

자연어처리

목차

1. 신경망 복습
2. 자연어와 단어의 분산 표현
3. word2vec
4. word2vec 속도 개선
5. 순환 신경망(RNN)
6. 게이트가 추가된 RNN
7. RNN을 사용한 문장 생성

1. 신경망 복습

1.1 수학과 파이썬 복습

1.2 신경망 추론

1.3 신경망의 학습

1.4 신경망으로 문제를 푼다.

벡터와 행렬의 예

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

1	2
3	4
5	6

행

열

벡터의 표현법

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

열벡터

$$[1 \quad 2 \quad 3]$$

행벡터

브로드캐스트의 예1 : 스칼라 값인 10이 2x2 행렬로 처리된다.

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|} \hline 10 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 10 & 10 \\ \hline 10 & 10 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 10 & 20 \\ \hline 30 & 40 \\ \hline \end{array}$$

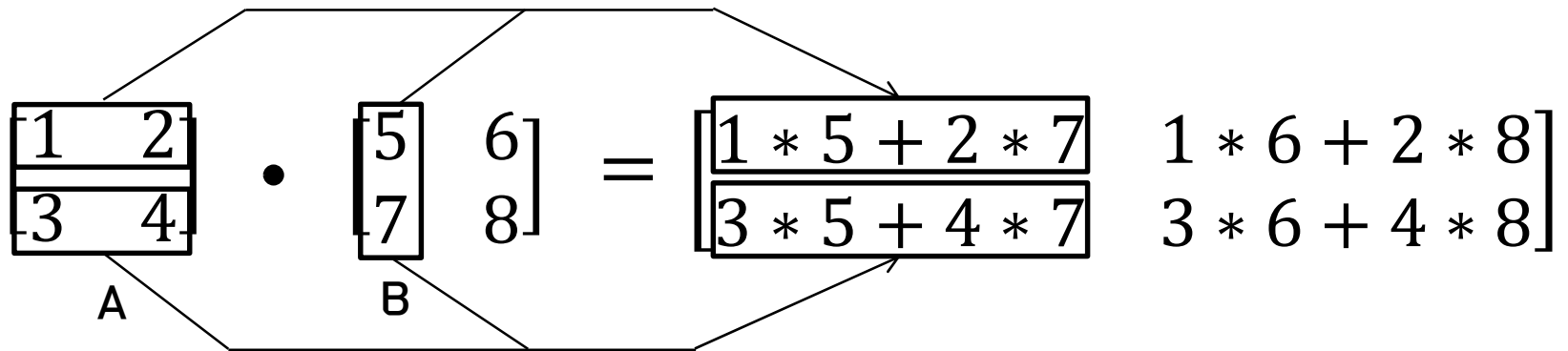
브로드캐스트의 예2 : 일차원 배열인 10,20이 2x2 행렬로 처리된다.

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 10 & 20 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 10 & 20 \\ \hline 10 & 20 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 10 & 40 \\ \hline 30 & 80 \\ \hline \end{array}$$

벡터의 내적

$$x \cdot y = x_1y_1 + x_2y_2 + \cdots + x_ny_n$$

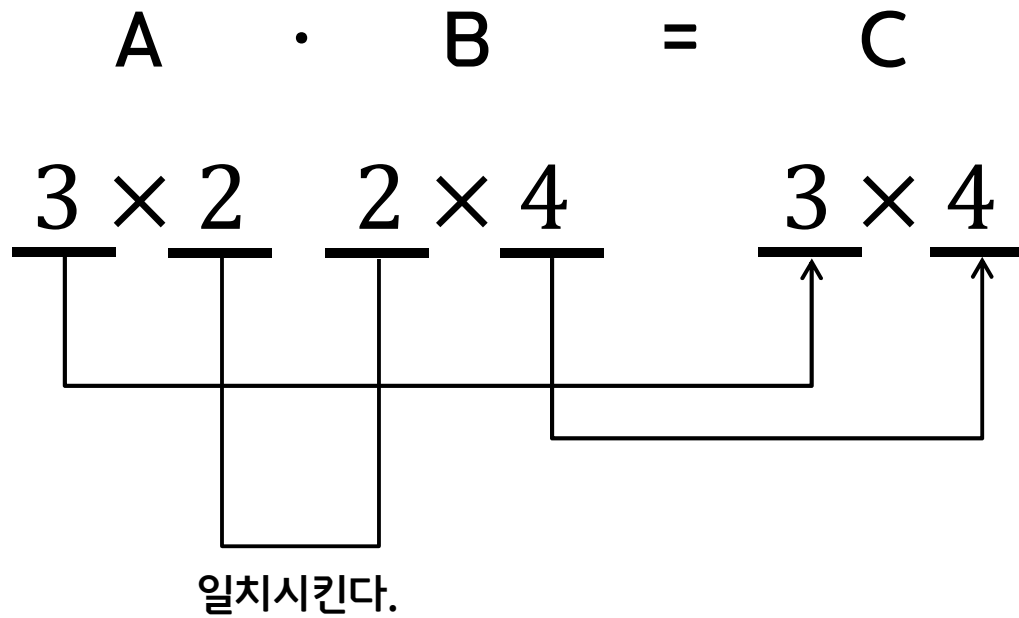
행렬의 곱



$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 * 5 + 2 * 7 & 1 * 6 + 2 * 8 \\ 3 * 5 + 4 * 7 & 3 * 6 + 4 * 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

형상 확인 : 행렬의 곱에서는 대응하는 차원의 원소 수를 일치시킨다.



1. 신경망 복습

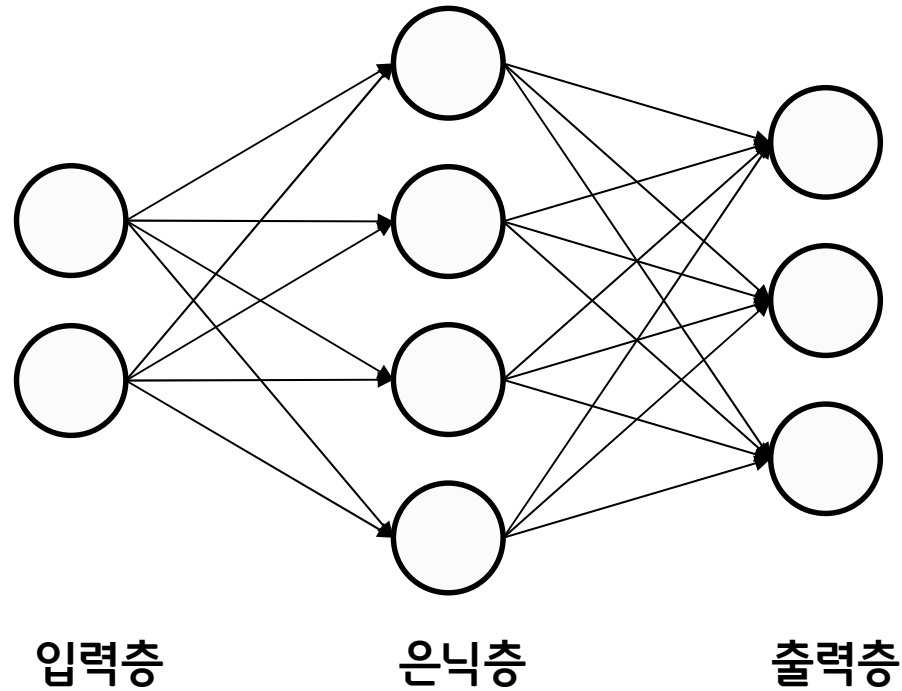
1.1 수학과 파이썬 복습

1.2 신경망 추론

1.3 신경망의 학습

1.4 신경망으로 문제를 푼다.

신경망의 예



신경망이 수행하는 계산 수식(단일층)

$$h_1 = x_1 w_{11} + x_2 w_{21} + b_1$$

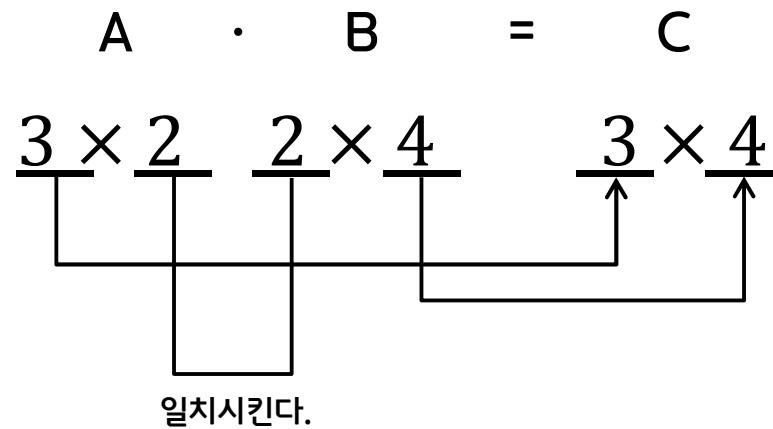
신경망이 수행하는 계산 수식(여러층)

$$(h_1, h_2, h_3, h_4) = (x_1, x_2) \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix} + (b_1, b_2, b_3, b_4)$$

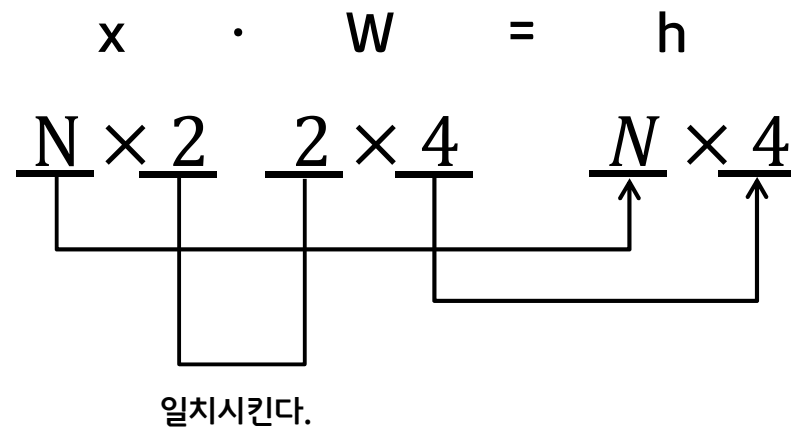


$$h = xW + b$$

형상 확인 : 대응하는 차원의 원소 수가 일치함(편향은 생략)

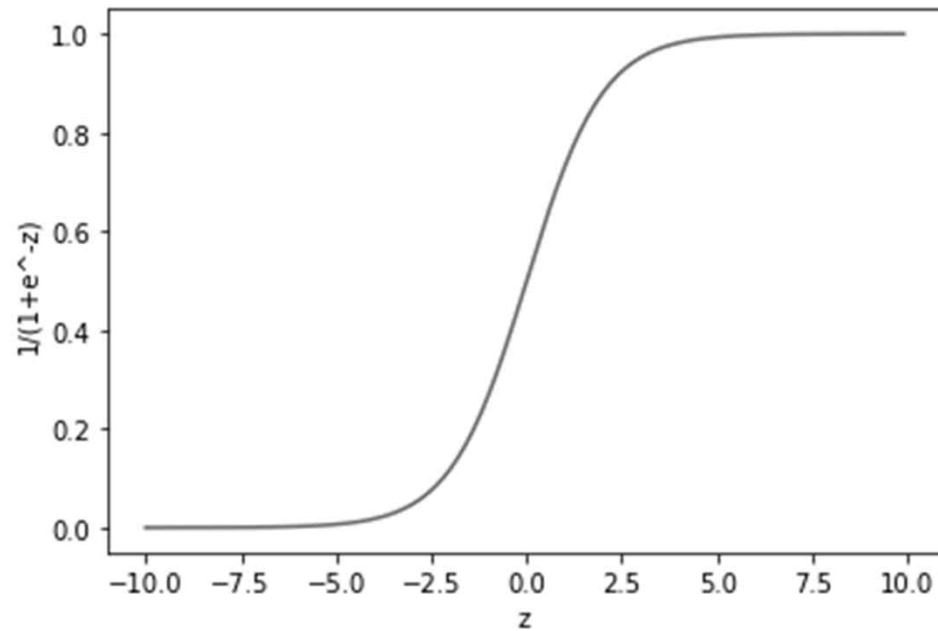


형상 확인 : 미니배치 버전의 행렬 곱(편향은 생략)



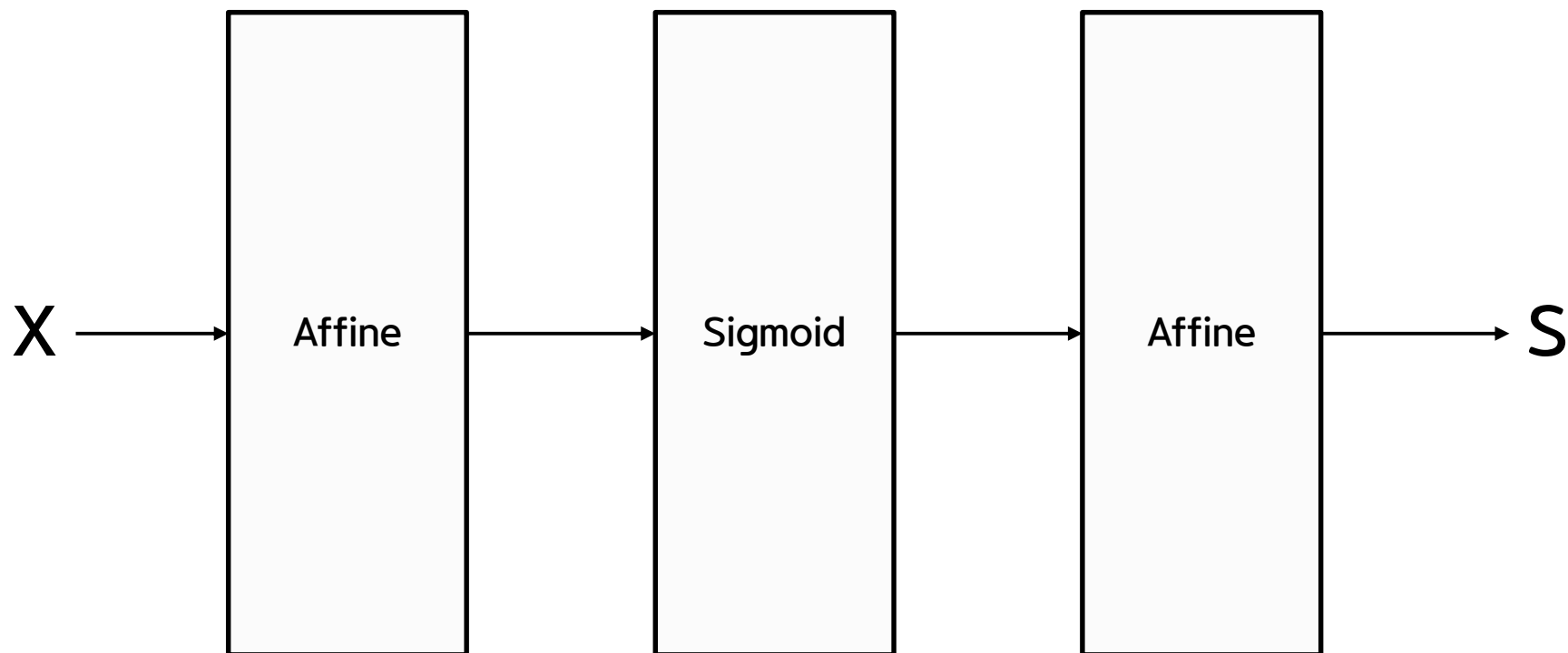
시그모이드 함수

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



- 모든 계층은 `forward()`와 `backward()` 메서드를 가진다.
- 모든 계층은 인스턴스 변수인 `param`와 `grads`를 가진다.

구현해볼 신경망의 계층 구성



1. 신경망 복습

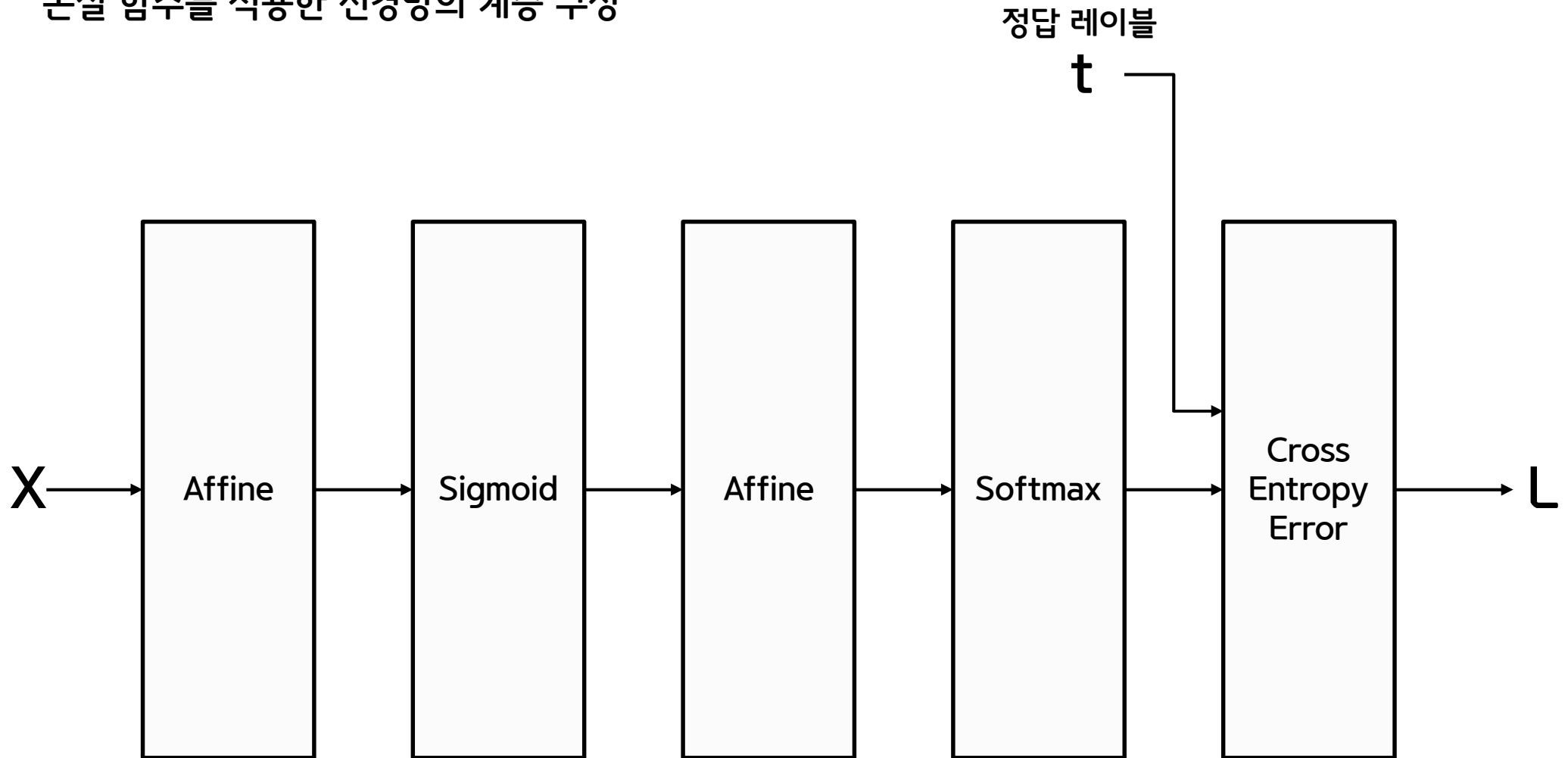
1.1 수학과 파이썬 복습

1.2 신경망 추론

1.3 신경망의 학습

1.4 신경망으로 문제를 푼다.

손실 함수를 적용한 신경망의 계층 구성



소프트맥스 함수 수식

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

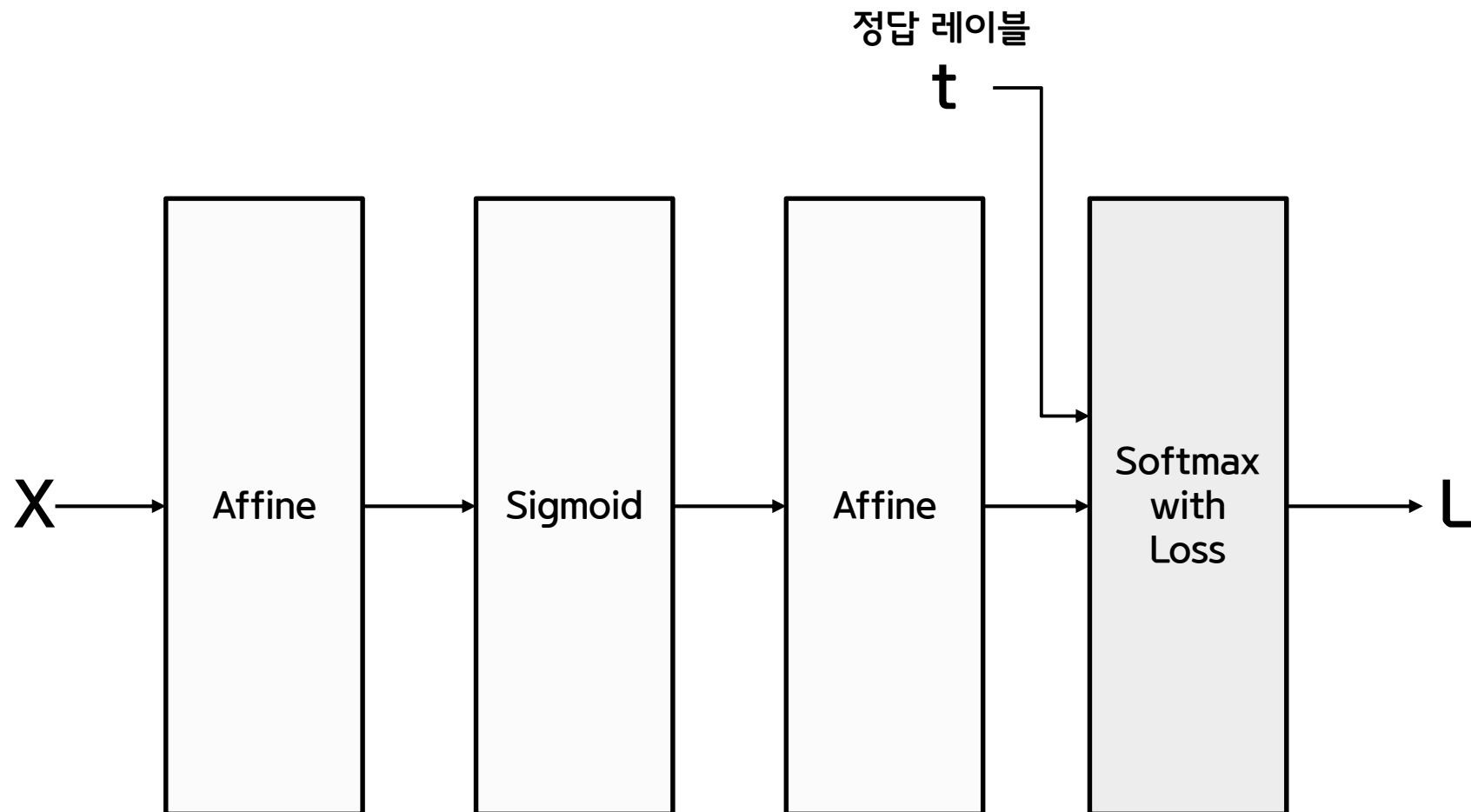
크로스 엔트로피 오차의 수식

$$L = - \sum_k t_k \log y_k$$

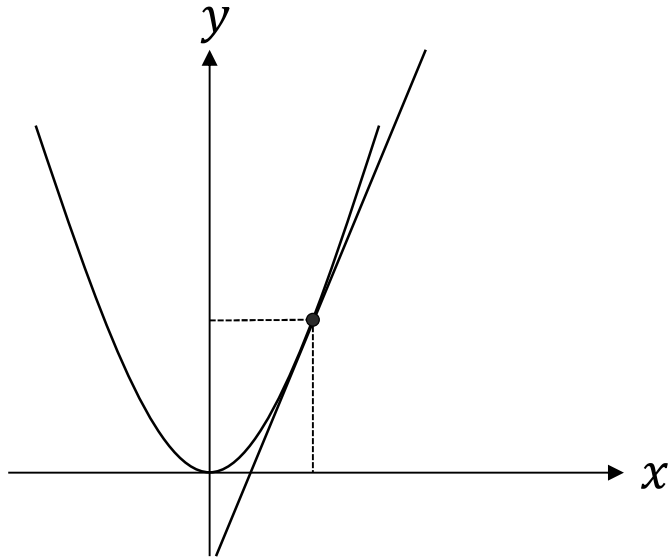
미니배치 처리를 고려한 크로스 엔트로피 오차의 수식

$$L = - \frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk}$$

Softmax with Loss 계층을 이용하여 손실을 출력



$y = x^2$ 의 미분은 각 x 에서의 기울기를 나타낸다.



벡터인 경우 미분

$$\frac{\partial L}{\partial \mathbf{x}} = \left(\frac{\partial L}{\partial x_1}, \frac{\partial L}{\partial x_2}, \dots, \frac{\partial L}{\partial x_n} \right)$$

행렬인 경우 미분

$$\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial W_{11}} & \cdots & \frac{\partial L}{\partial W_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial W_{m1}} & \cdots & \frac{\partial L}{\partial W_{m n}} \end{pmatrix}$$

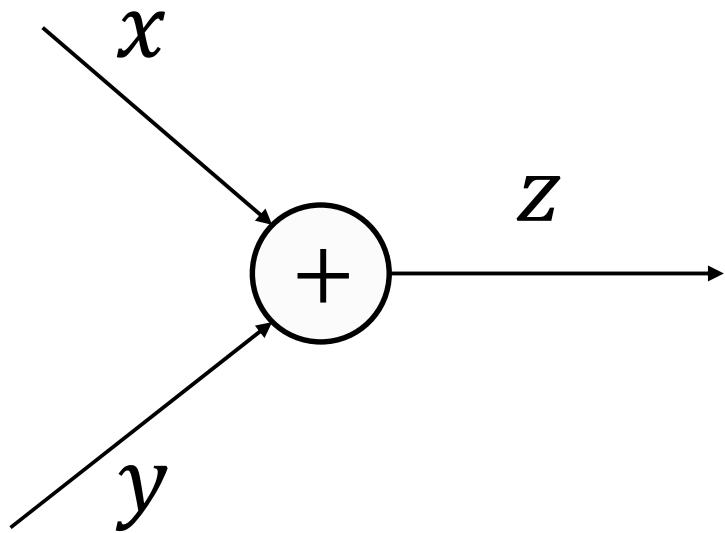
연쇄 법칙

$$\begin{aligned} y &= f(x) \\ z &= g(y) \end{aligned} \qquad z = g(f(x))$$

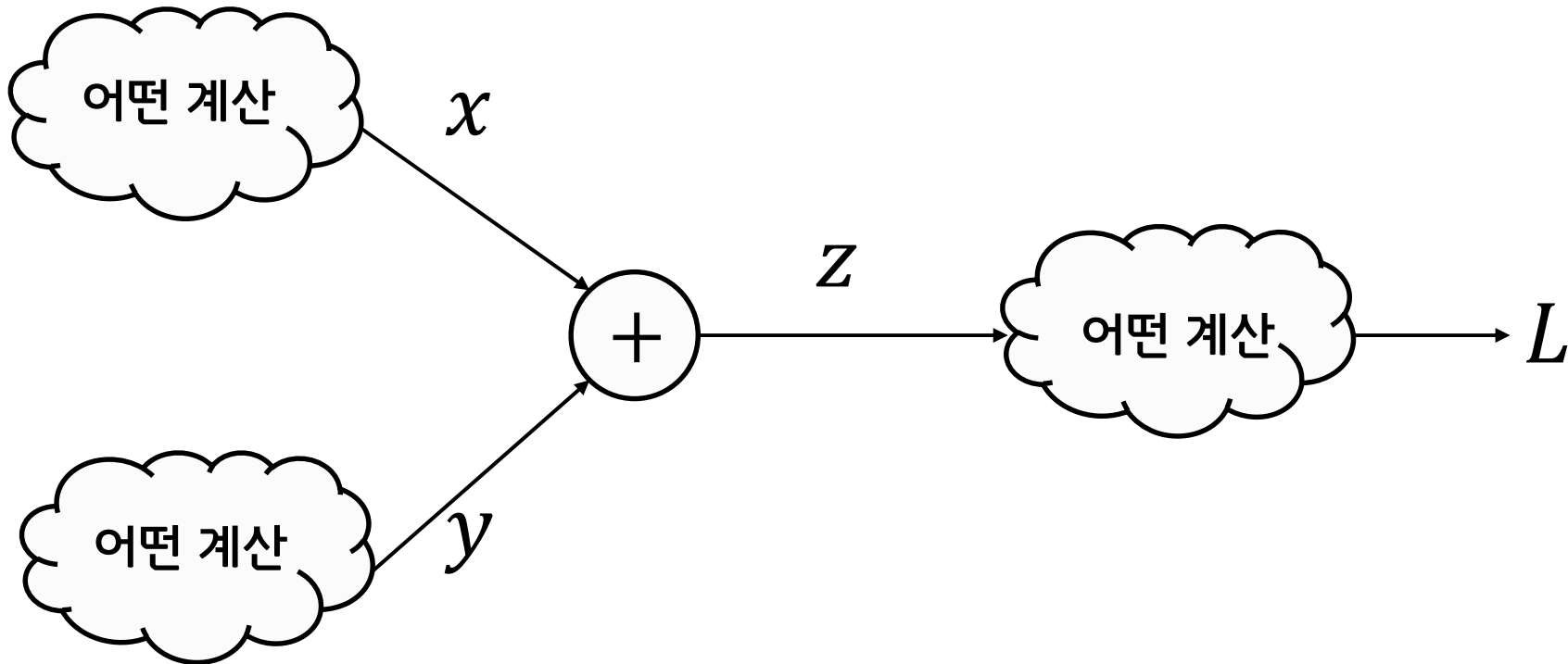
합성함수의 미분

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

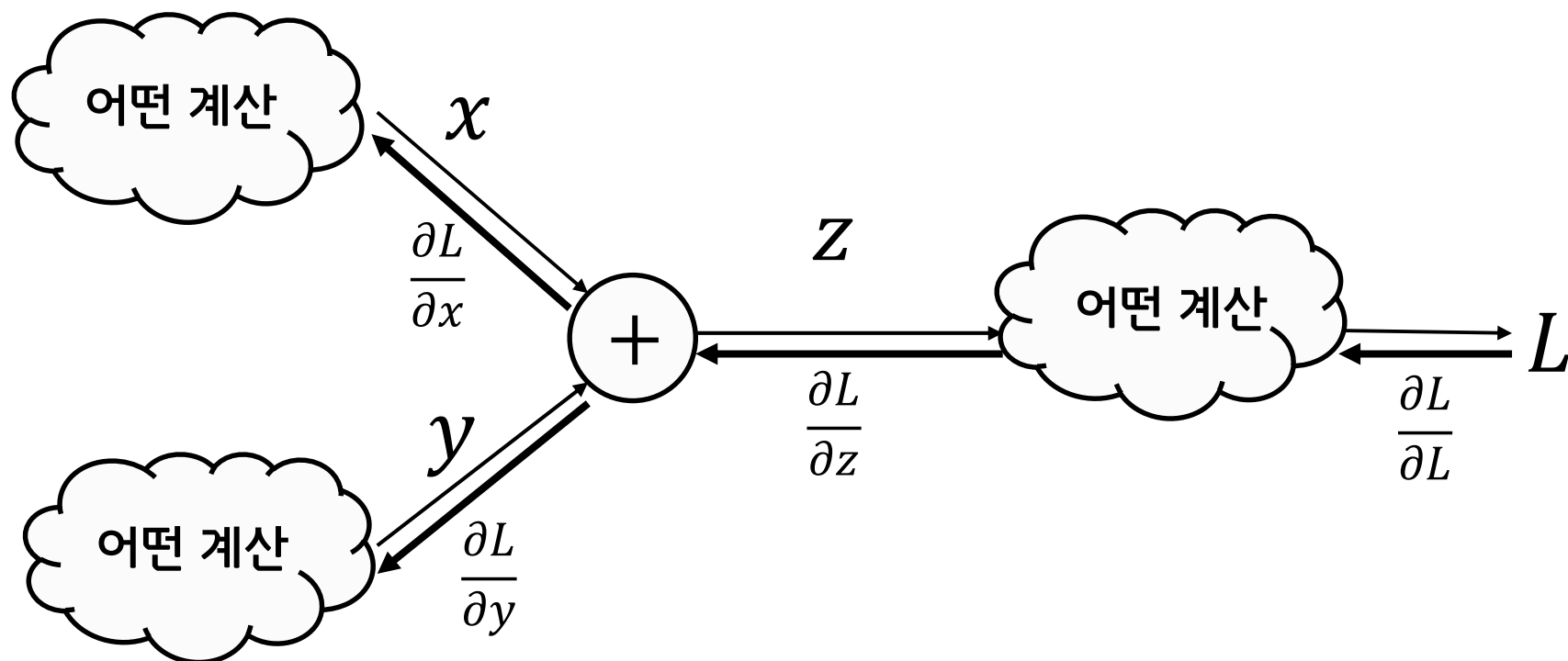
$z = x + y$ 를 나타내는 계산 그래프



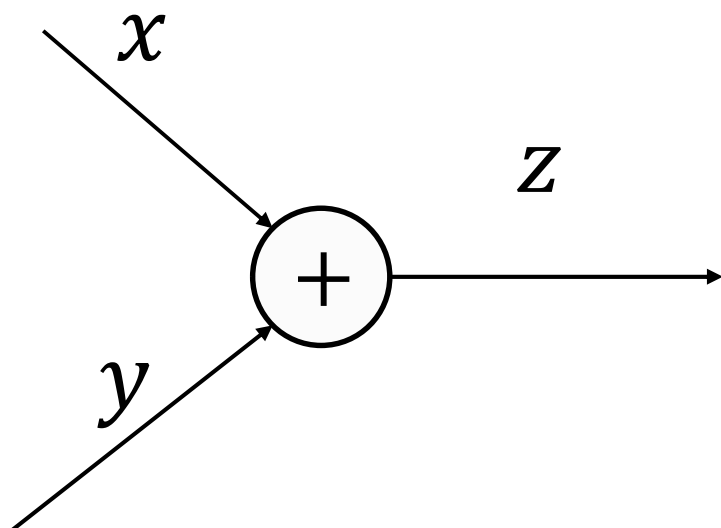
앞뒤로 추가된 노드는 '복잡한 전체 계산'의 일부를 구성한다.



