이는 것이다.

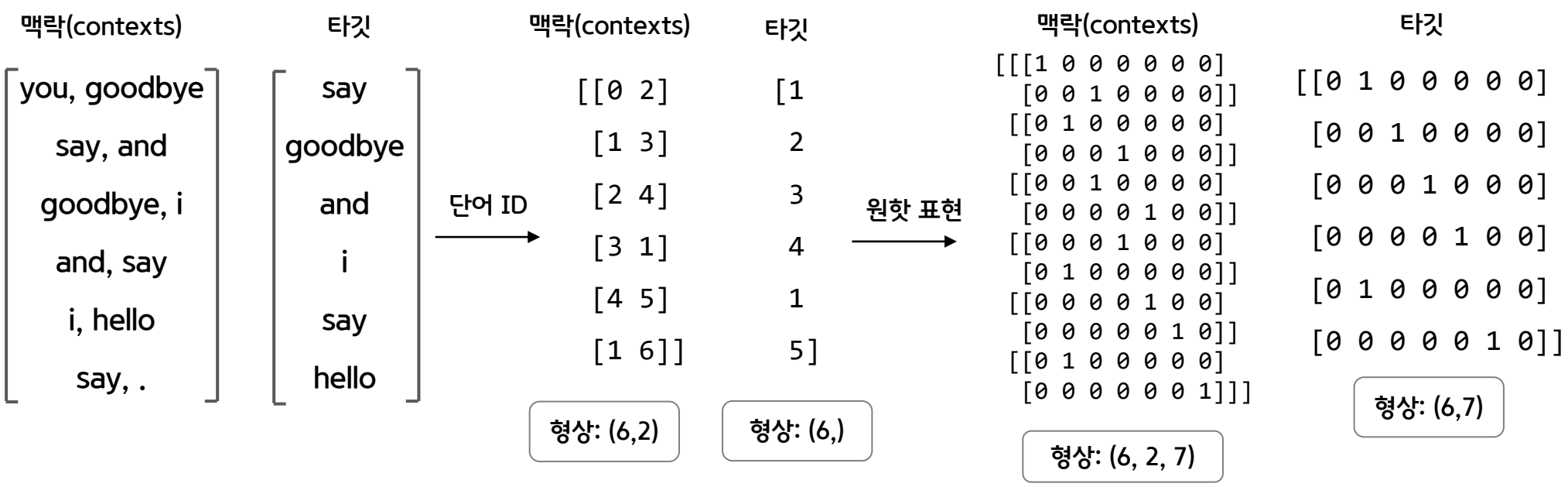
말뭉치에서 맥락과 타깃을 만드는 예

말뭉치	맥락(contexts)	타깃
you say goodbye and i say hello .	you, goodbye	say
you say goodbye and i say hello .	say, and	goodbye
you say goodbye and i say hello .	goodbye, i	and
you say goodbye and i say hello .	and, say	i
you say goodbye and i say hello .	i, hello	say
you say goodbye and i say hello .	say, .	hello

맥락과 타깃을 원핫 표현으로 변환하는 예

말뭉치	맥락(contexts)	타깃
[0 1 2 3 4 1 5 6]	[[0 2] [1 3] [2 4] [3 1] [4 5] [1 6]]	[1 2 3 4 1 5]
형상: (8,)	형상: (6,2)	형상: (6,)

맥락과 타깃을 원핫 표현으로 변환하는 예



```
def create_contexts_target(corpus, window_size=1):
    target = corpus[window_size:-window_size]
    contexts = []

    for idx in range(window_size, len(corpus)-window_size):
        cs = []
        for t in range(-window_size, window_size + 1):
            if t == 0:
                continue
            cs.append(corpus[idx + t])
        contexts.append(cs)

    return np.array(contexts), np.array(target)

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

```
                        corpus = [0 1 2 3 4 1 5 6]
cs = [0,2]              target = [1 2 3 4 1 5]
                        contexts = [[0,2], [1,3], [2,4], [3,1], [4,5], [1,6]]
```

```
def convert_one_hot(corpus, vocab_size):
    N = corpus.shape[0]

    if corpus.ndim == 1:
        one_hot = np.zeros((N, vocab_size), dtype=np.int32)
        for idx, word_id in enumerate(corpus):
            one_hot[idx, word_id] = 1

    elif corpus.ndim == 2:
        C = corpus.shape[1]
        one_hot = np.zeros((N, C, vocab_size), dtype=np.int32)
        for idx_0, word_ids in enumerate(corpus):
            for idx_1, word_id in enumerate(word_ids):
                one_hot[idx_0, idx_1, word_id] = 1

    return one_hot

target = convert_one_hot(target, vocab_size)
```

```
corpus = [1 2 3 4 1 5]

