

4. word2vec 속도 개선

4.1 word2vec 개선 I

4.2 word2vec 개선 II

4.3 개선판 word2vec 학습

word2vec의 속도 개선하는 법을 알아보겠다.

앞서 3장에서 보았던 CBOW(Continuous Bag of Words) 모델은 처리 효율이 떨어져 말뭉치에 포함된 어휘 수가 많아지면 계산량도 커진다.

따라서, 단순한 word2vec에 두가지 개선을 추가한다.

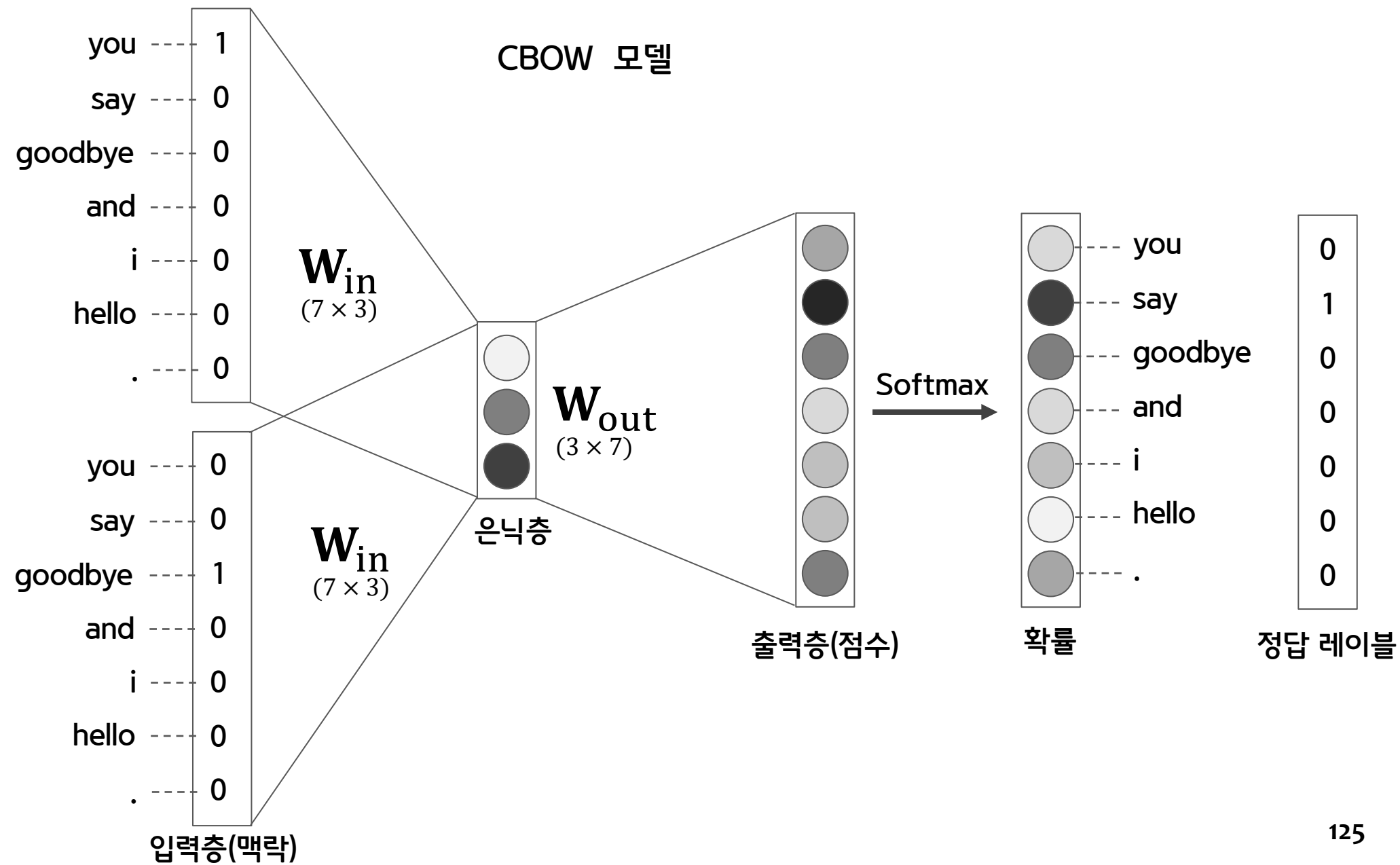
1. Embedding 이라는 새로운 계층을 만든다.
2. 네거티브 샘플링 이라는 새로운 손실함수를 도입한다.

이로써 '진짜'word2vec을 완성할 수 있다.

완성된 word2vec 모델을 가지고 PTB 데이터셋(=실용적인 크기의 말뭉치)를 가지고 학습을 수행하고, 결과를 평가해 보도록 하자.

CBOW 모델은, 복수 단어 문맥에 대한 문제 즉, 여러개의 단어를 나열한 뒤 이와 관련된 단어를 추정하는 문제이다.

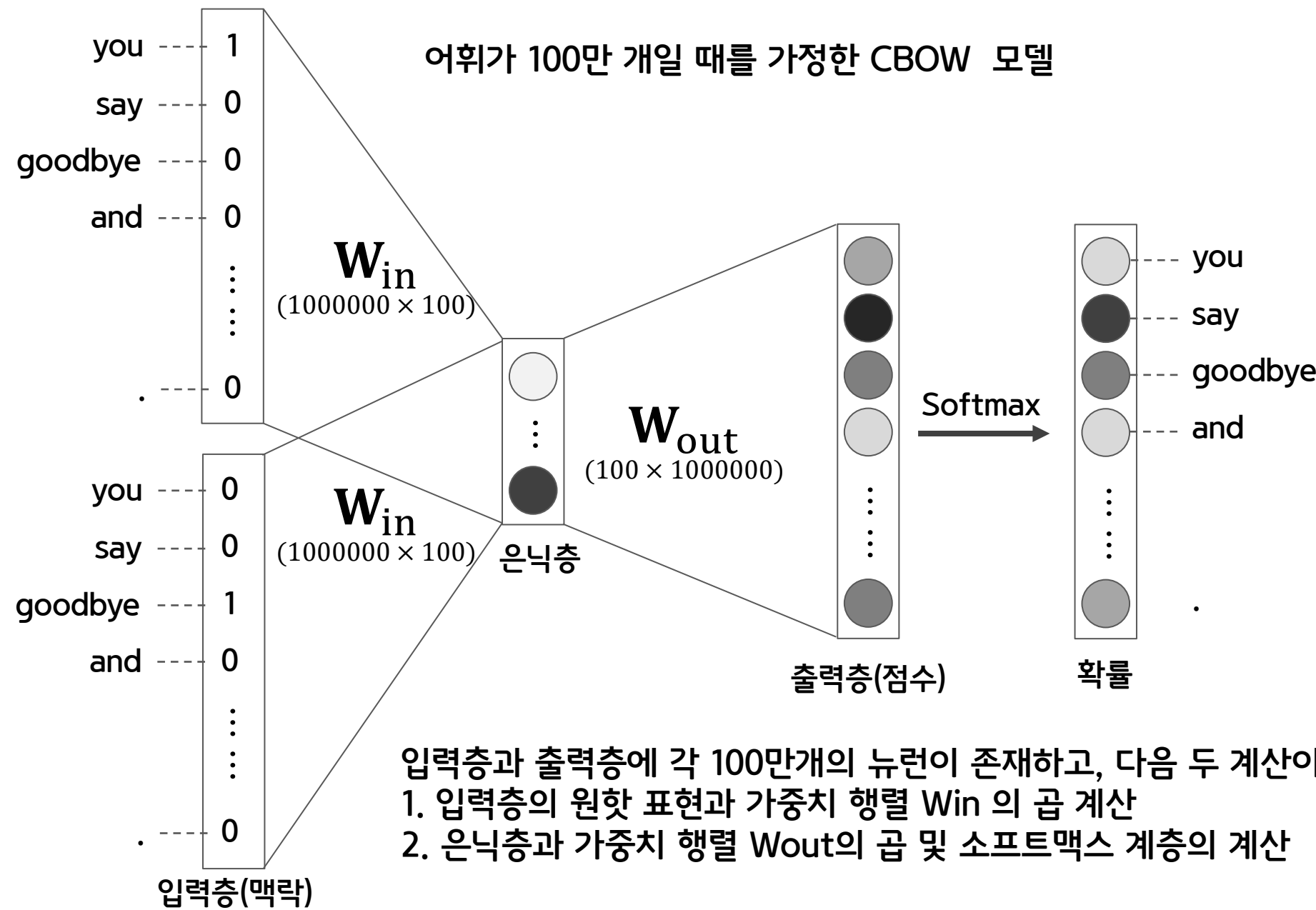
즉, 문자에서 나오는 n 개의 단어 열로부터 다음 단어를 예측하는 모델이다.