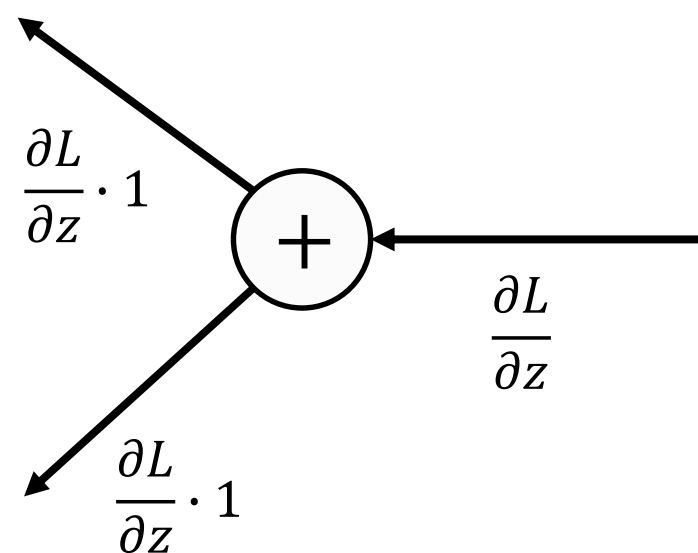
계산 그래프의 역전파



덧셈 노드의 순전파와 역전파

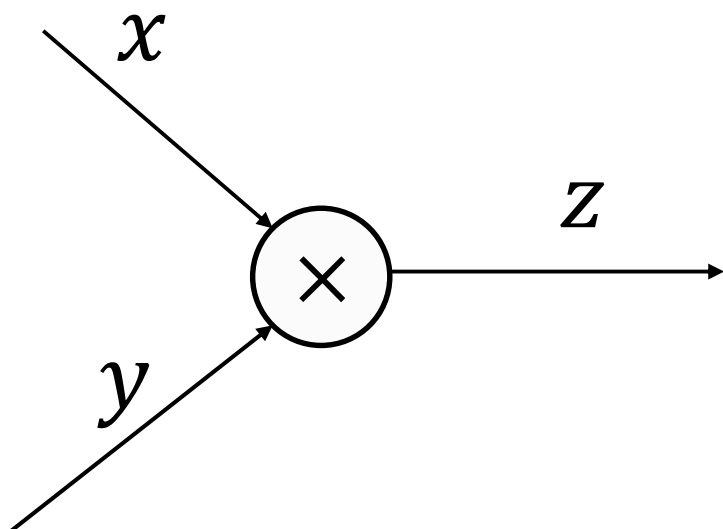


순전파

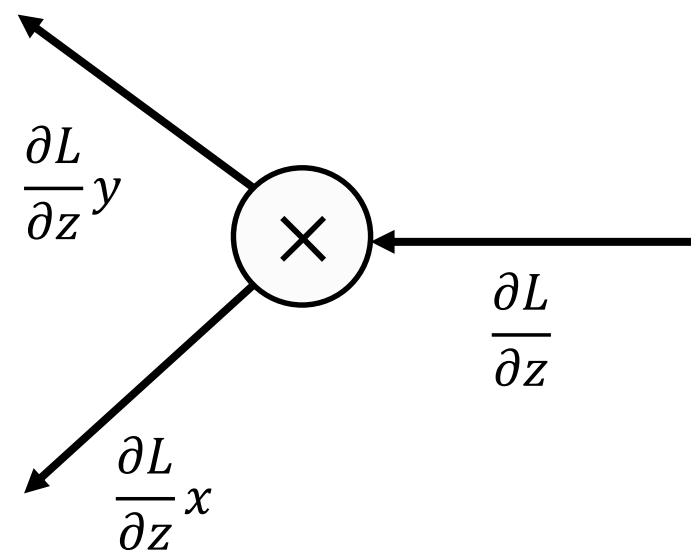


역전파

곱셈 노드의 순전파와 역전파

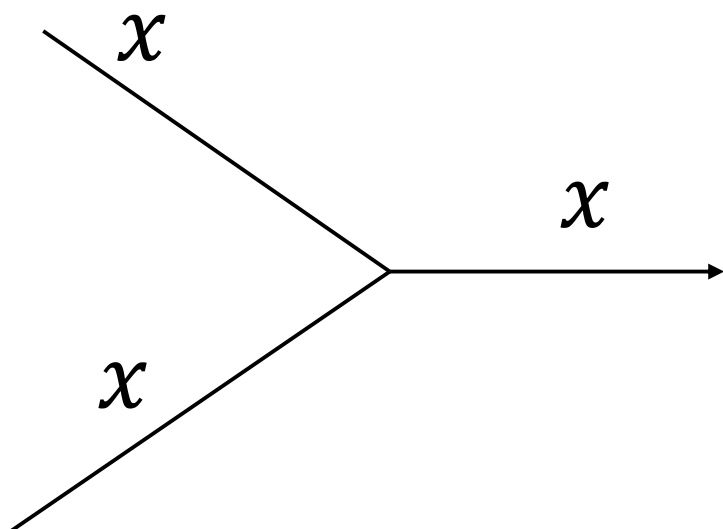


순전파

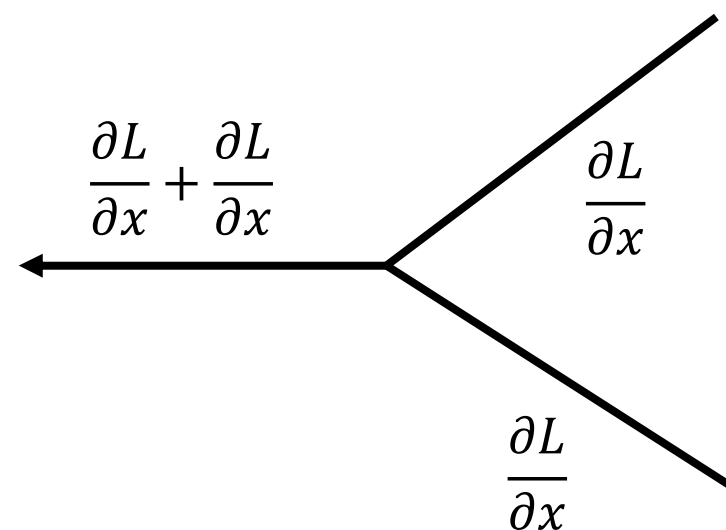


역전파

분기 노드의 순전파와 역전파

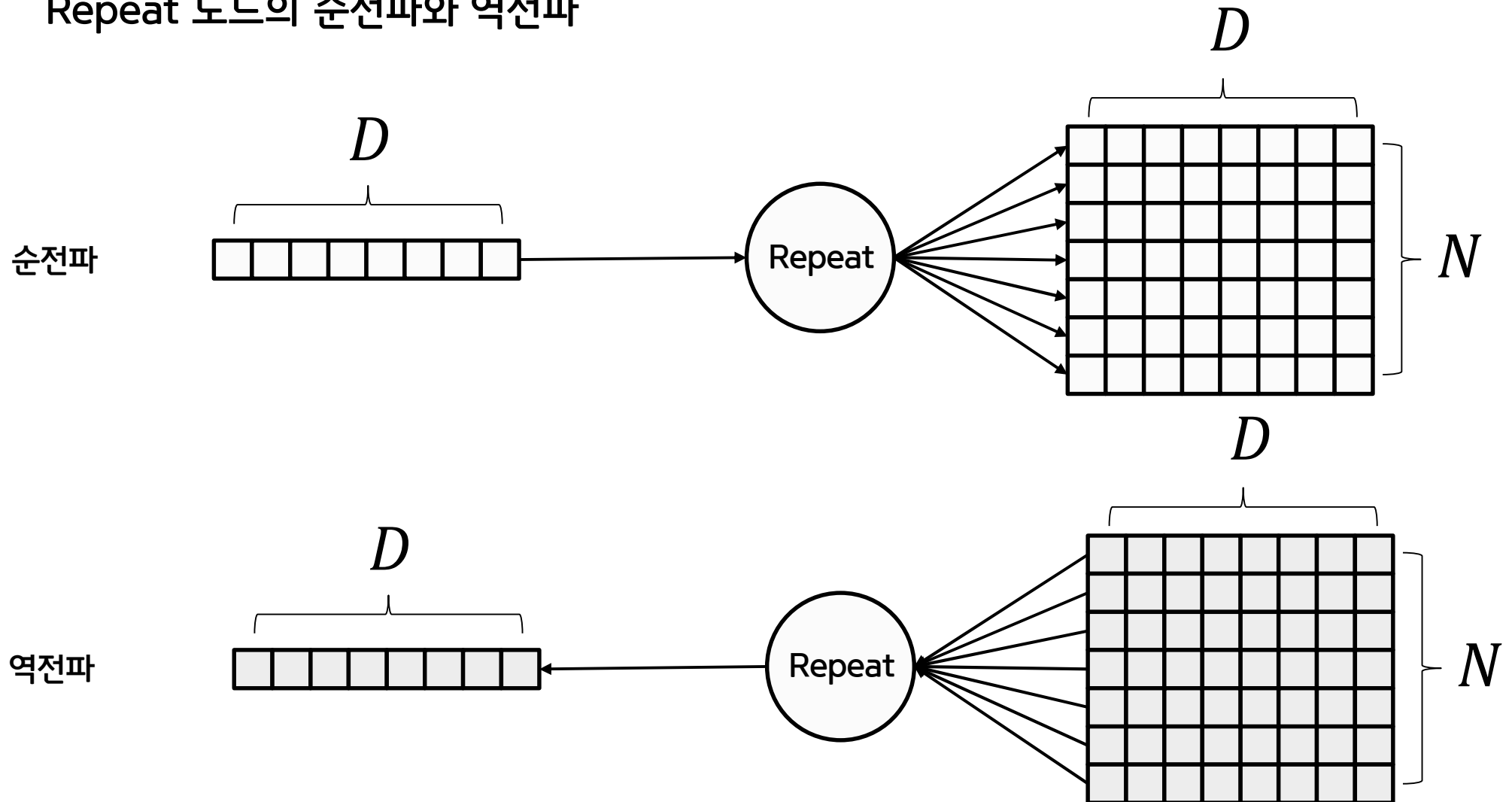


순전파

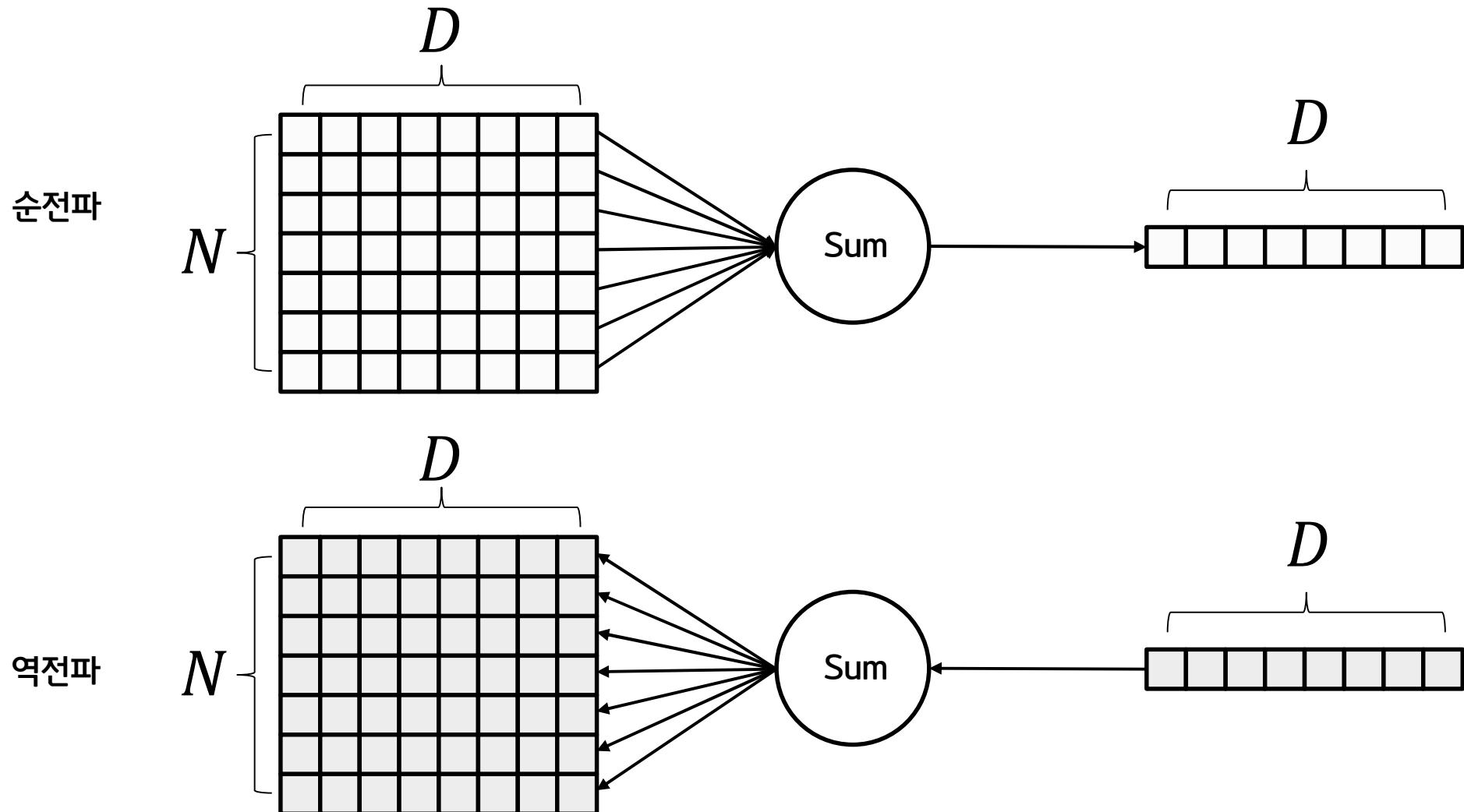


역전파

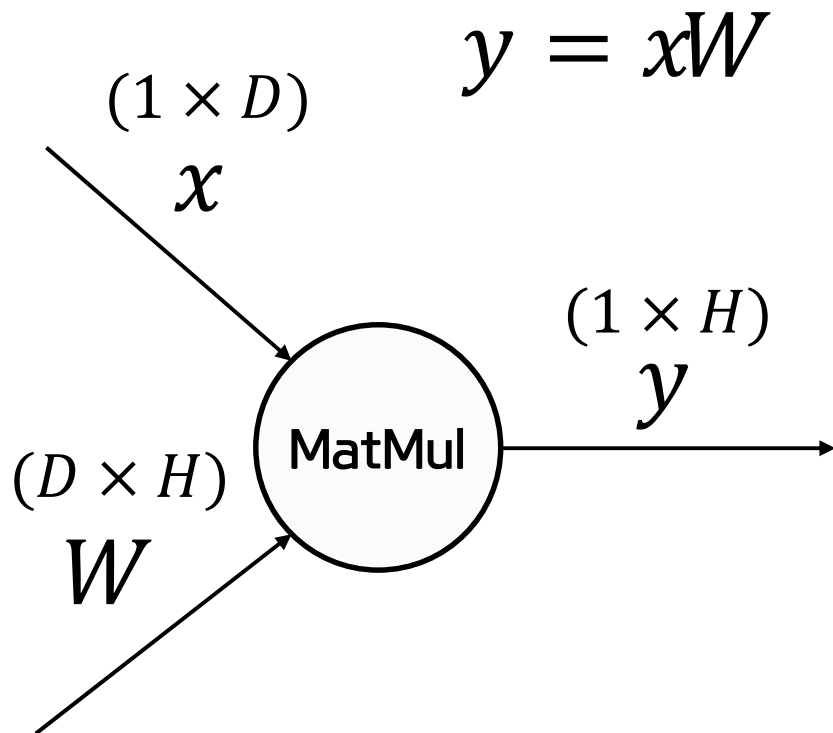
Repeat 노드의 순전파와 역전파



Sum 노드의 순전파와 역전파



MatMul 노드의 순전파



$$\frac{\partial L}{\partial x_i} = \sum_j \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

$$= \sum_j \frac{\partial L}{\partial y_j} W_{j\ddot{y}}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} W^T$$

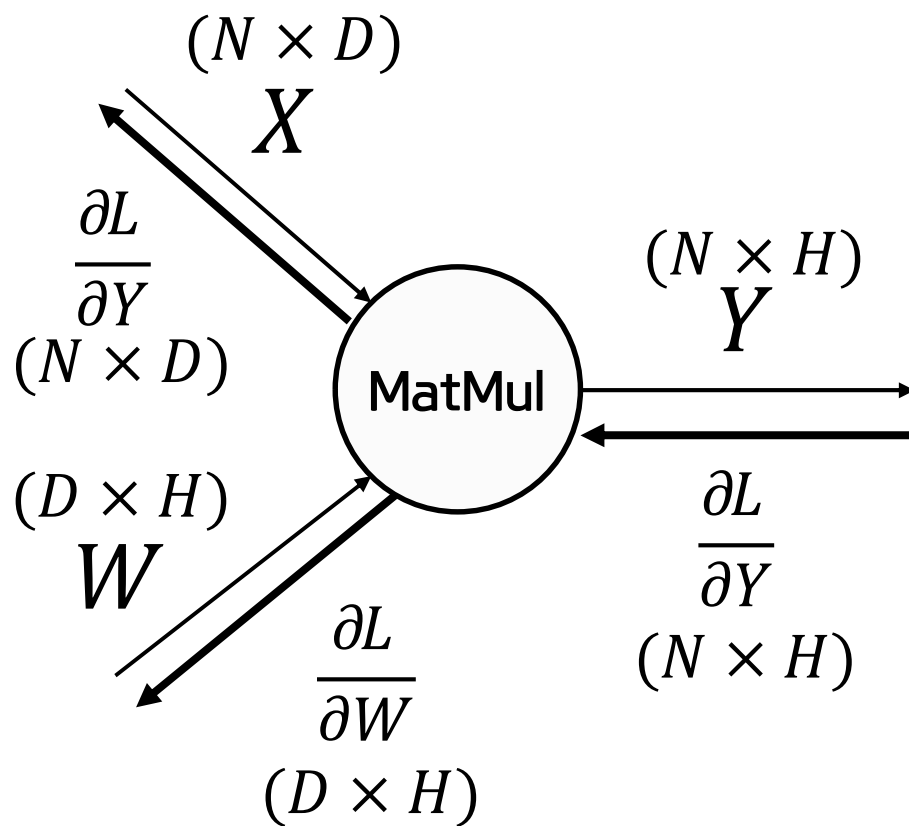
MatMul 미분

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} W^T$$

1 × D 1 × H H × D

일치시킨다.

MatMul 노드의 역전파



행렬의 형상을 확인하여 역전파 식을 유도 한다.

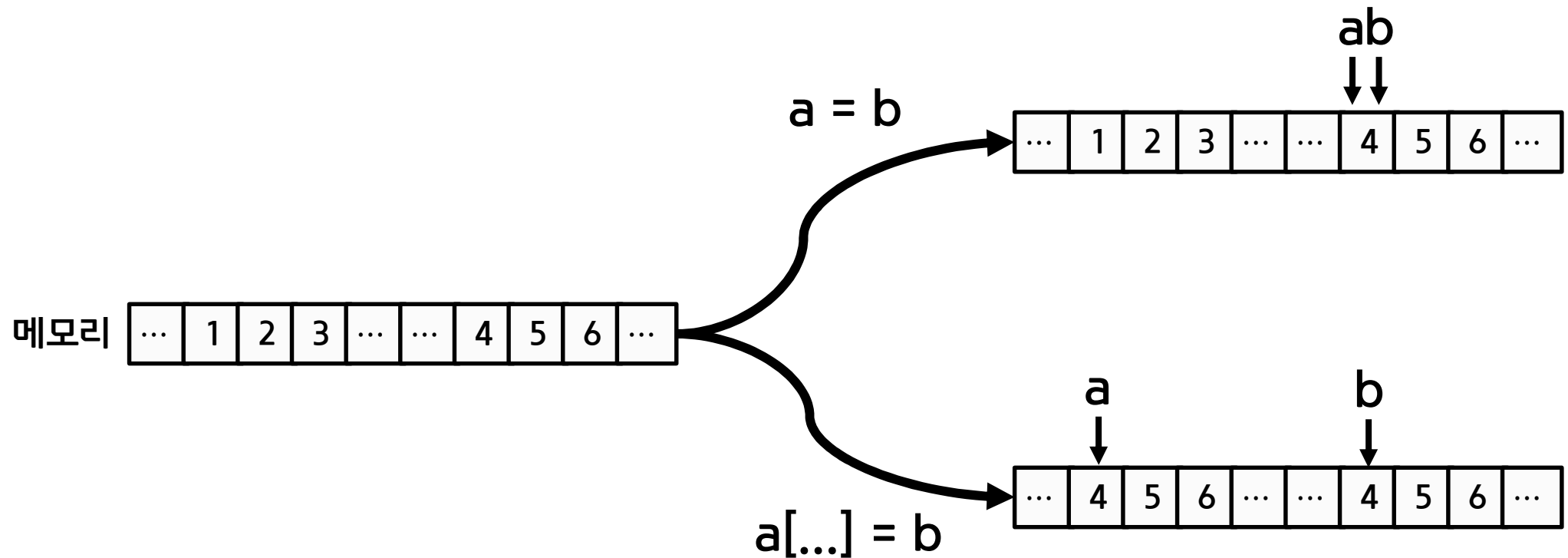
형상:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} W^T$$

형상:

$$\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Y}$$

```
a = np.array([1, 2, 3])  
b = np.array([1, 2, 3])
```



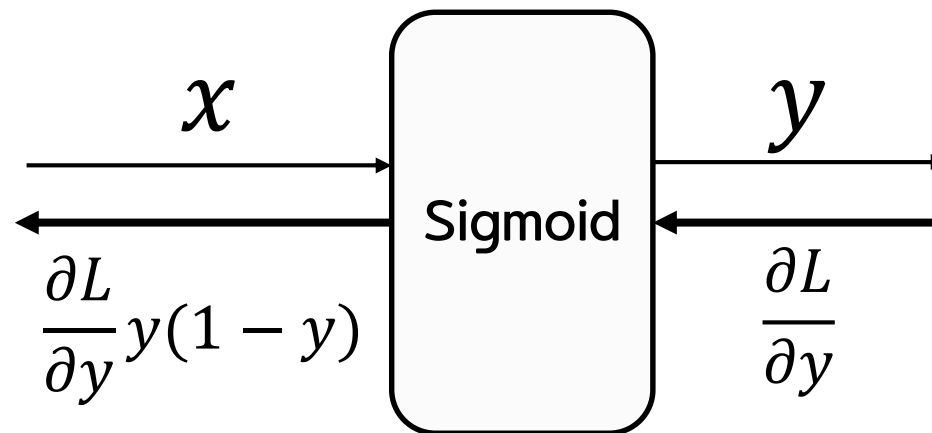
Sigmoid 계층

Sigmoid 식

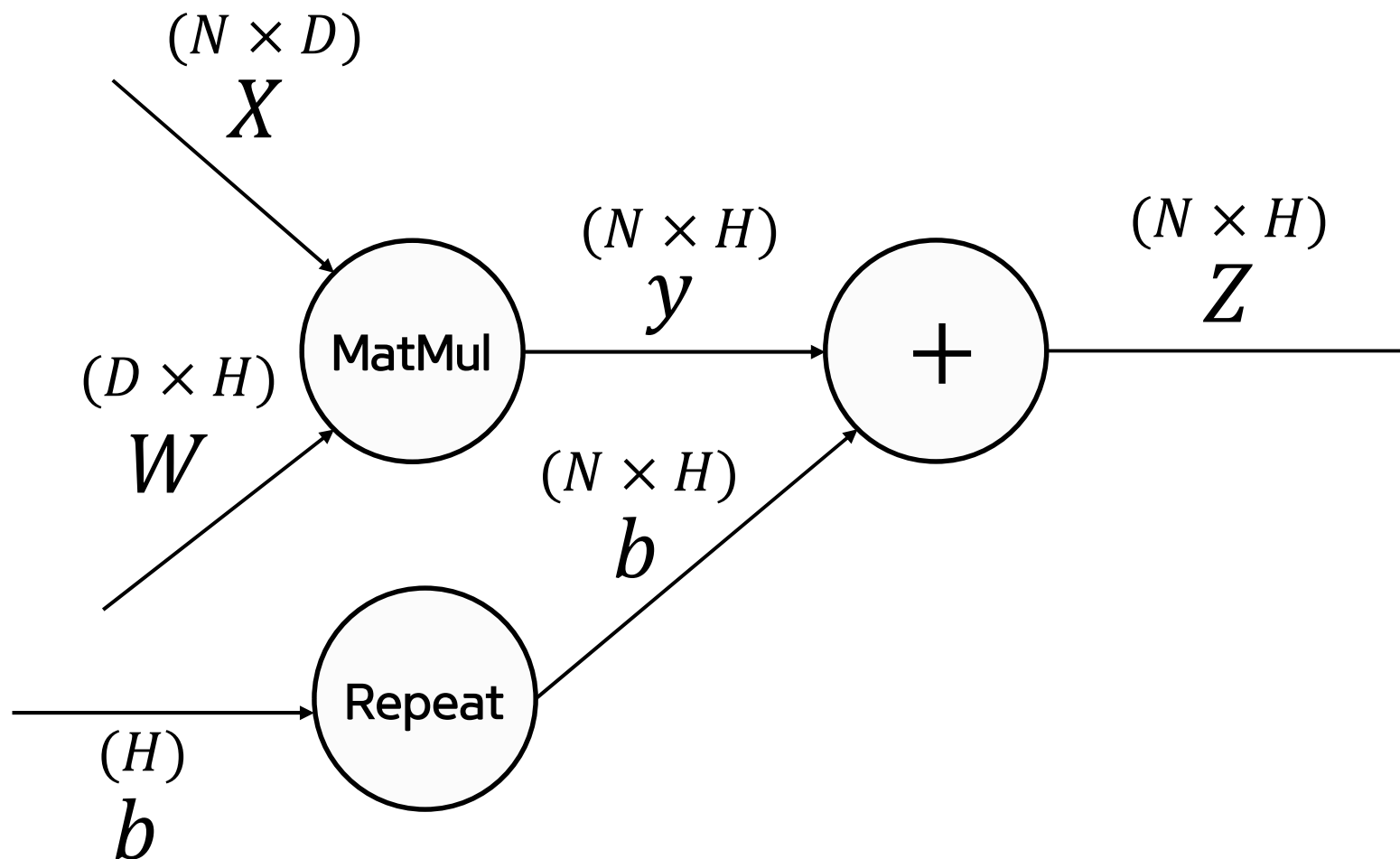
$$y = \frac{1}{1 + e^{-x}}$$

Sigmoid 미분식.
여기에 x 값을 입력하십시오.

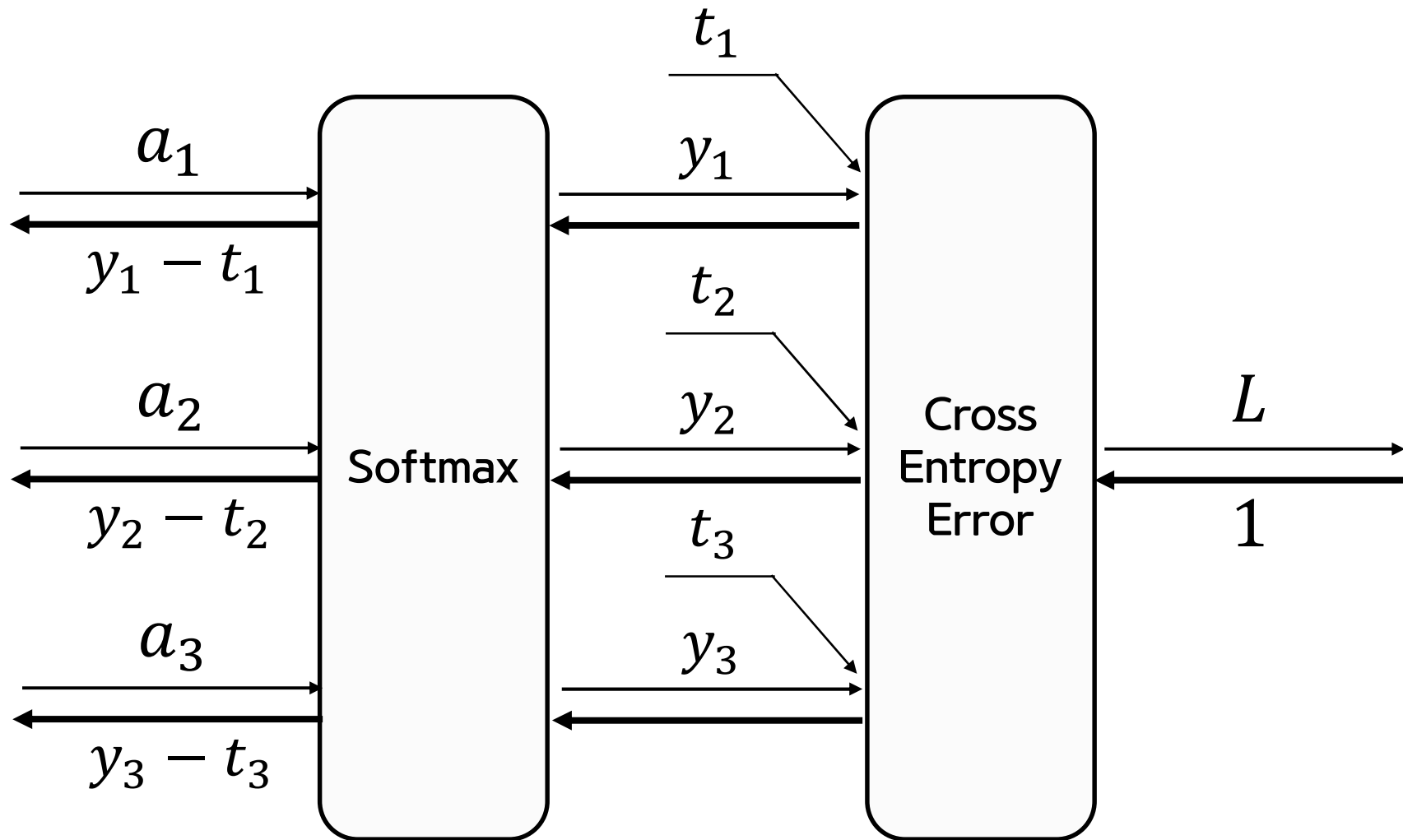
$$\frac{\partial y}{\partial x} = y(1 - y)$$



Affine 계층



Softmax with Loss 계층



- 1단계: 미니배치
훈련 데이터 중에서 무작위로 다수의 데이터를 골라낸다.
- 2단계: 기울기 계산
오차역전파법으로 각 가중치 매개변수에 대한 손실 함수의 기울기를 구한다.
- 3단계: 매개변수 갱신
기울기를 사용하여 가중치 매개변수를 갱신한다.
- 4단계: 반복
1~3 단계를 필요한 만큼 반복한다.

$$W \leftarrow W - \alpha \frac{\partial L}{\partial W}$$

1. 신경망 복습

- 1.1 수학과 파이썬 복습
 - 1.2 신경망 추론
 - 1.3 신경망의 학습
 - 1.4 신경망으로 문제를 푼다.**
-

인수	설명
x	입력 데이터
t	정답 레이블
max_epoch(=10)	학습을 수행하는 에폭 수
batch_size(=32)	미니배치 크기
eval_interval(=20)	결과(평균 손실 등) 출력하는 간격 예컨대 eval_interval=20으로 설정하면, 20번째 반복마다 손실의 평균을 구해 화면에 출력한다.
max_grad(=None)	기울기 최대의 노름 기울기 노름이 이 값을 넘어서면 기울기를 중인다(이를 기울기 클리핑이라 한다.)

2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

한국어와 영어 등 우리가 평소에 쓰는 말을 자연어라고 한다.

자연어 처리(NLP)를 풀어서 말하면

'우리의 말을 컴퓨터에게 이해시키기 위한 기술(분야)'이다.

자연어 처리가 추구하는 목표는 사람의 말을 부드럽게
컴퓨터가 이해하도록 만들어서,
컴퓨터가 우리에게 도움이 되는 일을 수행하게 하는 것이다.

우리의 말은 '문자로'로 구성되며, 말의 의미는 '단어'로 구성된다.

단어는 의미의 최소 단위이기 때문에 자연어를 컴퓨터에게 이해시키는 데는 '단어의 의미'를 이해시키는 것이 중요하다.

세가지 기법

- 시소러스를 활용한 기법
- 통계 기반 기법
- 추론 기반 기법(word2vec)

2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

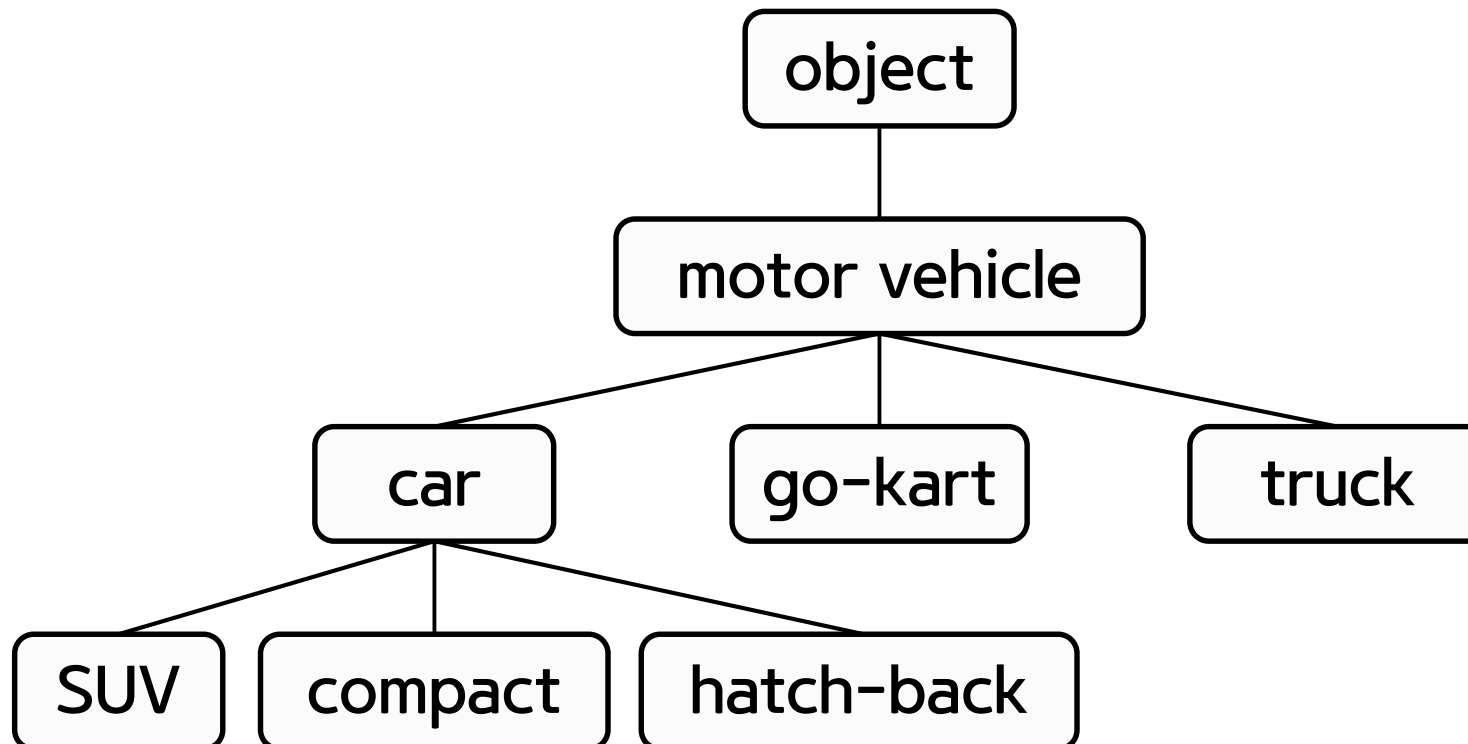
2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

동의어의 예: "car", "auto", "automobile" 등은 "자동차"를 뜻하는 동의어다.



단어들의 의미의 상/하위 관계에 기초해 그래프로 표현한다.



WordNet과 같은 시소러스에는 수많은 단어에 대한 동의어와 계층 구조 등의 관계가 정의되어 있다.

그리고 이 지식을 이용하면 '단어의 의미'를 컴퓨터에 전달할 수 있다.

하지만 사람이 수작업으로 레이블링하는 방식에는 문제들이 존재한다.

- 시대 변화에 대응하기 어렵다.

신조어 혹은 의미 변화된 단어들을 바로 적용 시키기 어렵다.

- 사람을 쓰는 비용이 든다.

현존하는 영어 단어의 수는 1,000만 개가 넘으며 WordNet에 등록된 단어는 20만 개 이상이다.

- 단어의 미묘한 차이를 표현할 수 없다.

가령, 빈티지와 레트로의 의미는 같으나 용법의 차이가 존재한다.

위 문제점들을 피하기 위해 '통계 기반 기법'과 신경망을 사용한 '추론 기반 기법'을 알아볼 것이다.

2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

자연어 처리에는 다양한 말뭉치가 사용되는데

예로는 구글 뉴스와 위키백과 등의 텍스트 데이터를 들 수 있다.

색에는 고유한 이름이 붙여진 다채로운 색들도 있고, RGB(Red/Green/Blue)라는 세가지 성분이 어떤 비율로 섞여 있느냐로 표현하는 방법이 있다. 전자는 색의 가짓수만큼의 이름을부여하는 반면에 후자는 색을 3차원의 벡터로 표현한다.

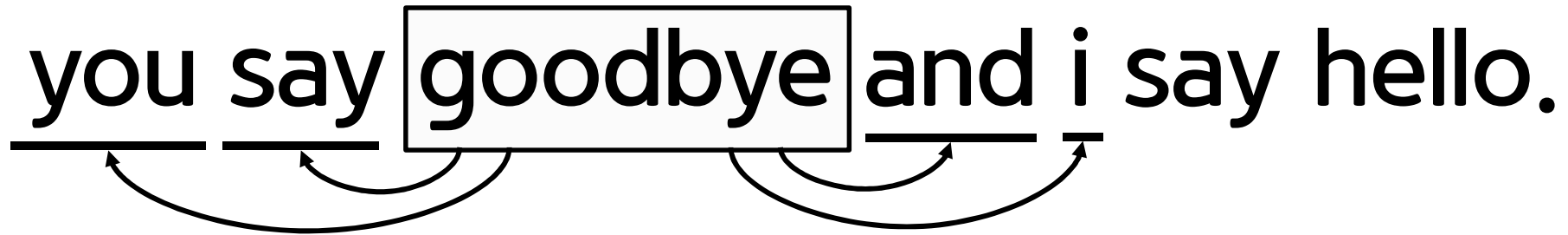
여기서 주목할 점은 RGB같은 벡터 표현이 단 3개의 성분으로 간결하게 표현할 수 있고, 색을 더 정확하게 명시할 수 있다는 점이다.

'색'을 벡터로 표현하듯 '단어'도 벡터로 표현할 수 있다. 이를 단어의 '분산 표현'이라고 한다.

분포 가설이란 단어의 의미는 주변 단어에 의해 형성된다는 것이다.
분포 가설이 말하고자 하는 것은 단어 자체에는 의미가 없고,
그 단어가 사용된 '맥락'이 의미를 형성한다는 것이다.

예를 들어, I drink beer를 I guzzle beer라고 해도 guzzle을 drink로 이해할 수 있다는 것이다.

윈도우 크기가 2인 '맥락'의 예. 단어 "goodbye"에 주목한다면,
그 좌우의 두 단어(총 네 단어)를 맥락으로 이용한다.



위 그림에서 goodbye를 기준으로 좌우의 두 단어씩이 '맥락'에 해당한다.
맥락의 크기를 '윈도우 크기'라고 한다. 여기서는 '윈도우 크기'가 2이기 때문에
좌우로 두 단어씩이 맥락에 포함된다.

분포 가설에 기초해 단어를 벡터로 나타내는 방법을 생각해보면
주변 단어를 세어보는 방법이 떠오를 것이며 이를 '통계 기반' 기법이라고 한다.

단어 "you"의 맥락을 세어본다.

you say goodbye and i say hello.



단어가 총 7개이며 윈도우 크기는 1로 하고 단어 ID가 0인 'you'부터
단어의 맥락에 해당하는 단어의 빈도를 세어보겠다.
'you'의 맥락은 'say'라는 단어 하나뿐이다.

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0

단어 "say"의 맥락을 세어본다.

you **say** goodbye and i **say** hello.

'say'라는 단어는 벡터 $[1, 0, 1, 0, 1, 1, 0]$ 으로 표현할 수 있다.

	you	say	goodbye	and	i	hello	.
you	1	0	1	0	1	1	0

모든 단어 각각의 맥락에 해당하는 단어의 빈도를 세어 표로 정리 한다.

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

위의 표는 모든 단어에 대해 동시발생하는 단어를 표에 정리한 것이다.
위 표의 각 행은 벡터이며 행렬의 형태를 띄어 동시발생 행렬이라 한다.

단어 벡터의 유사도를 나타낼 때는 코사인 유사도를 자주 이용한다.

$$\text{similarity}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{x_1 y_1 + \dots + x_n y_n}{\sqrt{x_1^2 + \dots + x_n^2} \sqrt{y_1^2 + \dots + y_n^2}}$$

위의 식처럼 정의되며 분자에는 벡터의 내적이 분모에는 벡터의 노름(크기)이 등장한다.
 위 식의 핵심은 벡터를 정규화하고 내적을 구하는 것이다.

코사인 유사도를 이용하여 어떤 단어가 주어지면,
그 검색어와 비슷한 단어를 유사도 순으로 출력하는 함수를 만들어 본다.

이를 구현하기 위한 코드에는 밑에 같은 함수의 인수들이 쓰인다.

인수명	설명
query	검색어(단어)
word_to_id	단어에서 단어 ID로으 딕셔너리
id_to_word	단어 ID에서 단어로의 딕셔너리
word_matrix	단어 벡터들을 한데 모은 행렬, 각 행에는 대응하는 단어의 벡터가 저장되어 있다고 가정한다.
top	상위 몇 개까지 출력할지 설정

2. 자연어와 단어의 분산 표현

2.1 자연어 처리란

2.2 시소러스

2.3 통계 기반 기법

2.4 통계 기반 기법 개선하기

동시발생 행렬의 원소는 두 단어가 동시에 발생한 횟수를 나타내지만 '발생' 횟수라는 것은 사실 좋은 특징이 아니다.
예를 들어 'the' 와 'car'의 동시발생을 생각해보자.
'...the car...'라는 문구가 자주 보일 것이며 'car'와 'drive'는 관련이 깊다.
하지만 'the'가 고빈도 단어이기 때문에 'car'와 더 관련이 있어 보이게 결과가 나올 수 있다.

이를 해결하기 위해 점별 상호정보량이라는 척도를 사용한다.
PMI(Pointwise Mutual Information)는 확률 변수 x 와 y 에 대해 다음 식으로 정의 된다.

$$PMI(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

$P(x)$ 는 x 가 일어날 확률, $P(y)$ 는 y 가 일어날 확률, $P(x, y)$ 는 x, y 가 동시에 일어날 확률이다.
PMI값이 높을수록 관련성이 높다는 의미이다.

$$PM\ I(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} = \log_2 \frac{\frac{C(x, y)}{N}}{\frac{C(x)}{N} \frac{C(y)}{N}} = \log_2 \frac{C(x, y) \cdot N}{C(x)C(y)}$$

여기서 C는 동시발생 행렬, C(x,y)는 단어 x와 y가 동시발생하는 횟수, C(x)와 C(y)는 각각 단어 x와 y의 등장 횟수이며 N은 말뭉치에 포함된 단어 수이다.

이 식을 토대로 1,000번 등장한 'the', 20번 등장한 'car'와 10번 등장한 'drive'를 계산해보자.

$$PM\ I("the" , "car") = \log_2 \frac{10 \cdot 10000}{1000 \cdot 20} \approx 2.32$$

$$PM\ I("car" , "drive") = \log_2 \frac{5 \cdot 10000}{20 \cdot 10} \approx 7.97$$

두 PMI의 결과를 살펴보면 'car'와 'drvie'의 관계성이 강하다는 것을 볼 수 있다. 이러한 결과가 나온 이유는 단어가 단독으로 출현하는 횟수가 고려되었기 때문이다. 이 예에서는 'the'가 자주 출현하였기 때문에 PMI값이 낮아진 것이다.

하지만 PMI에도 문제가 하나 있다.
이는 두 단어의 동시발생 횟수가 0이면 $\log(0,2) = -\infty$ 가 된다.
이 문제를 피하기 위해 실제 구현할 때는 양의 상호정보량(PPMI)를 사용한다.

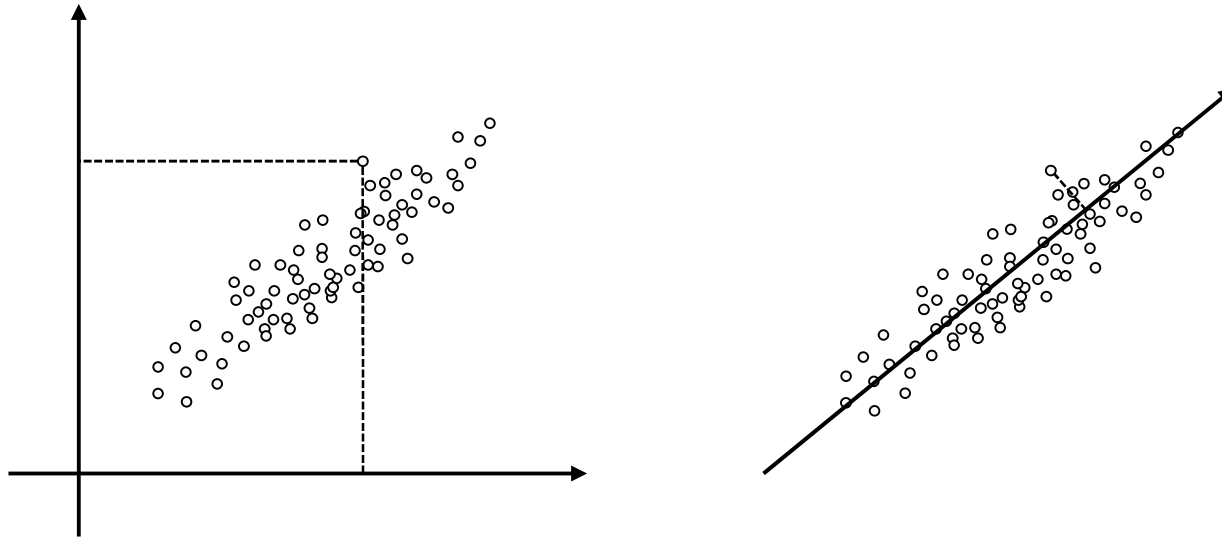
$$PPMI(x, y) = \max(0, PMI(x, y))$$

이 식에 따라 PMI가 음수인 때는 0으로 취급하며 단어 사이의 관련성을 0 이상의 실수로 나타낼 수 있다.

하지만 PPMI 행렬에도 문제가 있는데 말뭉치의 어휘 수가 증가함에 따라 각 단어 벡터의 차원 수도 증가한다는 문제이다.

이 문제를 대처하고자 자주 수행하는 기법이 '벡터의 차원 감소'이다.

그림으로 이해하는 차원 감소: 2차원 데이터를 1차원으로 표현하기 위해 중요한 축을 찾는다.