N=6

vocab_size = 7
```

target

0	1	0	0	0	0	0
0	0	1	0	0	0	0
			1			
				1		
	1					
					1	

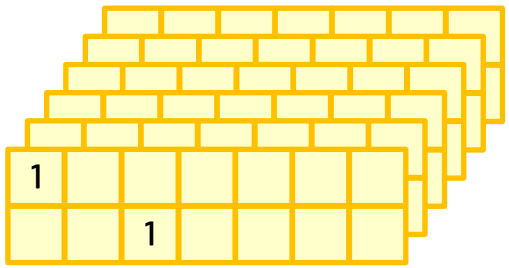
one_hot

```
def convert_one_hot(corpus, vocab_size):
    N = corpus.shape[0]

    if corpus.ndim == 1:
        one_hot = np.zeros((N, vocab_size), dtype=np.int32)
        for idx, word_id in enumerate(corpus):
            one_hot[idx, word_id] = 1

    elif corpus.ndim == 2:
        C = corpus.shape[1]
        one_hot = np.zeros((N, C, vocab_size), dtype=np.int32)
        for idx_0, word_ids in enumerate(corpus):
            for idx_1, word_id in enumerate(word_ids):
                one_hot[idx_0, idx_1, word_id] = 1

    return one_hot

contexts = convert_one_hot(contexts, vocab_size)
```

```
vocab_size = 7
```

(6,2) N=6 C=2

```
corpus = [[0,2], [1,3], [2,4], [3,1], [4,5], [1,6]]
```



contexts => (6,2,7)

3. word2vec

3.1 추론 기반 기법과 신경망

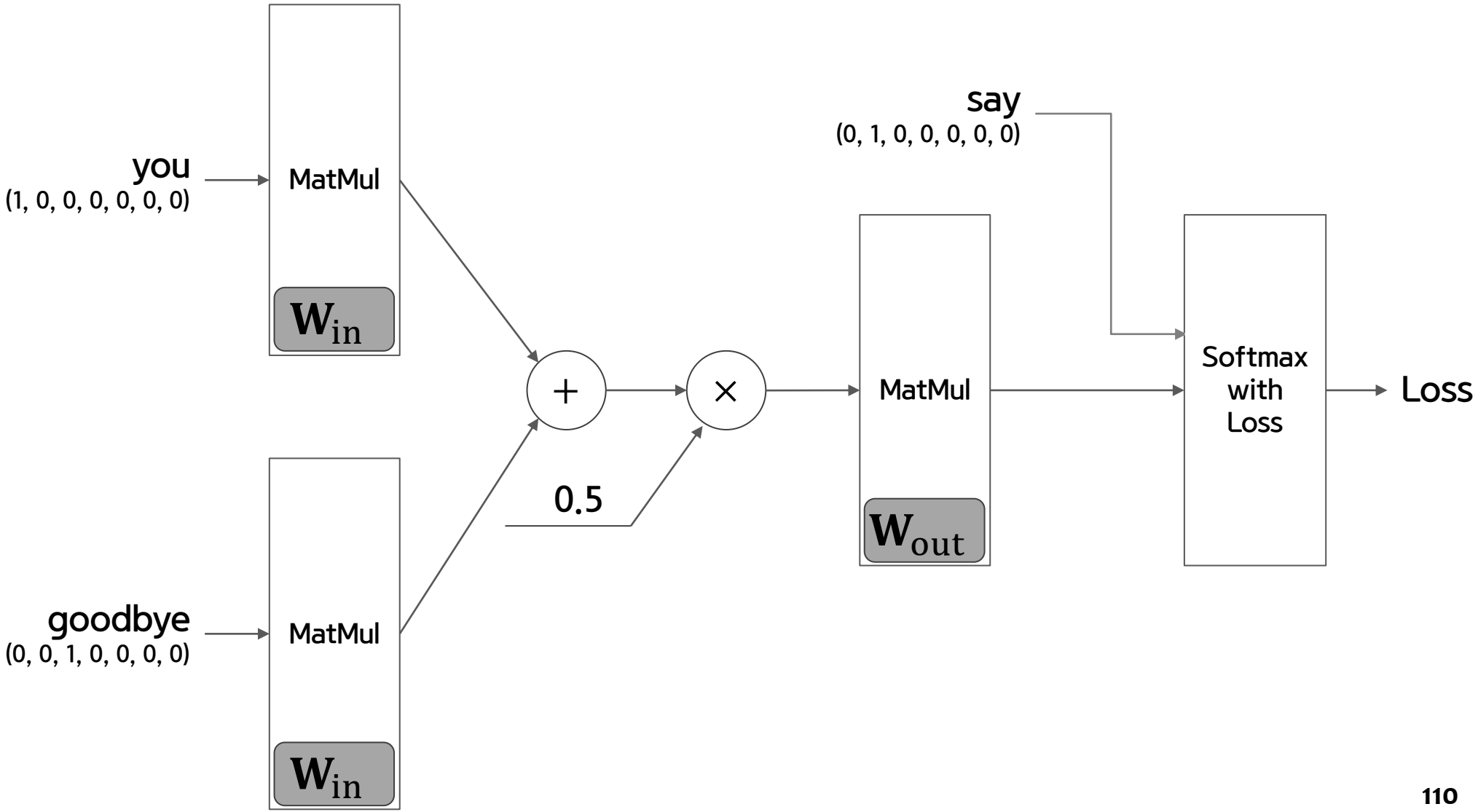
3.2 단순한 word2vec

3.3 학습 데이터 준비

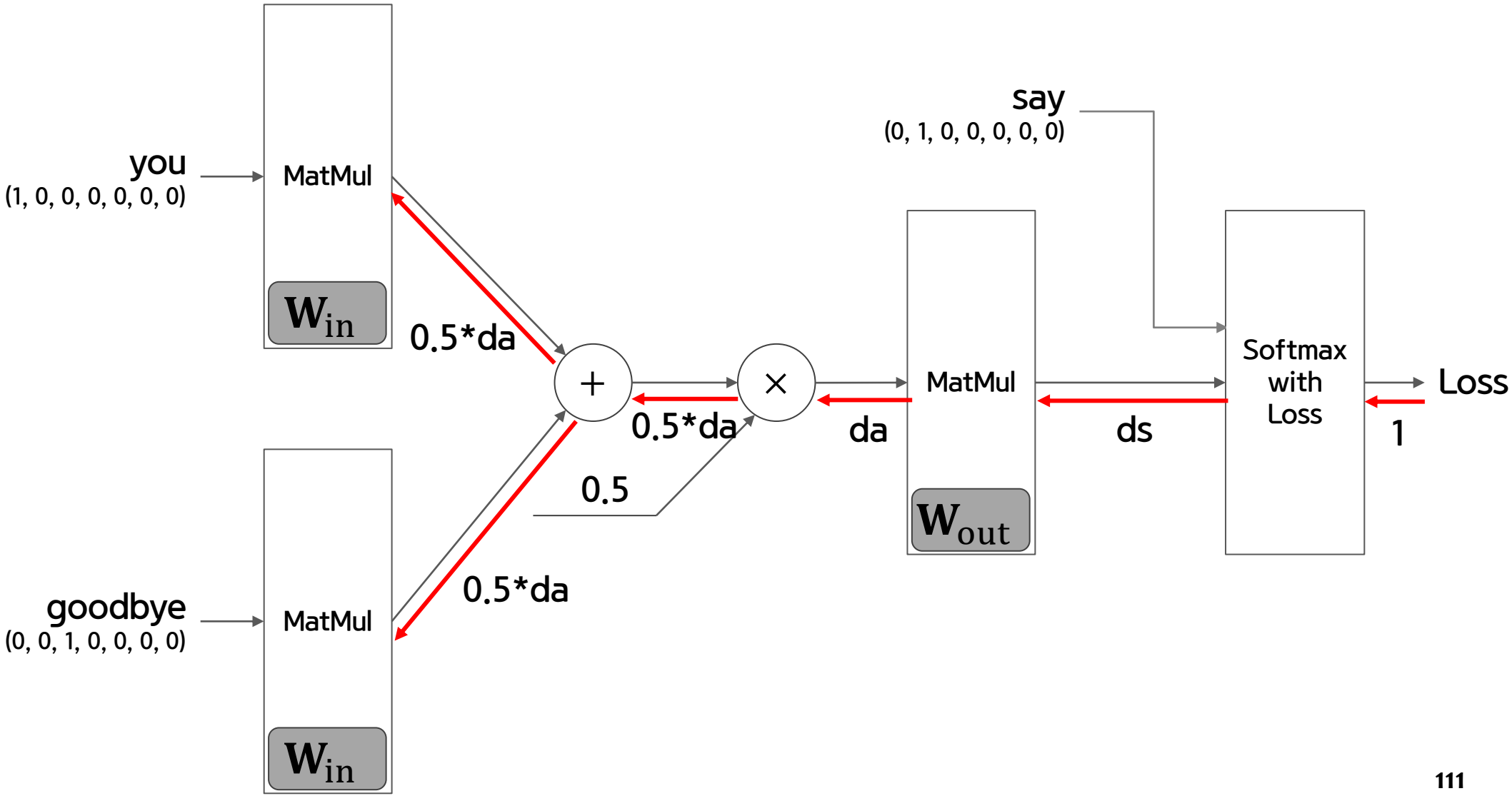
3.4 CBOW 모델 구현

3.5 word2vec 보충

CBOW 모델의 신경망 구성



CBOW 모델의 역전파


you
(1, 0, 0, 0, 0, 0, 0)
MatMul
W_{in}
goodbye
(0, 0, 1, 0, 0, 0, 0)
MatMul
W_{in}
0.5*da
0.5*da
+
×
0.5
da
MatMul
W_{out}
say
(0, 1, 0, 0, 0, 0, 0)
ds
Softmax with Loss
Loss
1

3. word2vec

3.1 추론 기반 기법과 신경망

3.2 단순한 word2vec

3.3 학습 데이터 준비

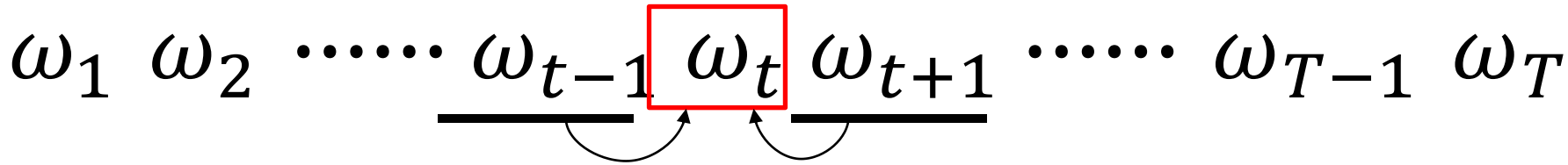
3.4 CBOW 모델 구현

3.5 word2vec 보충