입력 층 가중치와 행렬 곱으로 은닉층이 계산되고,
다시 출력층 가중치와의 행렬 곱으로 각 단어 점수를 계산,
소프트맥스 함수를 적용해 각 단어의 출현 확률을 얻어 정답 레이블과 비교하여 손실을 구한다.

위의 그림은 다루는 어휘가 7개일 때이다.

만약 어휘가 100만개, 은닉층의 뉴런이 100개인 CBOW 모델을 생각해 보면,



Embedding이란, 텍스트를 구성하는 하나의 단어를 수치화하는 방법의 일종이다.
 텍스트 분석에서 흔히 사용하는 방식은 단어 하나에 인덱스 정수를 할당하는 Bag of Words 방법이다.
 이 방법을 사용하면 문서는 단어장에 있는 단어의 갯수와 같은 크기의 벡터가 되고 단어장의
 각 단어가 그 문서에 나온 횟수만큼 벡터의 인덱스 위치의 숫자를 증가시킨다.

즉 단어장이 "I", "am", "a", "boy", "girl" 다섯개의 단어로 이루어진 경우 각 단어에 다음과 같이 숫자를 할당한다.

"I": 0

"am": 1

"a": 2

"boy": 3

"girl": 4

이 때 "I am a girl" 이라는 문서는 다음과 같이 벡터로 만들 수 있다.

[1, 1, 1, 0, 1]

단어 임베딩은 하나의 단어를 하나의 인덱스 정수가 아니라 실수 벡터로 나타낸다.

예를 들어 2차원 임베딩을 하는 경우 다음과 같은 숫자 벡터가 될 수 있다.

"I": (0.3, 0.2)

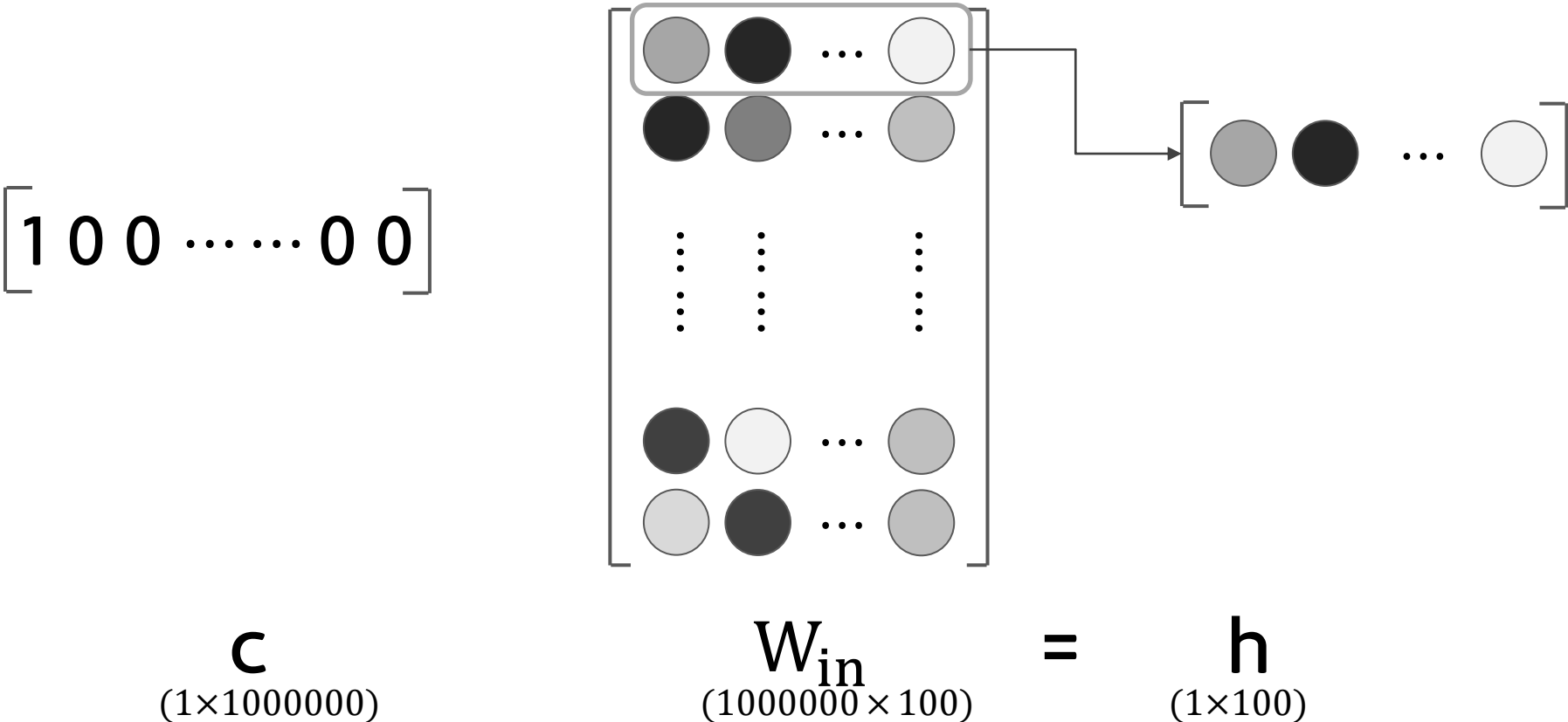
"am": (0.1, 0.8)

"a": (0.5, 0.6)

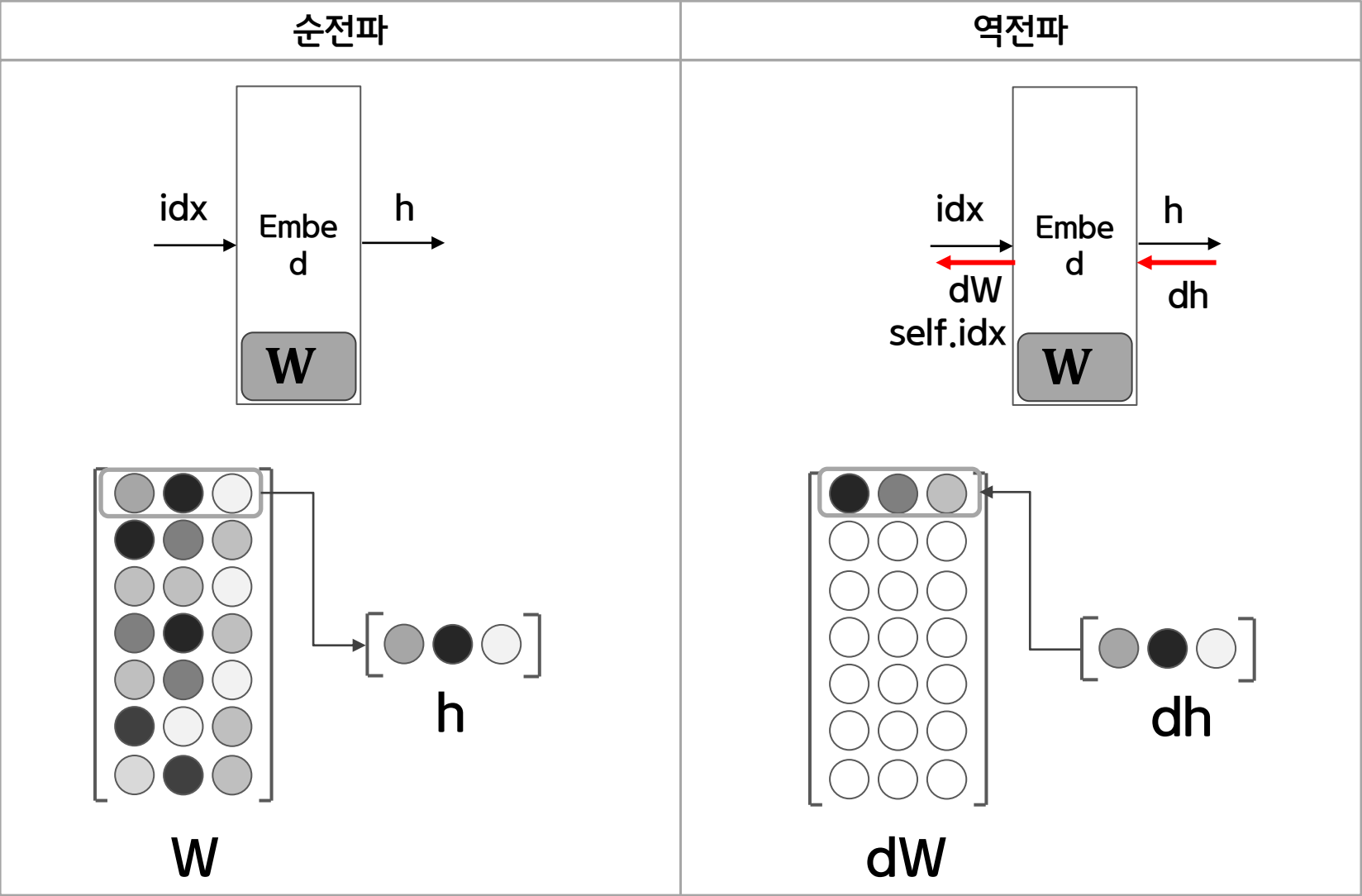
"boy": (0.2, 0.9)

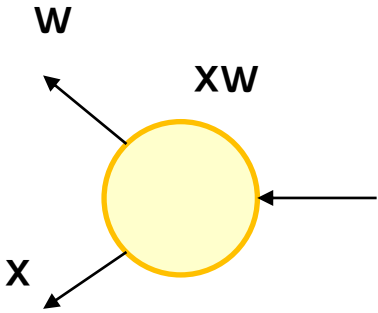
"girl": (0.4, 0.7)

맥락(원핫 표현)과 MatMul 계층의 가중치를 곱한다.