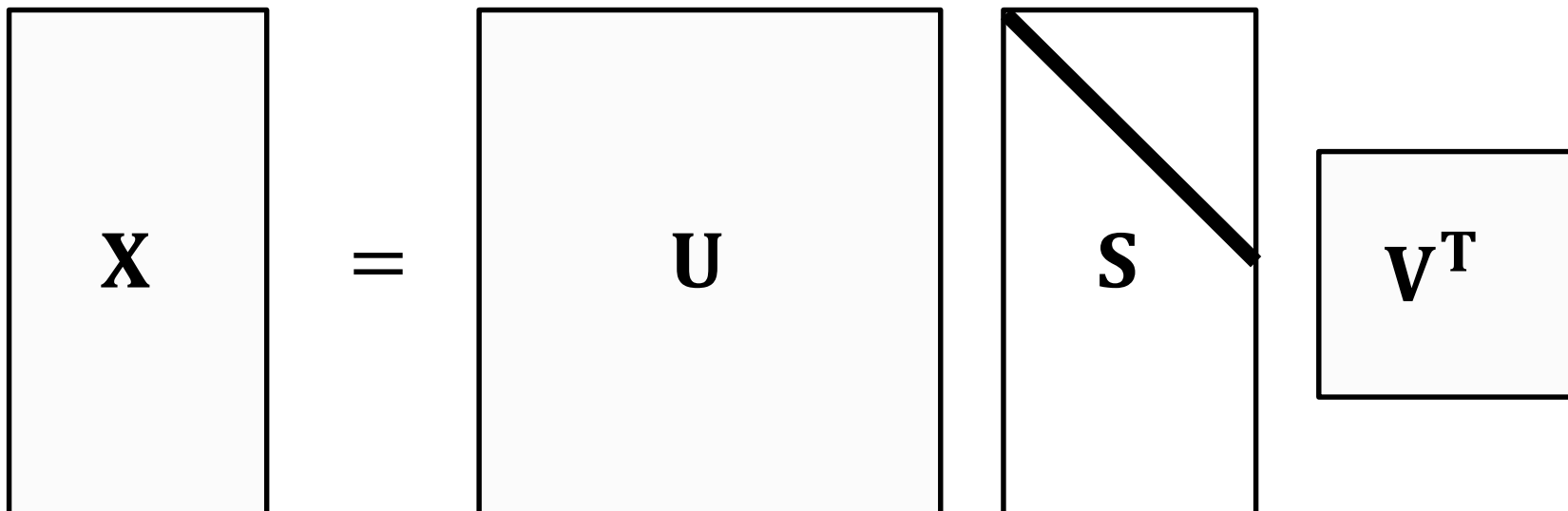
위의 그림 예시처럼 데이터의 분포를 고려해 중요한 '축'을 찾는 일을 수행한다.
왼쪽 그림은 데이터점들을 2차원 좌표에 표시한 모습이고
오른쪽 그림은 새로운 축을 도입하여 똑같은 데이터를 좌표축 하나만으로 표시했다.

여기서 중요한 것은 가장 적합한 축을 찾아내는 일로,
1차원 값만으로 데이터의 본질적인 차이를 구별할 수 있어야 한다.
그리고 다차원 데이터에 대해서도 수행 가능하다.

차원을 감소시키는 방법 중 하나인 특잇값분해(SVD)는 임의의 행렬을 세 행렬의 곱으로 분해하며, 수식으로는 다음과 같다.

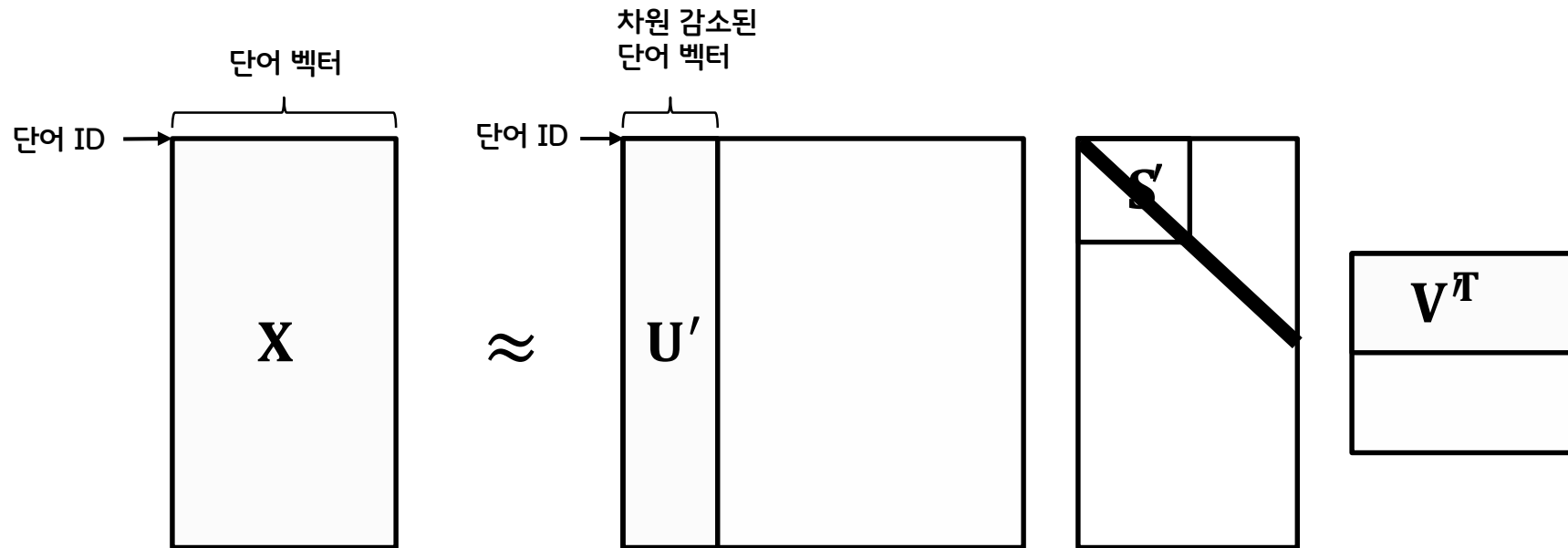
$$X = USV^T$$

SVD에 대한 행렬의 변환(행렬의 '흰 부분'은 원소가 0임을 뜻함)



행렬 S 에서 특 값이 작다면 중요도가 낮다는 뜻이므로
 행렬 U 에서 여분의 열벡터를 깎아내려 원래의 행렬을 근사할 수 있다.

SVD에 의한 차원의 감소



이를 '단어의 PPMI 행렬'에 적용하면 행렬 X 의 각 행에는 해당 단어 ID의 단어 벡터가 저장되어 있으며, 그 단어 벡터가 행렬 U' 라는 차원 감소된 벡터로 표현된다.

우리가 사용할 PTB(펜 트리뱅크) 말뭉치는 word2vec의 발명자인 토마스 미콜로프의 웹 페이지에서 받을 수 있다.

결과적으로 말뭉치를 사용해 맥락에 속한 단어의 등장 횟수를 센 후 PPMI 행렬로 변환하고 다시 SVD를 이용해 차원을 감소시킴으로서 더 좋은 단어 벡터를 얻었다. 이것이 단어의 분산 표현이고, 각 단어는 고정 길이의 밀집벡터로 표현되었다.

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

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.

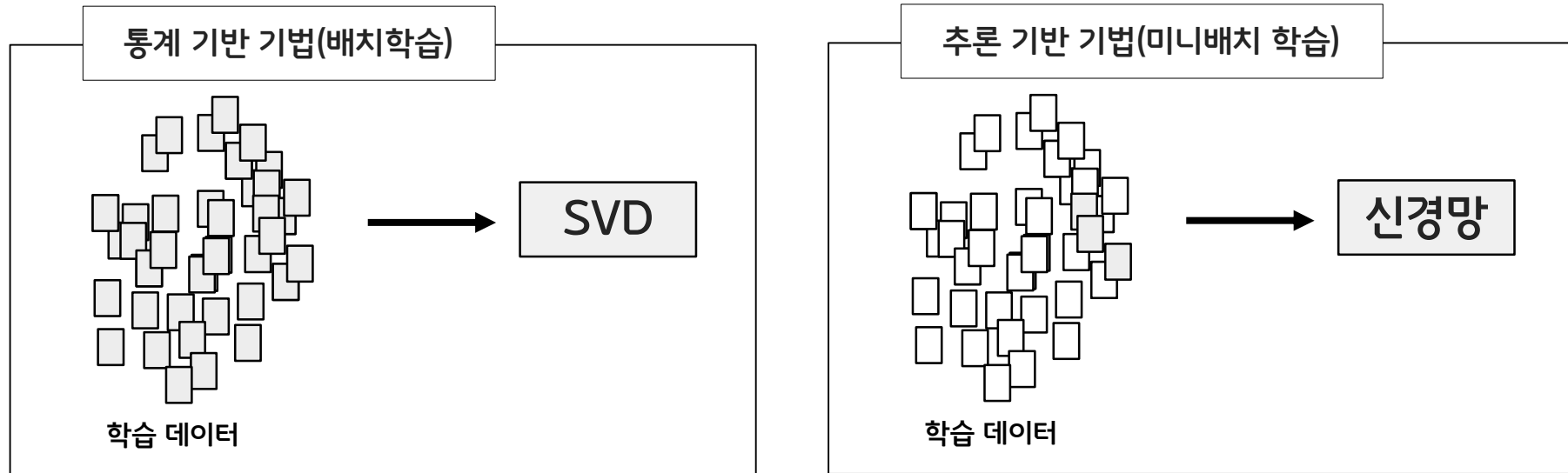
이번 절에서는 통계 기반 기법의 문제를 지적하고,
그 대안인 초론 기반 기법의 이점을 거시적 관점에서 설명한다.

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 $n \times n$ 행렬에 적용하는 비용은 $O(n^3)$ 이다.

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

주변 단어들의 맥락으로 사용해 "?"에 들어갈 단어를 추측한다.

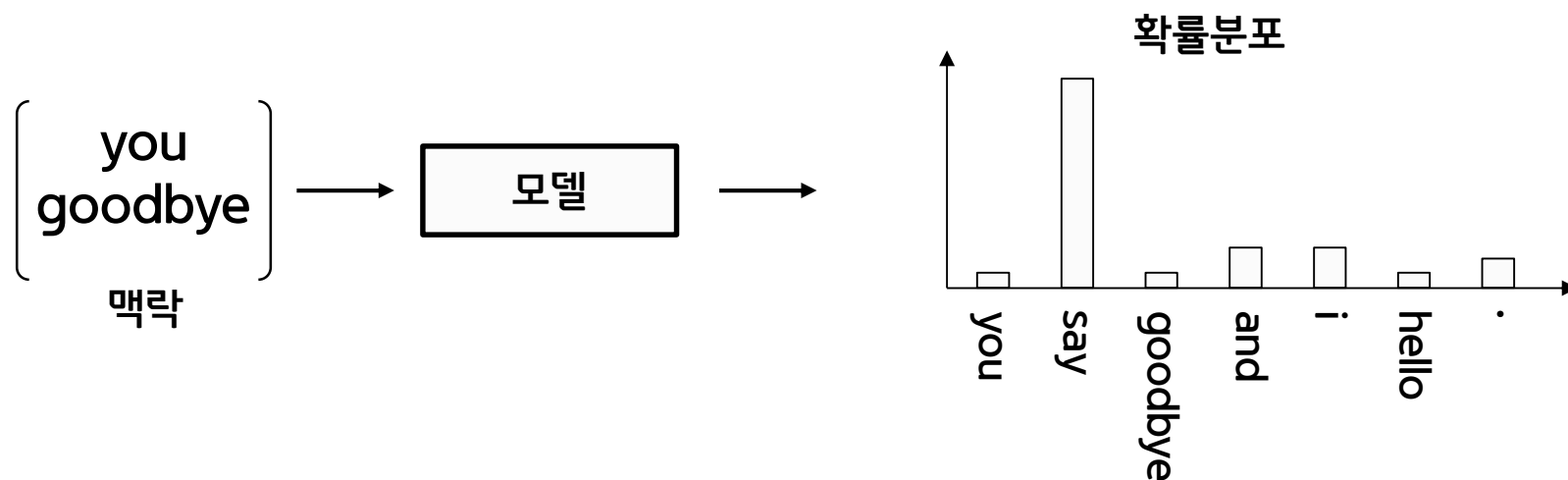
you say goodbye and i say hello.



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

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.

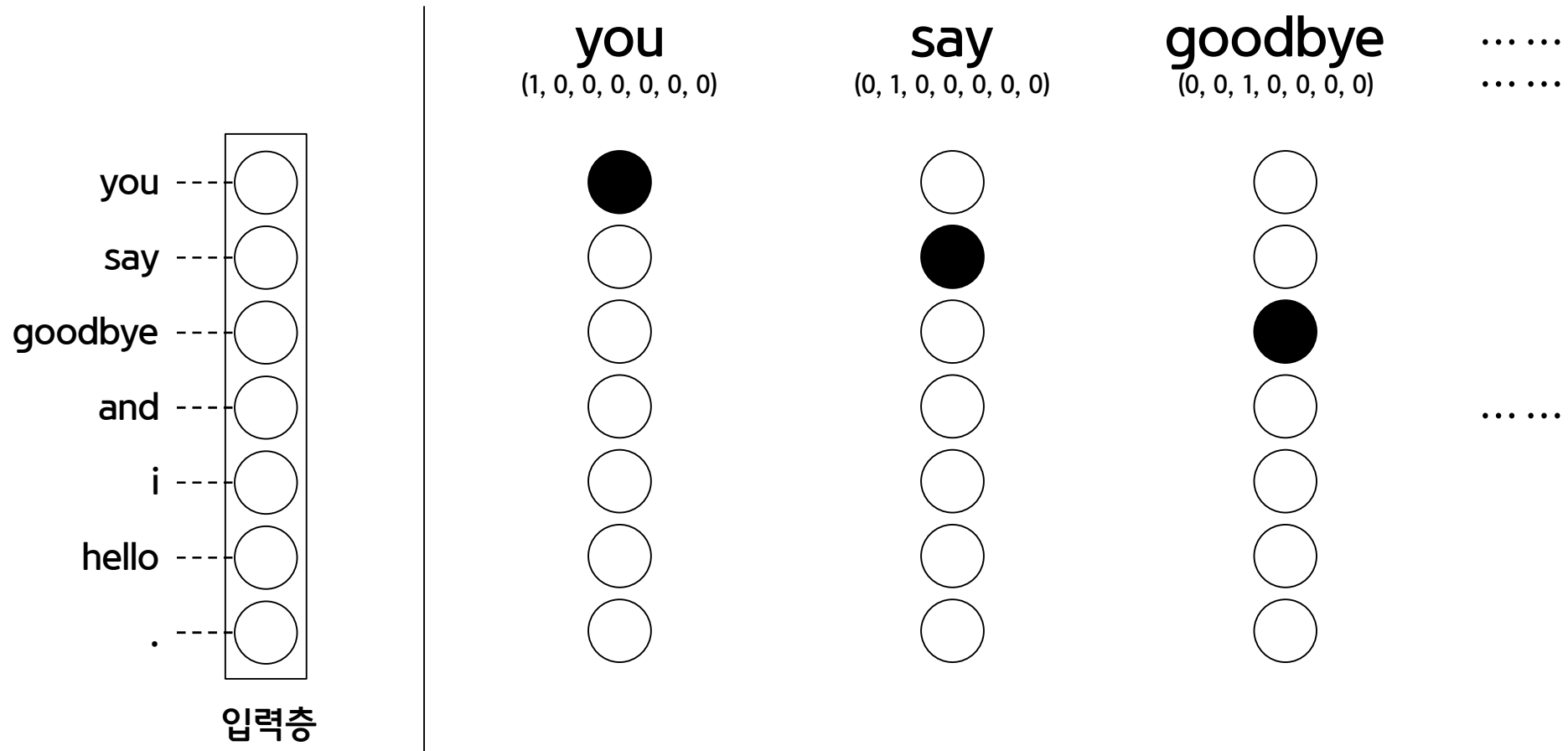
지금부터 신경망을 이용해 단어를 처리해보자.
 단어를 있는 그대로 처리할 수 없으니 고정 길이의 벡터로 변환해야 한다.
 이때 사용하는 대표적인 방법이 단어를 원 핫 표현으로 변환하는 것이다.
 원 핫 표현이란, 벡터의 원소 중 하나만 1이고, 나머지는 모두 0인 벡터를 말한다.

단어(텍스트)	단어 ID	원 핫 표현
$\begin{bmatrix} \text{you} \\ \text{goodbye} \end{bmatrix}$	$\begin{bmatrix} 0 \\ 2 \end{bmatrix}$	$\begin{bmatrix} (1, 0, 0, 0, 0, 0, 0) \\ (0, 0, 1, 0, 0, 0, 0) \end{bmatrix}$

단어를 원 핫 표현으로 변환하는 방법

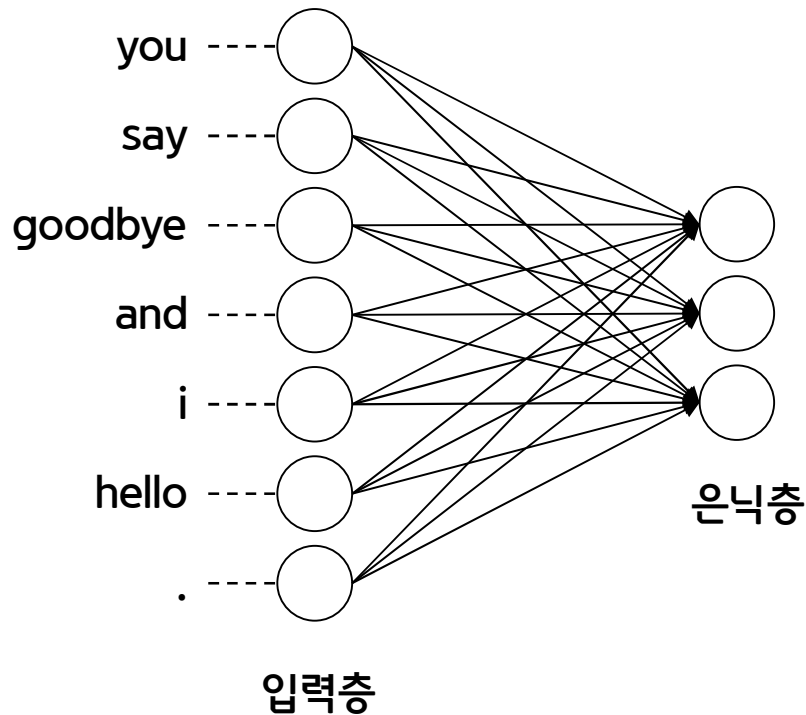
- 먼저 총 어휘 수만큼의 원소를 갖는 벡터를 준비하고,
 - 인덱스가 단어 ID 와 같은 원소를 1로, 나머지는 모두 0으로 설정한다.
- 이처럼 단어를 고정 길이 벡터로 변환하면, 신경망의 입력층은 뉴런의 수를 고정할 수 있다.

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



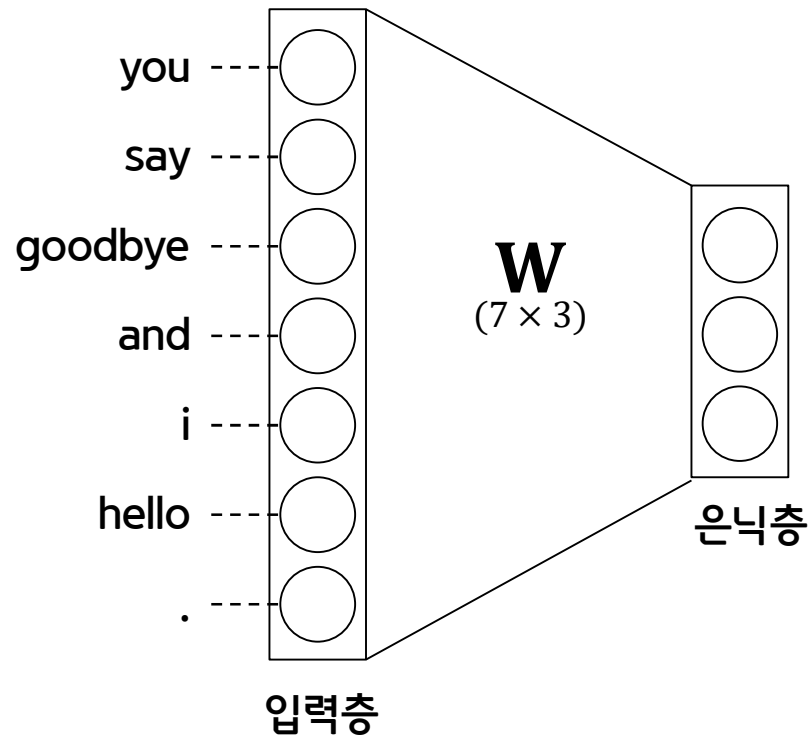
단어를 벡터로 나타낼 수 있고, 신경망을 구성하는 계층들은 벡터를 처리할 수 있다.
다시 말해, 단어를 신경망으로 처리할 수 있다는 뜻이다.

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

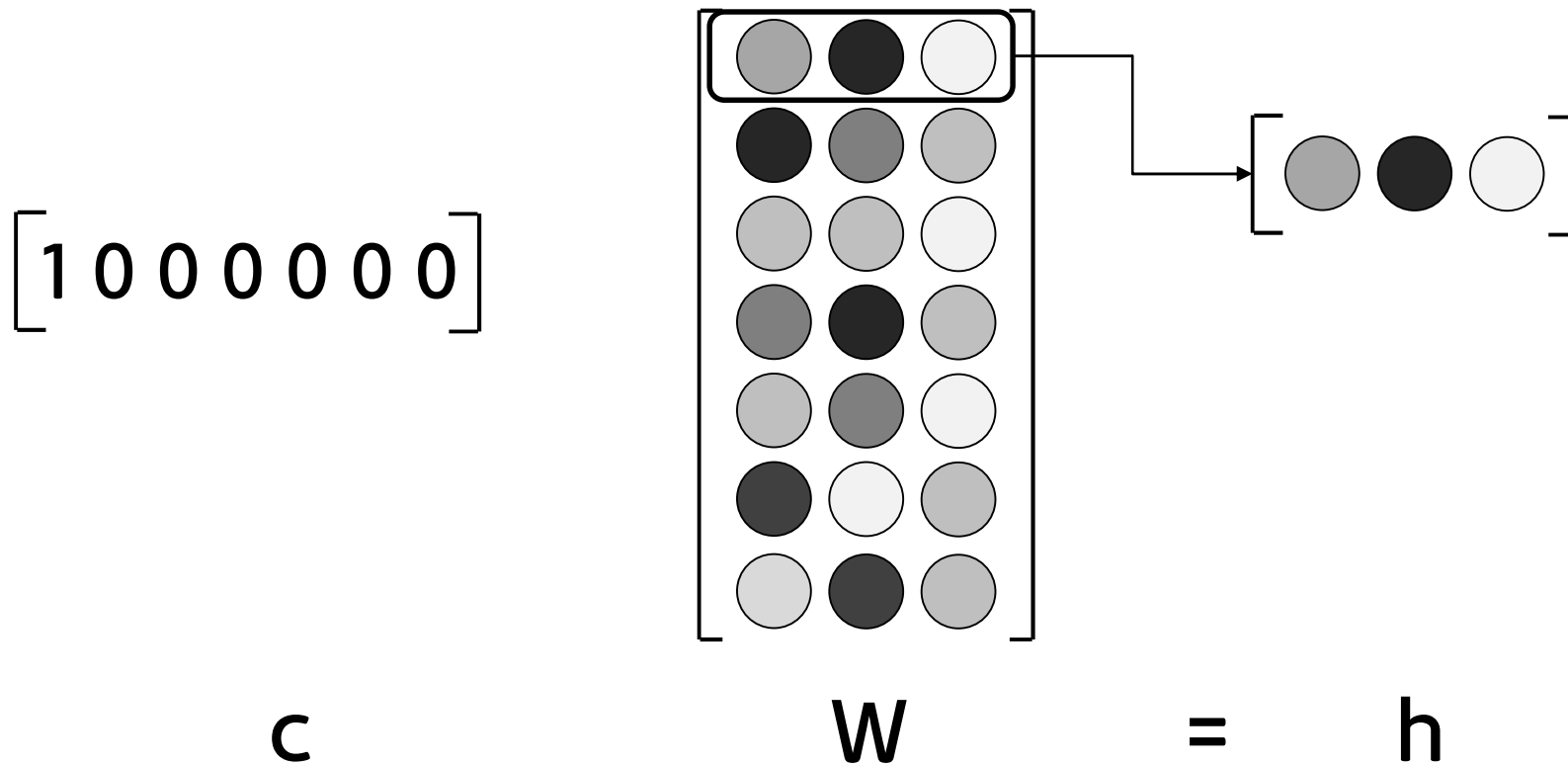


화살표에는 가중치:매개변수가 존재하여, 입력층 뉴런과의 가중합이 은닉층 뉴런이 된다.
간단한 설명을 위해 완전연결계층에서는 편향을 생략했다.
편향을 이용하지 않은 완전연결계층은 행렬 곱 계산에 해당한다.

완전연결층에 의한 변환을 단순화한 그림(완전연결계층의 가중치를 7×3 크기의 W 라는 행렬로 표현)



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



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

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.

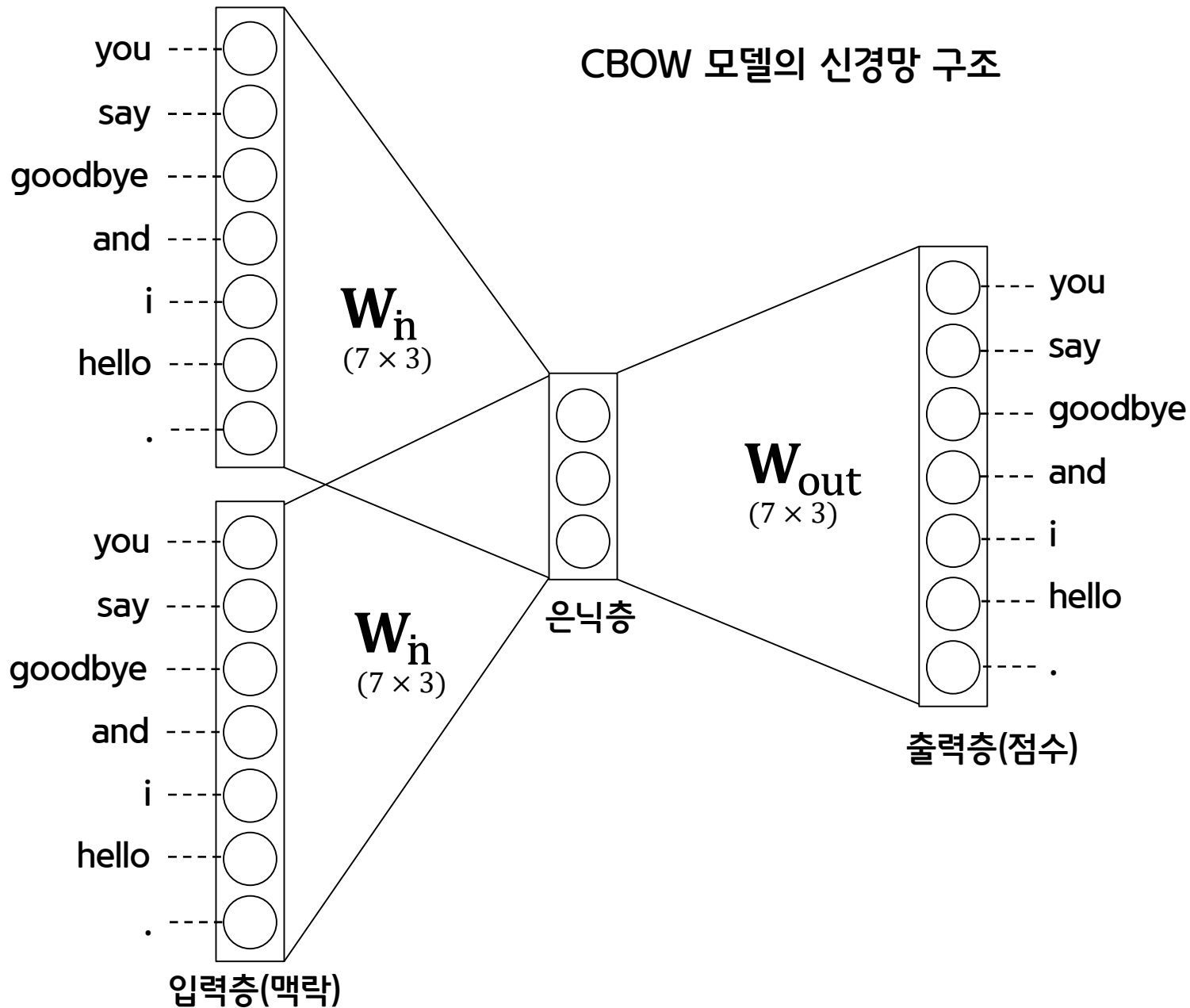
CBOW 모델은 맥락으로부터 타깃을 추측하는 용도의 신경망이다.

타깃은 중앙 단어이고, 그 주변 단어들이 맥락이다.

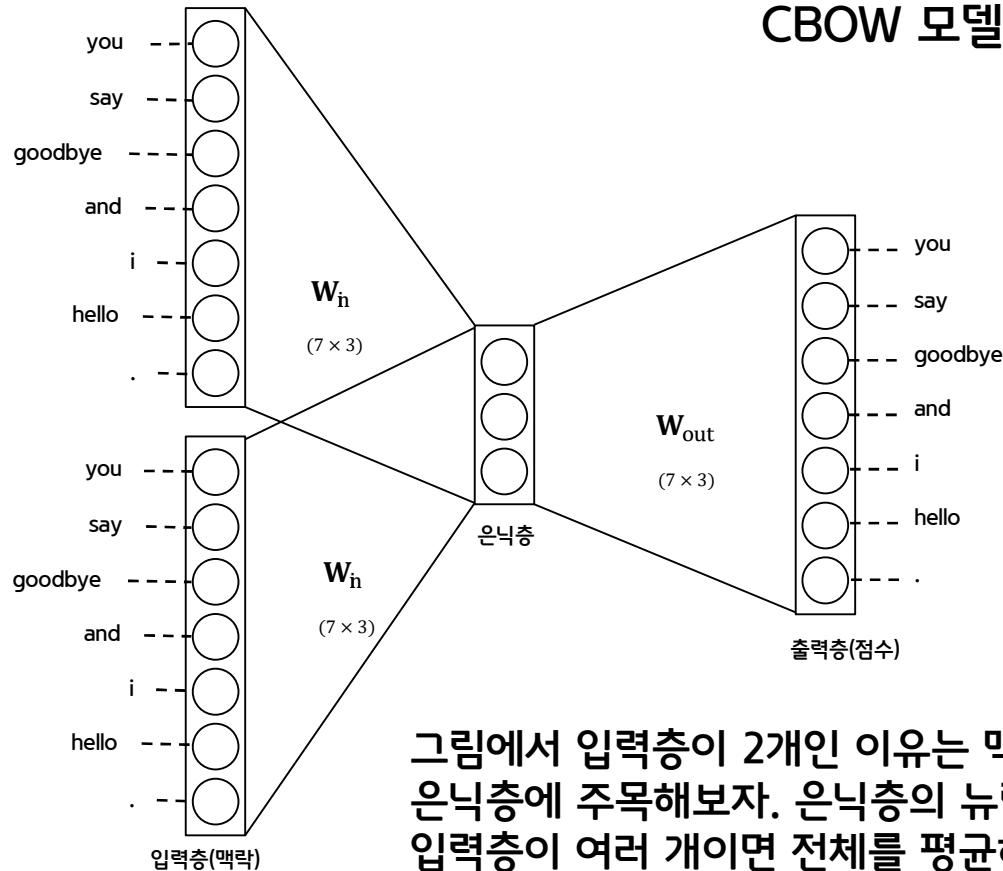
우리는 이 CBOW 모델이 가능한 한 정확하게 추론하도록 훈련시켜서 단어의 분산 표현을 얻어낼 것이다.

CBOW 모델의 입력은 맥락이다.

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



CBOW 모델의 신경망 구조



그림에서 입력층이 2개인 이유는 맥락으로 고려할 단어를 2개로 정했기 때문이다.

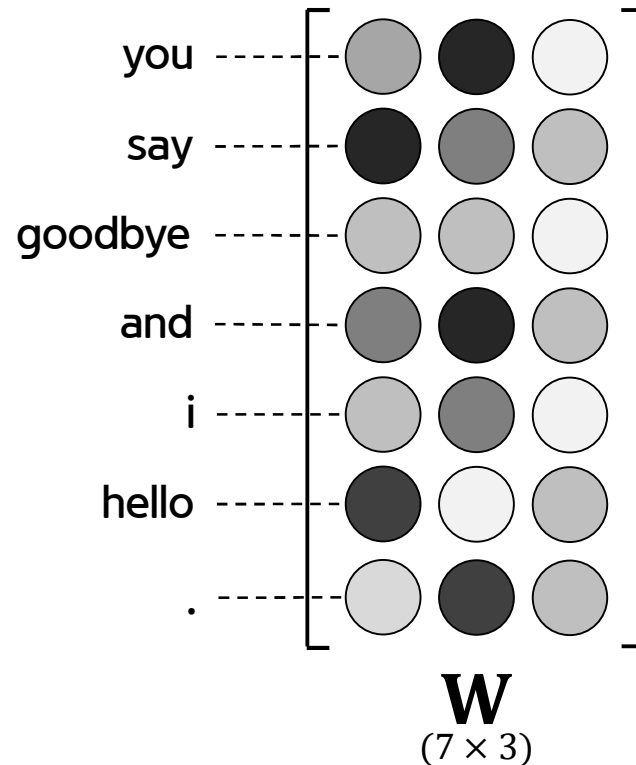
은닉층에 주목해보자. 은닉층의 뉴런은 입력층의 완전연결계층에 의해 변환된 값이 되는데, 입력층이 여러 개이면 전체를 평균하면 된다.

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.



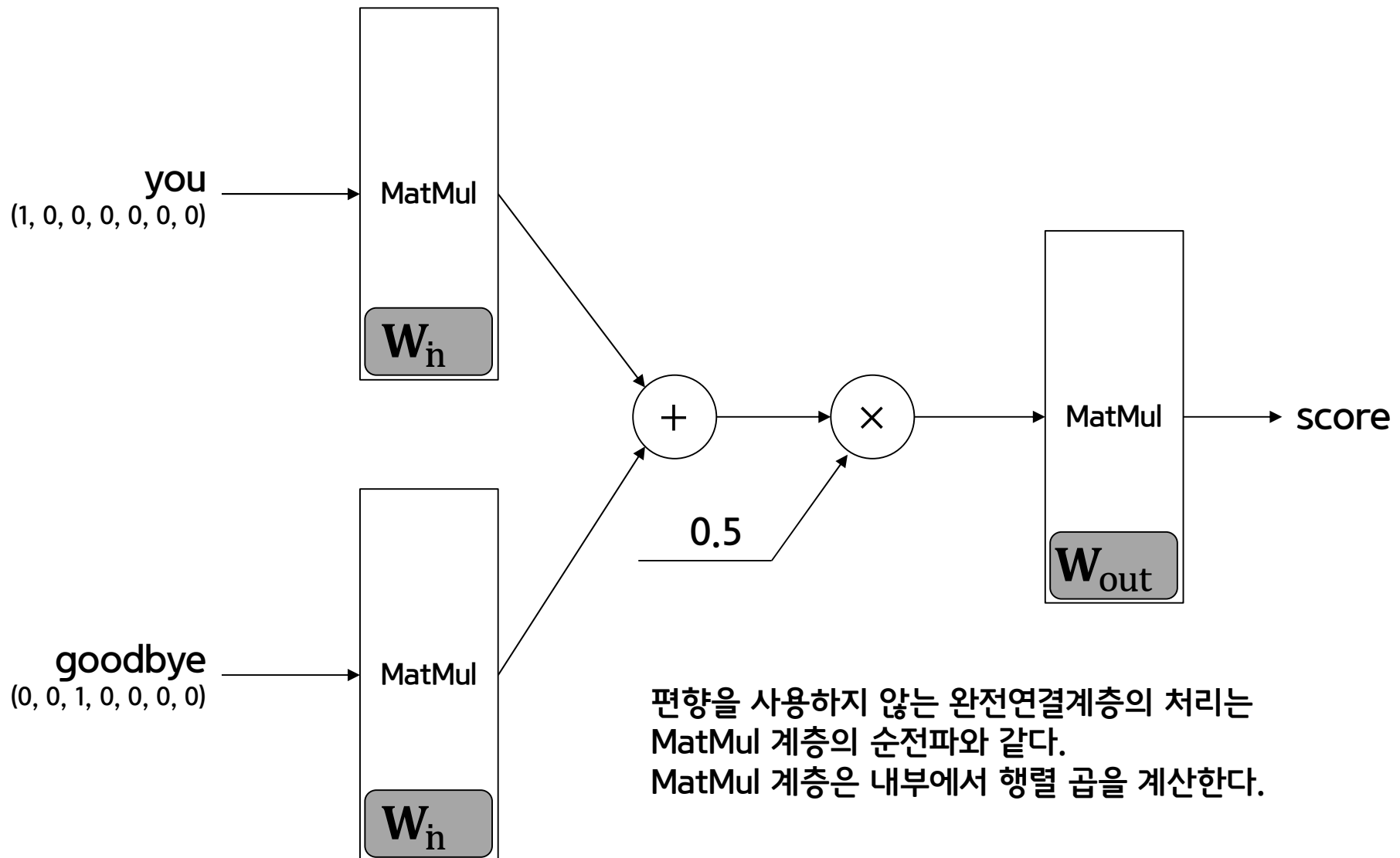
은닉층의 뉴런 수를 입력 층의 뉴런 수보다 적게 하는 것이 중요한 핵심이다. 이렇게 해야 은닉층에는 단어 예측에 필요한 정보를 간결하게 담게 되며, 결과적으로 밀집벡터 표현을 얻을 수 있다.

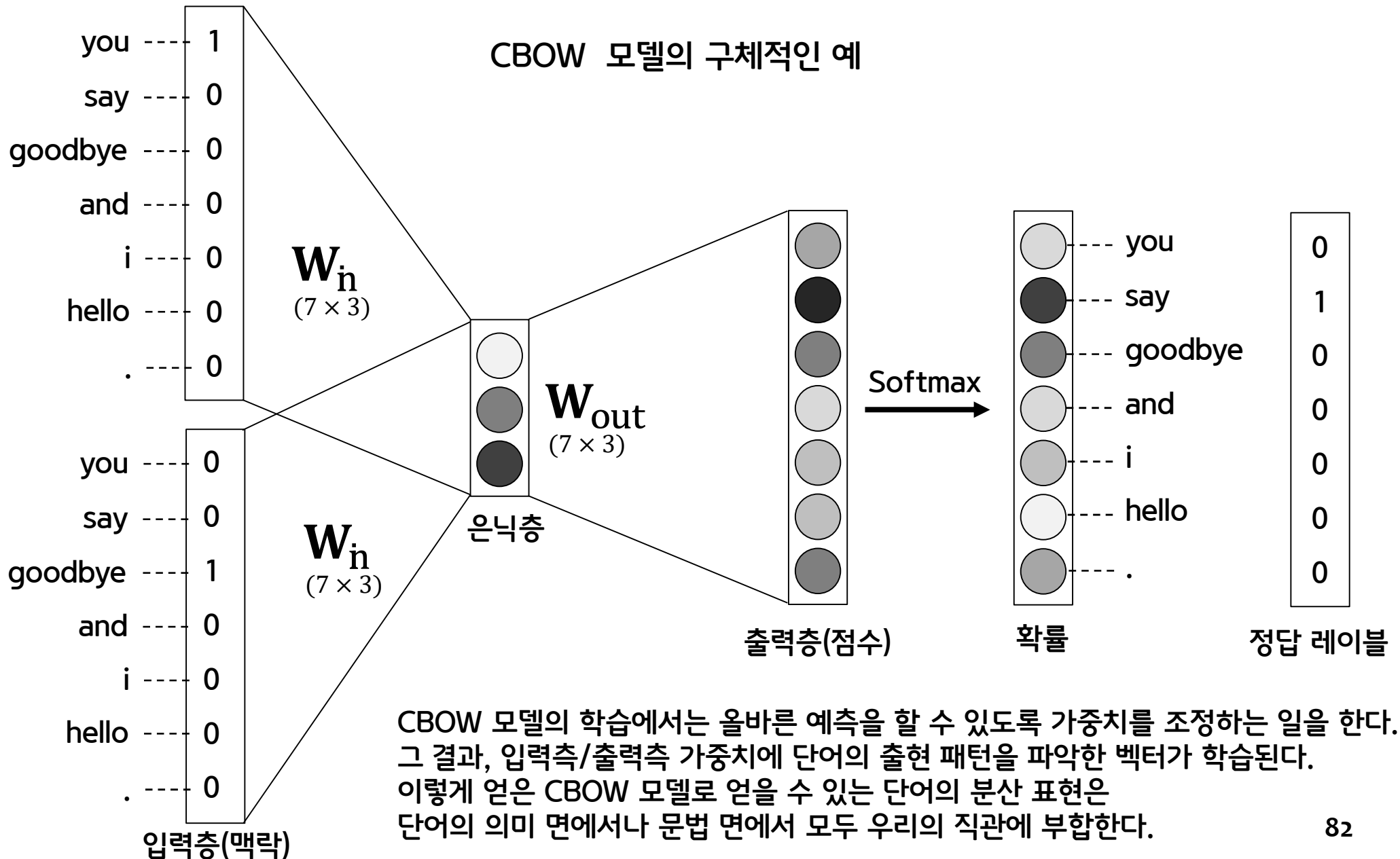
이때 은닉층 정보는 인간이 이해할 수 없는 코드로 쓰여 있다. (인코딩)

한편, 은닉층의 정보로부터 원하는 결과를 얻는 작업은 디코딩이라고 한다.