확률의 표기법을 간단하게 살펴보자.
확률 P()
동시 확률 P(A,B), A와 B가 동시에 일어날 확률.
사후 확률 P(A|B), 사건이 일어난 후의 확률.
B라는 정보가 주어졌을 때, A가 일어날 확률.

그럼 CBOW 모델을 확률 표기법으로 기술해보자.
CBOW 모델이 하는 일은, 맥락을 주면 타깃 단어가 출현할 확률을 출력하는 것이다.

word2vec 의 CBOW 모델 – 맥락의 단어로부터 타깃 단어를 추측



CBOW 모델은 다음 식을 모델링하고 있다.

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

위 식을 이용하면 CBOW 모델의 손실 함수도 간결하게 표현할 수 있다.
교차 엔트로피 오차를 적용해보자.
다음 식을 유도할 수 있다.

$$L = -logP(w_t|w_{t-1}, w_{t+1})$$

이 식을 보듯, CBOW 모델의 손실 함수는 단순히 식의 확률에 log 를 취한 다음 마이너스를 붙이면 된다.
덧붙여 식은 샘플 데이터 하나에 대한 손실 함수이며, 이를 말뭉치 전체로 확장하면 다음 식이 된다.

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

CBOW 모델의 학습이 수행하는 일은, 손실 함수 식을 가능한 작게 만드는 것이다.
그리고 이때의 가중치 매개변수가 우리가 얻고자 하는 단어의 분산 표현이다.