Embedding 계층의 forward와 backward 처리



$$W = \frac{\partial L}{\partial w}$$


$$X^T dh$$

(10,1) (1,3)

(10,3)

1

0

0

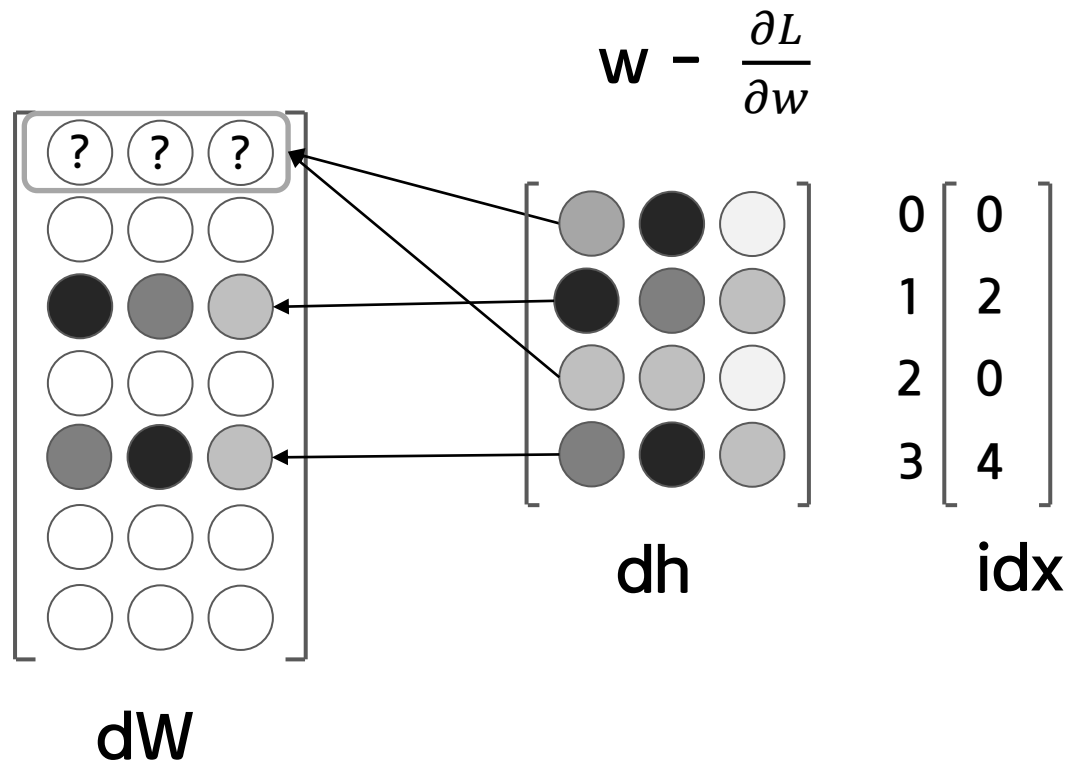
0

0

0

(a,b,c)

idx 배열의 원소 중 값(행 번호)이 같은 원소가 있다면, dh를 해당 행에 할당할 때 문제가 생긴다.



4. word2vec 속도 개선

4.1 word2vec 개선 I

4.2 word2vec 개선 II

4.3 개선판 word2vec 학습

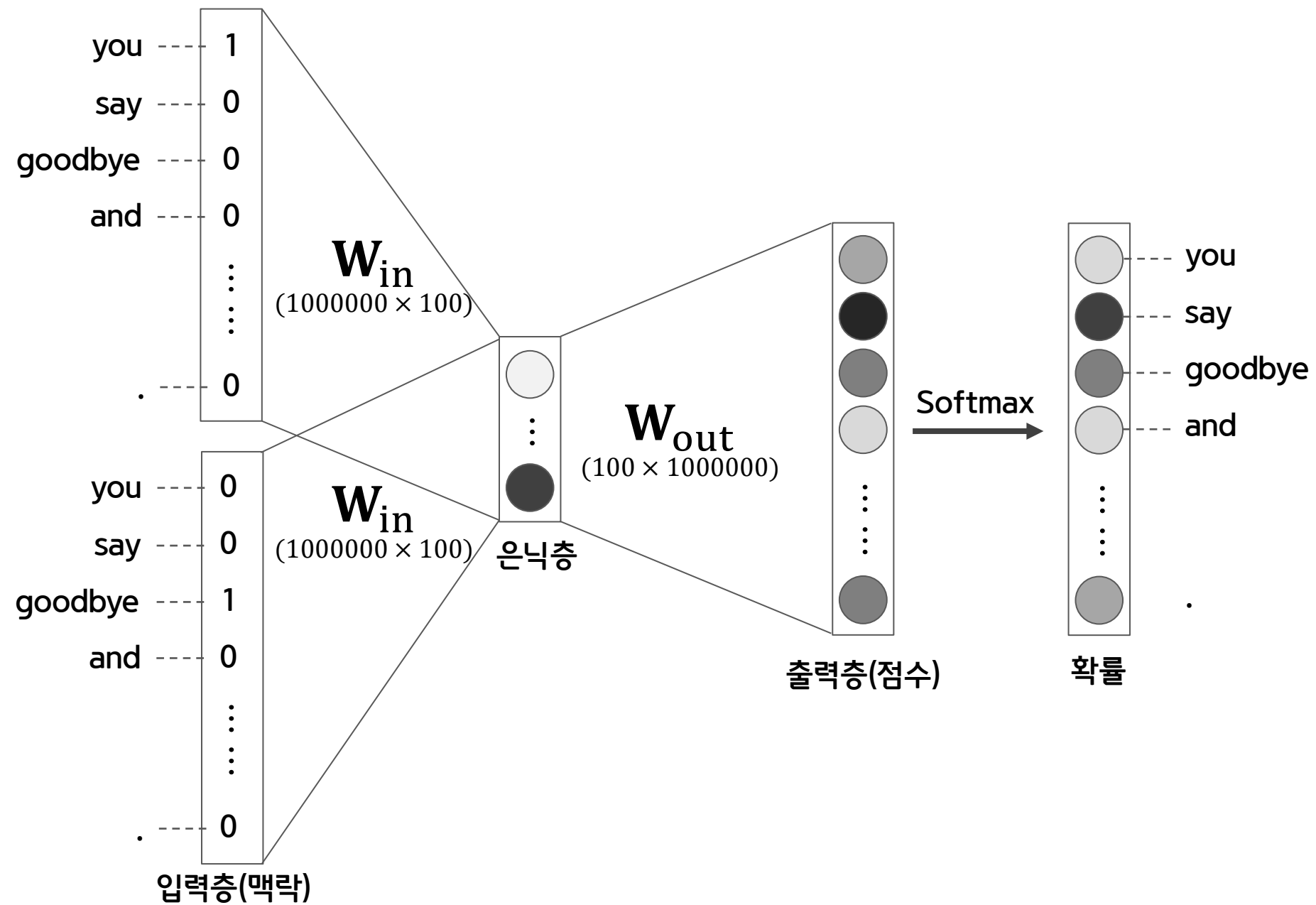
남은 병목은 은닉층 이후의 처리(행렬 곱과 softmax 계층의 계산) 이다.

어휘가 100만개일 때를 가정한 word2vec 모델이 있었는데,
은닉층의 뉴런과 가중치 행렬의 곱을 할때,
크기가 100인 은닉층 뉴런과 100*100만인 가중치 행렬을 곱해야 하고,
역전파 때도 같은 계산을 수행한다.