즉, 디코딩이란 인코딩된 정보를 인간이 이해할 수 있는 표현으로 복원하는 작업이다.

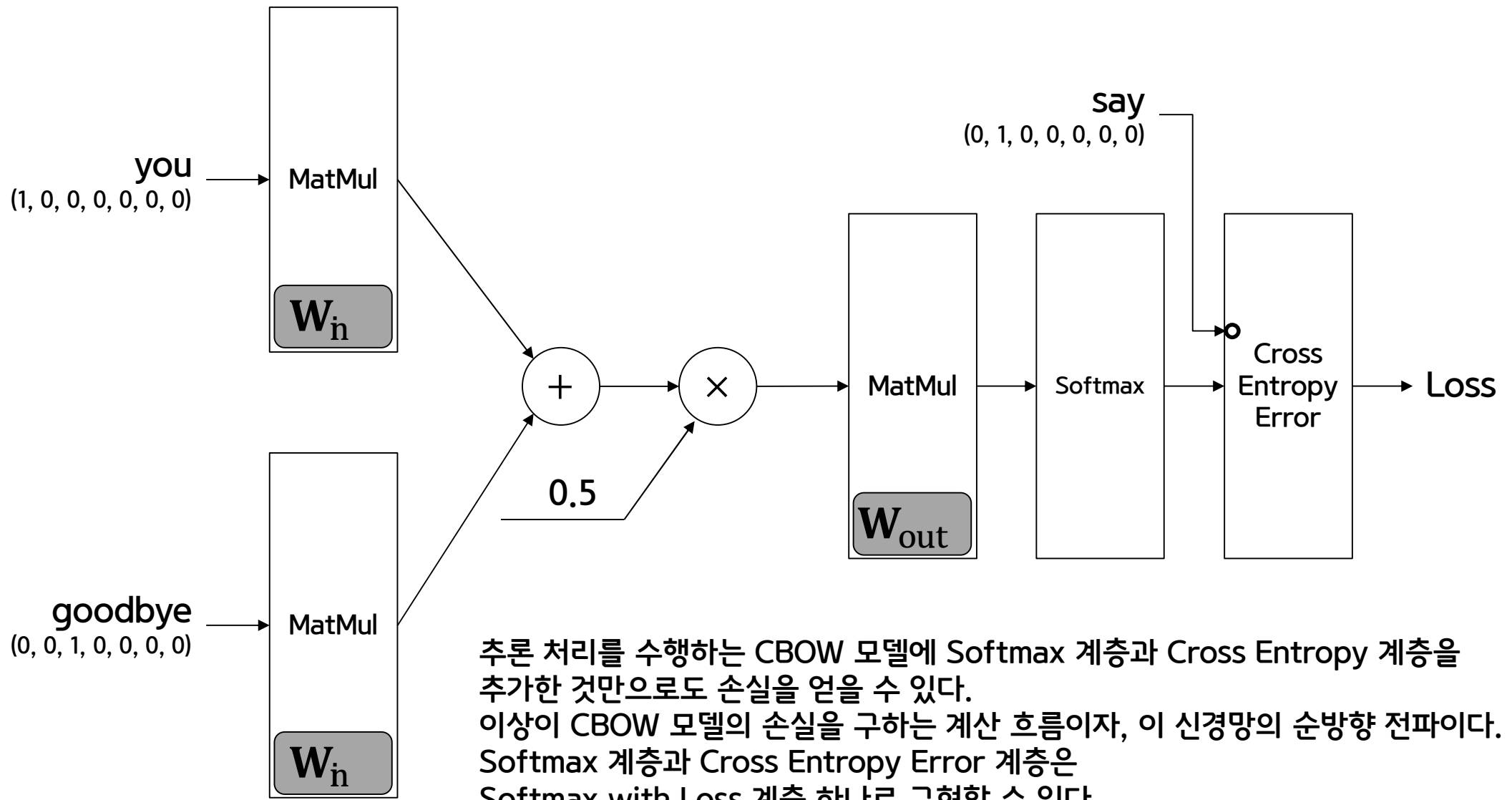
계층 관점에서 본 CBOW 모델의 신경망 구성

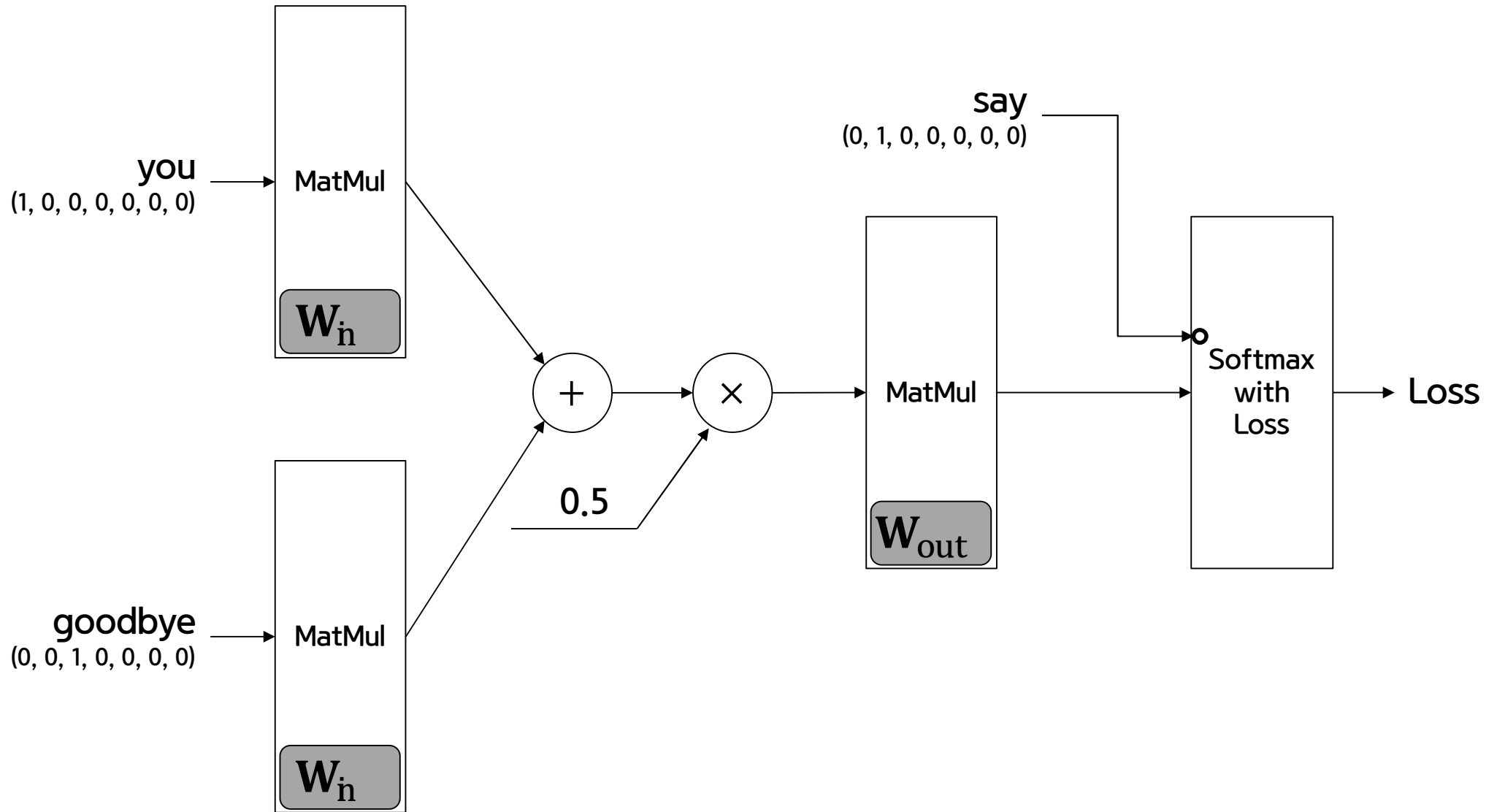




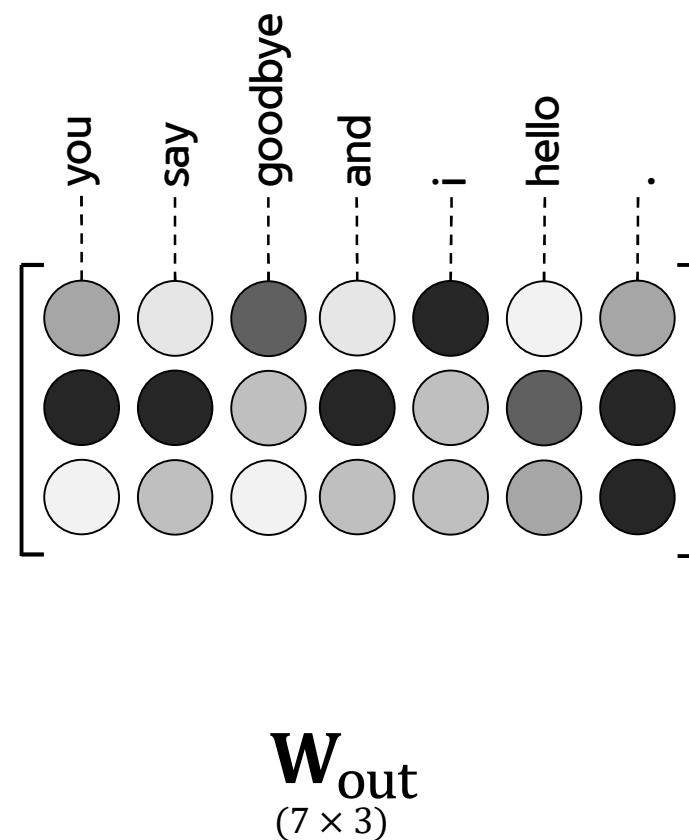
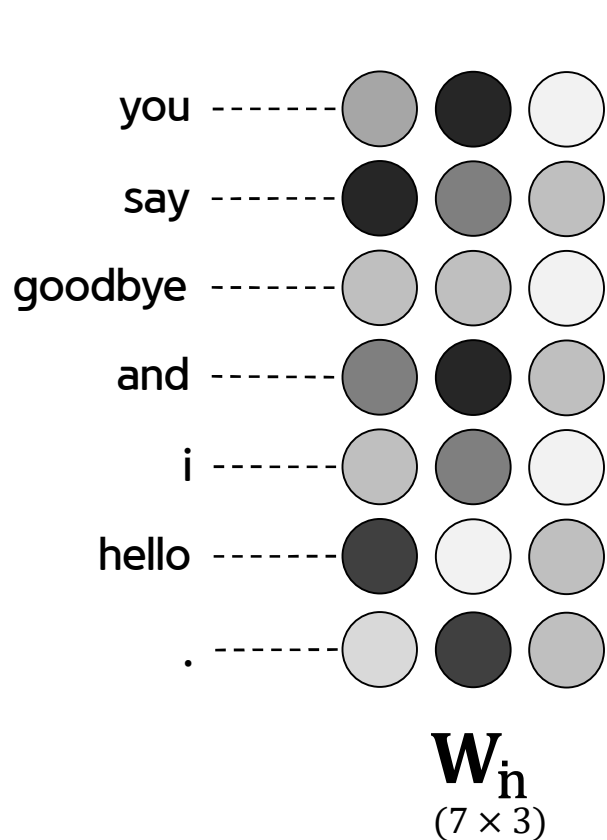
신경망의 학습에 대해 생각해보자.
우리가 다루는 모델은 다중 클래스 분류를 수행하는 신경망이다.
따라서 이 신경망을 학습하려면, 소프트맥스 함수와 교차 엔트로피 오차만 이용하면 된다.

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





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



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.

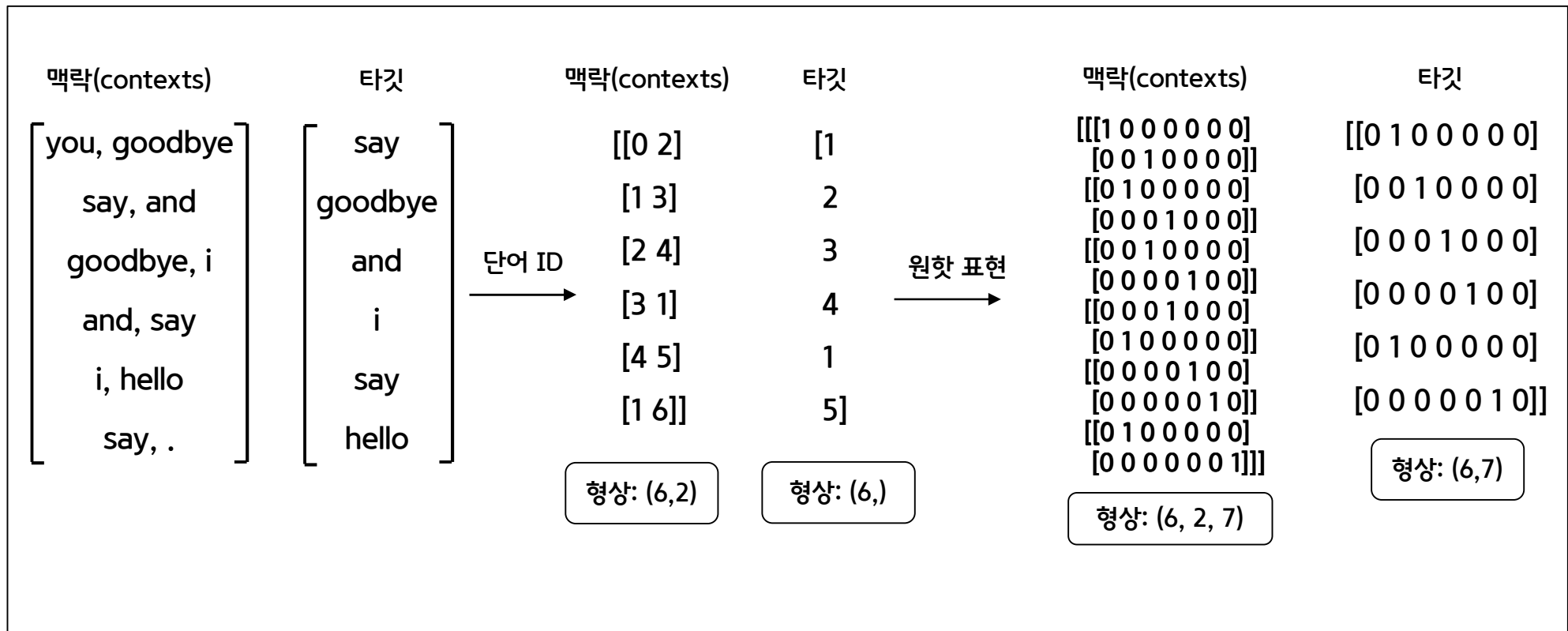
말뭉치에서 맥락과 타깃을 만드는 예

말뭉치	맥락(contexts)	타깃
you <u>say</u> <u>goodbye</u> and i say hello .	[you, goodbye	[say
you <u>say</u> <u>goodbye</u> <u>and</u> i say hello .	say, and	goodbye
you say <u>goodbye</u> <u>and</u> i say hello .	goodbye, i	and
you say goodbye <u>and</u> <u>i</u> <u>say</u> hello .	and, say	i
you say goodbye and i <u>say</u> <u>hello</u> .	i, hello	say
you say goodbye and i <u>say</u> <u>hello</u> .	say, .	hello

맥락과 타깃을 원핫 표현으로 변환하는 예

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

맥락과 타깃을 원핫 표현으로 변환하는 예



3. word2vec

3.1 추론 기반 기법과 신경망

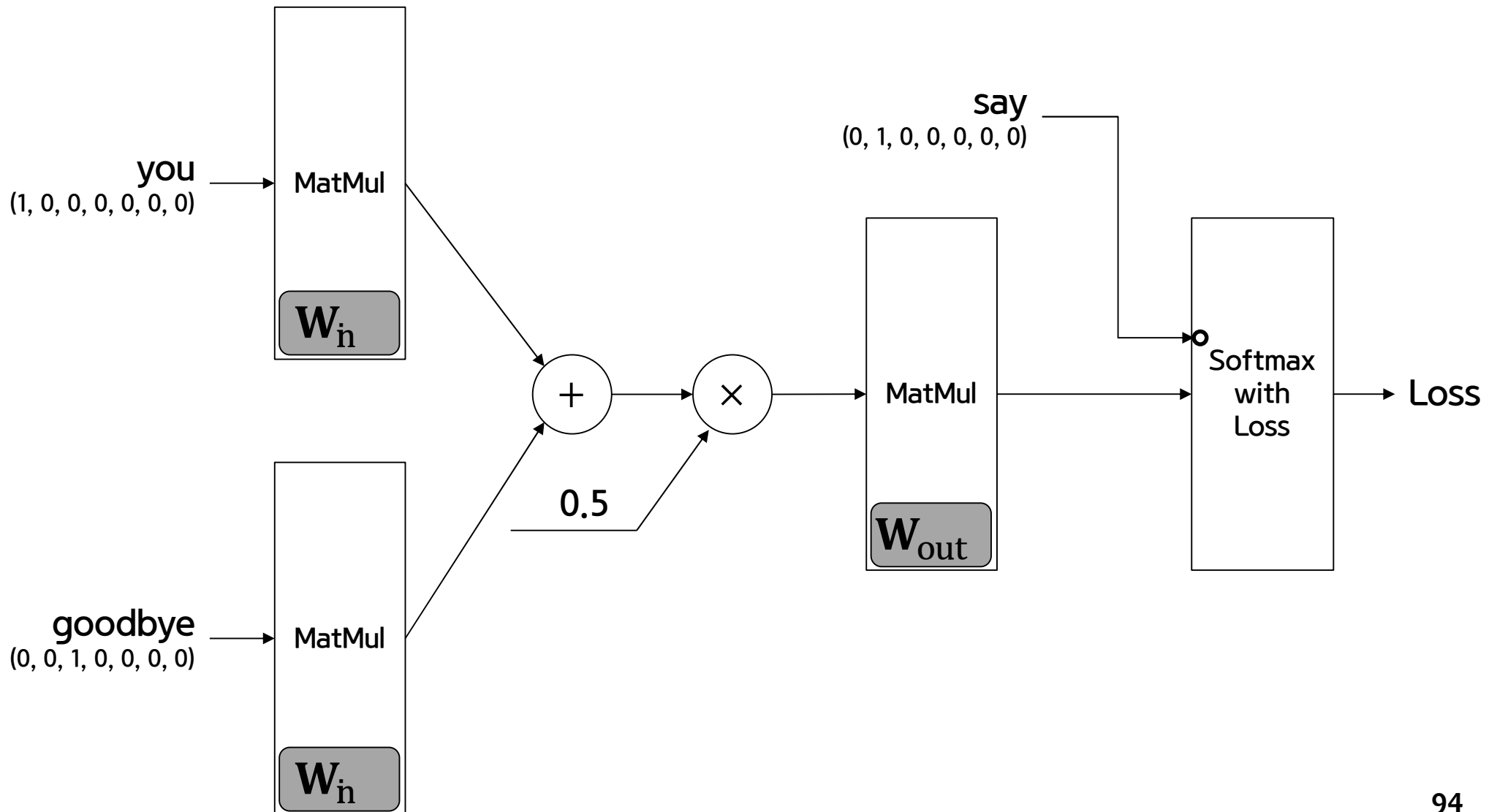
3.2 단순한 word2vec

3.3 학습 데이터 준비

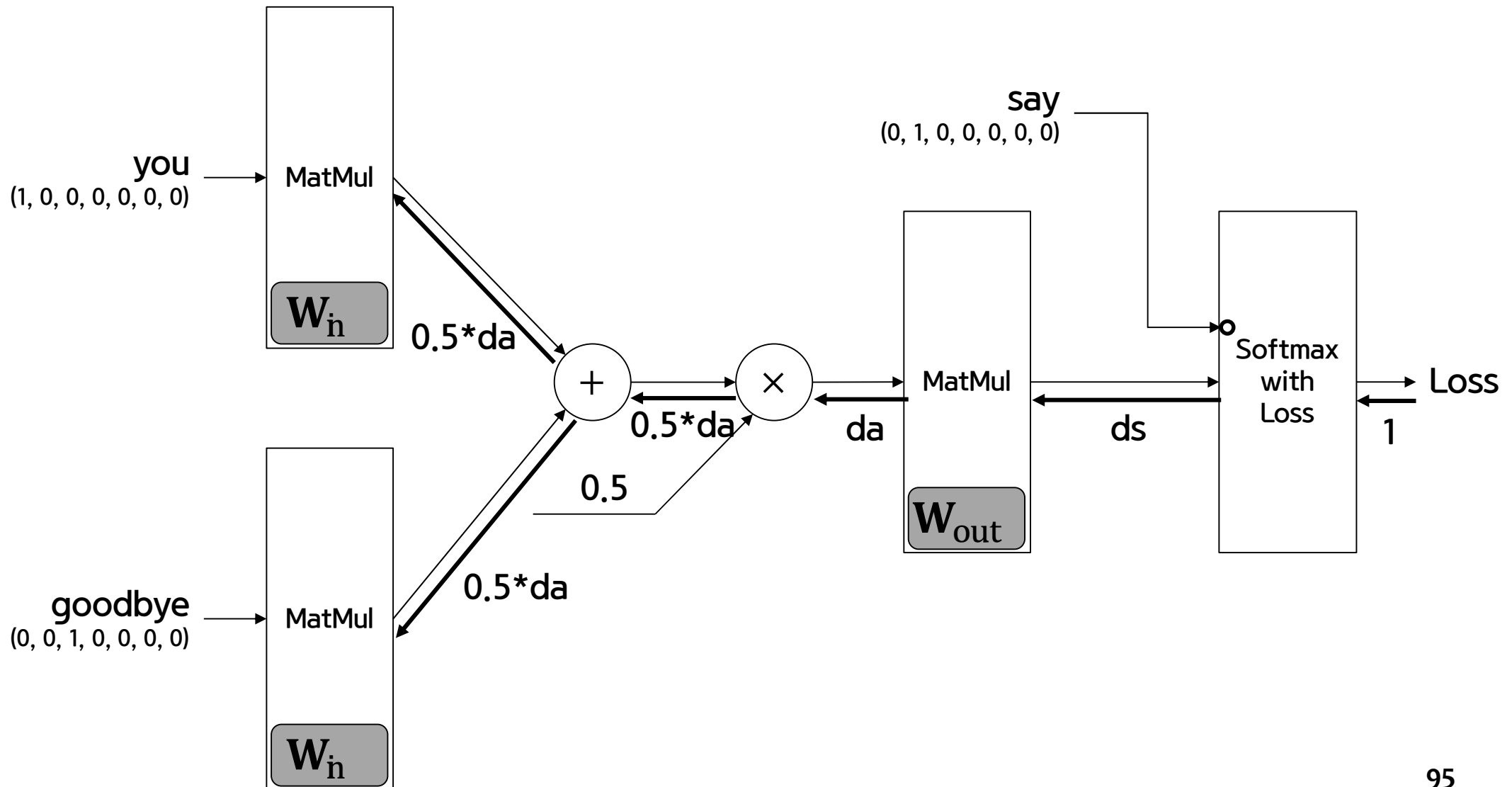
3.4 CBOW 모델 구현

3.5 word2vec 보충

CBOW 모델의 신경망 구성



CBOW 모델의 역전파



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.

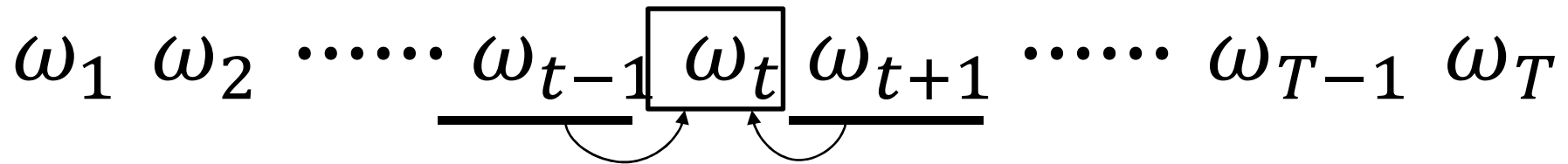
사후 확률 $P(A|B)$, 사건이 일어난 후의 확률.

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.

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

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.

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

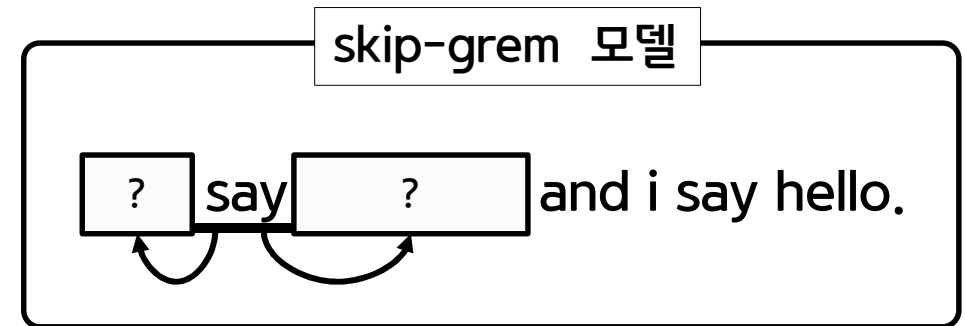
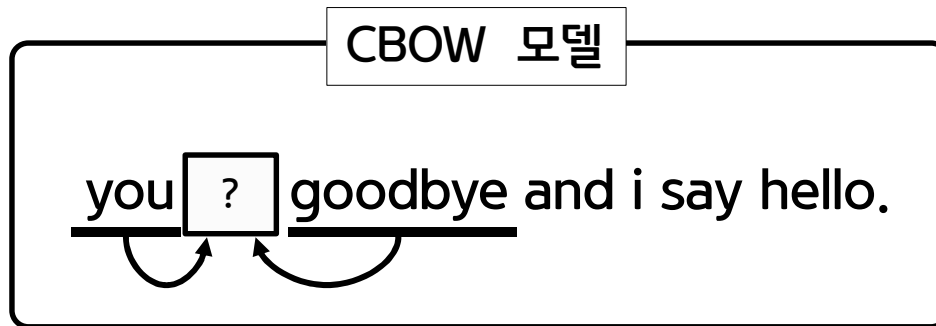
CBOW 모델의 학습이 수행하는 일은, 손실 함수 식을 가능한 작게 만드는 것이다. 그리고 이때의 가중치 매개변수가 우리가 얻고자 하는 단어의 분산 표현이다.

word2vec은 2개의 모델을 제안하고 있다.

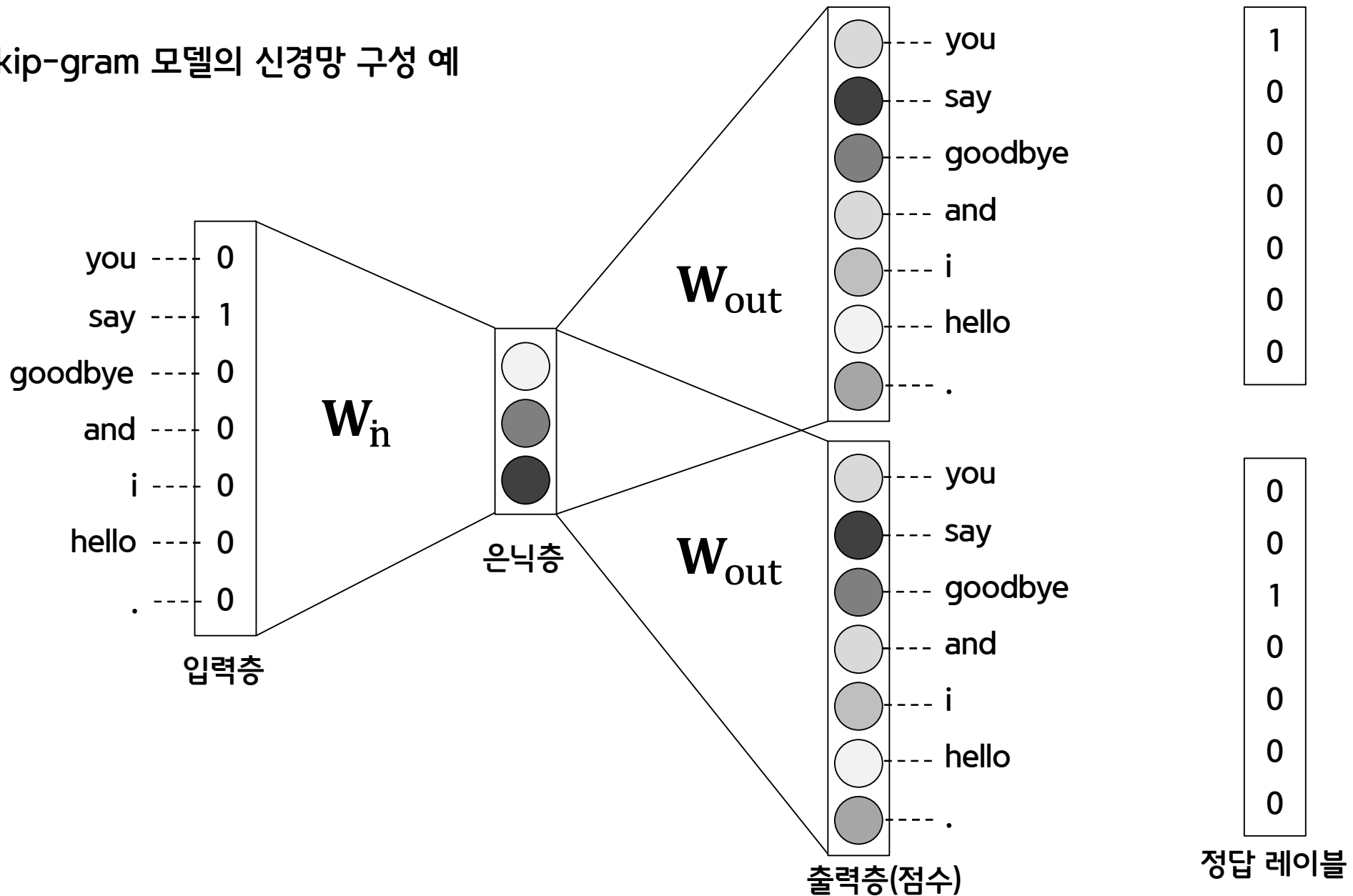
1. CBOW 모델
2. skip-gram 모델

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.

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

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.

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

위 식을 CBOW 모델의 식과 비교해보자.

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

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 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.
GloVe의 기본 아이디어는, 말뭉치 전체의 통계 정보를 손실 함수에 도입해 미니매치 학습을 하는 것이다.

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.

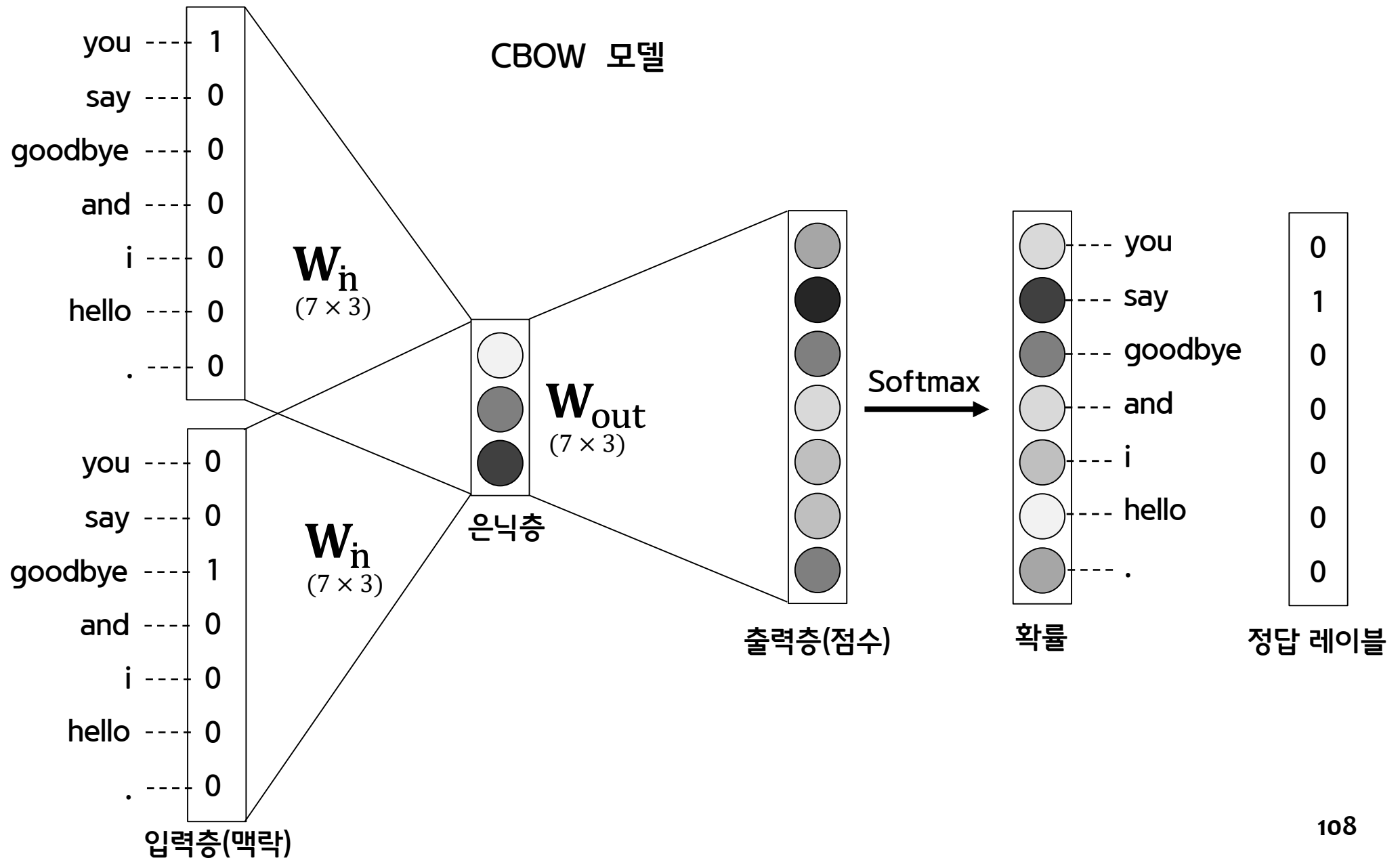
1. Embedding 이라는 새로운 계층을 만든다.
2. 네거티브 샘플링 이라는 새로운 손실함수를 도입한다.

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)

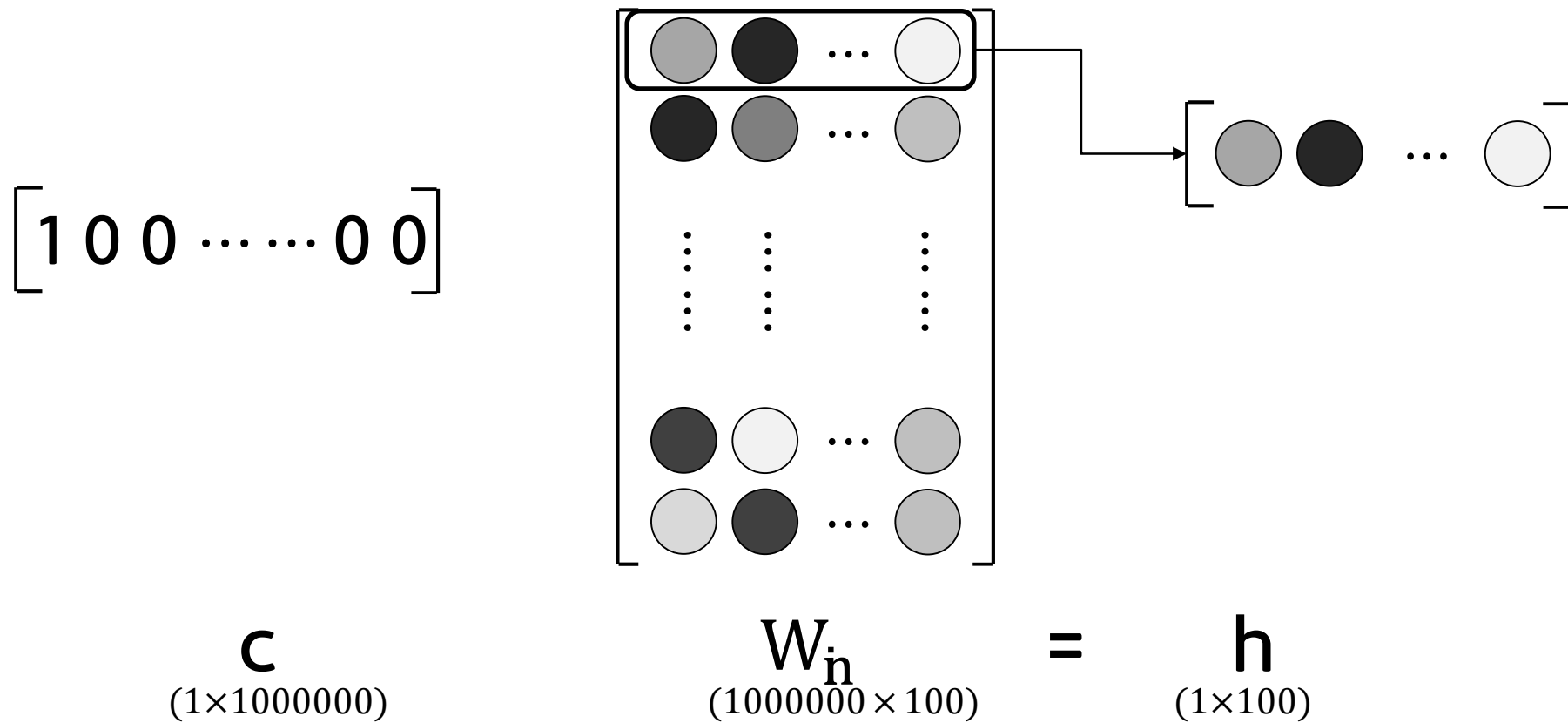
"am": (0.1, 0.8)

"a": (0.5, 0.6)

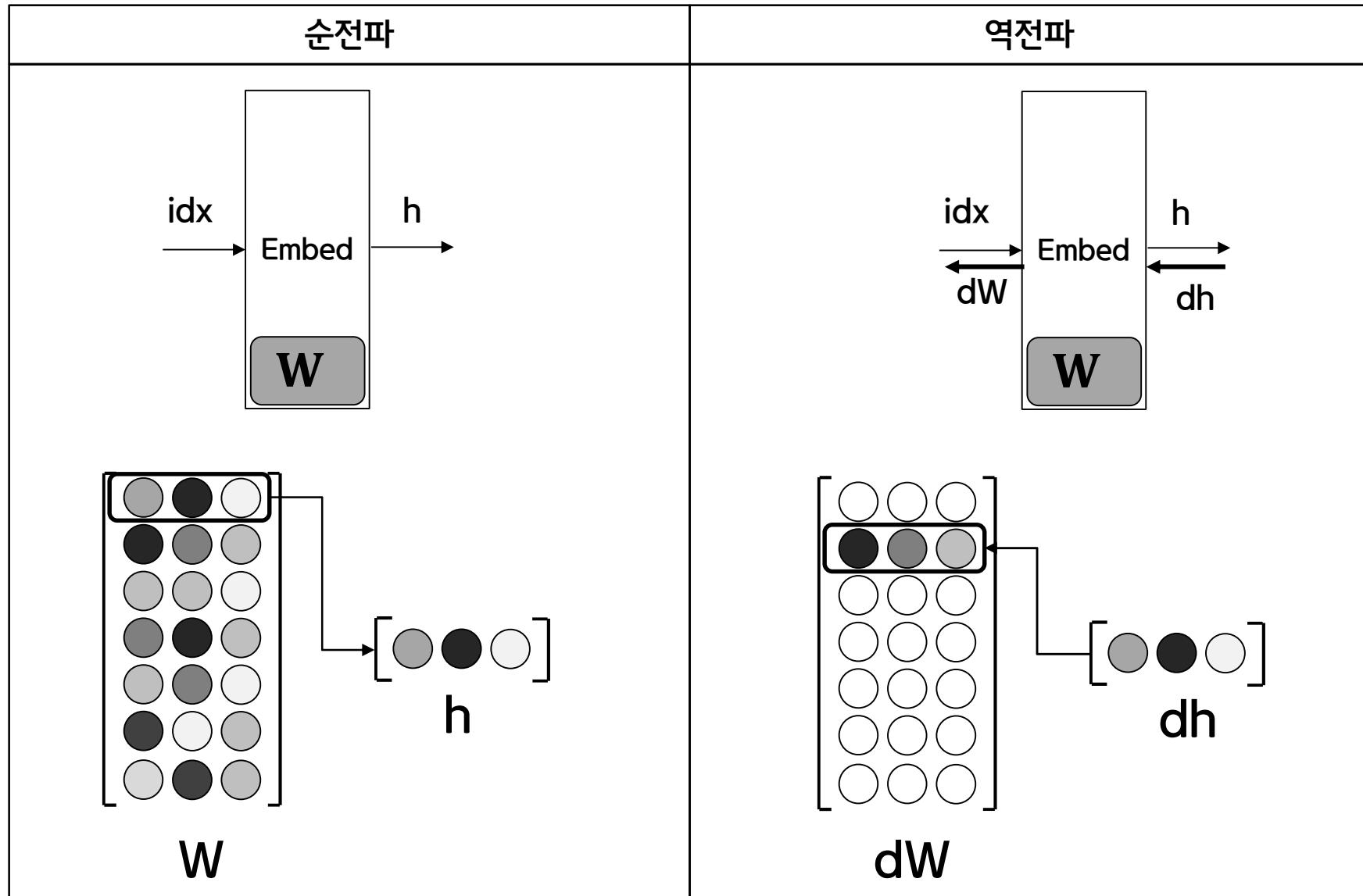
"boy": (0.2, 0.9)

"girl": (0.4, 0.7)

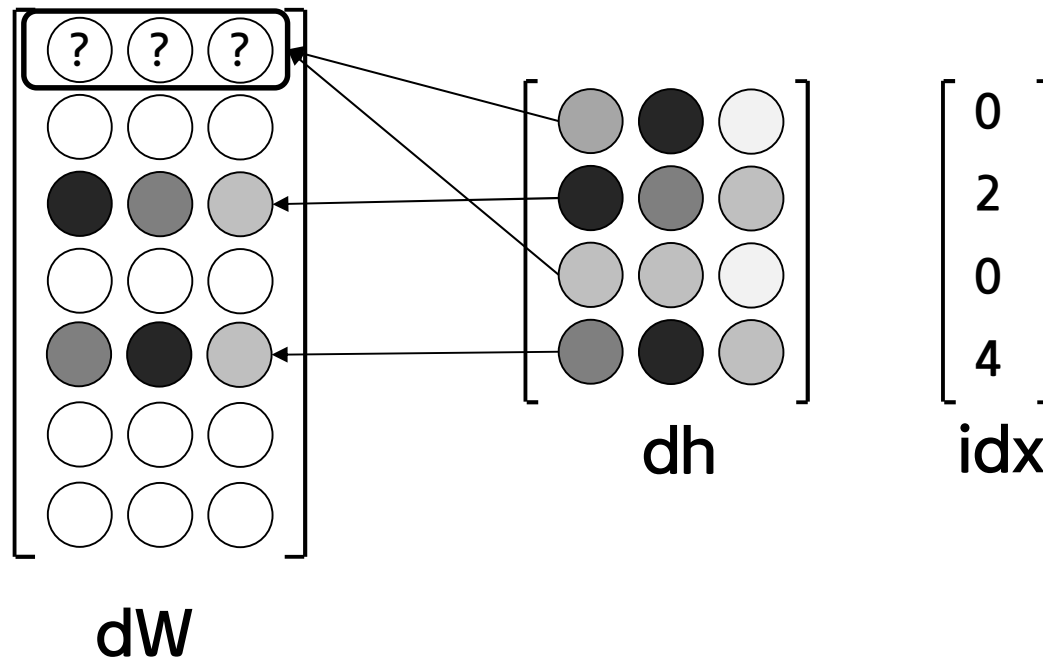
맥락(원핫 표현)과 MatMul 계층의 가중치를 곱한다.