```
def preprocess(text):
    text = text.lower()
    text = text.replace('.', ' .')
    words = text.split(' ')

    word_to_id = {}
    id_to_word = {}
    for word in words:
        if word not in word_to_id:
            new_id = len(word_to_id)
            word_to_id[word] = new_id
            id_to_word[new_id] = word

    corpus = np.array([word_to_id[w] for w in words])

    return corpus, word_to_id, id_to_word

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

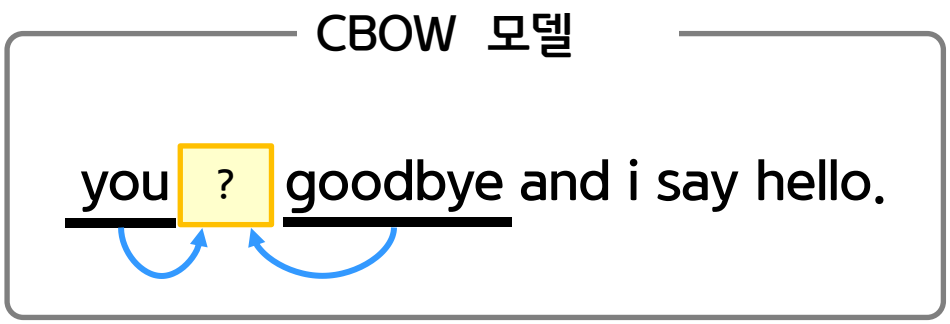
word_to_id	id_to_word
"you":0	0:"you"
"say":1	1:"say"
"goodbye":2	2:"goodbye"
"and":3	3:"and"
"i":4	4:"i"
"hello":5	5:"hello"
".":6	6:"."

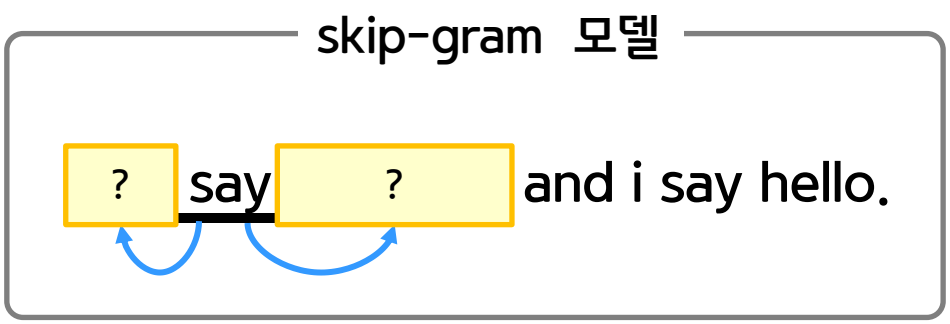
corpus	0	1	2	3	4	1	5	6

word2vec은 2개의 모델을 제안하고 있다.
1. CBOW 모델
2. skip-gram 모델

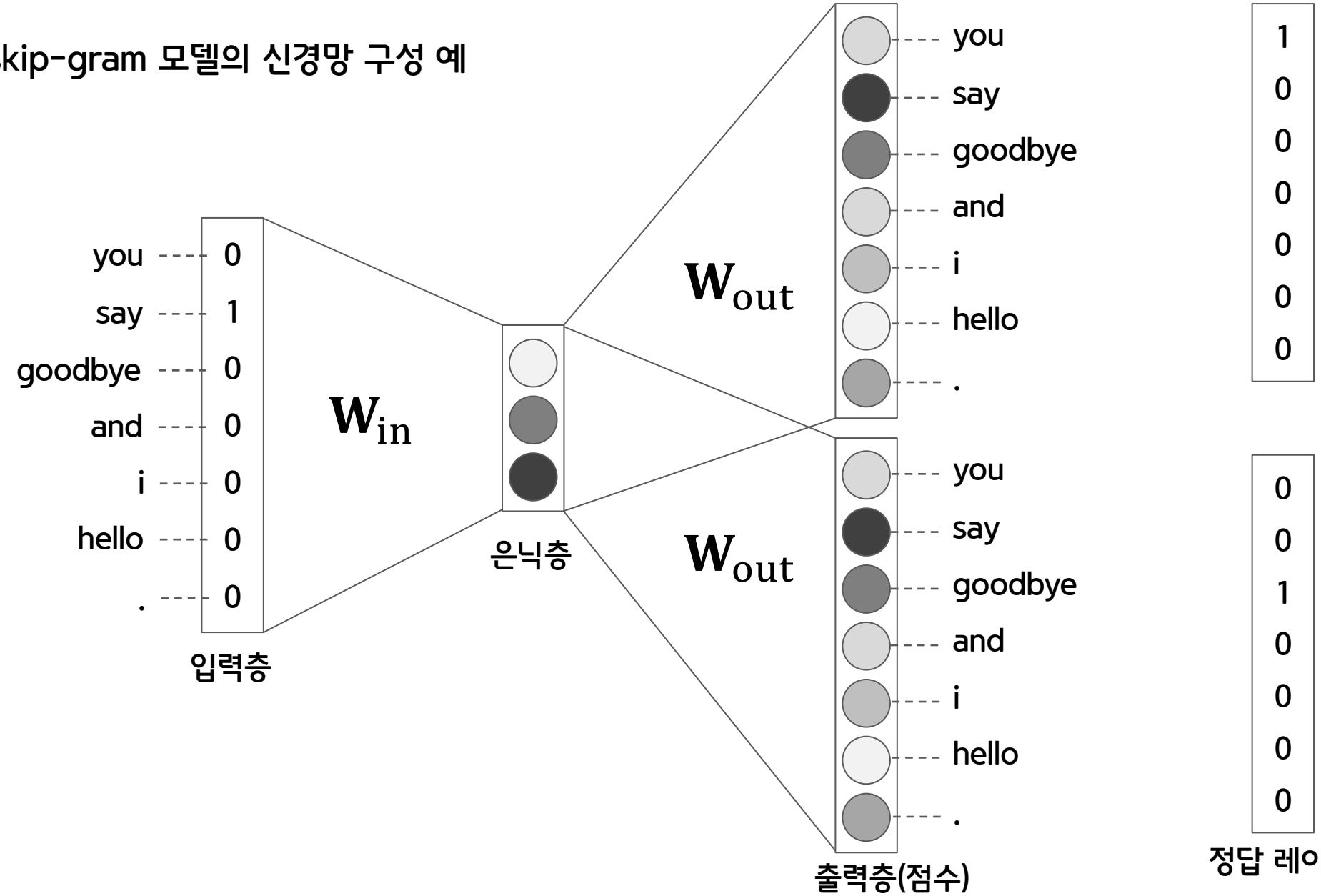
skip-gram 은 CBOW 에서 다루는 맥락과 타깃을 역전시킨 모델.