또한, 소프트맥스의 계산량도 exp 계산만 100만번 수행해야 한다. 따라서,

1. 은닉층의 뉴런과 가중치 행렬의 곱
2. 소프트맥스 계층의 계산

을 가볍게 해야한다.



어휘가 많아지면 Softmax의 계산량이 증가한다.

$$y_k = \frac{\exp(s_k)}{\sum_{i=1}^{1000000} \exp(s_i)}$$

네거티브 샘플링

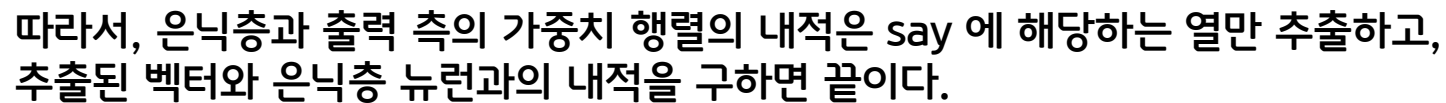
네거티브 샘플링의 핵심은 다중분류를 이진분류로 근사하는 것이다.

예를 들면,

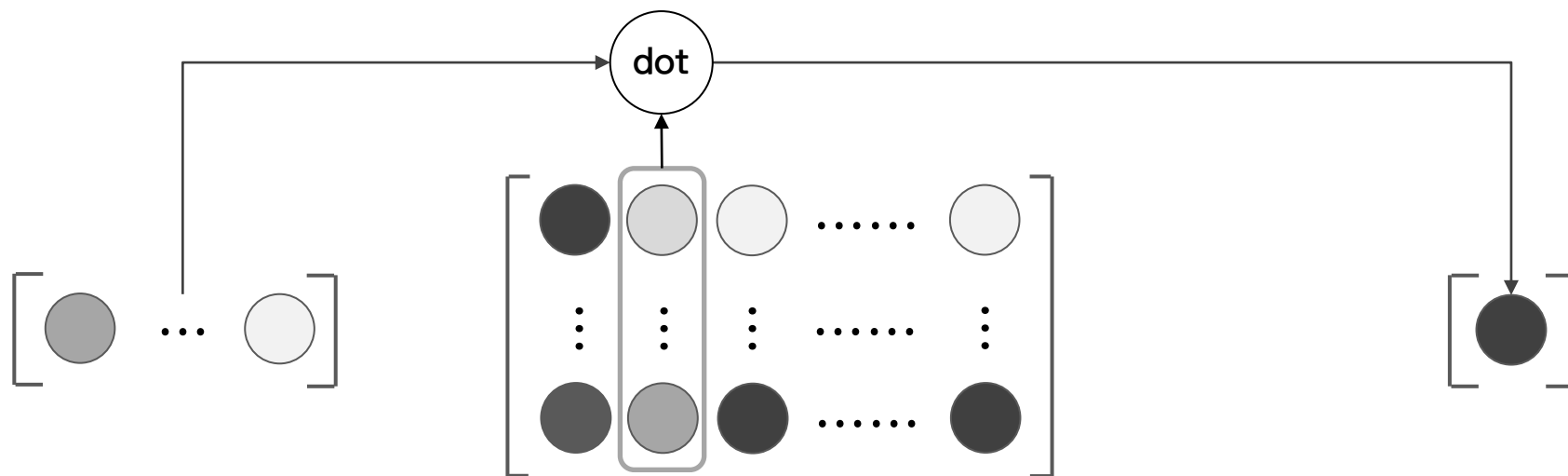
다중 분류는 맥락이 you 와 goodbye 일때, 타깃 단어는 무엇입니까?
에 대답하는 것이고,

이진 분류는 맥락이 you 와 goodbye 일때, 타깃 단어는 say 입니까?
에 대답하는 것이다.

이런식으로 하면 출력층에 뉴런을 하나만 준비하면 된다.
출력층의 이 뉴런이 say 의 점수를 출력하는 것이다.



"say"에 해당하는 열벡터와 은닉층 뉴런의 내적을 계산한다.

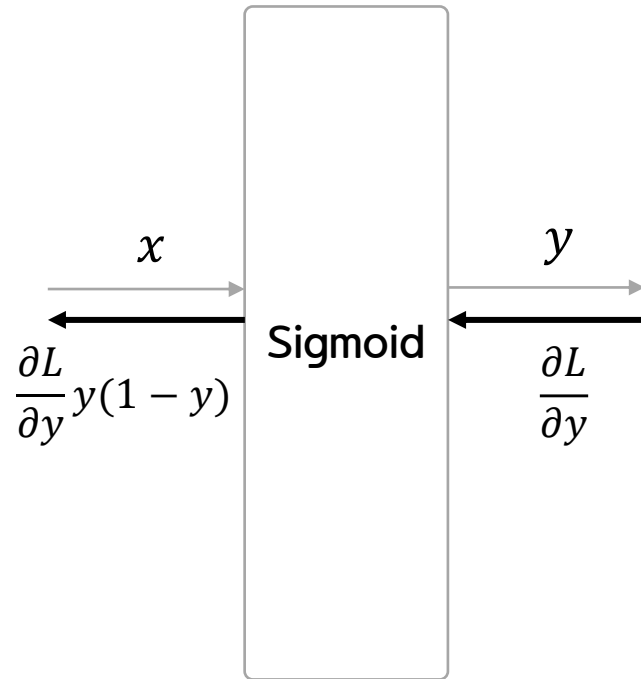


형상 : h
(1×100)

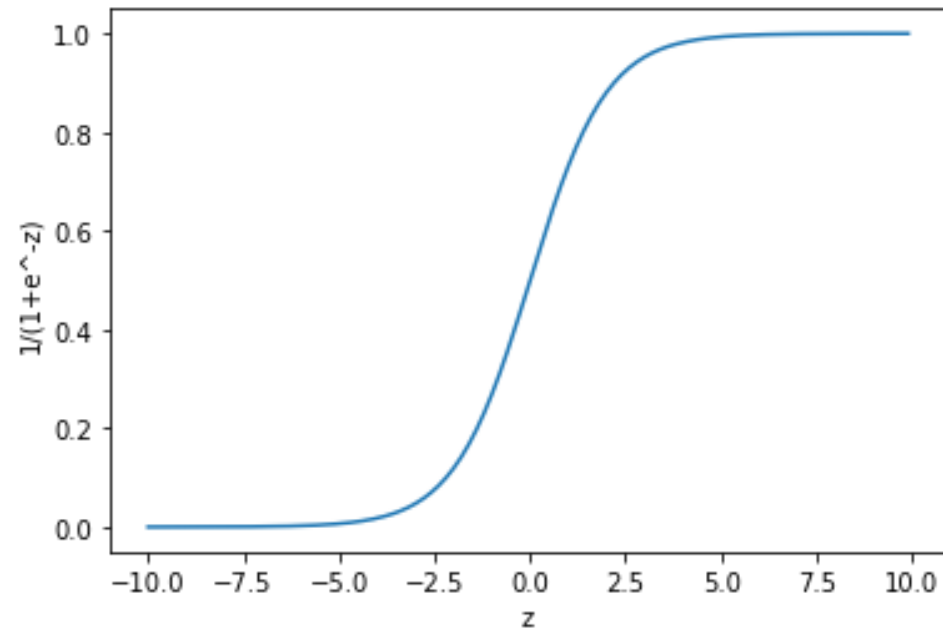
W_{out}
(100 × 1000000)

s
(1×1)