Embedding 계층의 forward와 backward 처리



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

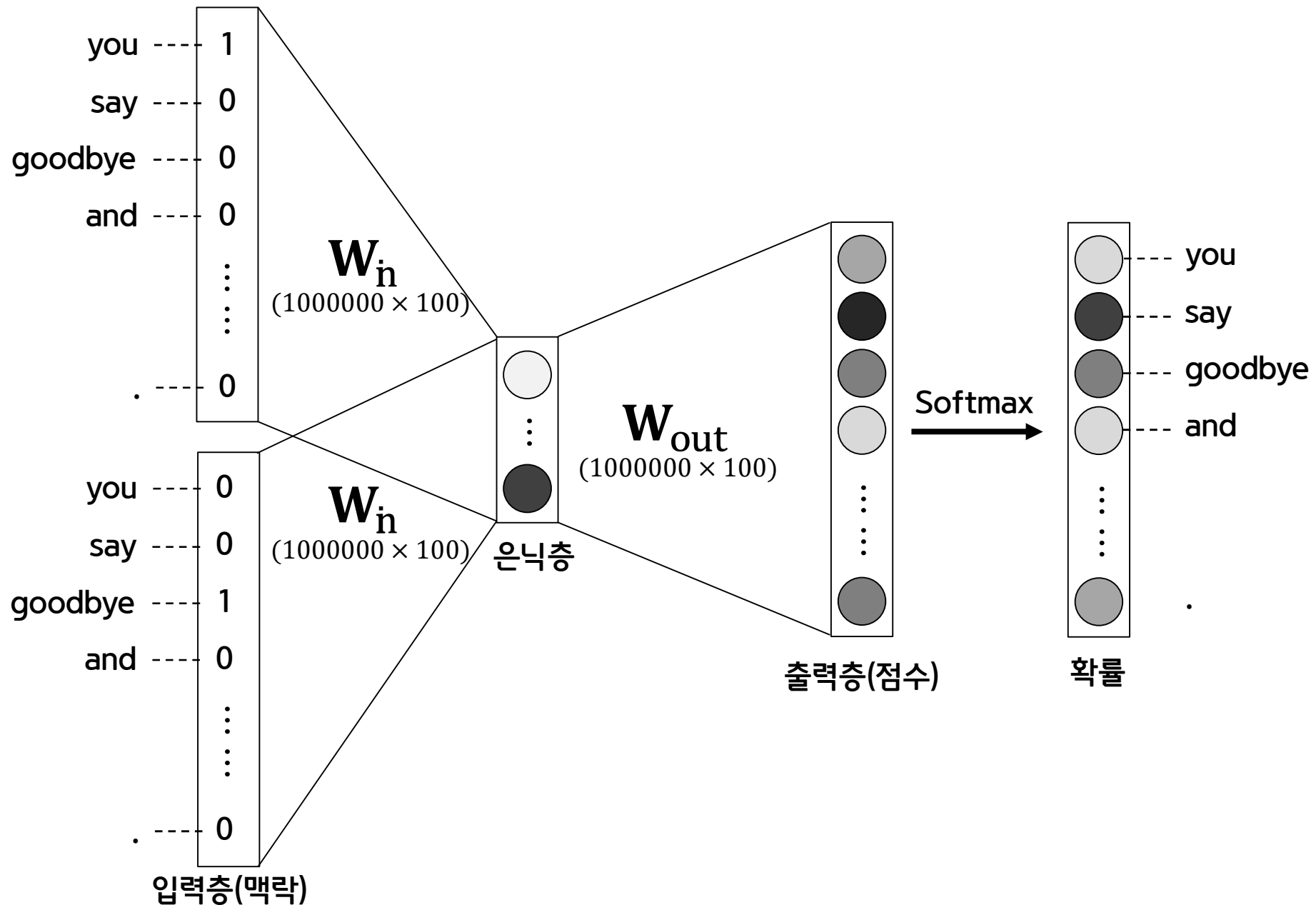
남은 병목은 은닉층 이후의 처리(행렬 곱과 softmax 계층의 계산) 이다.

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

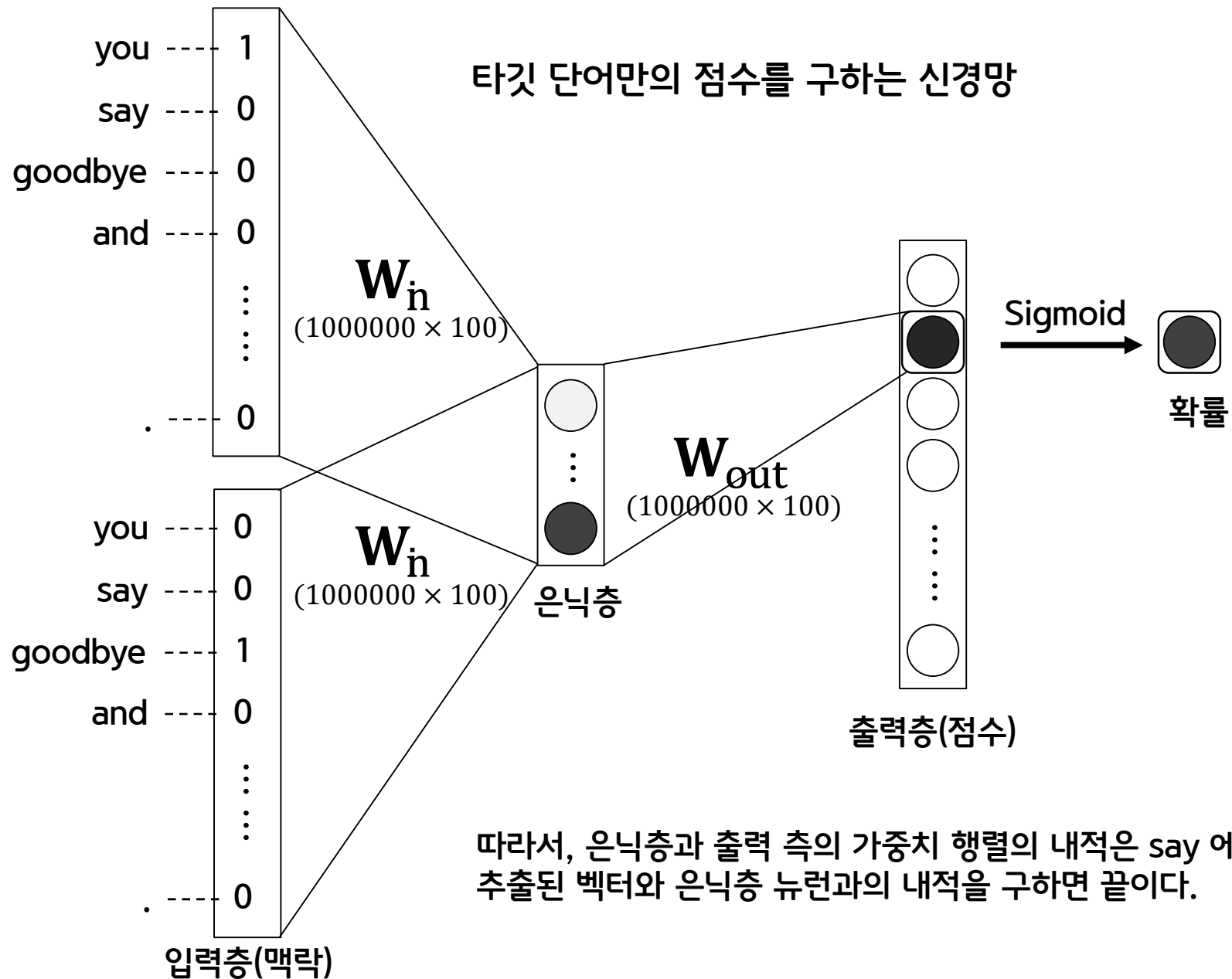
네거티브 샘플링의 핵심은 다중분류를 이진분류로 근사하는 것이다.

예를 들면,

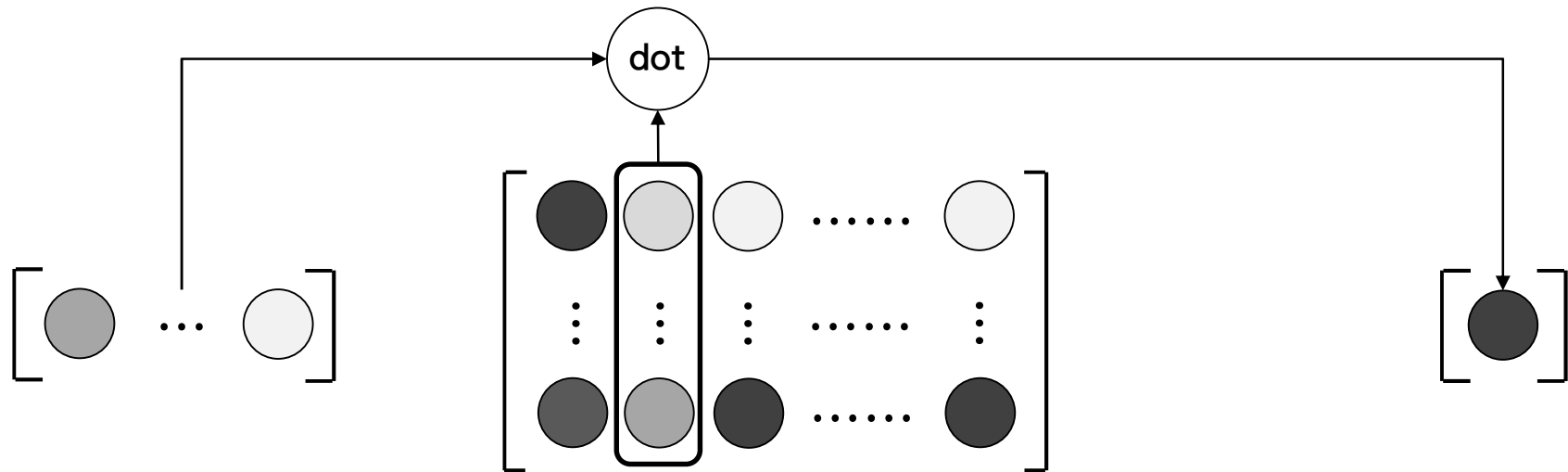
다중 분류는 맥락이 you 와 goodbye 일때, 타깃 단어는 무엇입니까?
에 대답하는 것이고,

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.

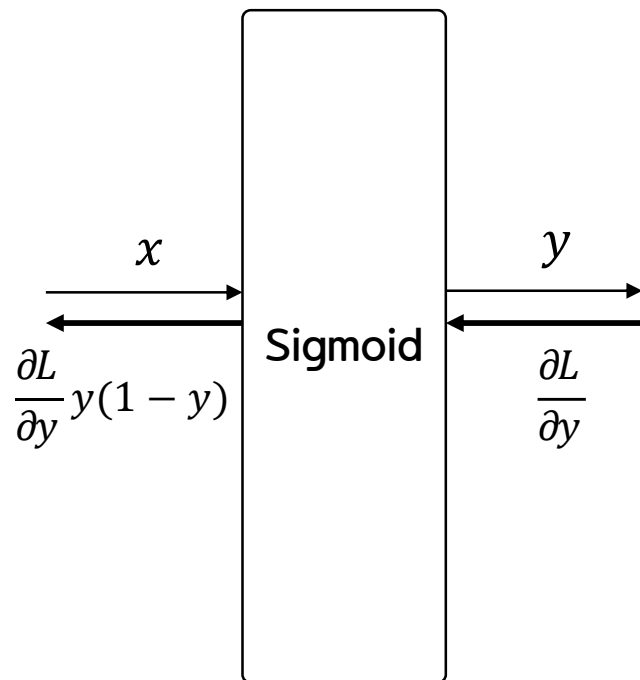


형상 : h
(1×100)

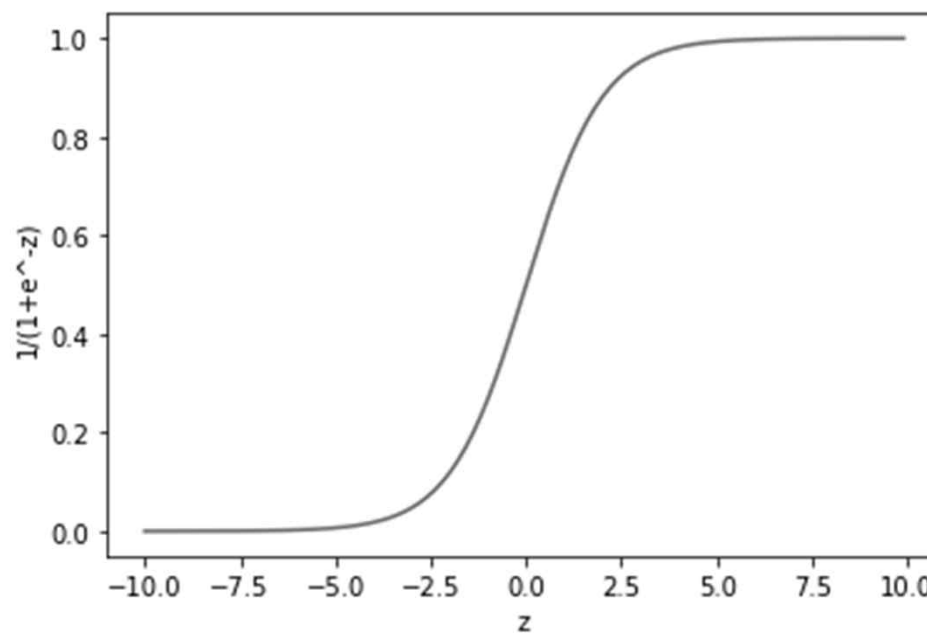
W_{out}
(100 × 1000000)

s
(1×1)

Sigmoid 계층과 시그모이드 함수의 그래프

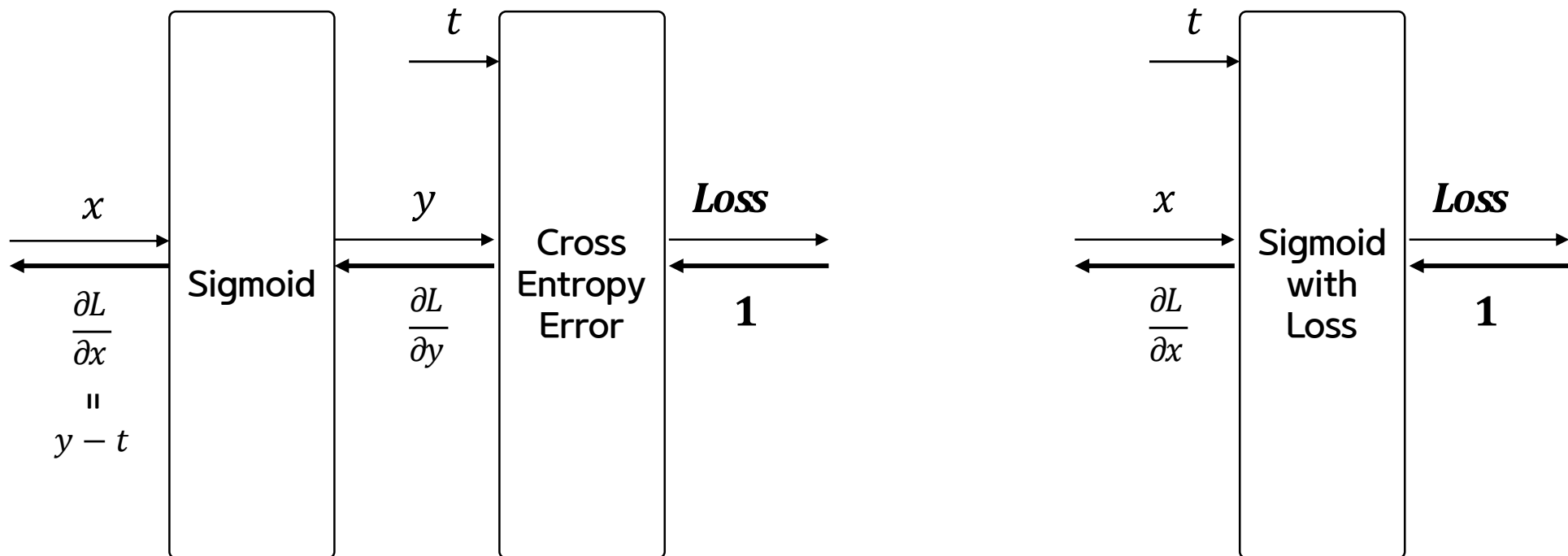


$$y = \frac{1}{1 + e^{-x}}$$

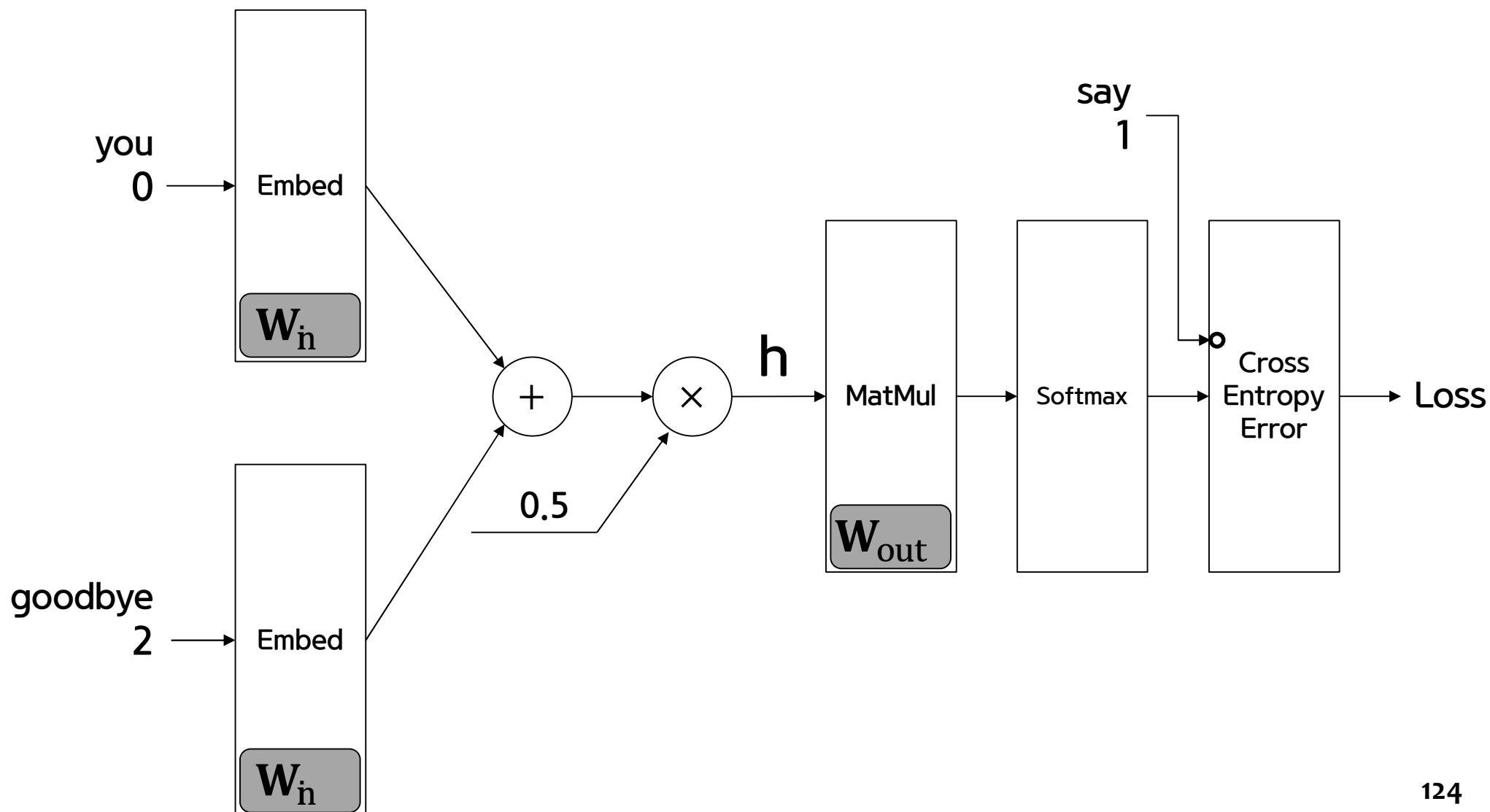


Sigmoid 계층과 Cross Entropy Error 계층의 계산 그래프

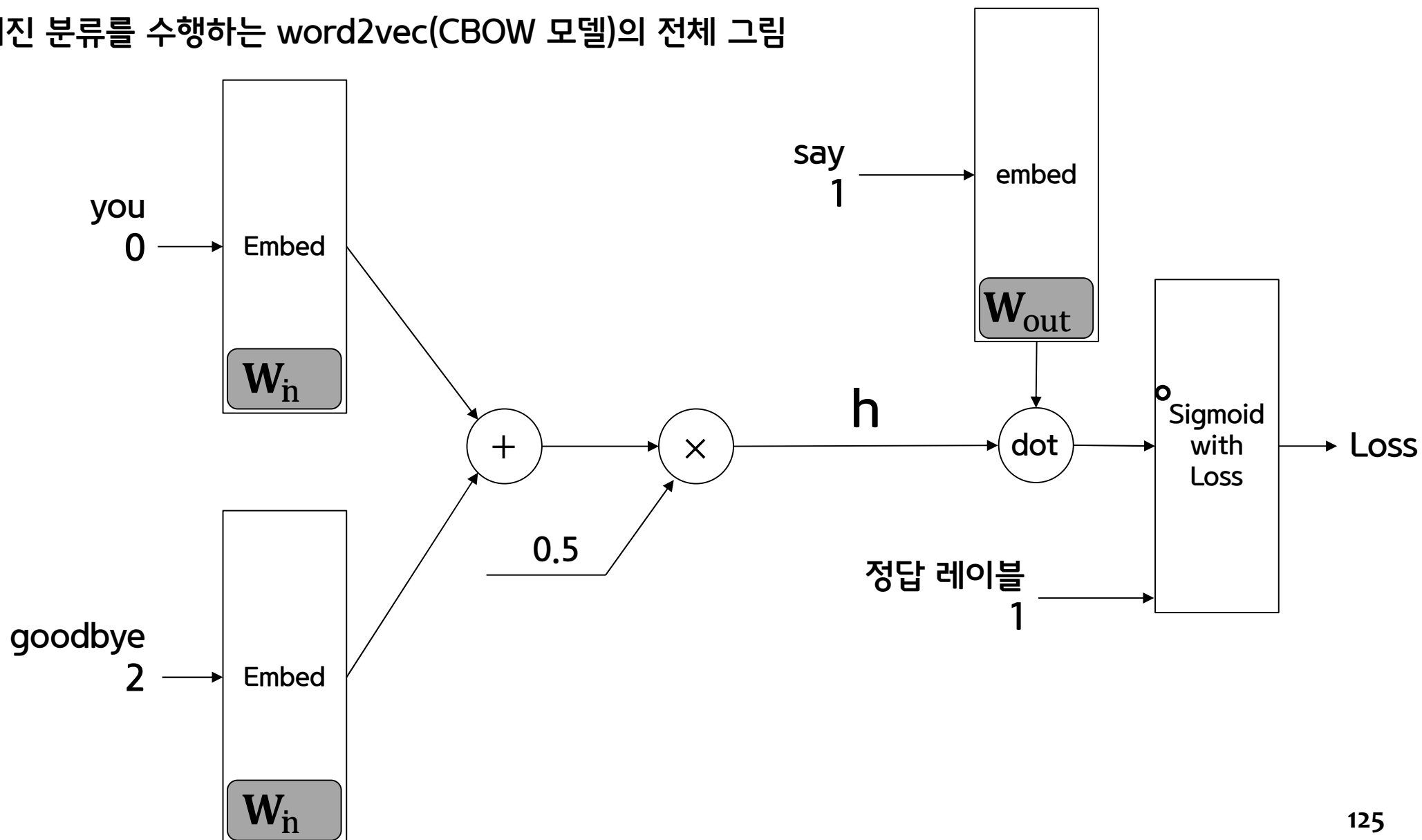
$$L = -(t \log y + (1 - t) \log(1 - t))$$



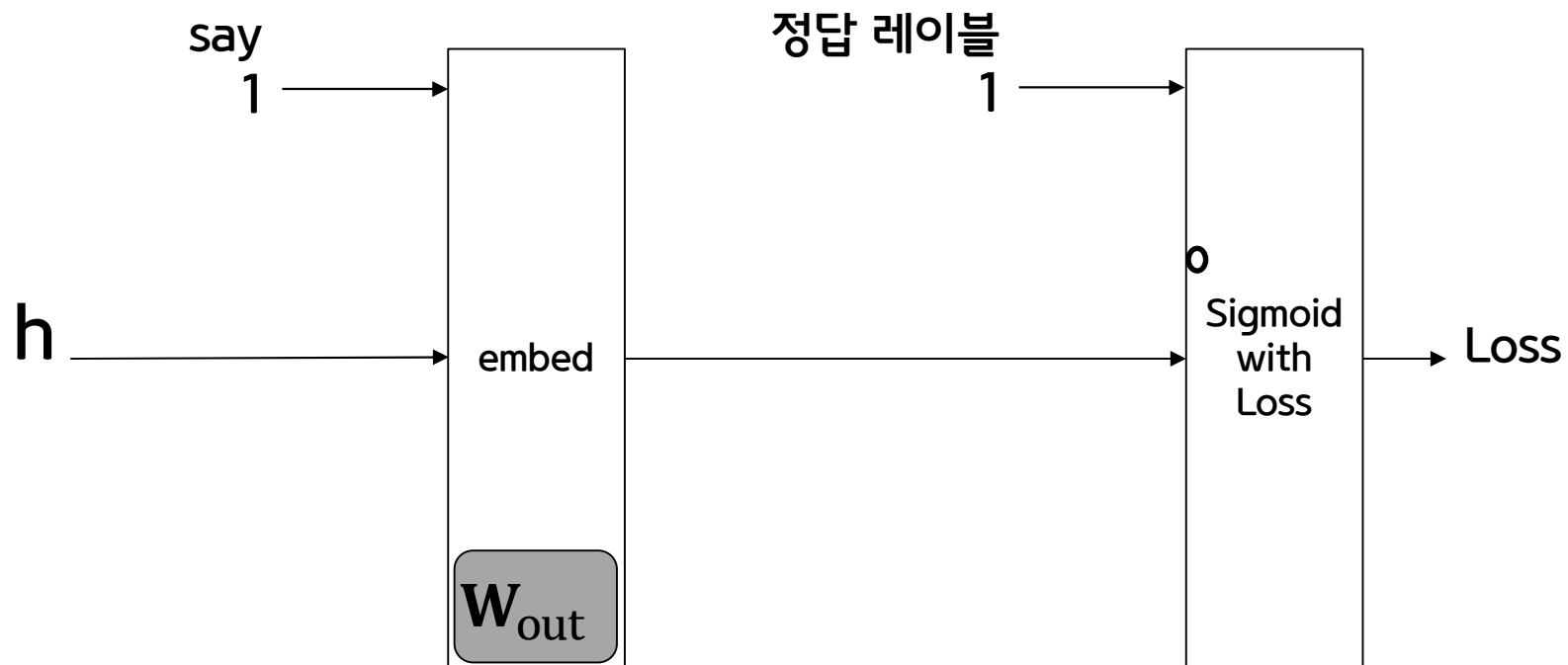
다중 분류를 수행하는 CBOW 모델의 전체 그림



이진 분류를 수행하는 word2vec(CBOW 모델)의 전체 그림



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

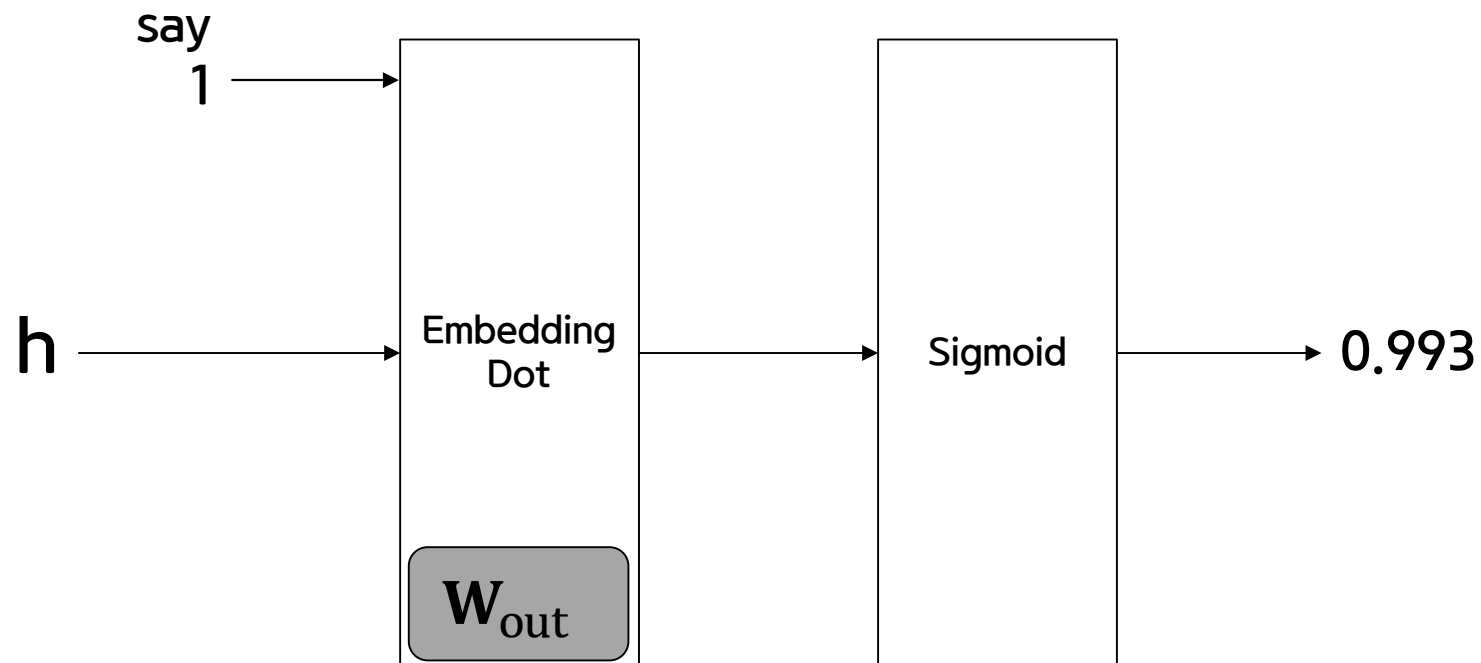


Embedding Dot 계층의 각 변수의 구체적인 값

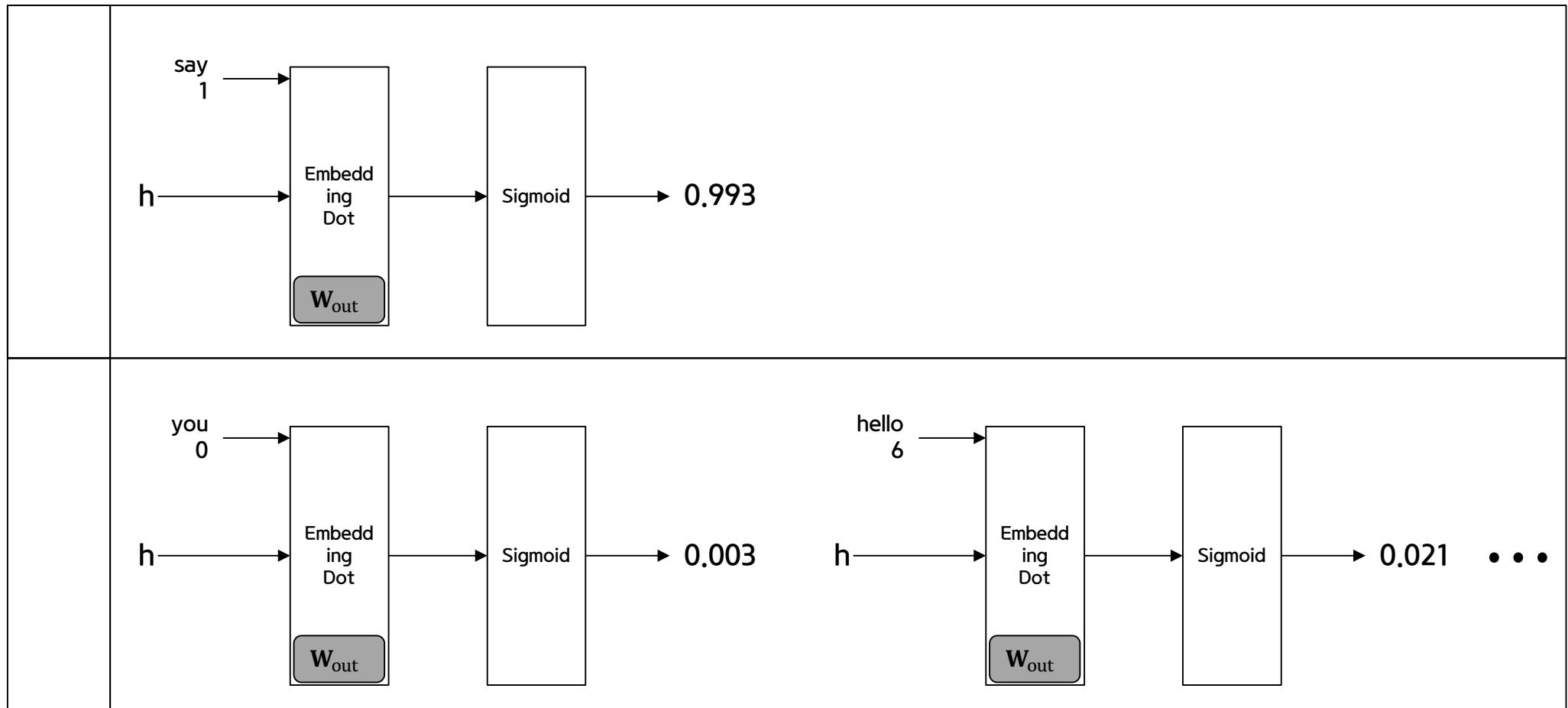
```
embed = Embedding(W)
target_W = embed.forward(idx)
out = np.sum(target_W*h, axis=1)
```

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

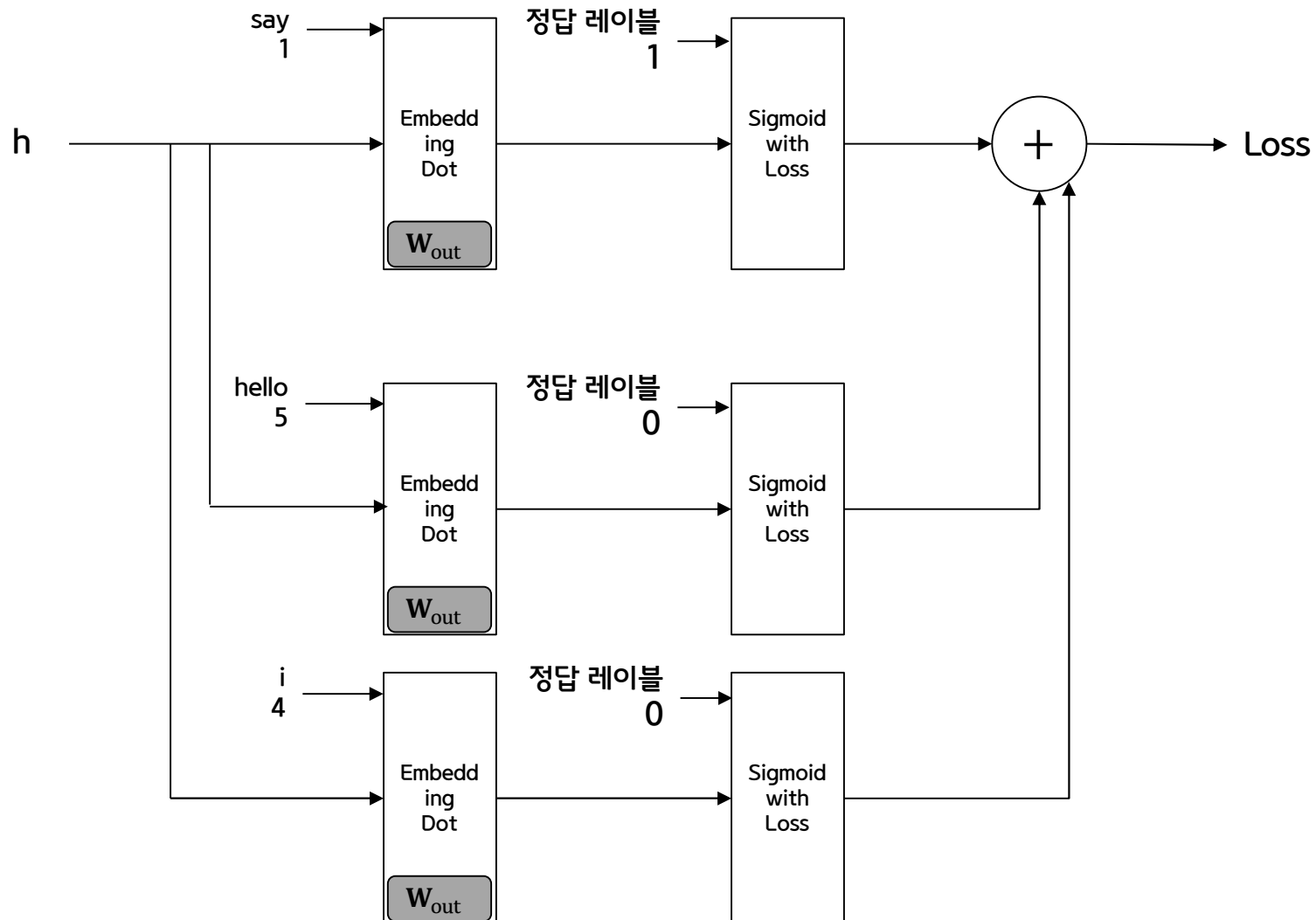
CBOW 모델의 은닉층 이후의 처리 예: 맥락은 "you"와 "goodbye"이고, 타겟이 "say"일 확률은 0.993(99.3%)이다.