CBOW 모델과 skip-gram 모델이 다루는 문제





skip-gram 모델의 신경망 구성 예



skip-gram 모델을 확률 표기로 나타내보자.
skip-gram 은 다음 식을 모델링한다.

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

skip-gram 모델에서는 맥락의 단어들 사이에 관련성이 없다고 가정하고, 다음과 같이 분해한다.

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

위 식을 교차 엔트로피 오차에 적용하여 skip-gram 모델의 손실 함수를 유도할 수 있다.

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

위 식에서 알 수 있듯, skip-gram 모델의 손실 함수는 맥락별 손실을 구한 다음 모두 더한다.
위 식은 샘플 데이터 하나짜리 skip-gram 의 손실 함수이다.

이를 말뭉치 전체로 확장하면 skip-gram 모델의 손실 함수는 다음과 같다.

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

위 식을 CBOW 모델의 식과 비교해보자.

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

skip-gram 모델은 맥락의 수만큼 추측하기 때문에,
그 손실 함수는 각 맥락에서 구한 손실의 총합이어야 한다.
반면, CBOW 모델은 타깃 하나의 손실을 구한다.

그렇다면, CBOW 모델과 skip-gram 모델 중 어느 것을 사용해야 할까?
답은 skip-gram.
단어 분산 표현의 정밀도 면에서 skip-gram 모델의 결과가 더 좋은 경우가 많기 때문이다.
특히 말뭉치가 커질수록 저빈도 단어나 유추 문제의 성능 면에서 skip-gram 모델이 더 뛰어난 경향이 있다.
반면, 학습 속도 면에서는 CBOW 모델이 더 빠르다.
skip-gram 모델은 손실을 맥락의 수만큼 구해야 해서 계산 비용이 그만큼 커지기 때문이다.

다행히 CBOW 모델의 구현을 이해할 수 있다면, skip-gram 모델의 구현도 특별히 어려울 게 없다.

통계 기반 vs. 추론 기반

통계 기반 기법에서는 주로 단어의 유사성이 인코딩된다.
한편 word2vec, 특히 skip-gram 모델에서는 단어의 유사성은
물론, 한층 복잡한 단어 사이의 패턴까지도 파악되어 인코딩된다.

추론 기반 기법이 통계 기반 기법보다 정확하다고 흔히 오해하곤 한다.
하지만 단어의 유사성을 정량 평가해본 결과, 추론 기반과 통계 기반 기법의 우열을 가릴 수 없었다고 한다.

추론 기반 기법과 통계 기반 기법은 서로 관련되어 있다.

word2vec 이후 추론 기반 기법과 통계 기반 기법을 융합한 GloVe 기법이 등장했다.
GloVe 의 기본 아이디어는, 말뭉치 전체의 통계 정보를 손실 함수에 도입해 미니배치 학습을 하는 것이다.