Sigmoid 계층과 시그모이드 함수의 그래프

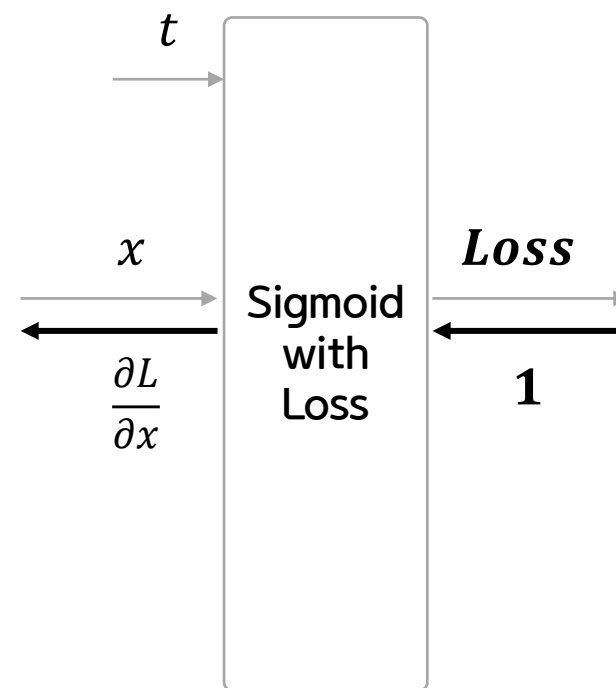
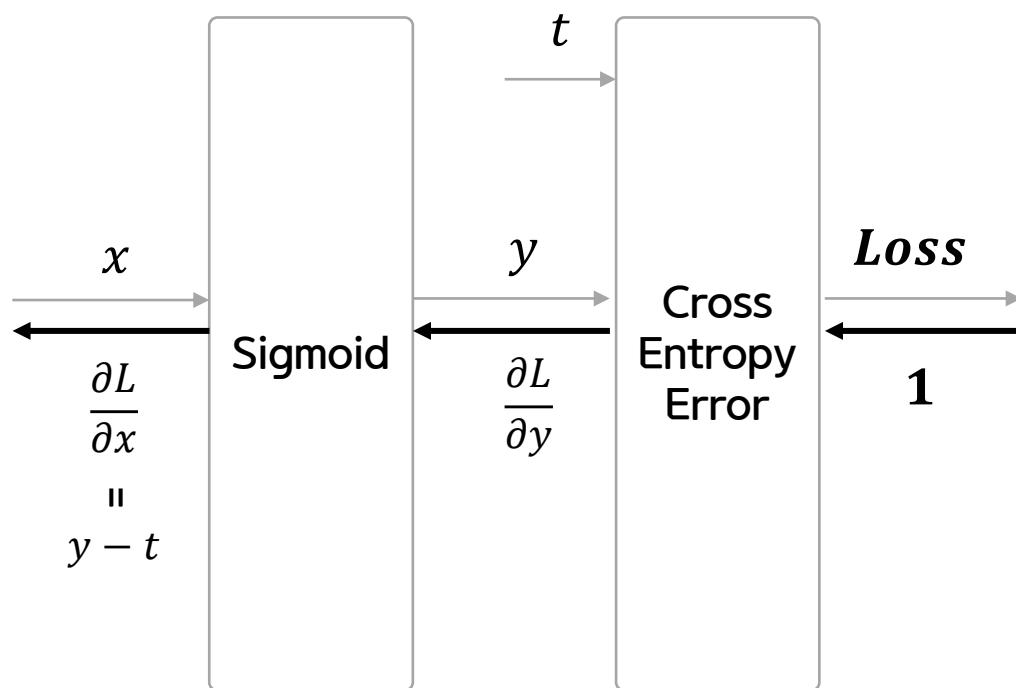


$$y = \frac{1}{1 + e^{-x}}$$

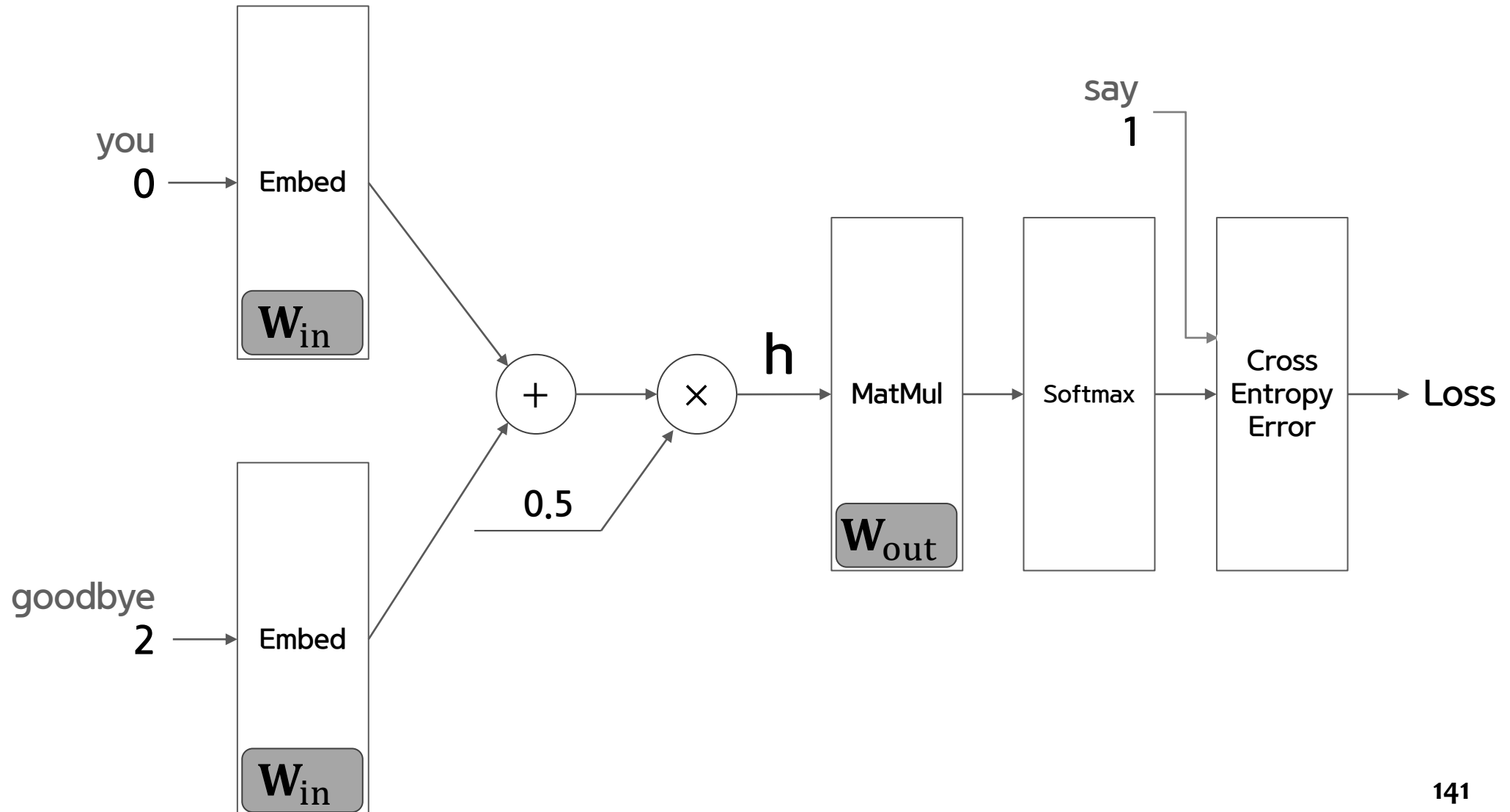


Sigmoid 계층과 Cross Entropy Error 계층의 계산 그래프

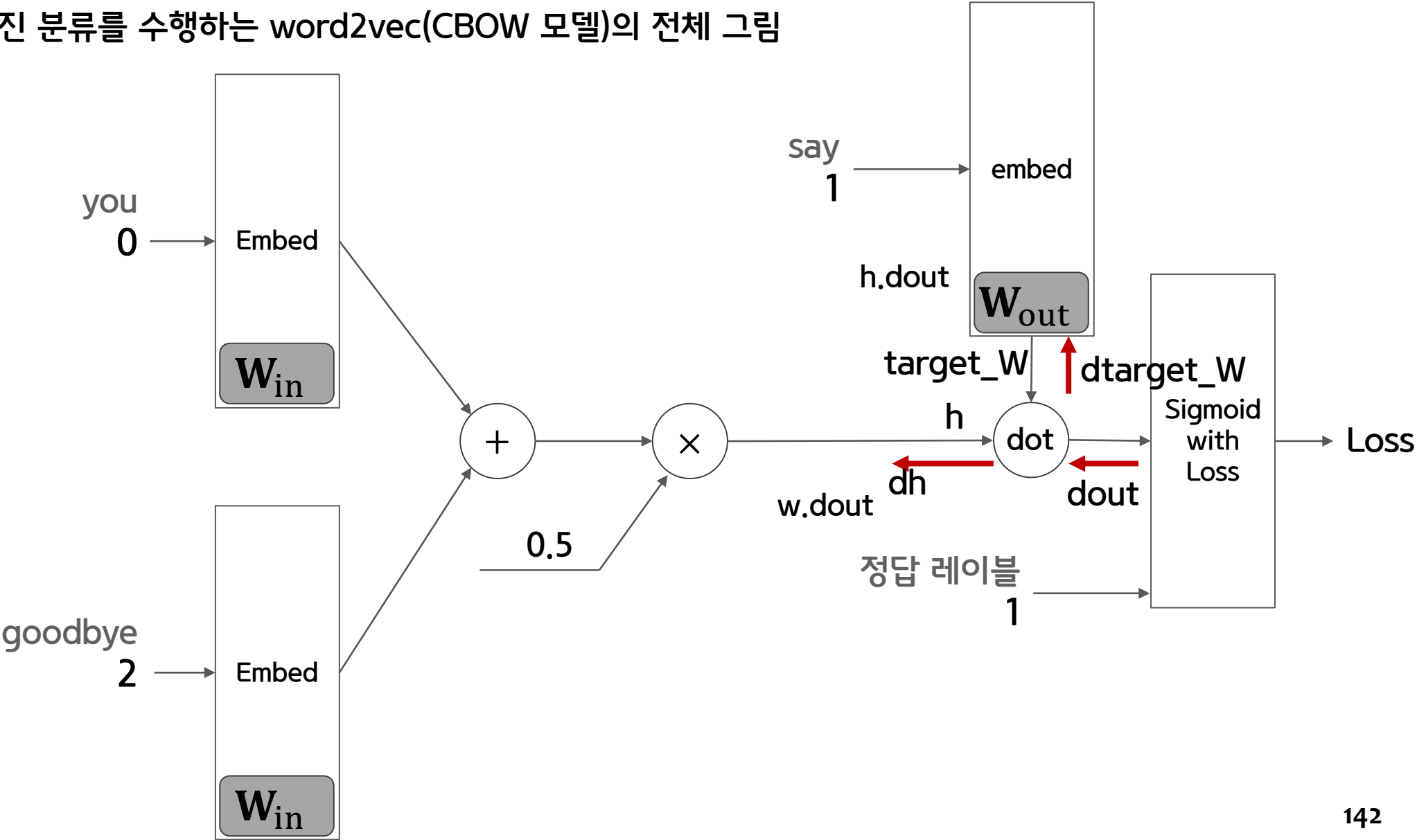
$$L = -(t \log y + (1 - t) \log(1 - t))$$