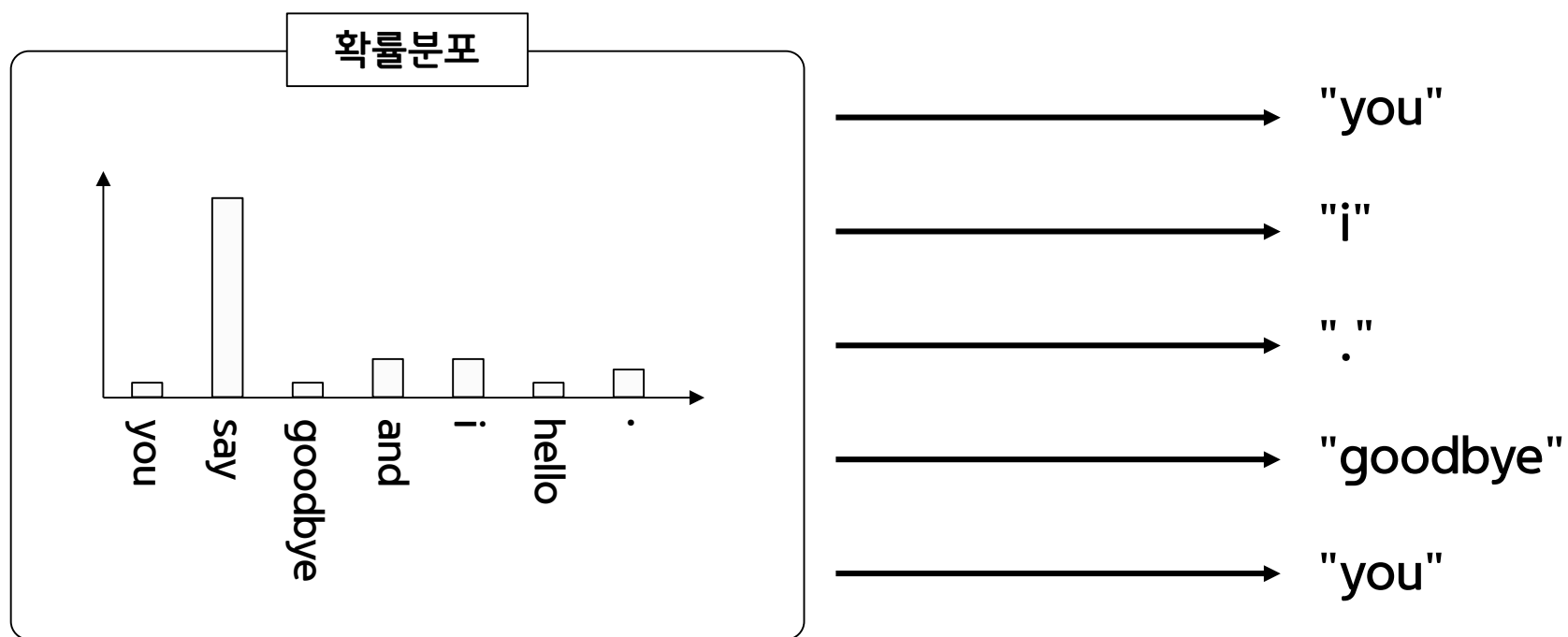
긍정적 예(정답)를 "say"라고 가정하면, "say"를 입력했을 때의 Sigmoid 계층 출력은 1에 가깝고, "say" 이외의 단어를 입력했을 때의 출력은 0에 가까워야 한다. 이런 결과를 내어주는 가중치가 필요하다.



네거티브 샘플링의 예



확률분포에 따라 샘플링을 여러 번 수행한다.



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

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.

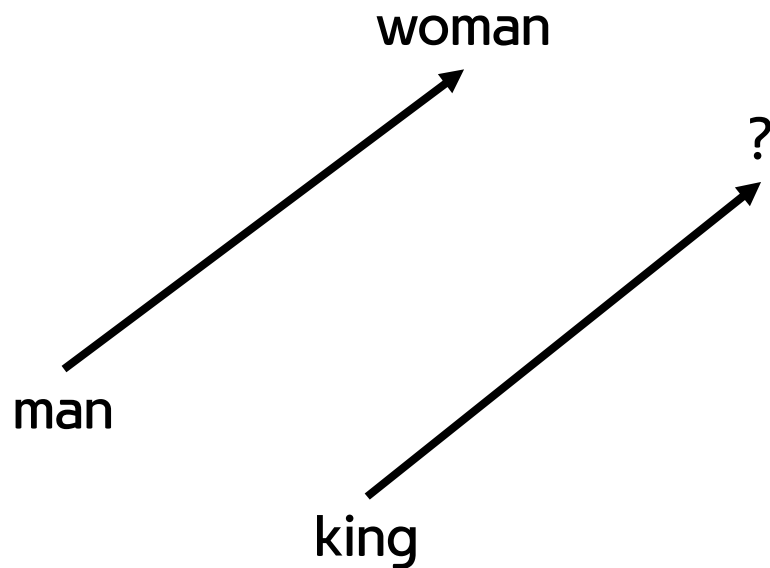
다음은 개선된 CBOW 모델 코드다.

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

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]

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



5. 순환 신경망(RNN)

5.1 확률과 언어 모델

5.2 RNN 이란

5.3 RNN 구현

5.4 시계열 데이터 처리 계층 구현

5.5 RNNLM 학습과 평가

피드포워드(feed forward) 신경망

- 흐름이 단방향
- 시계열 데이터의 성질(패턴)을 충분히 학습할 수 없음

순환 신경망(Recurrent Neural Network, RNN)의 등장

CBOW(Continuous bag-of-words)모델

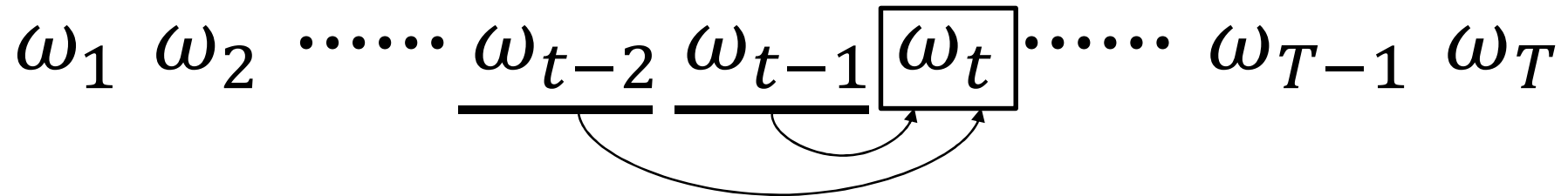
- CBOW 모델의 학습 -> 손실함수(말뭉치 전체의 손실함수의 총합)를 최소화하는 가중치 매개변수를 찾는다. -> 맥락으로부터 타깃을 더 정확하게 추측 가능
- 맥락 안의 단어 순서가 무시된다는 한계가 있음
말뭉치 : $w_1, w_2, w_3, \dots, w_t$

word2vec의 CBOW 모델 : 맥락의 단어로부터 타깃 단어를 추측한다.

$$\omega_1 \quad \omega_2 \quad \cdots \quad \omega_{t-1} \quad \boxed{\omega_t} \quad \omega_{t+1} \quad \cdots \quad \omega_{T-1} \quad \omega_T$$

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

왼쪽 윈도우만 맥락으로 고려한다.



$w(t-2)$ 과 $w(t-1)$ 이 주어졌을 때 타겟이 w_t 가 될 확률(CBOW 모델이 출력할 확률)을 수식으로 표현하면

$$P(w_t | w_{t-2}, w_{t-1})$$

CBOW모델이 다루는 손실함수

$$L = -bgP(w_t | w_{t-2}, w_{t-1})$$

단어 나열에 확률을 부여
특정한 단어의 시퀀스에 대해서 그 시퀀스가 일어날 가능성이
어느 정도인지(얼마나 자연스러운 단어 순서인지)를 확률로 평가한다.

- 언어 모델의 사용
 기계 번역과 음성 인식
 새로운 문장을 생성

w_1, \dots, w_m 이라는 m 개 단어로 된 문장이 있을 때
 w_1, \dots, w_m 순서로 출현할 확률 $P(w_1, \dots, w_m)$
(여러 사건이 동시에 일어날 확률이므로 동시확률이라고 한다.)

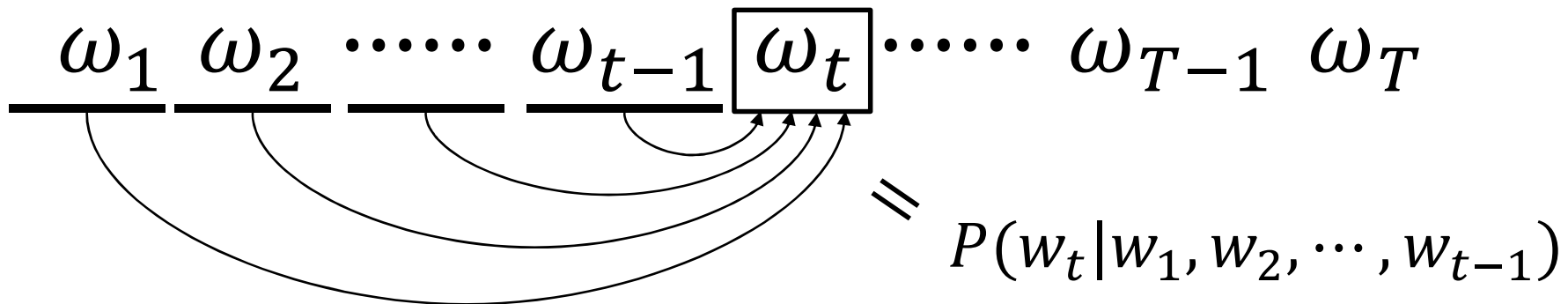
$$\begin{aligned} P(w_1, \dots, w_m) &= P(w_m | w_1, \dots, w_{m-1}) P(w_{m-1} | w_1, \dots, w_{m-1}) \\ &\quad \dots P(w_3 | w_1, w_2) P(w_2 | w_1) P(w_1) \\ &= \prod_{t=1}^m P(w_t | w_1, \dots, w_{t-1}) \end{aligned}$$

$$P(A, B) = P(A|B)P(B)$$

A, B가 모두 일어날 확률 $P(A, B)$ 는 B가 일어날 확률 $P(B)$ 와 B가 일어난 후 A가 일어날 확률 $P(A|B)$ 를 곱한 값과 같다.

$$P(\underbrace{w_1, \dots, w_{m-1}}_A, w_m) = P(A, w_m) = P(w_m|A)P(A)$$

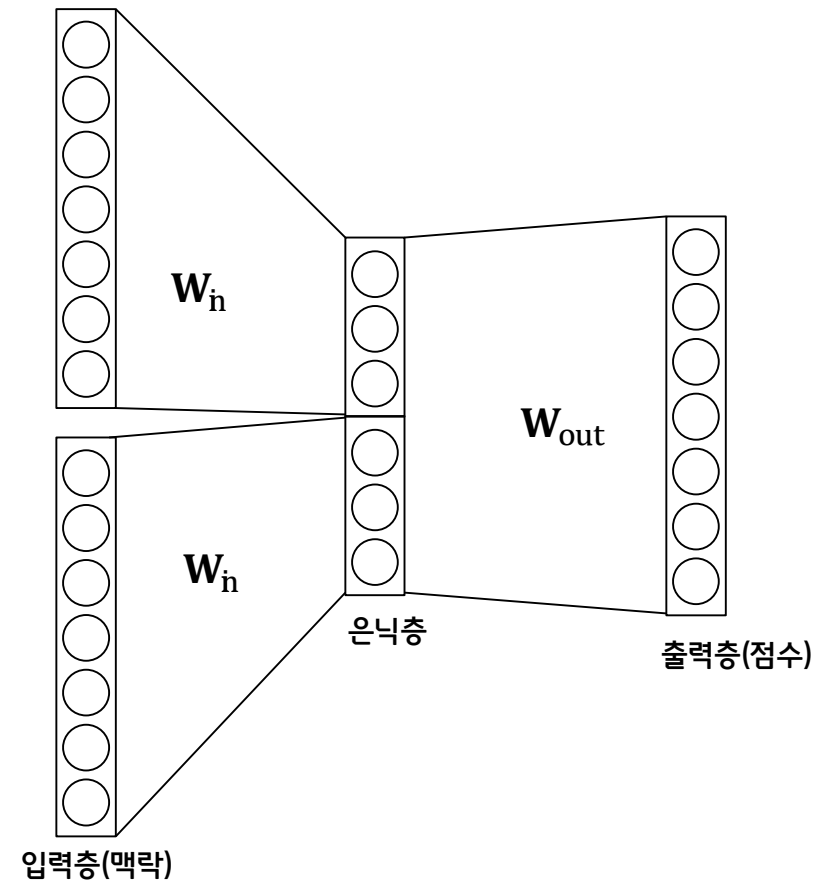
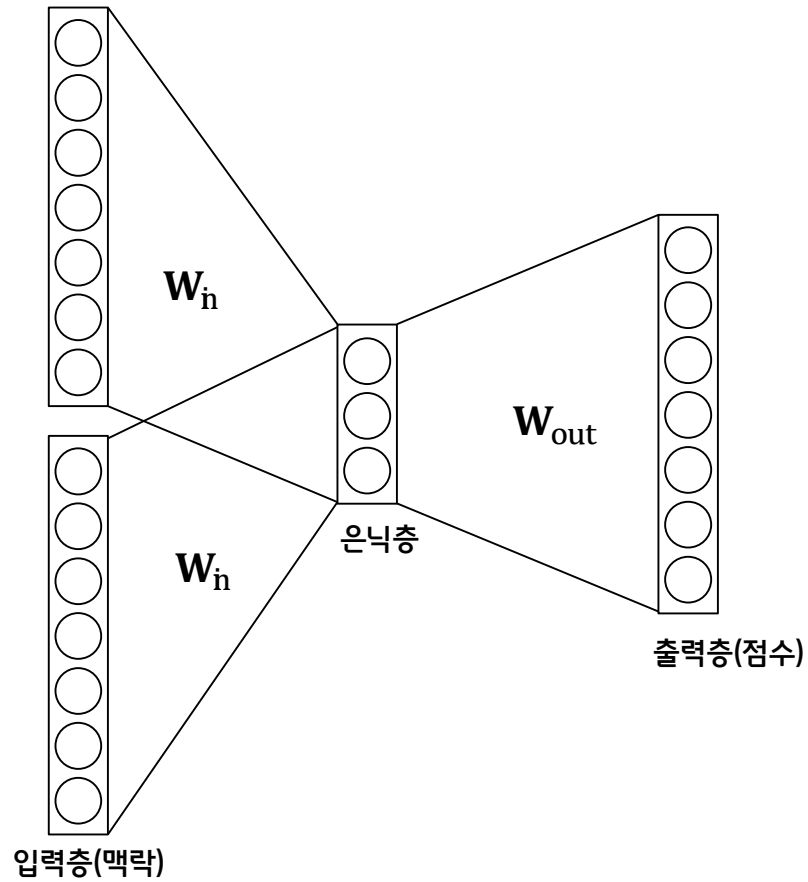
$$P(A) = P(\underbrace{w_1, \dots, w_{m-2}, w_{m-1}}_{A'}, w_m) = P(A', w_{m-1}) = P(w_{m-1}|A'')P(A')$$



- word2vec의 CBOW모델을 언어 모델에 적용하려면 맥락의 크기를 특정 값으로 한정하여 근사적으로 나타낼 수 있다.
- 맥락의 크기는 임의 길이로 설정할 수 있지만 결국 특정 길이로 '고정'된다.
 - 예를 들어 왼쪽 10개의 단어를 맥락으로 CBOW 모델을 만든다고 하면 그 맥락보다 더 왼쪽에 있는 단어의 정보는 무시된다.
- CBOW모델의 맥락 크기를 키울 수는 있으나 맥락 안의 단어 순서가 무시된다는 한계가 있다.
- 맥락의 단어 순서를 고려하기 위해 맥락의 단어 벡터를 은닉층에서 연결(concatenate)하는 방식을 생각할 수 있으나 맥락의 크기에 비례해 가중치 매개변수가 늘어난다는 문제가 발생한다.

그래서 순환 신경망 즉 RNN이 등장하게 되었는데 RNN은 맥락이 아무리 길더라도

맥락의 정보를 기억하는 메커니즘을 갖추고 있기에 아무리 긴 시계열 데이터에도 대응할 수 있다.



5. 순환 신경망(RNN)

5.1 확률과 언어 모델

5.2 RNN 이란

5.3 RNN 구현

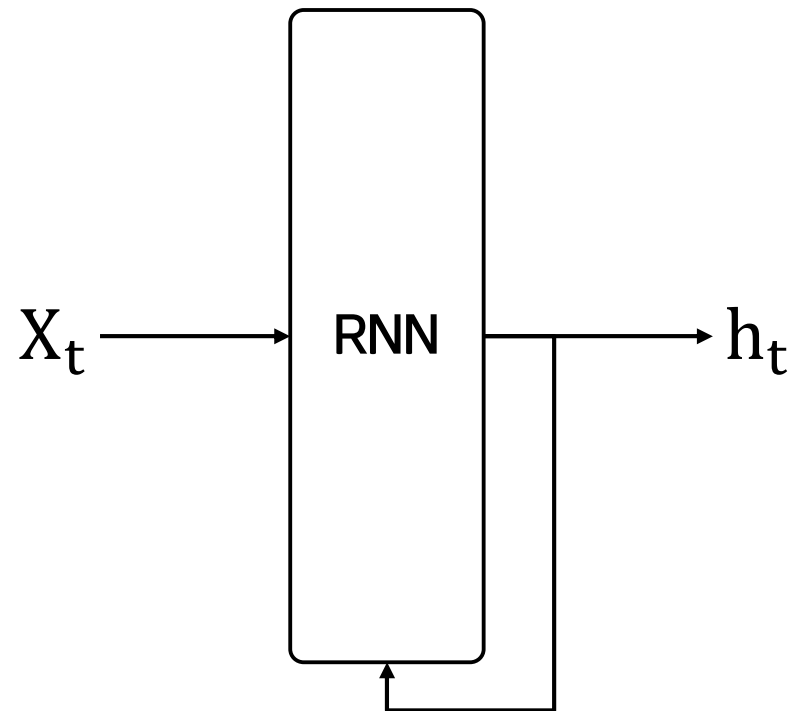
5.4 시계열 데이터 처리 계층 구현

5.5 RNNLM 학습과 평가

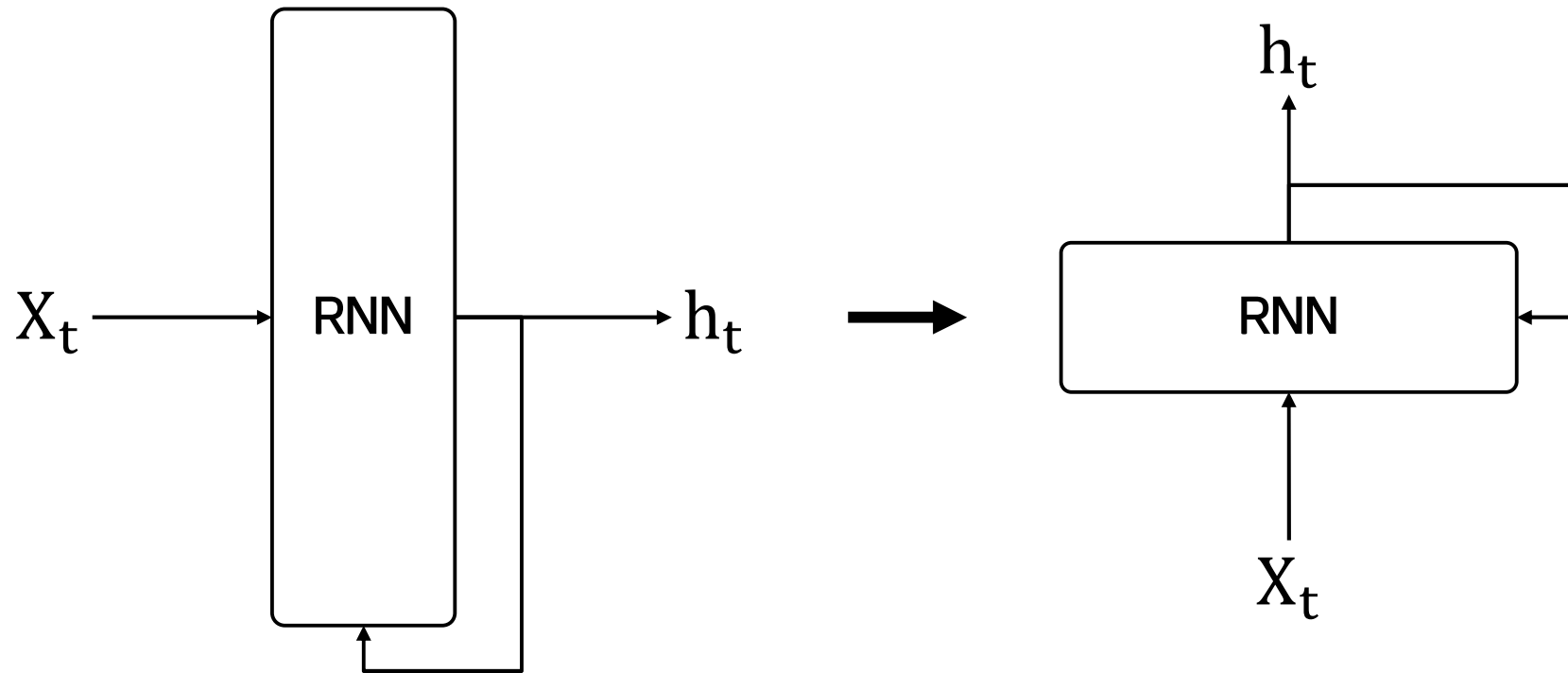
순환하기 위해서는 닫힌 경로가 필요하다.

닫힌 경로 혹은 순환하는 경로가 존재해야 데이터가 같은 장소를 반복해 왕래할 수 있고 데이터가 순환하면서 과거의 정보를 기억하는 동시에 최신 데이터로 갱신 될 수 있다.

순환 경로를 포함하는 RNN 계층



계층을 90도 회전시켜 그린다.



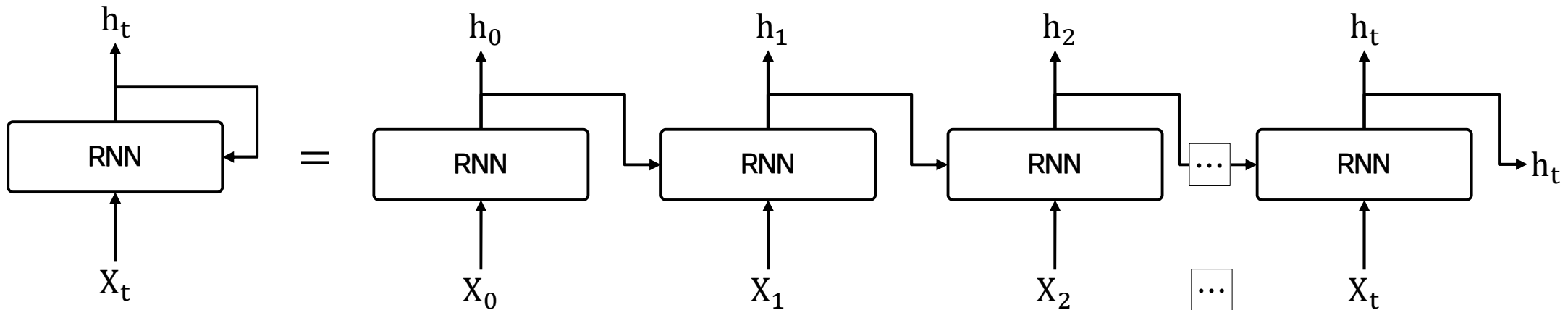
t: 시각

시계열 데이터($x_0, x_1, \dots, x_t, \dots$)가 RNN계층에 입력되고 이에 대응해 ($h_0, h_1, \dots, h_t, \dots$)가 출력된다.

각 시각에 입력되는 x_t 를 벡터라고 가정했을 때

문장(단어 순서)을 다루는 경우를 예로 든다면 각 단어의 분산 표현(단어 벡터)이 x_t 가 되며 이 분산 표현이 순서대로 하나씩 RNN계층에 입력된다.

RNN 계층의 순환 구조 펼치기



RNN계층의 순환 구조를 펼침으로써 오른쪽으로 성장하는 긴 신경망으로 변신
 피드포워드 신경망(데이터가 한 방향으로만 흐른다)과 같은 구조이지만 위 그림에서는
 다수의 RNN계층 모두가 실제로는 '같은 계층'인 것이 지금까지의 신경망과는 다른 점이다.

각 시각의 RNN계층은 그 계층으로의 입력과 1개 전의 RNN계층으로부터의 출력을 받는데
 이 두 정보를 바탕으로 현 시각의 출력을 계산한다.

$$h_t = \tanh(h_{t-1}W_h + x_tW_x + b)$$

W_x : 입력 x 를 출력 h 로 변환하기 위한 가중치

W_h : 1개의 RNN출력을 다음 시각의 출력으로 변환하기 위한 가중치

b : 편향

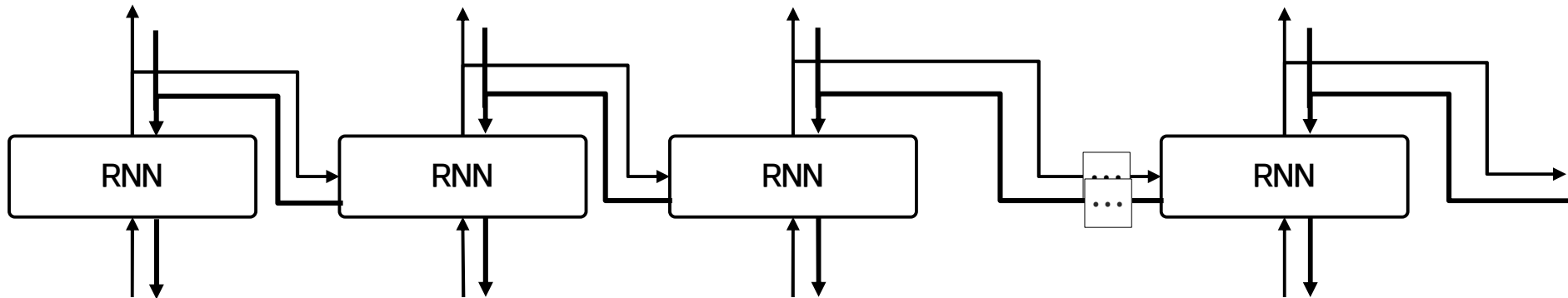
$h(t-1)$, x_t : 행벡터

h_t 는 다른 계층을 향해 위쪽으로 출력되는 동시에 다음 시각의 RNN계층(자기 자신)을 향해 오른쪽으로도 출력된다.

RNN의 출력 h_t 는 은닉상태(hidden state) 혹은 은닉 상태 벡터(hidden state vector)라고 한다.

RNN은 h 라는 '상태'를 가지고 있으며 위의 식의 형태로 갱신된다고 해석할 수 있다.
RNN계층을 '상태를 가지는 계층' 혹은 '메모리(기억력)가 있는 계층'이라고 한다.

순환 구조를 펼친 RNN 계층에서의 오차역전파



순환 구조를 펼친 후의 RNN에는 (일반적인) 오차역전파법을 적용할 수 있다.
먼저 순전파를 수행하고 이어서 역전파를 수행하여 원하는 기울기를 구할 수 있다.
여기서의 오차역전파법은 '시간 방향으로 펼친 신경망의 오차역전파법'이란 뜻으로
BPTT(Backpropagation Through Time)라고 한다.

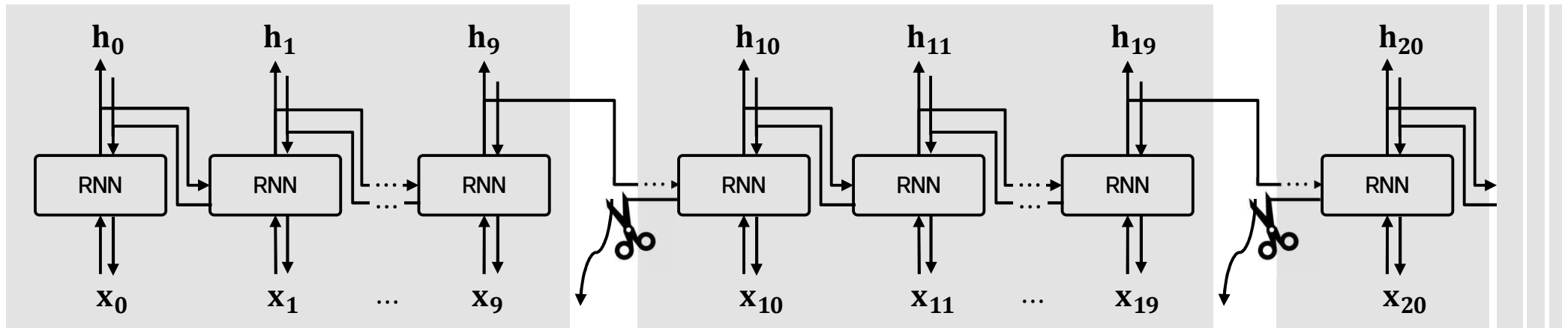
문제점

- 시계열 데이터의 시간 크기가 커지는 것에 비례하여 BPTT가 소비하는 컴퓨팅 자원도 증가
- 시간 크기가 커지면 역전파 시의 기울기가 불안정해짐

Truncated BPTT : 시간축 방향으로 너무 길어진 신경망을 적당한 지점에서 잘라내어 작은 신경망 여러 개로 만들어 잘라낸 작은 신경망에서 오차역전파법을 수행한다.

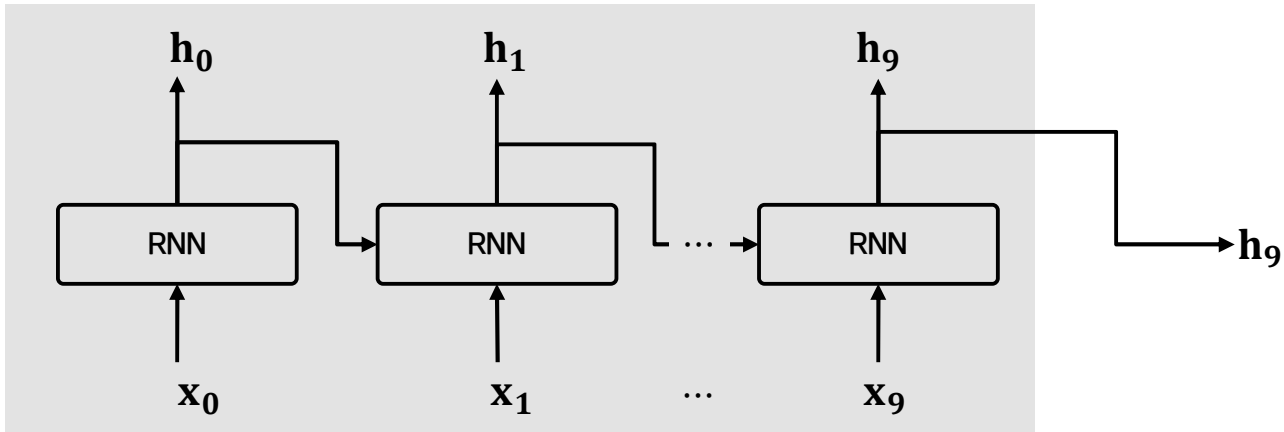
- 계층이 너무 길면 계산량과 메모리 사용량 등이 문제가 되고 계층이 길어짐에 따라 신경망을 하나 통과할 때마다 기울기 값이 조금씩 작아져서 이전 시각 t 까지 역전파되기 전에 0이 되어 소멸할 수도 있다.
- 순전파의 연결을 그대로 유지하면서(데이터를 순서대로 입력해야 한다) 역전파의 연결은 적당한 길이로 잘라내 잘라낸 신경망 단위로 학습을 수행한다.
- 역전파의 연결을 잘라버리면 그보다 미래의 데이터에 대해서는 생각할 필요가 없어지기 때문에 각각의 블록 단위로 미래의 블록과는 독립적으로 오차역전파법을 완결시킨다.
 - 블록: 역전파가 연결되는 일련의 RNN계층
- 순전파를 수행하고 그 다음 역전파를 수행하여 원하는 기울기를 구한다.
- 다음 역전파를 수행할 때 앞 블록의 마지막 은닉 상태인 h_t 가 필요하다.
 h_t 로 순전파가 계속 연결될 수 있다.

역전파의 연결을 적당한 지점에서 끊는다.

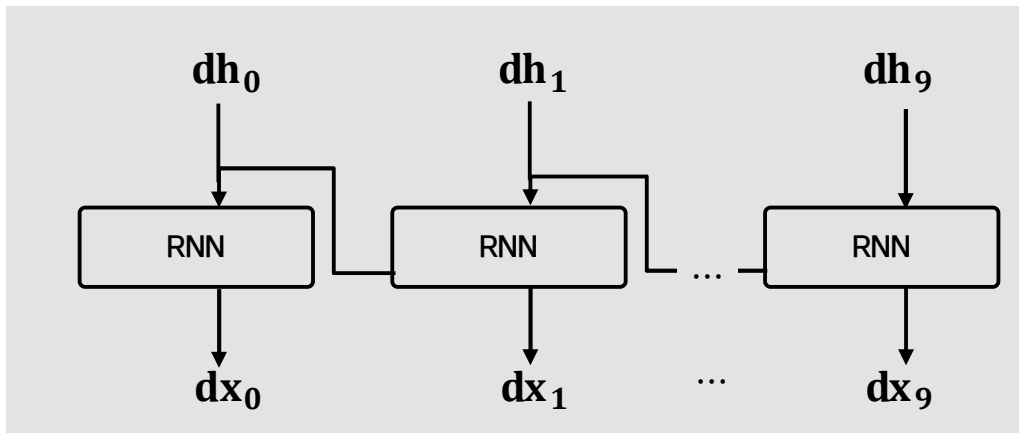


첫 번째 블록의 순전파와 순전파 : 이보다 앞선 시각으로부터의 기울기는 끊겼기 때문에 이 블록 내에서만 오차역전파법이 완결된다.

순전파

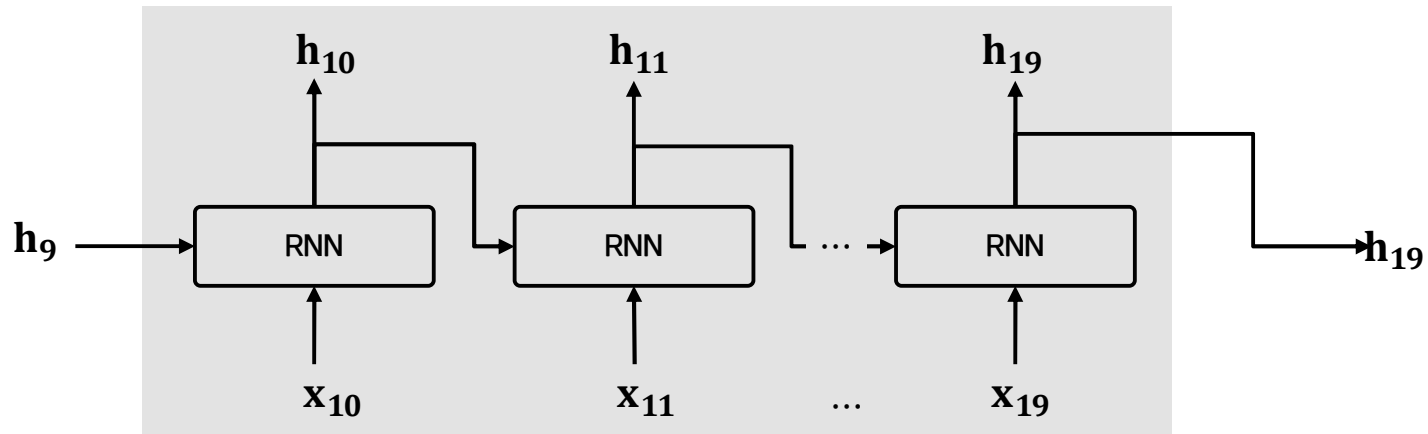


역전파

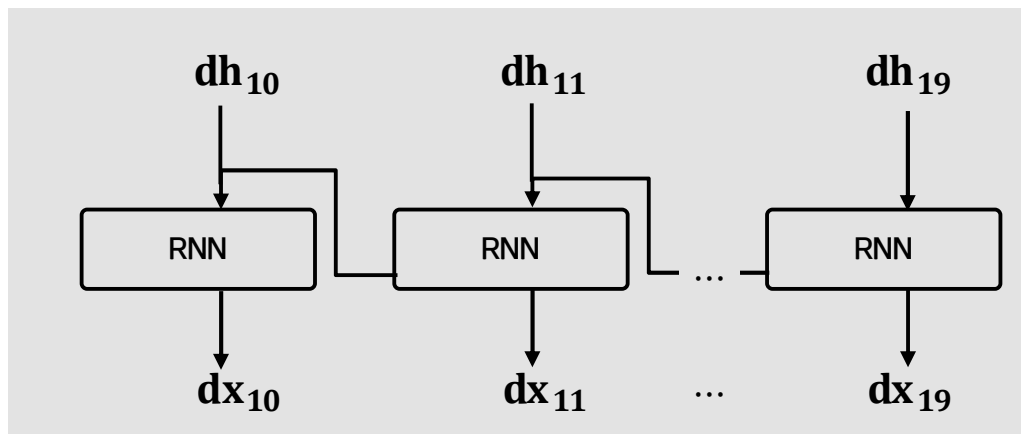


두 번째 블록의 순전파와 순전파

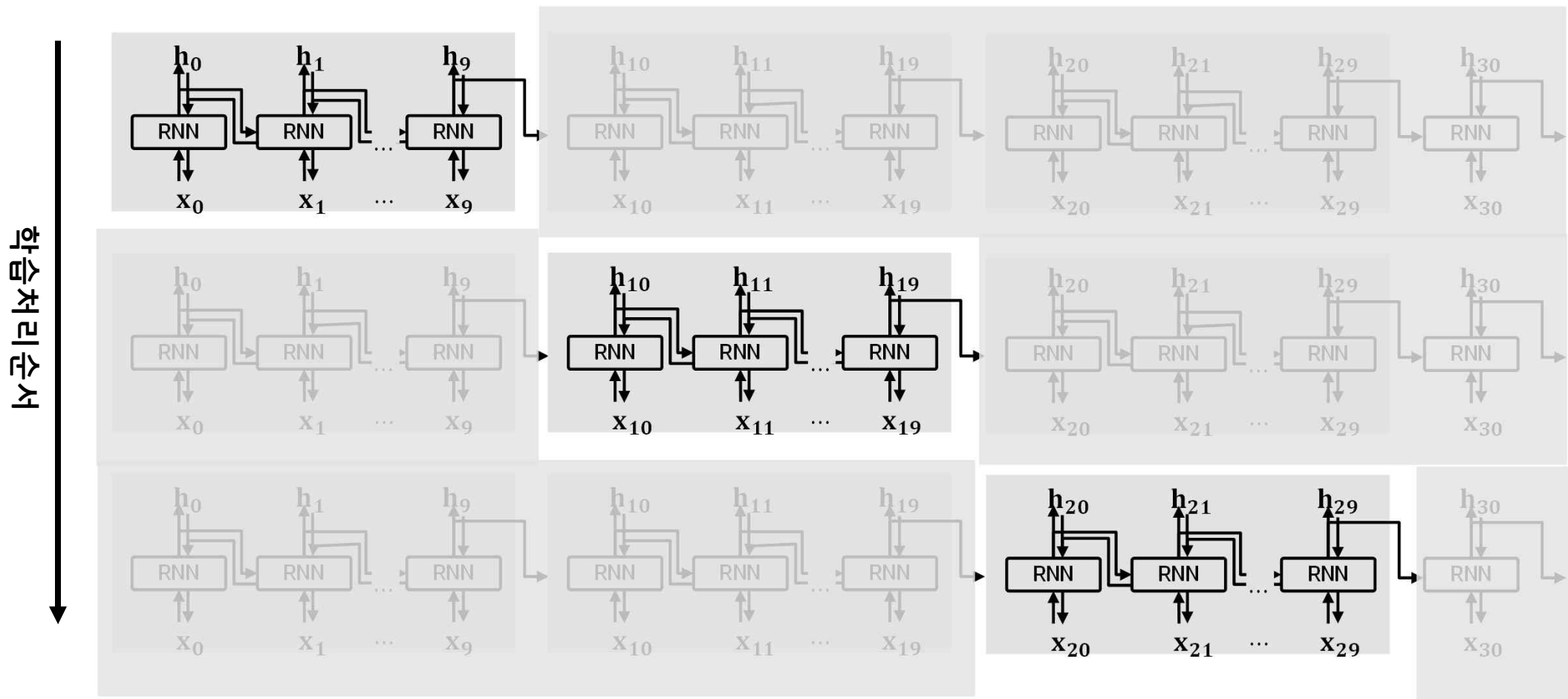
순전파



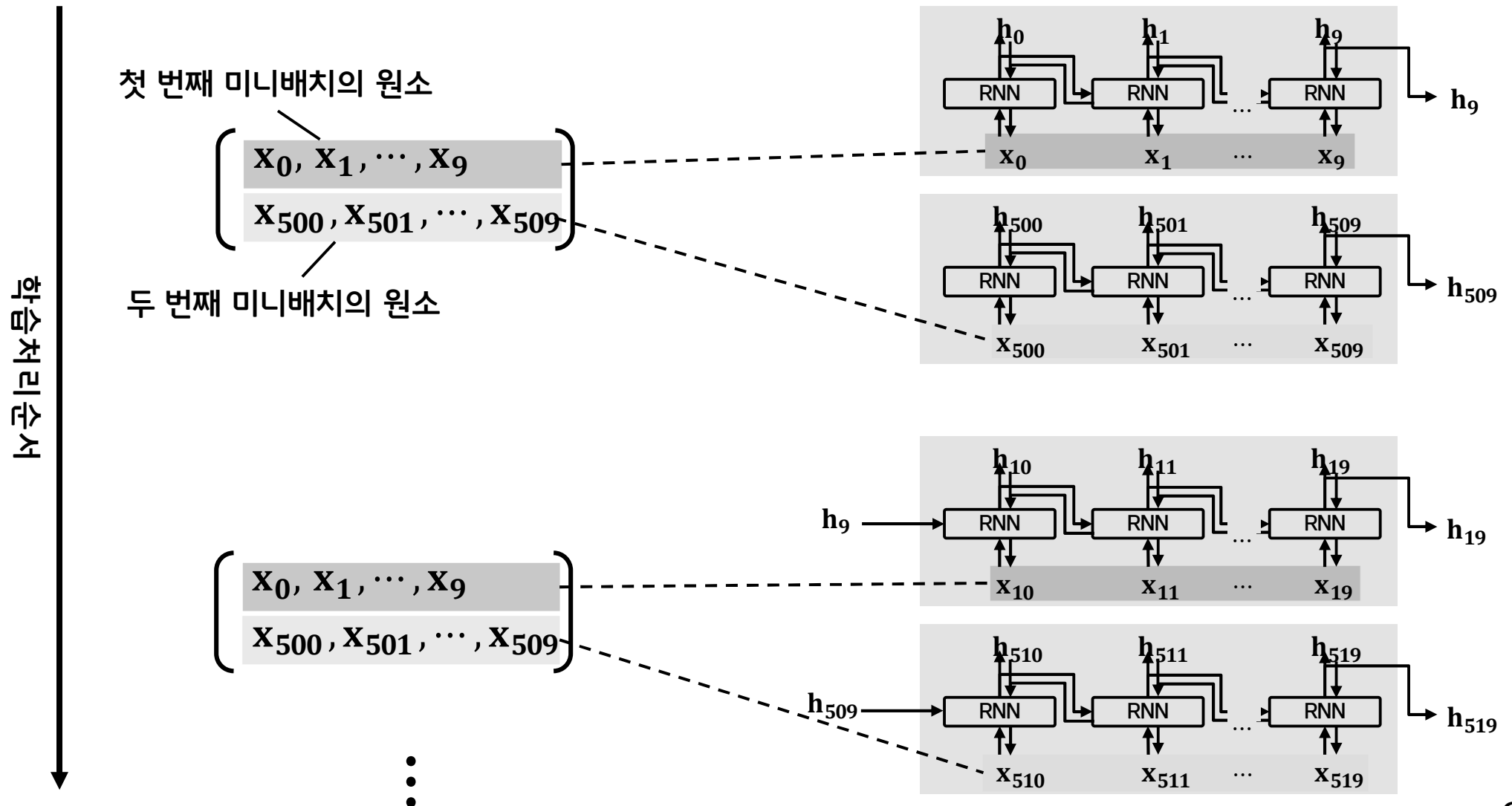
역전파



Truncated BPTT의 데이터 처리 순서



미니배치 학습 시 데이터를 제공하는 시작 위치를 각 미니배치로 옮긴다.