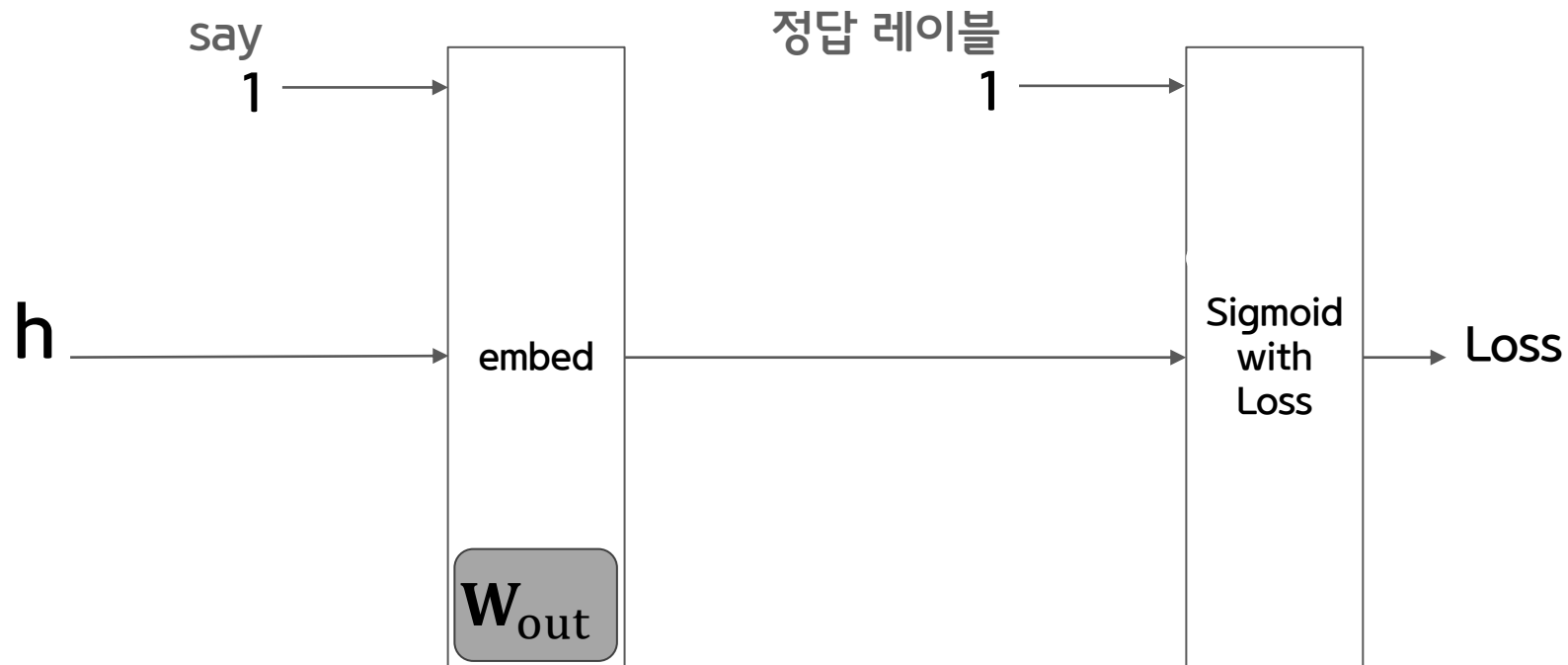
다중 분류를 수행하는 CBOW 모델의 전체 그림



이진 분류를 수행하는 word2vec(CBOW 모델)의 전체 그림



은닉층 이후 처리(Embedding Dot 계층을 사용하여 Embedding 계층과 내적 계산을 한 번에 수행)

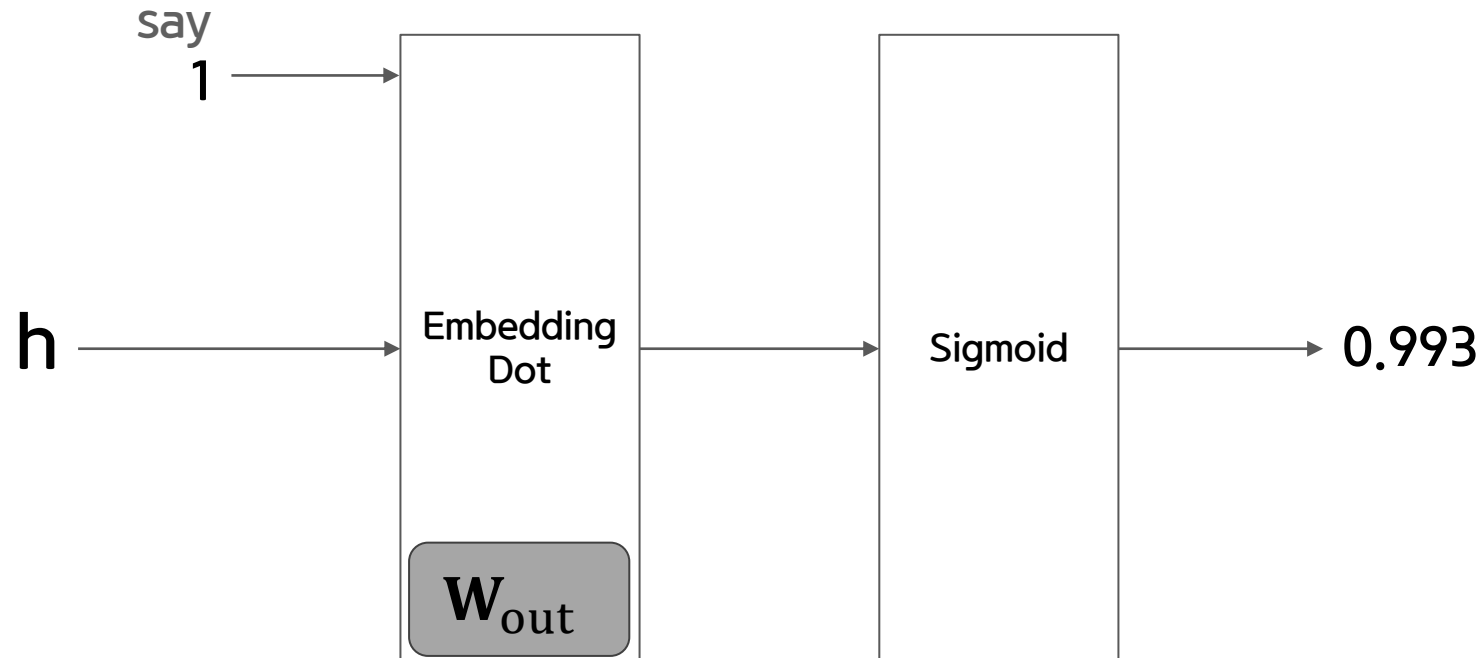


Embedding Dot 계층의 각 변수의 구체적인 값

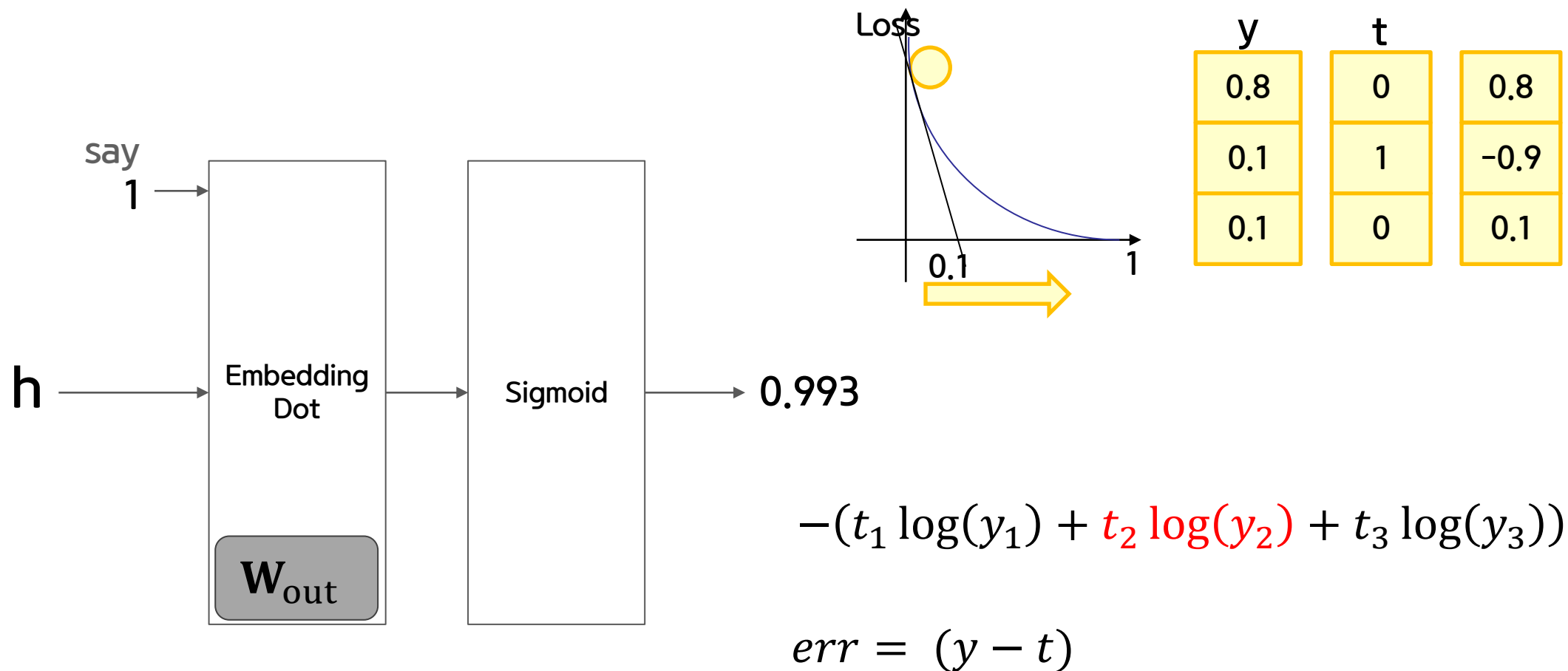
```
embed = Embedding(W)
target_W = embed.forward(idx)
out = np.sum(target_W*h, axis=1)
```

W	idx	target_W	h	target_W * h	out
<div>[[0 1 2] [3 4 5] [6 7 8] [9 10 11] [12 13 14] [15 16 17] [18 19 20]]</div>	<div>[[0 3 1]]</div>	<div>[[0 1 2] [9 10 11] [3 4 5]]</div>	<div>[[0 1 2] [3 4 5] [6 7 8]]</div>	<div>[[0 1 4] [27 40 55] [18 28 40]]</div>	<div>[[5 122 86]]</div>

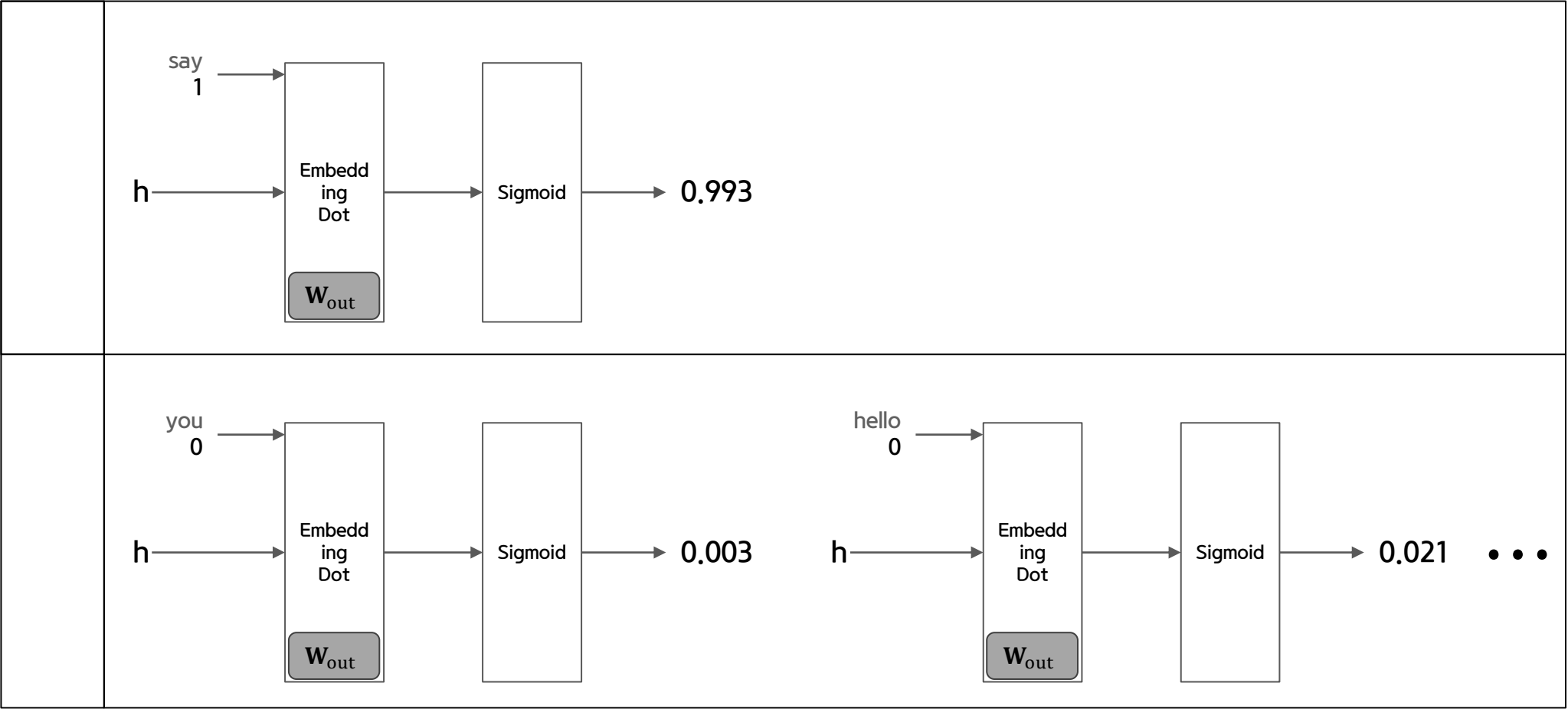
CBOW 모델의 은닉층 이후의 처리 예: 맥락은 "you"와 "goodbye"이고, 타겟이 "say"일 확률은 0.993(99.3%)이다.