5. 순환 신경망(RNN)

5.1 확률과 언어 모델

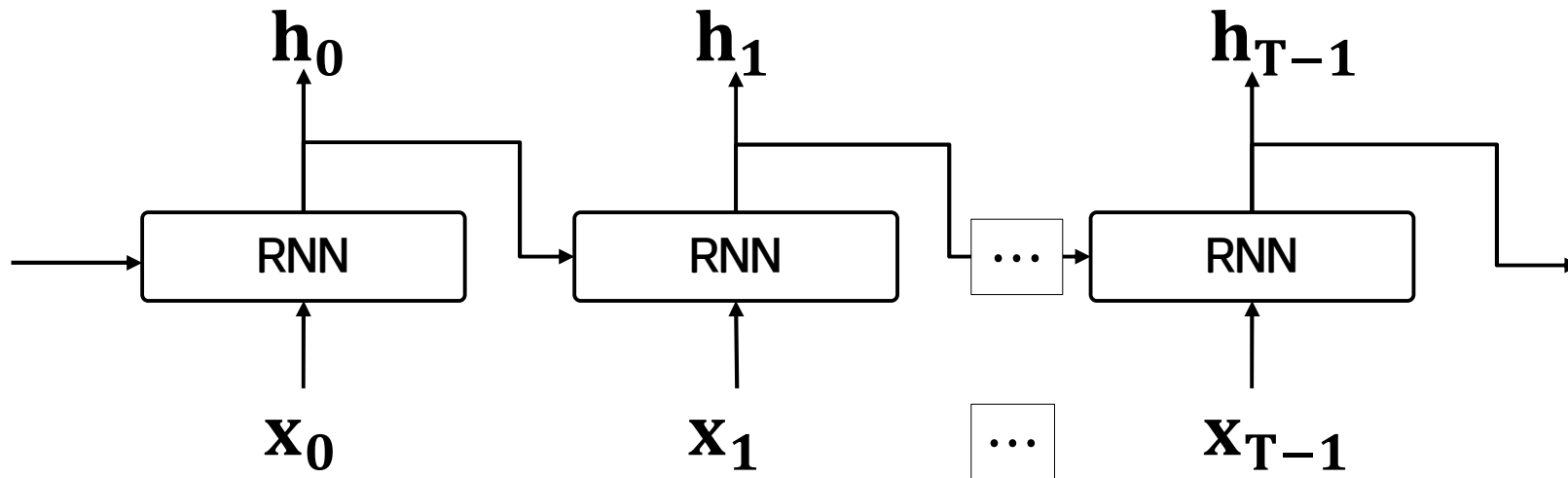
5.2 RNN 이란

5.3 RNN 구현

5.4 시계열 데이터 처리 계층 구현

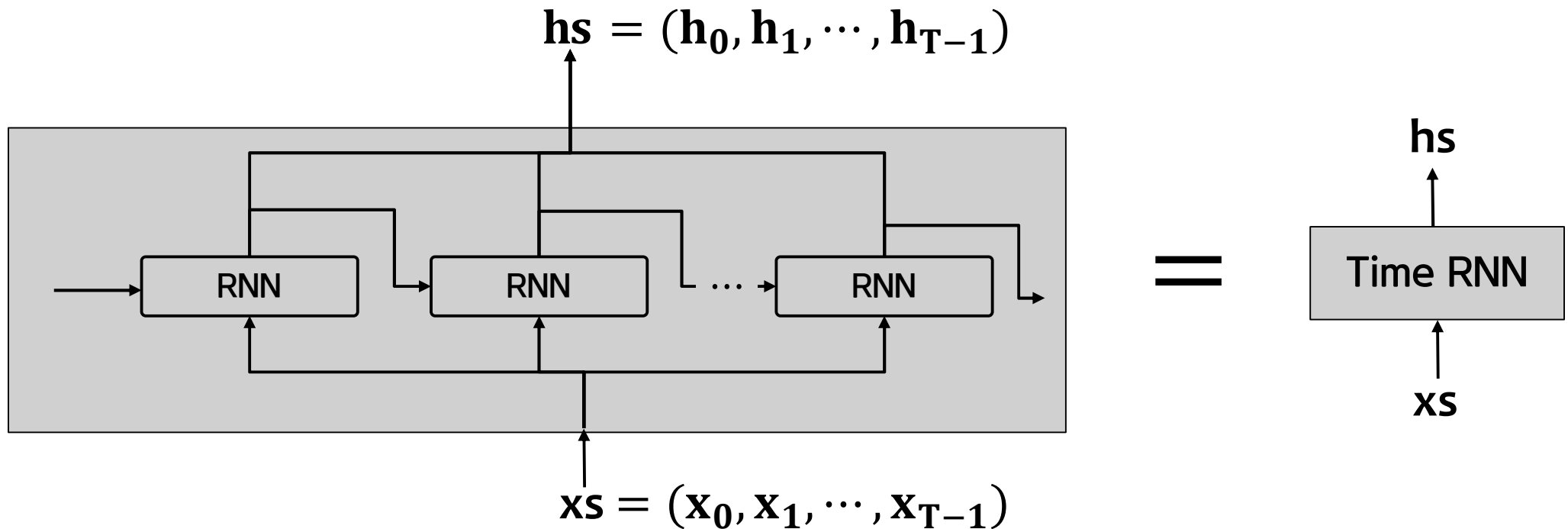
5.5 RNNLM 학습과 평가

RNN에서 다루는 신경망



길이가 T 인 시계열 데이터를 받는다.
각 시각의 은닉 상태를 T 개 출력한다.
모듈화를 생각해 위의 그림의 신경망을 '하나의 계층'으로 구현한다.

Time RNN 계층 : 순환 구조를 펼친 후의 계층들을 하나의 계층으로 간주한다.

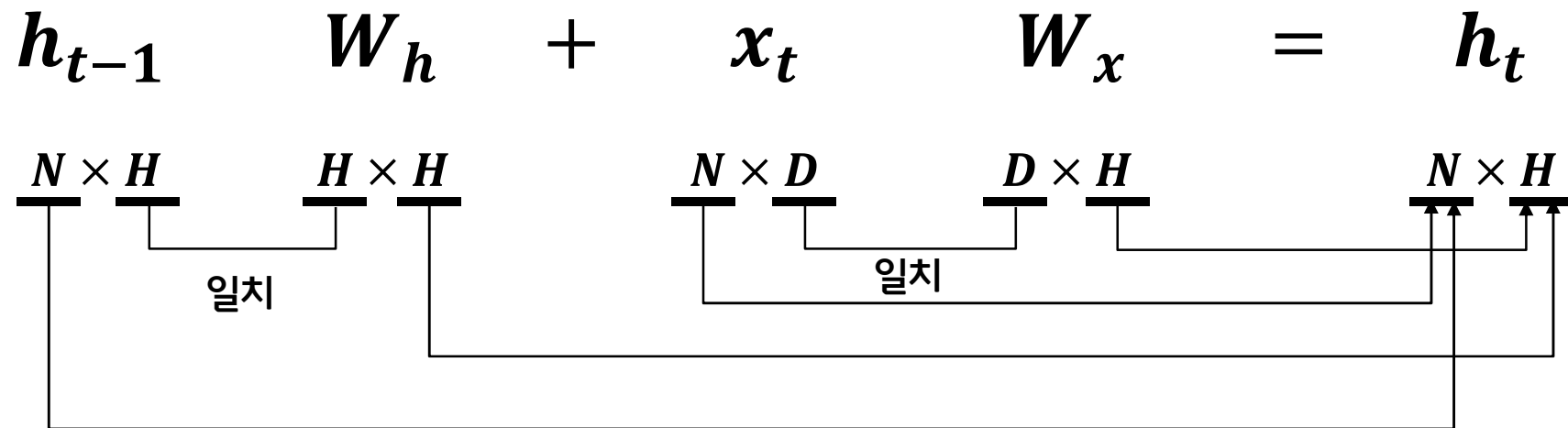


x_s 를 입력하면 h_s 를 출력하는 단일 계층

Time RNN계층 내에서 한 단계의 작업을 수행하는 계층을 'RNN계층'이라 하고
T개 단계분의 작업을 한꺼번에 처리하는 계층을 'Time RNN계층'이라 한다.

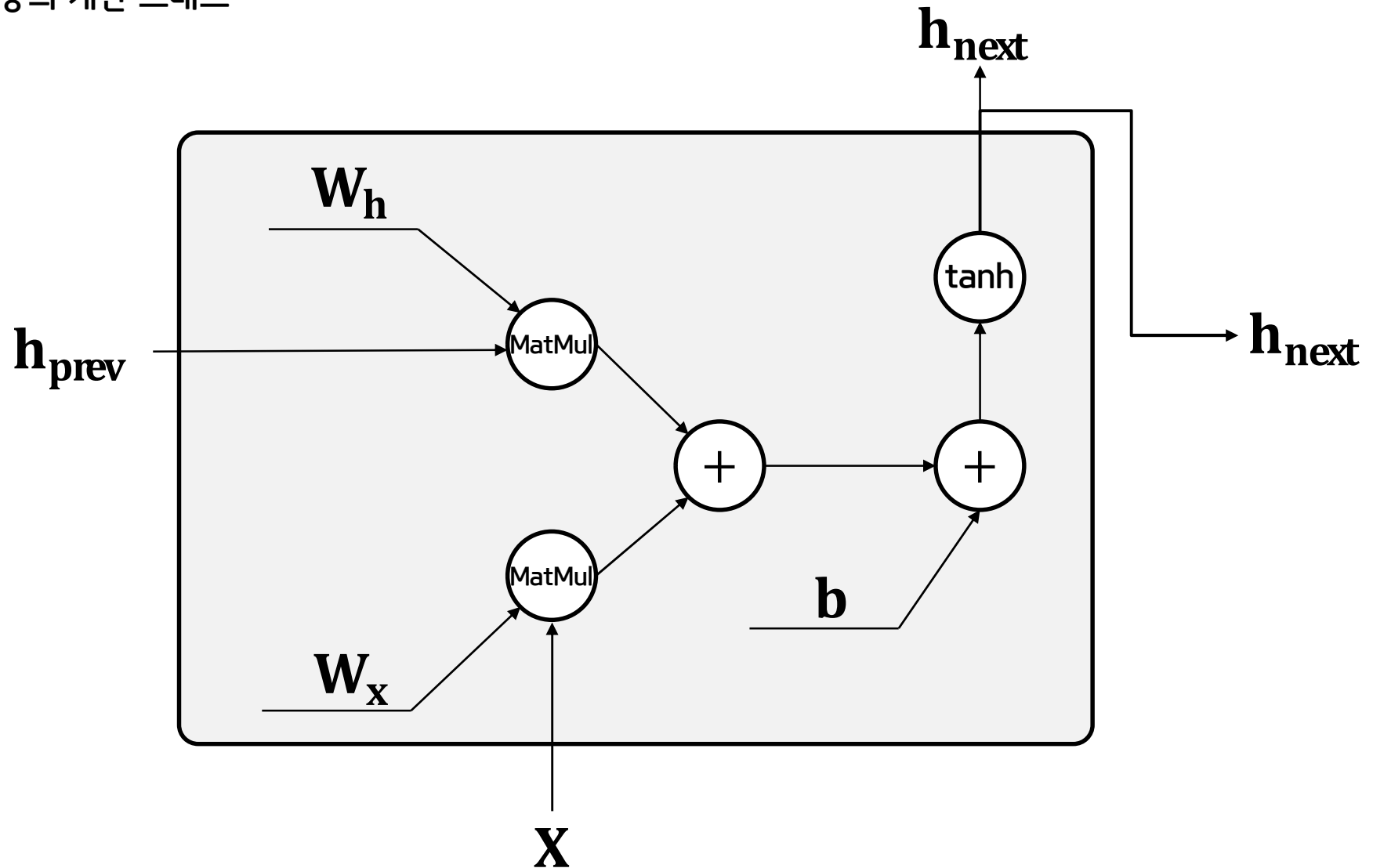
$$h_t = \tanh(h_{t-1}W_h + x_tW_x + b)$$

형상 확인 : 형렬 곱에서는 대응하는 차원의 원소 수를 일치시킨다.

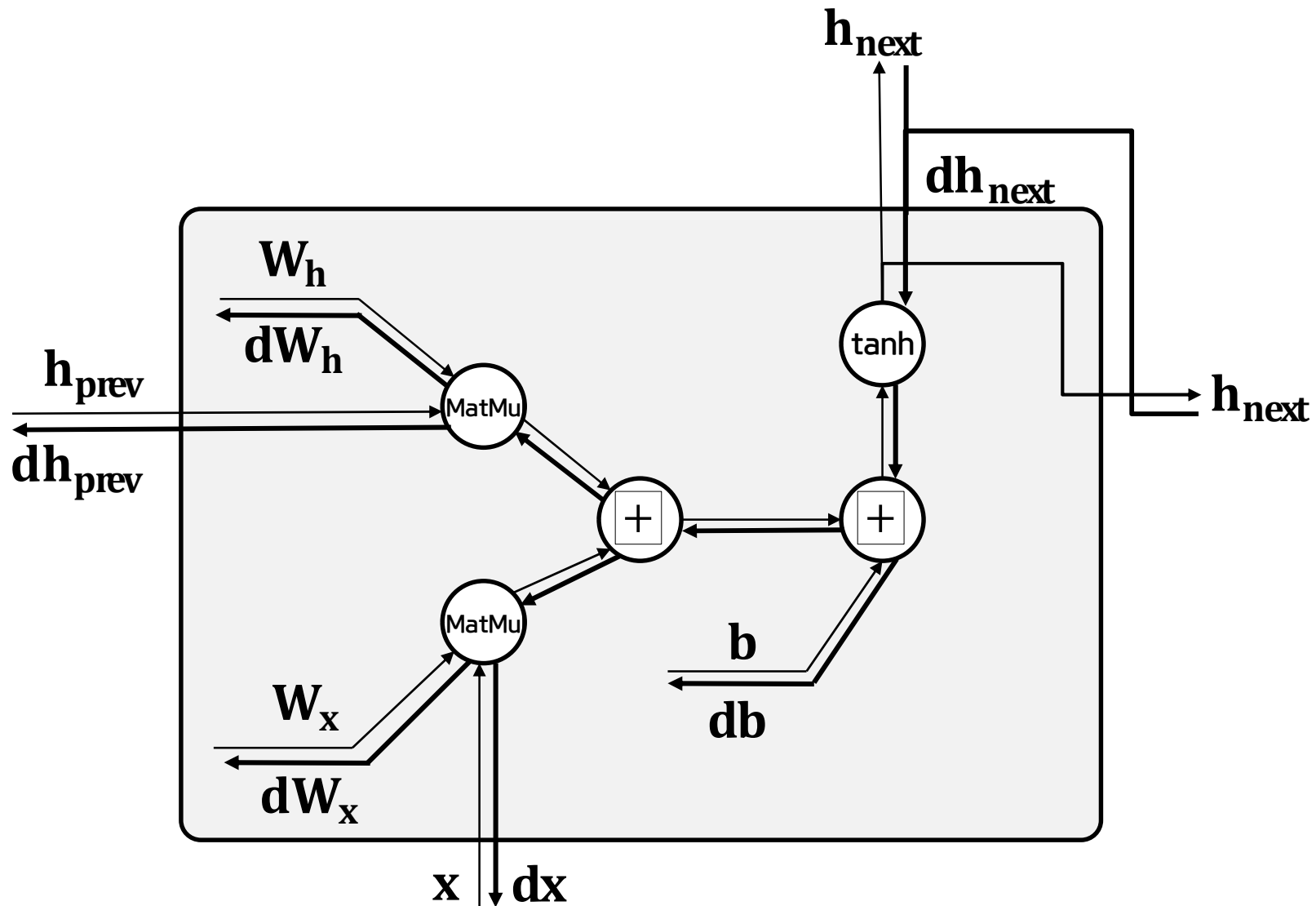


N: 미니배치 크기 D: 입력 벡터의 차원 수 H: 은닉 상태 벡터의 차원 수

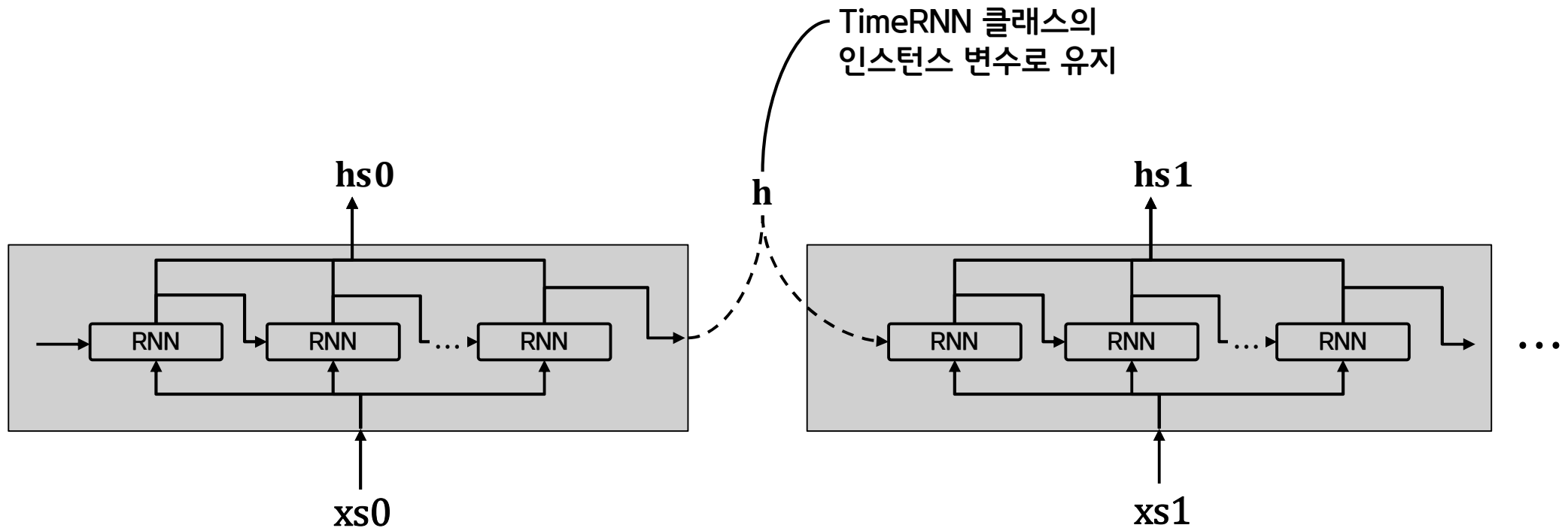
RNN 계층의 계산 그래프



RNN 계층의 계산 그래프(역전파 포함)

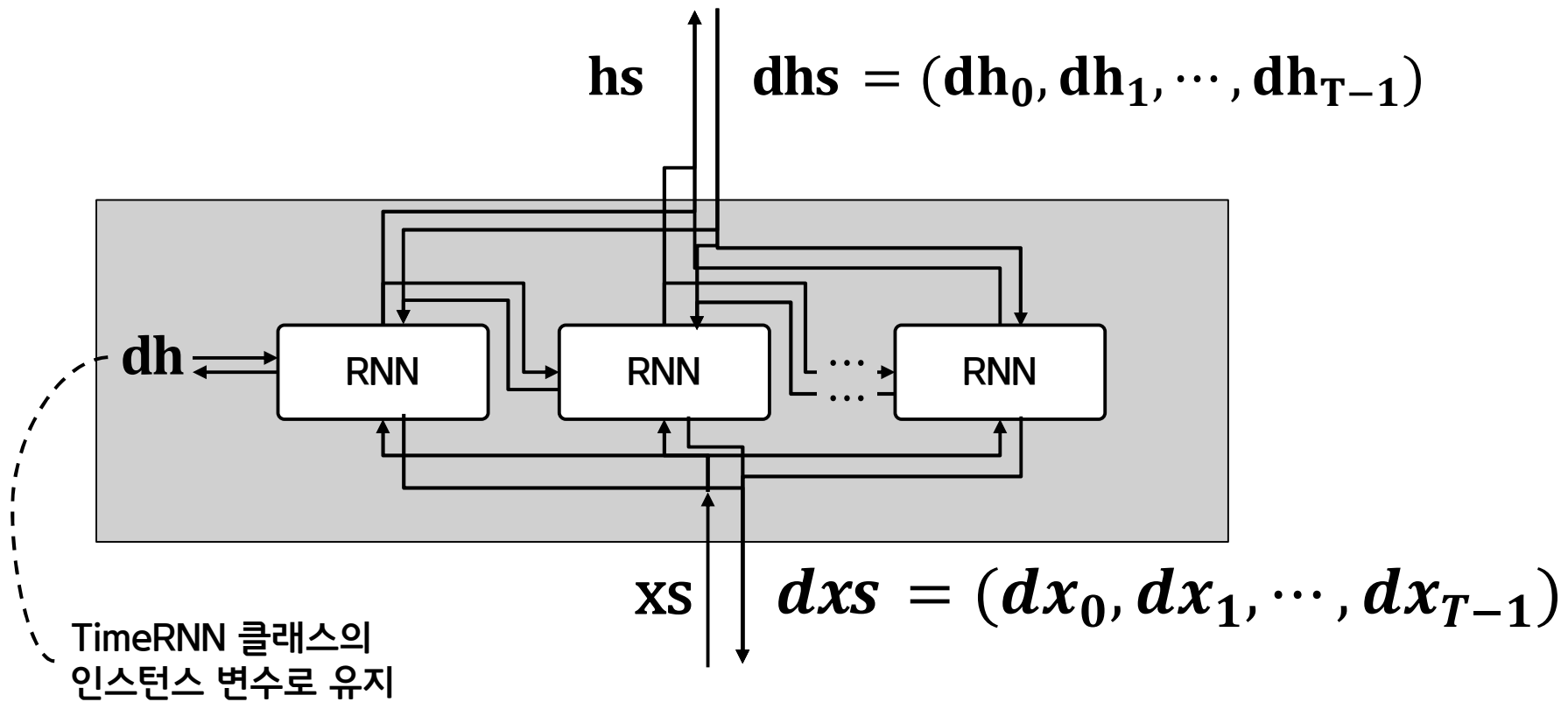


Time RNN 계층은 은닉 상태를 인스턴스 변수 h 로 보관한다. 그러면 은닉 상태를 다음 블록에 인계할 수 있다.

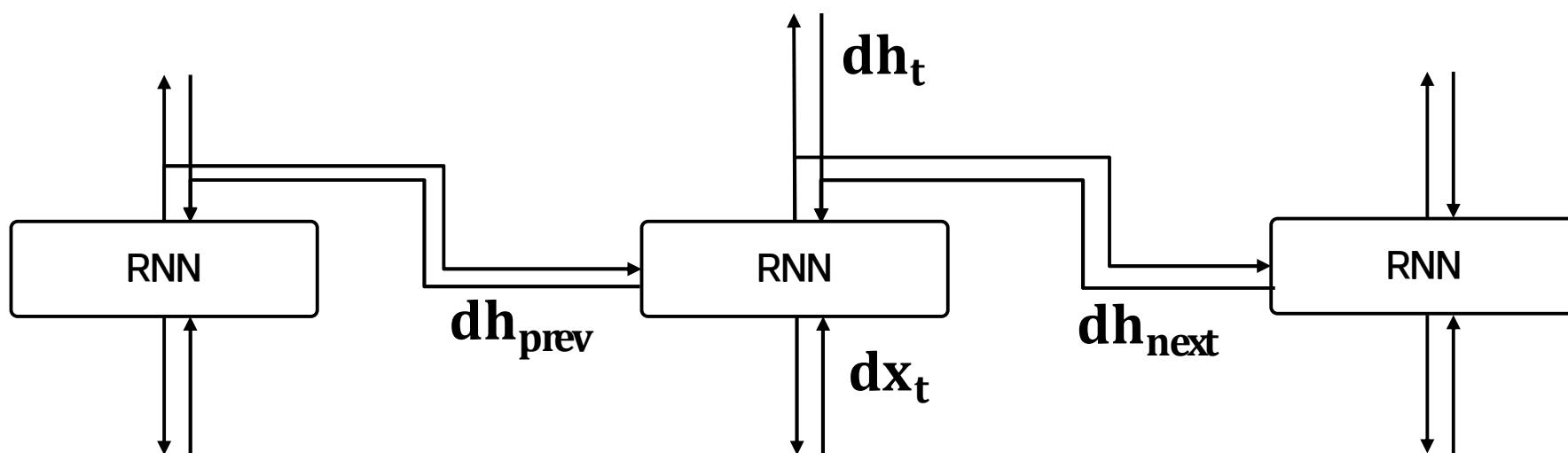


RNN계층의 은닉 상태를 Time RNN계층에서 관리하면 Time RNN사용자는 RNN계층 사이에서 은닉 상태를 '인계하는 작업'을 생각하지 않아도 된다.

Time RNN 계층의 역전파



t번째 RNN 계층의 역전파



5. 순환 신경망(RNN)

5.1 확률과 언어 모델

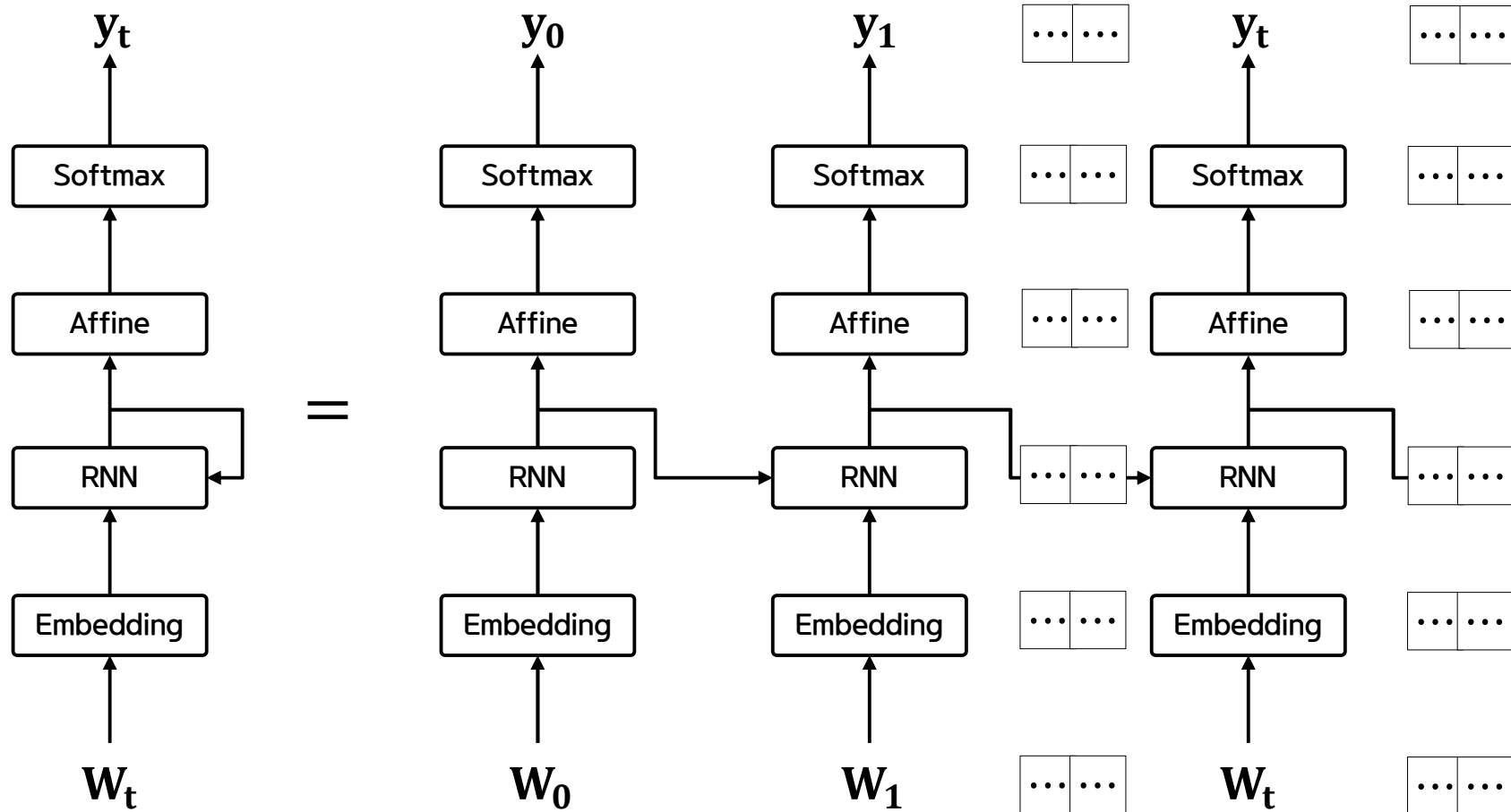
5.2 RNN 이란

5.3 RNN 구현

5.4 시계열 데이터 처리 계층 구현

5.5 RNNLM 학습과 평가

RNNLM의 신경망



RNNLM의 신경망

Embedding: 단어 ID를 단어의 분산 표현(단어 벡터)으로 변환

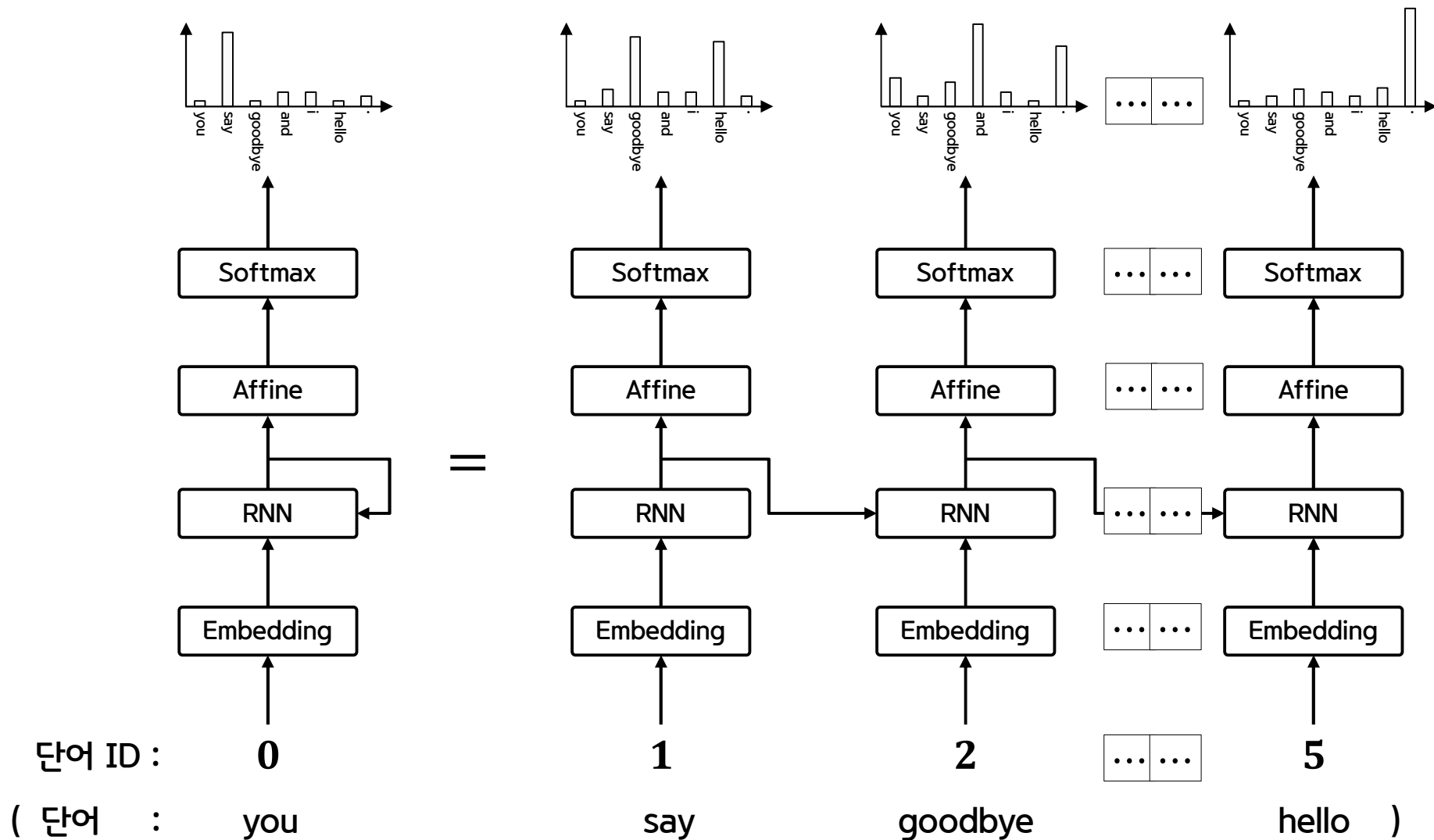
RNN계층: 은닉 상태를 다음 층으로(위쪽으로) 출력함과 동시에 다음 시각의 RNN 계층으로(오른쪽으로) 출력한다.

RNN계층이 위로 출력한 은닉 상태는 Affine 계층을 거쳐 Softmax 계층으로 전해진다.

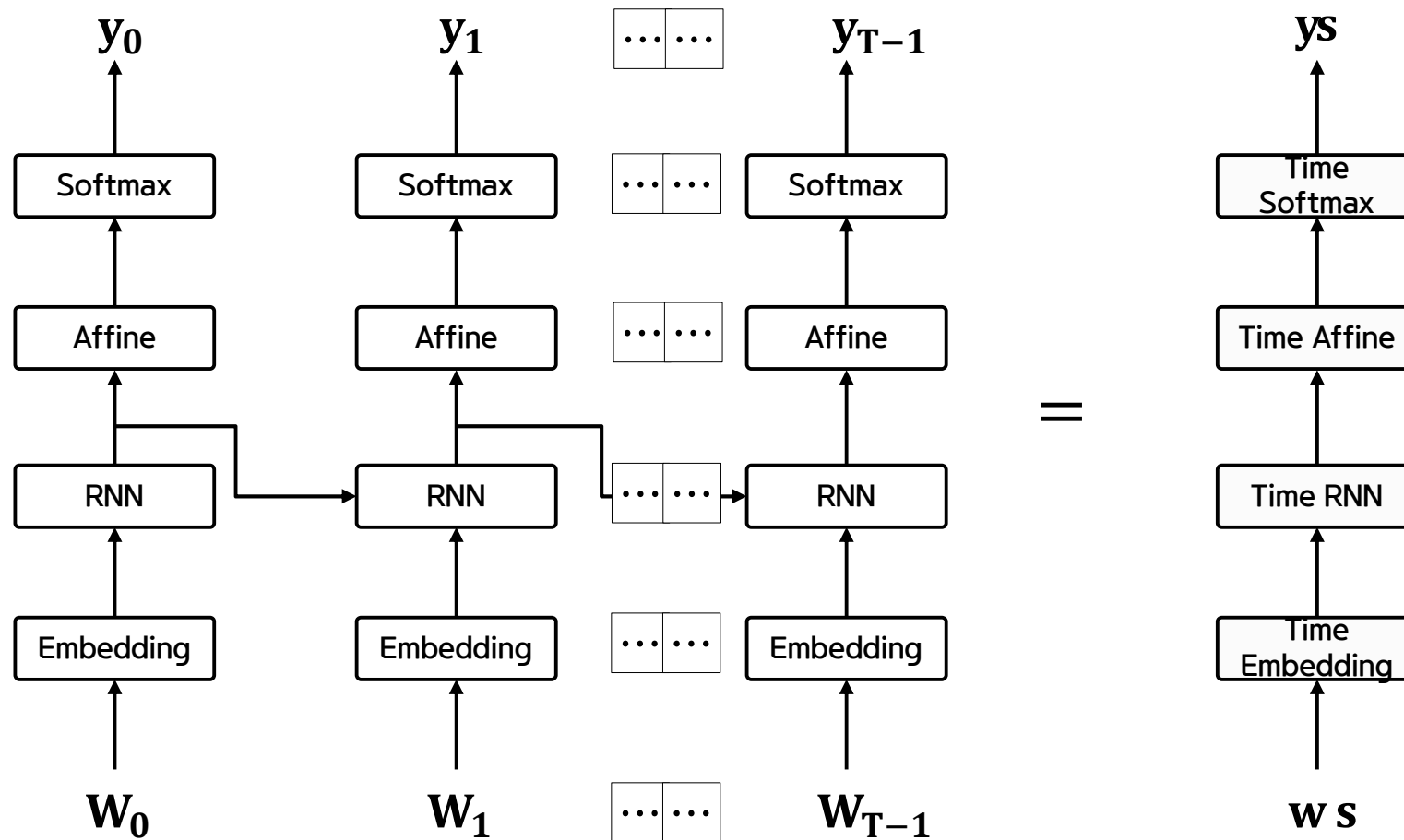
RNNLM은 지금까지 입력된 단어를 '기억'하고 그것을 바탕으로 다음에 출현할 단어를 예측한다.

RNN계층이 과거에서 현재로 데이터를 계속 흘려 보내 줌으로써 과거의 정보를 인코딩해 저장(기억) 할 수 있다.