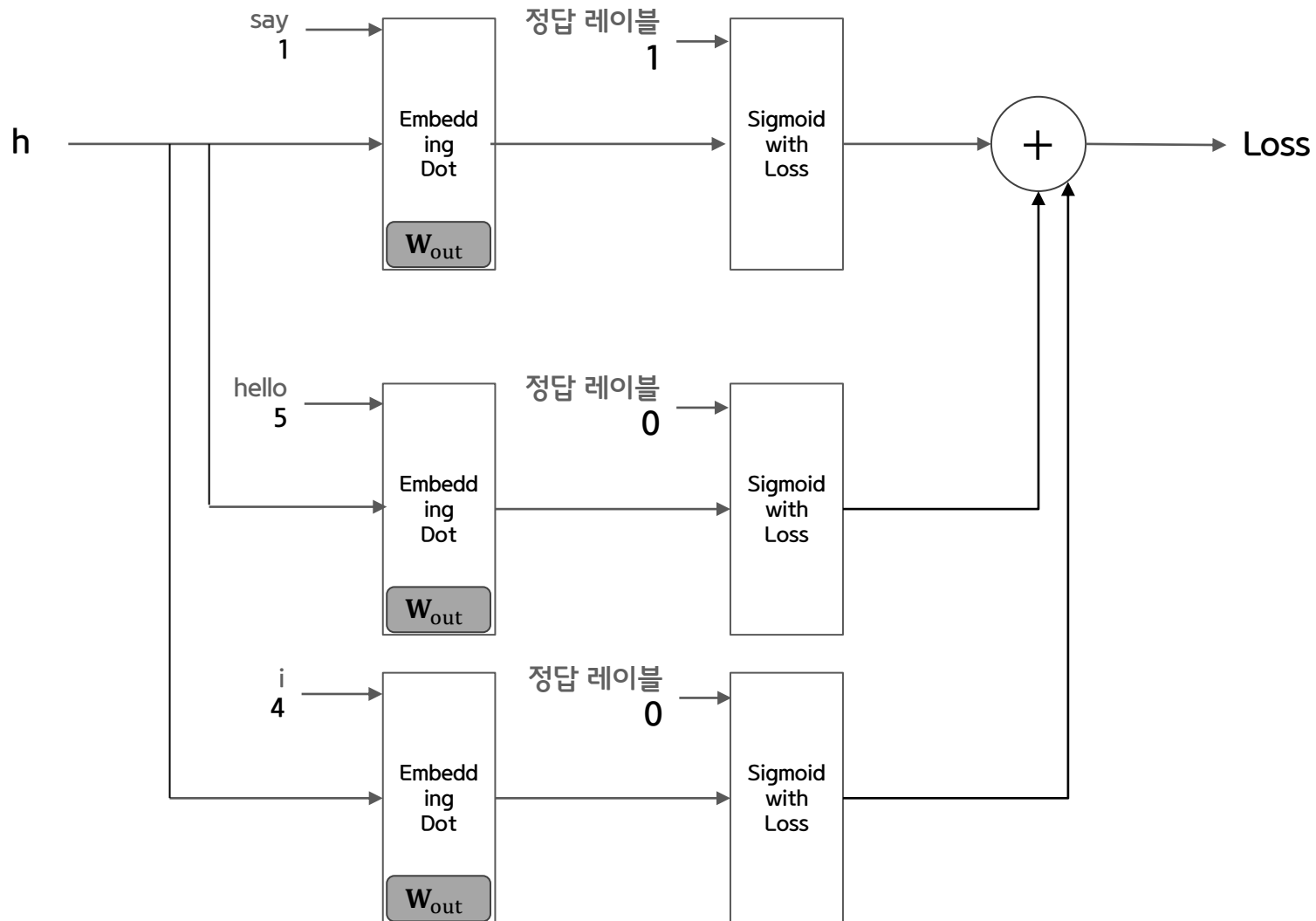
CBOW 모델의 은닉층 이후의 처리 예: 맥락은 "you"와 "goodbye"이고, 타겟이 "say"일 확률은 0.993(99.3%)이다.



긍정적 예(정답)를 "say"라고 가정하면, "say"를 입력했을 때의 Sigmoid 계층 출력은 1에 가깝고, "say" 이외의 단어를 입력했을 때의 출력은 0에 가까워야 한다. 이런 결과를 내어주는 가중치가 필요하다.



네거티브 샘플링의 예



```

correct_label = np.ones(batch_size, dtype=np.int32)
loss = self.loss_layers[0].forward(score, correct_label)

self.y = 1 / (1 + np.exp(-x))
self.loss = cross_entropy_error(np.c_[1 - self.y, self.y], self.t)

def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    # 정답 데이터가 원핫 벡터일 경우 정답 레이블 인덱스로 변환
    if t.size == y.size:
        t = t.argmax(axis=1)

    batch_size = y.shape[0]

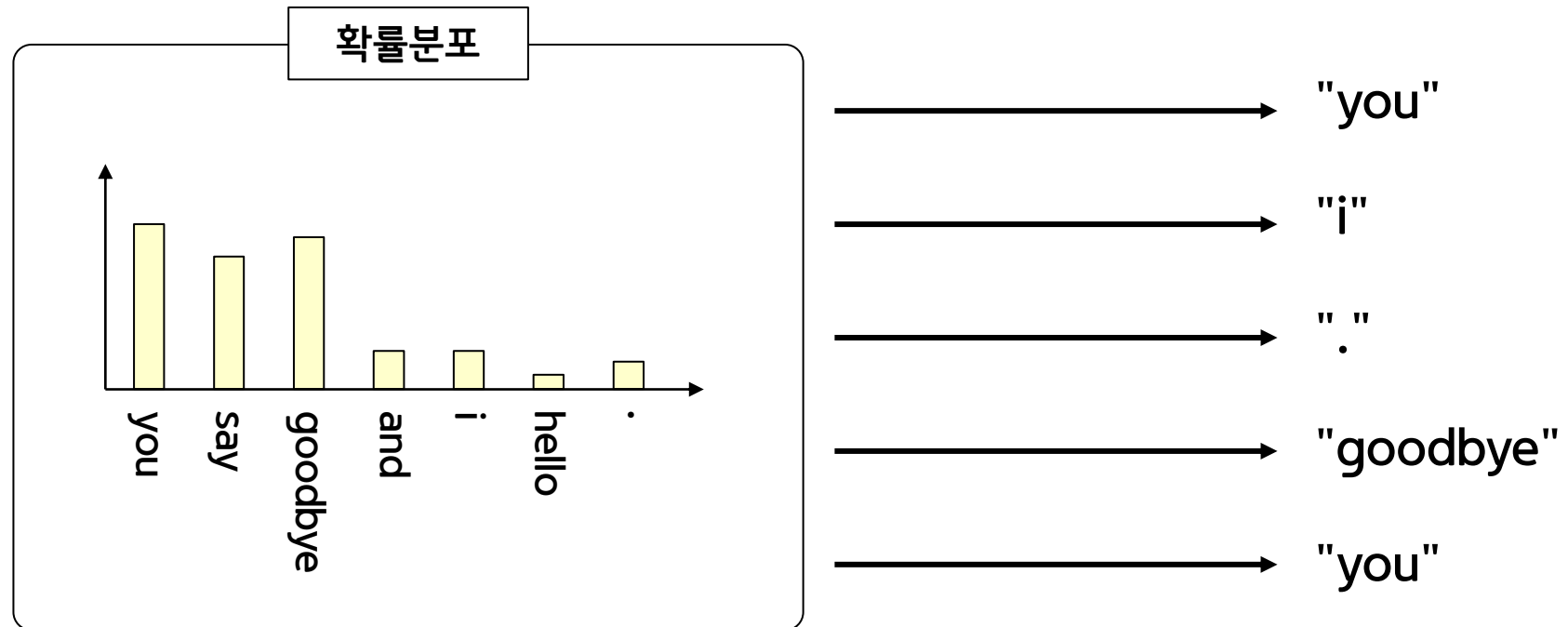
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size

```

1-y	y	correct_label
0.1	0.9	1

$$-(t \log(y) + (1 - t) \log(1 - y))$$

확률분포에 따라 샘플링을 여러 번 수행한다.



$$P'(w_i) = \frac{P(w_i)^{0.75}}{\sum_j^n P(w_j)^{0.75}}$$

```
p = [0.7, 0.29, 0.01]
new_p = np.power(p, 0.75)
new_p /= np.sum(new_p)
print(new_p)
[0.64196878 0.33150408 0.02652714]
```

0.7	0.29	0.01
-----	------	------

0.76	0.39	0.03
------	------	------

0.64	0.33	0.02
------	------	------

※ 코드 참조

4. word2vec 속도 개선

4.1 word2vec 개선 I

4.2 word2vec 개선 II

4.3 개선판 word2vec 학습

임베딩 계층과 네거티브 샘플링 기법을 사용하여 개선된 신경망 모델에 PTB 데이터셋을 사용해 학습시키고, 실용적인 단어의 분산 표현을 얻어보겠다.

다음은 개선된 CBOW 모델 코드다.

앞의 단순한 CBOW 모델에서 임베딩 계층과 네거티브 샘플링 손실함수 계층을 적용했다.

※ 코드 참조

맥락과 타깃을 단어 ID로 나타낸 예(맥락의 윈도우 크기는 1)

맥락(contexts)	타깃		맥락(contexts)	타깃
you, goodbye	say	단어 ID →	[[0 2]	[[1
say, and	goodbye		[1 3]	2
goodbye, i	and		[2 4]	3
and, say	i		[3 1]	4
i, hello	say		[4 5]	1
say, .	hello		[1 6]]	5]

※ 코드 참조

※ 코드 참조

"man : woman = king : ?" 유추 문제 풀기(단어 벡터 공간에서 각 단어의 관계성)