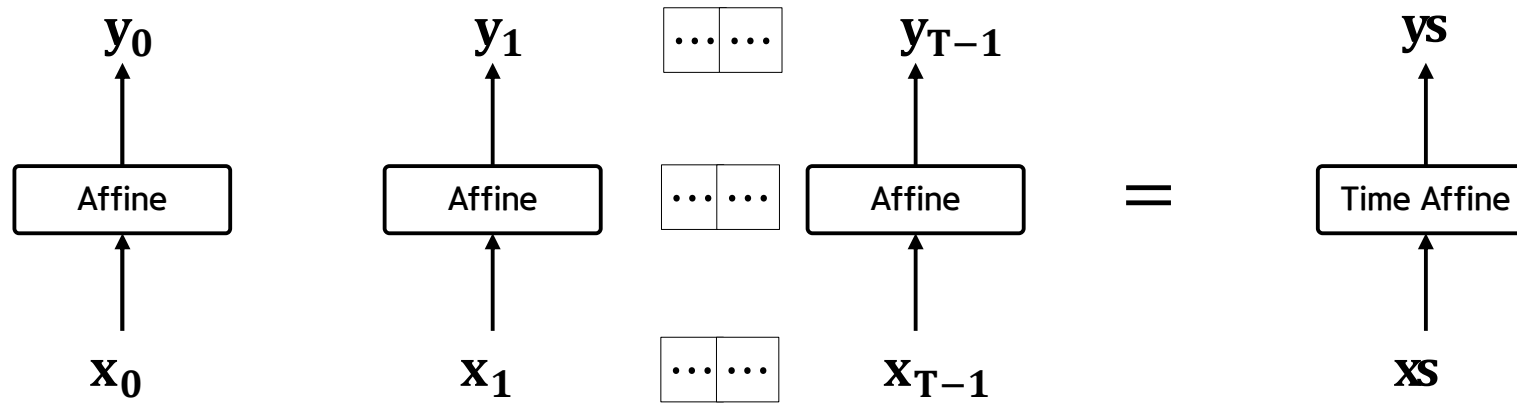
샘플 말뭉치로 "you say goodbye and i say hello."를 처리하는 RNNLM의 예



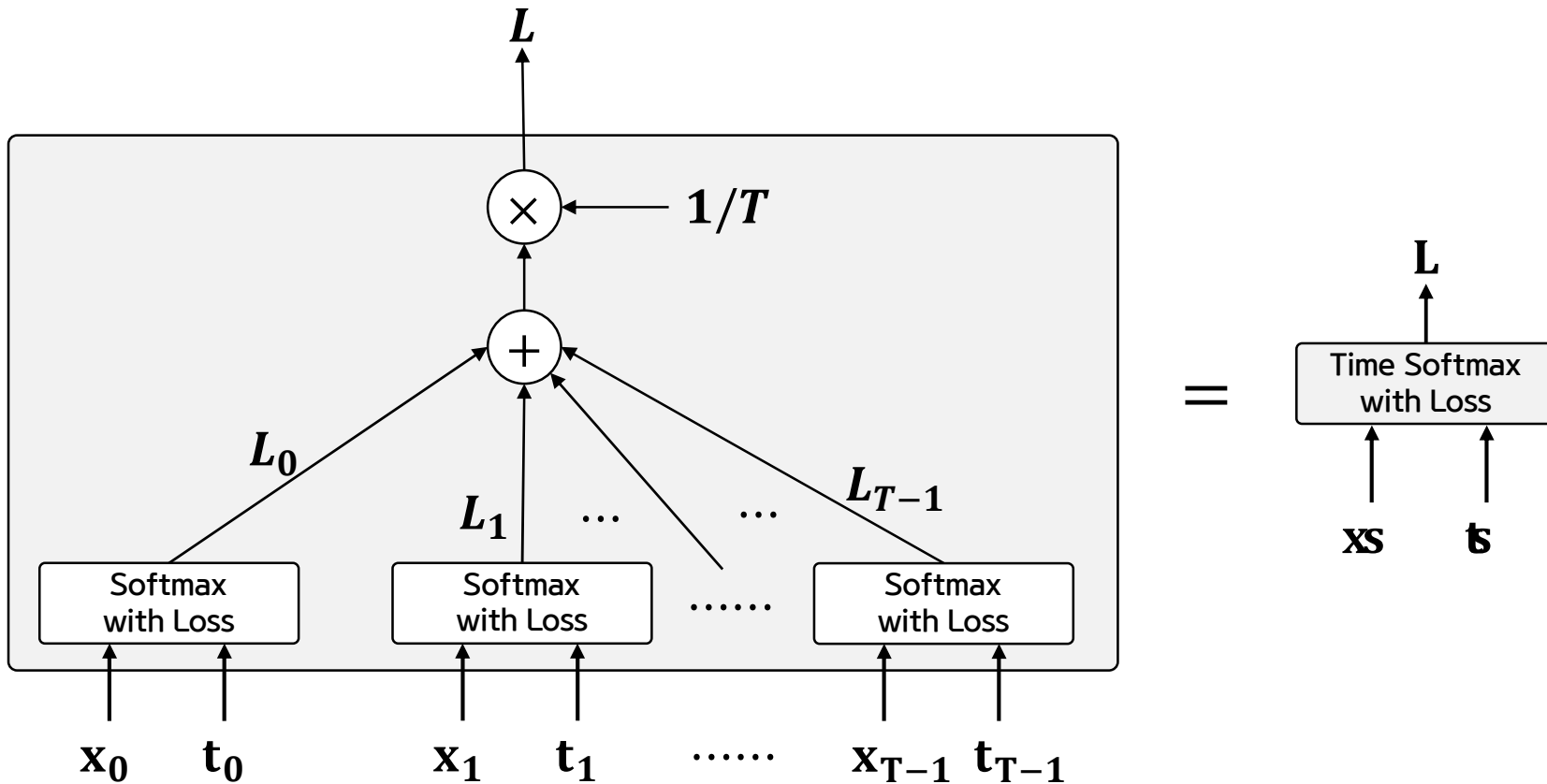
시계열 데이터를 한꺼번에 처리하는 계층을 Time XX 계층으로 구현



Time Affine 계층은 T개의 Affine 계층의 집합으로 구현



Time Softmax with Loss 계층의 전체 그림



x_0, x_1, \dots : 아래층에서부터 전해지는 점수(확률로 정규화 되기 전의 값)

t_0, t_1, \dots : 정답 레이블

T개의 Softmax with Loss 계층 각각이 손실을 산출하고 그 손실들을 합산해 평균한 값이 최종 손실이 된다.

$$L = \frac{1}{T} (L_0 + L_1 + \cdots + L_{T-1})$$

Time Softmax with Loss 계층도 시계열에 대한 평균을 구하는 것으로 데이터 1개당 평균 손실을 구해 최종 출력으로 내보낸다.

5. 순환 신경망(RNN)

5.1 확률과 언어 모델

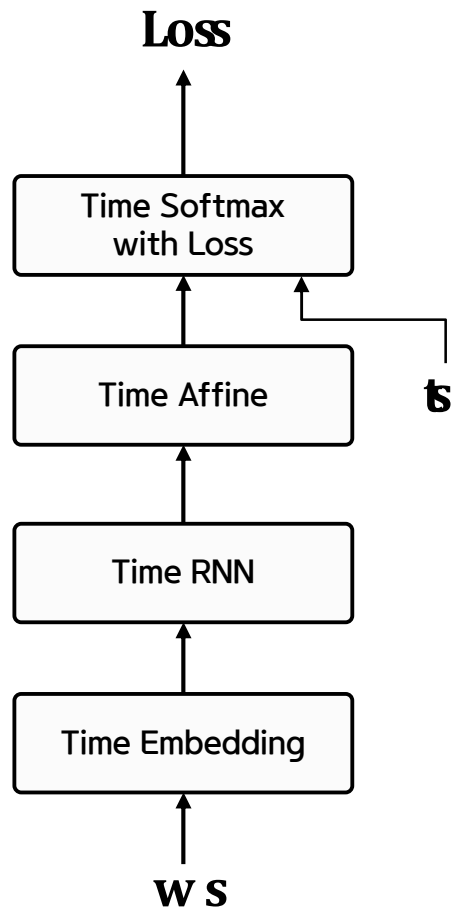
5.2 RNN 이란

5.3 RNN 구현

5.4 시계열 데이터 처리 계층 구현

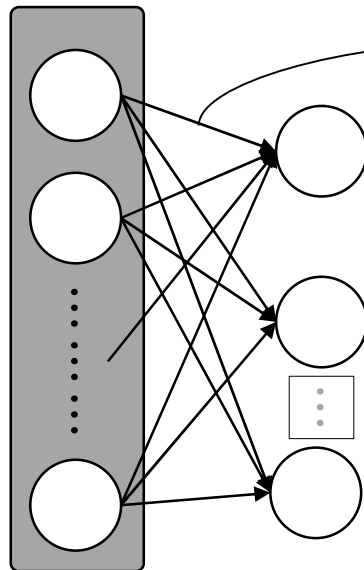
5.5 RNNLM 학습과 평가

SimpleRnnlm의 계층 구성 : RNN 계층의 상태는 클래스 내부에서 관리



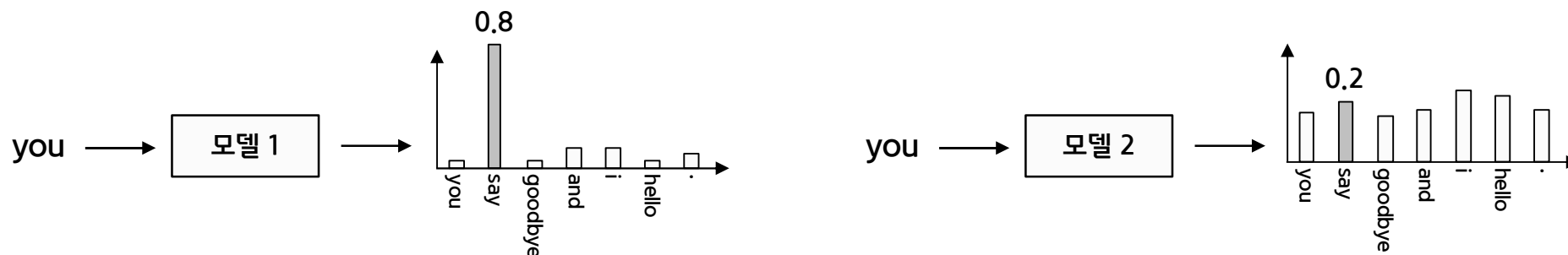
Xavier 초기값 : 이전 계층의 노드가 n 개라면 표준편차가 $\frac{1}{\sqrt{n}}$ 인 분포를 초기값으로 사용

n개의 노드



표준편차가 $\frac{1}{\sqrt{n}}$ 인 분포로 초기화

단어 "you"를 입력하여 다음에 출현할 단어의 확률분포를 출력하는 모델의 예



언어 모델은 주어진 과거 단어(정보)로부터 다음에 출현할 단어의 확률분포를 출력한다.
이때 언어 모델의 예측 성능을 평가하는 척도로 혼란도(perplexity)를 자주 이용한다.

혼란도(perplexity) : 간단히 말하면 '확률의 역수'이다.(데이터 수가 하나일 때에 정확히 일치한다.)
작을수록 좋은 값이다.

분기수(number of branches): 다음에 취할 수 있는 선택사항의 수(다음에 출현할 수 있는 단어의 후보 수)

예)

분기수가 1.25 -> 다음에 출현할 수 있는 단어의 후보를 1개 정도로 좁혔다(좋은 모델)

분기수가 5 -> 후보가 아직 5개(나쁜 모델)

입력 데이터가 여러 개일 때

$$L = -\frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk}$$

$$\text{perplexity} = e^L$$

N:데이터의 총 개수

t_n : 원 핫 벡터로 나타낸 정답 레이블

t_{nk} : n번째 데이터의 k번째 값

y_{nk} : 확률분포(신경망에서는 Softmax의 출력)

L: 신경망의 손실. 교차 엔트로피 오차를 뜻하는 식과 같은 식

6. 게이트가 추가된 RNN

6.1 RNN의 문제점

6.2 기울기 소실과 LSTM

6.3 LSTM 구현

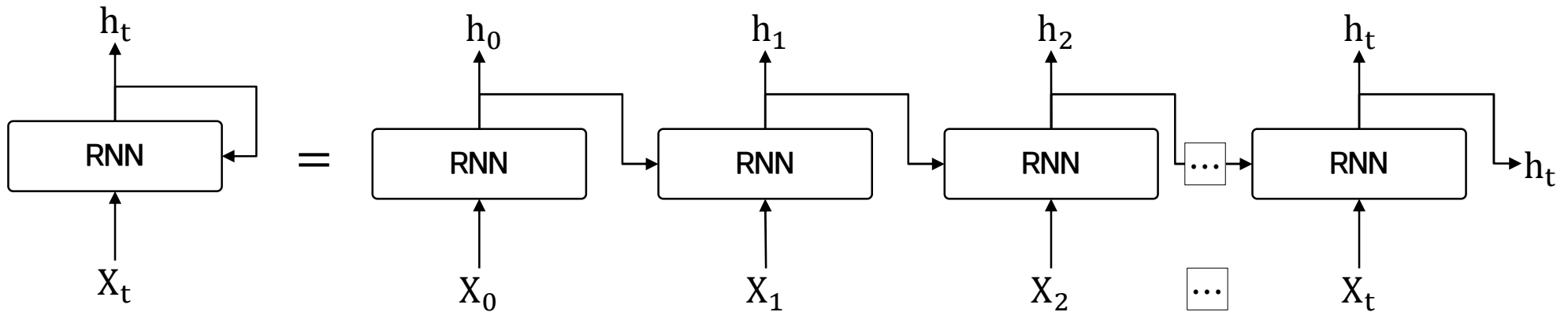
6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

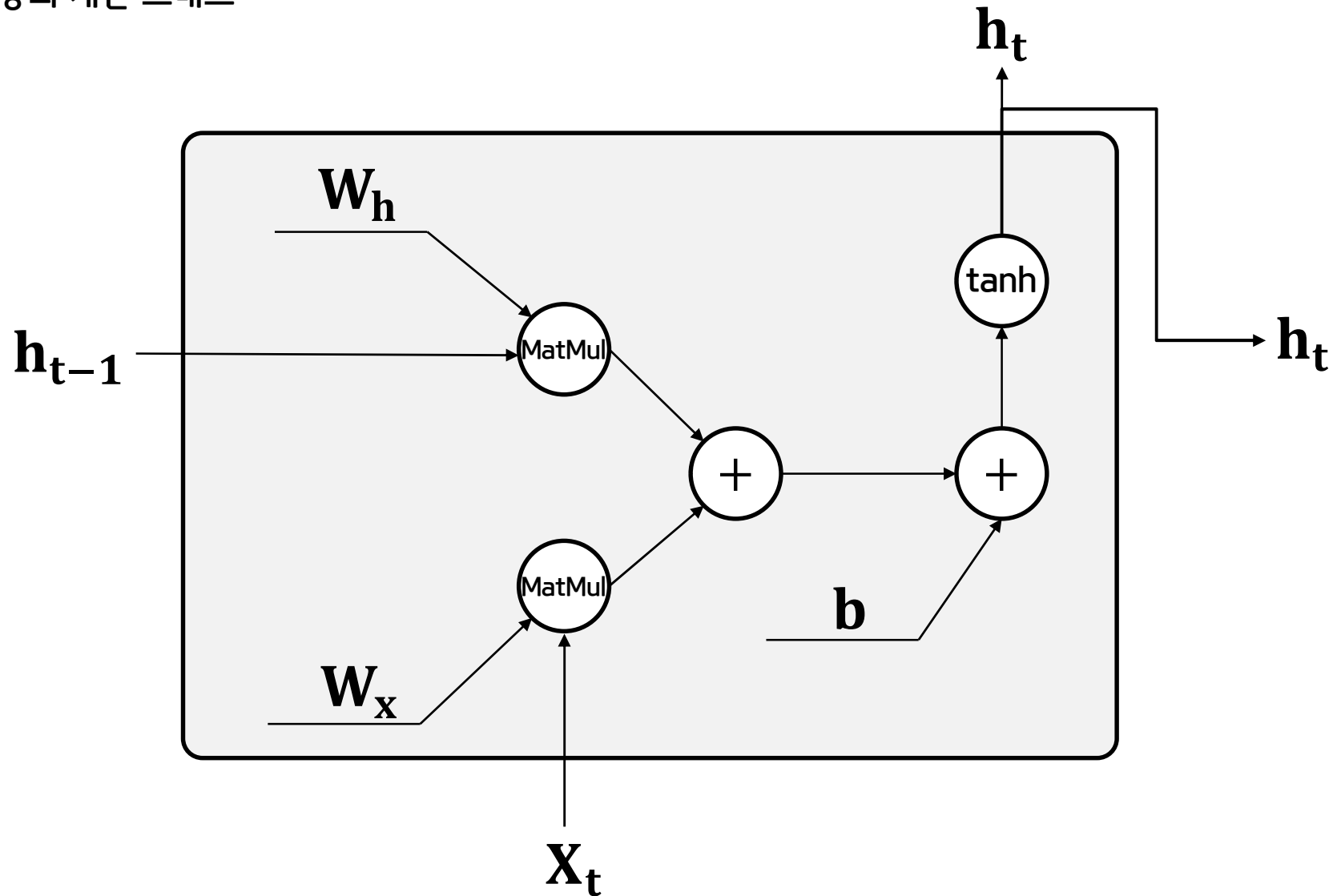
RNN의 문제점

RNN은 시계열 데이터의 장기 의존 관계를 학습하기 어렵다.
그 이유는 BPTT에서 기울기 소실 혹은 기울기 폭발이 일어나기 때문이다.

RNN 계층 : 순환을 펼치기 전과 후



RNN 계층의 계산 그래프



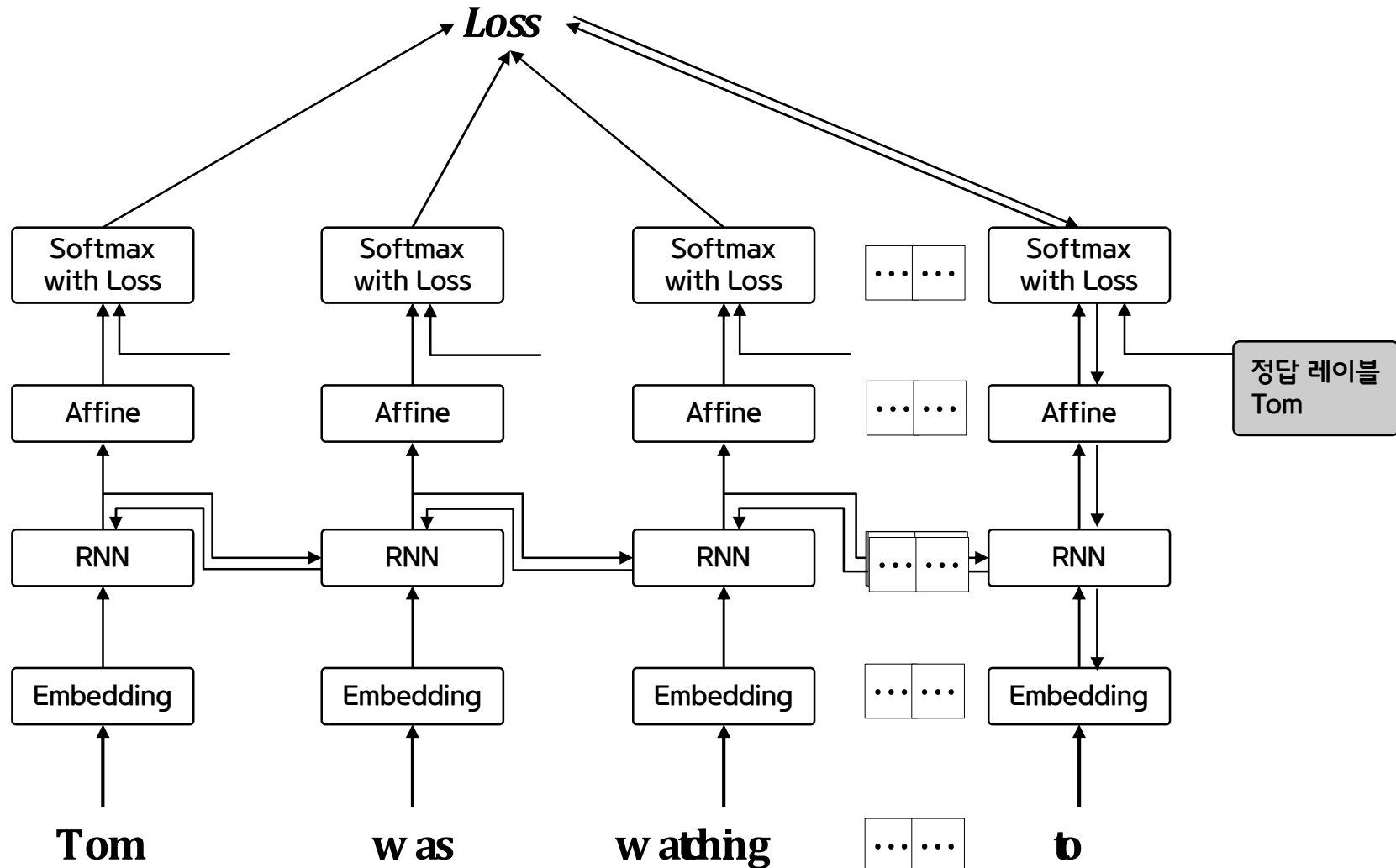
"?"에 들어갈 단어는 ? : (어느 정도의)장기 기억이 필요한 문제의 예

Tom was watching TV in his room. Mary came into the room. Mary said hi to ?

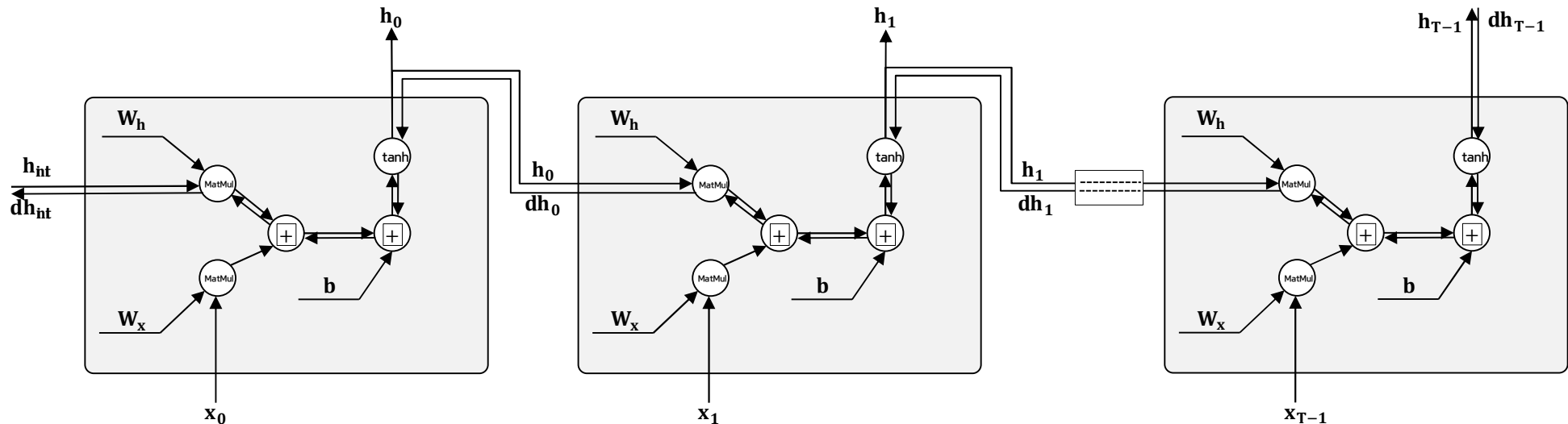
언어 모델은 주어진 단어들을 기초로 다음에 출현할 단어를 예측하는 일을 한다.
이번 절에서는 RNNLM의 단점을 확인하는 차원에서 다음의 문제를 다시 생각해보았다.

Tom was watching TV in his room. Mary came into the room. Mary said hi to ?
여기서 ?에 들어갈 단어는 Tom이다. RNNLM이 이 문제에 올바르게 답하라면,
현재 맥락에서 'Tom이 방에서 TV를 보고 있음'과 '그 방에 Mary가 들어옴'이란 정보를 기억해야 한다.
즉, 이런 정보를 RNN 계층의 은닉 상태에 인코딩해 보관해야 한다.

정답 레이블이 "Tom"임을 학습할 때의 기울기 흐름



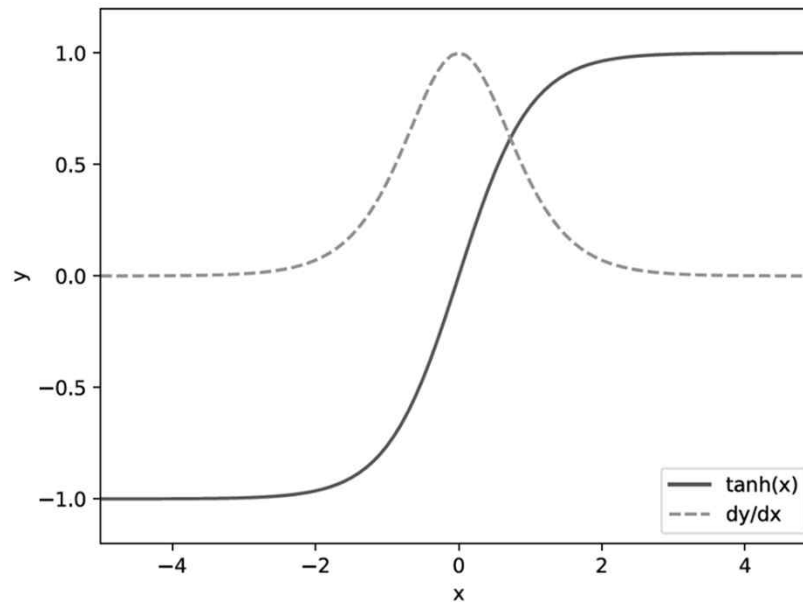
RNN 계층에서 시간 방향으로의 기울기 전파



위의 RNN 계층의 그림에서 시간 방향 기울기 전파에만 주목해보자.
 길이가 T인 시계열 데이터를 가정하여 T번째 정답 레이블로부터 전해지는 기울기가 어떻게 변하는지 보자.
 앞의 문제에 대입하면 T번째 정답 레이블이 'Tom'인 경우에 해당한다.
 이때 시간 방향 기울기에 주목하면 역전파로 전해지는 기울기는 차례로 'tanh', '+', 'MatMul(행렬 곱)' 연산을 통과한다는 것을 알 수 있다.

'+'의 역전파는 상류에서 전해지는 기울기를 그대로 하류로 흘려보내 기울기는 변하지 않는다.

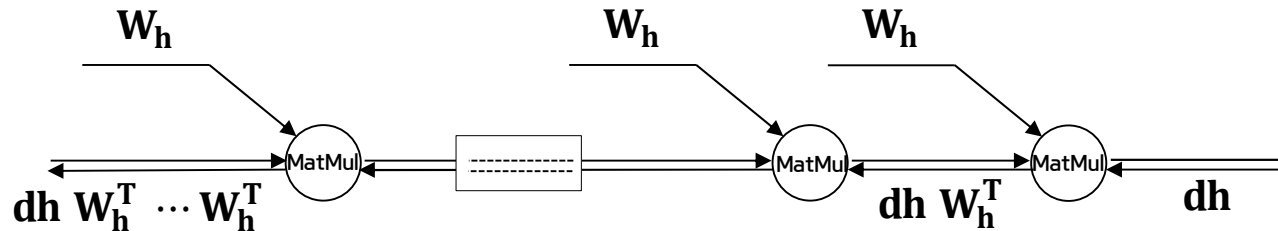
$y=\tanh(x)$ 의 그래프(점선은 미분)



그림에서 점선이 $y=\tanh(x)$ 의 미분이고 값은 1.0 이하이며, x 가 0으로부터 멀어질수록 작아진다. 즉, 역전파에서 기울기가 \tanh 노드를 지날 때마다 값은 계속 작아진다는 의미이다. 그리고 \tanh 함수를 T 번 통과하면 기울기도 T 번 반복해서 작아진다.

'MatMul(행렬 곱)' 노드의 경우 \tanh 노드를 무시하기로 한다. 그러면 RNN 계층의 역전파 시 기울기는 'MatMul' 연산에 의해서만 변화하게 된다.

RNN 계층의 행렬 곱에만 주목했을 때의 역전파의 기울기



상류로부터 dh 라는 기울기가 흘러온다고 가정하고 이때 MatMul 노드에서의 역전파는 $dh(W_h^T)$ 라는 행렬 곱으로 기울기를 계산한다.

그리고 같은 계산을 시계열 데이터의 시간 크기만큼 반복한다.
주의할 점은 행렬 곱셈에서 매번 똑같은 W_h 가중치를 쓴다는 것이다.

즉, 행렬 곱의 기울기는 시간에 비례해 지수적으로 증가/감소함을 알 수 있으며
증가할 경우 기울기 폭발이라고 한다. 기울기 폭발이 일어나면 오버플로를 일으켜 NaN 같은 값을 발생시킨다.
반대로 기울기가 감소하면 기울기 소실이 일어나고 이는 일정 수준 이하로 작아지면 가중치 매개변수가
더 이상 갱신되지 않으므로 장기 의존 관계를 학습할 수 없게 된다.

기울기 클리핑(gradients clipping)

$$\text{if } \|\hat{g}\| \geq \text{threshold} :$$

$$\hat{g} = \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}$$

6. 게이트가 추가된 RNN

6.1 RNN의 문제점

6.2 기울기 소실과 LSTM

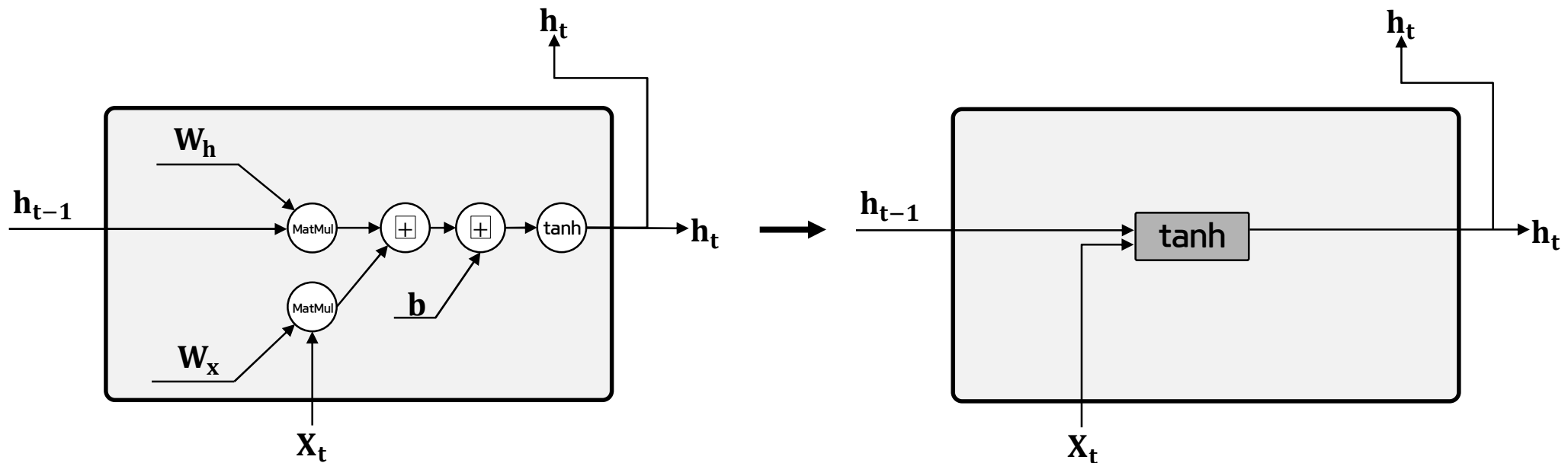
6.3 LSTM 구현

6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

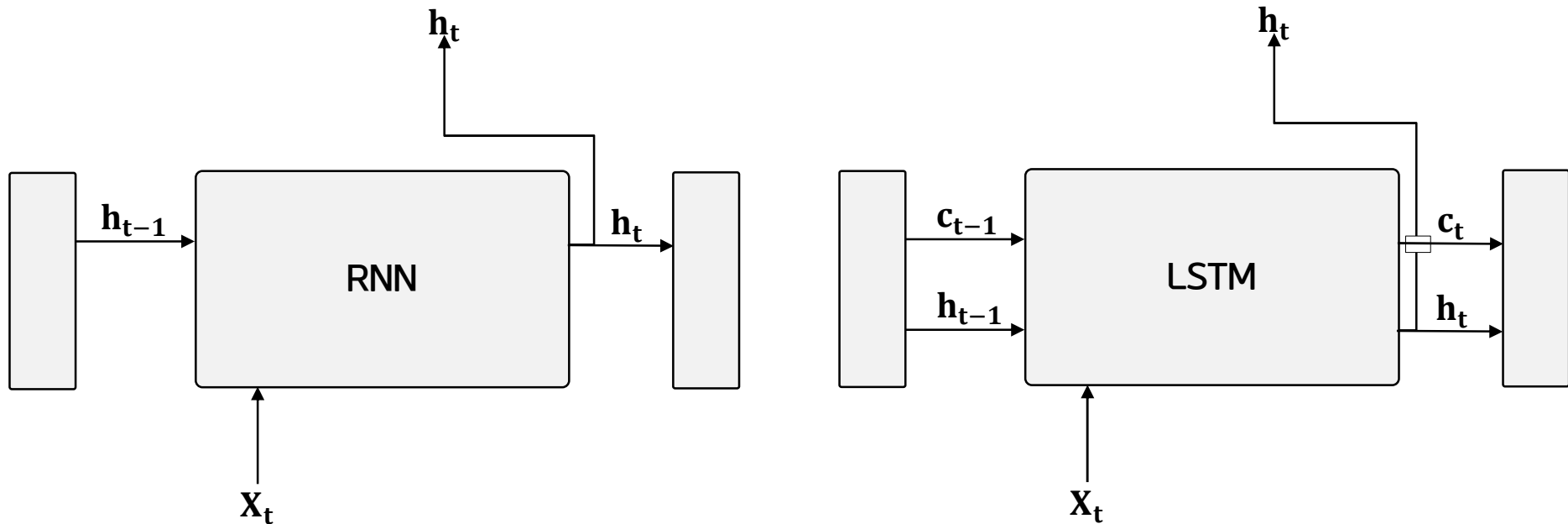
RNN 학습에서 기울기 소실도 큰 문제이다.
이 문제를 해결하려면 RNN 계층의 아키텍처를 근본부터 뜯어고쳐야 한다.

단순화한 도법을 적용한 RNN 계층



여기서는 $\tanh(h_{t-1}W_h + x_tW_x + b)$ 계산을 \tanh 라는 직사각형 노드 하나로
그리고 직사각형 노드 안에 행렬 곱과 편향의 합, 그리고 \tanh 함수에 의한 변환이 모두 포함된 것이다.

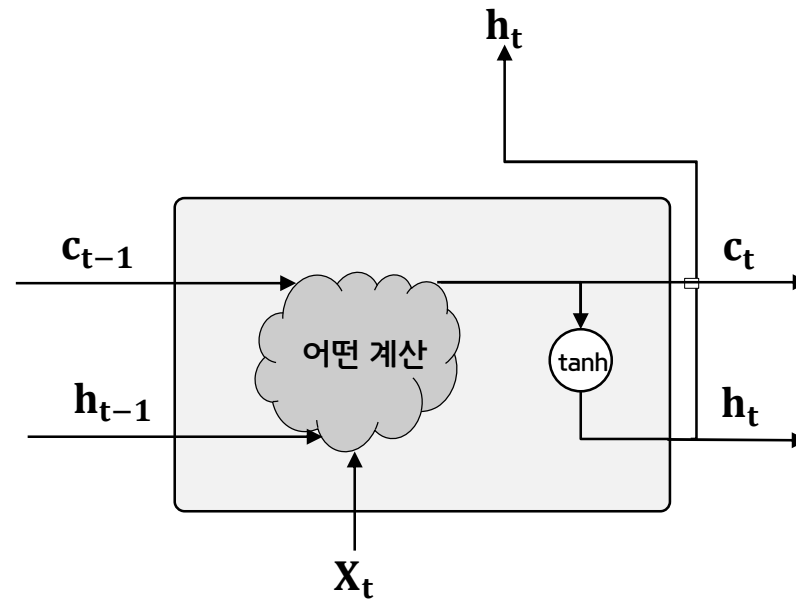
RNN 계층과 LSTM 계층 비교



이 그림에서는 LSTM 계층의 인터페이스에는 c 라는 경로가 있다는 차이가 있다. 여기서 c 를 기억 셀이라 하며 LSTM 전용의 기억 메커니즘이다.

기억 셀의 특징은 데이터를 LSTM 계층 내에서만 주고받는다라는 것이다. 다른 계층으로는 출력하지 않는다는 것이다. 반면, LSTM의 은닉 상태 h 는 RNN 계층과 마찬가지로 다른 계층, 위쪽으로 출력된다.

기억 셀 c_t 를 바탕으로 은닉 상태 h_t 를 계산하는 LSTM 계층



그림에서 현재의 기억 셀 c_t 는 3개의 입력 (c_{t-1} , h_{t-1} , x_t)으로부터 '어떤 계산'을 수행하여 구할 수 있다.

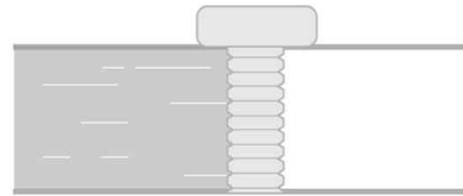
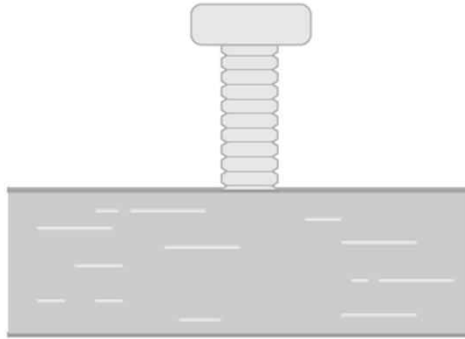
여기서 핵심은 갱신된 c_t 를 사용해 은닉 상태 h_t 를 계산한다는 것이다.

또한 이 계산은 $h_t = \tanh(c_t)$ 인데, 이는 c_t 의 각 요소에 \tanh 함수를 적용한다는 뜻이다.

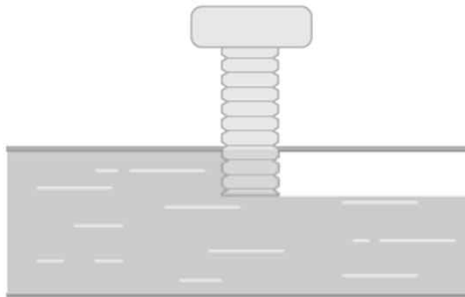
'게이트'란 우리나라로 '문'을 의미하는 단어이다.

문은 열거나 닫을 수 있듯이, 게이트는 데이터의 흐름을 제어한다.

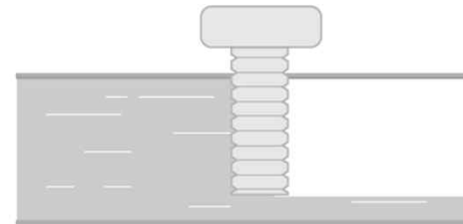
비유하자면 게이트는 물의 흐름을 제어한다.



물이 흐르는 양을 0.0~1.0 범위에서 제어한다.



0.7(70%)



0.2(20%)

$\tanh(c_t)$ 의 각 원소에 대해 '그것이 다음 시각의 은닉 상태에 얼마나 중요한가'를 조정한다.
한편, 이 게이트는 다음 은닉 상태 h_t 의 출력을 담당하는 게이트이므로 output 게이트라고 한다.

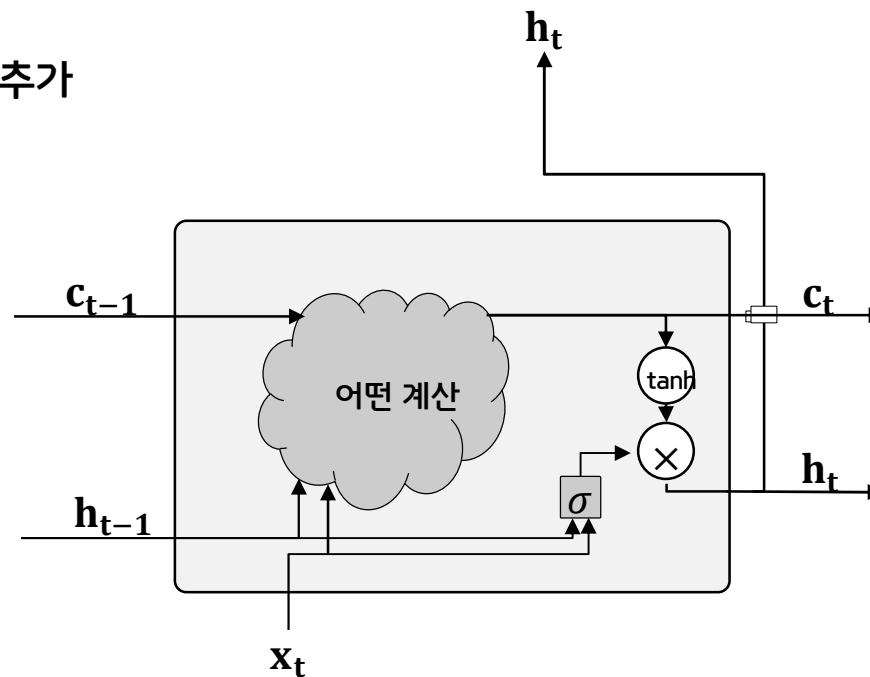
output 게이트의 열림 상태는 입력 x_t 와 이전 상태 h_{t-1} 로부터 구한다.

이때의 식은 밑에 식과 같다.

$$o = \sigma(x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)})$$

밑에 그림에서 output 게이트에서 수행하는 식의 계산을 sigma로 표기했다.
sigma의 출력을 o라고 하면 h_t 는 o와 $\tanh(c_t)$ 의 곱으로 계산된다.

output 게이트 추가

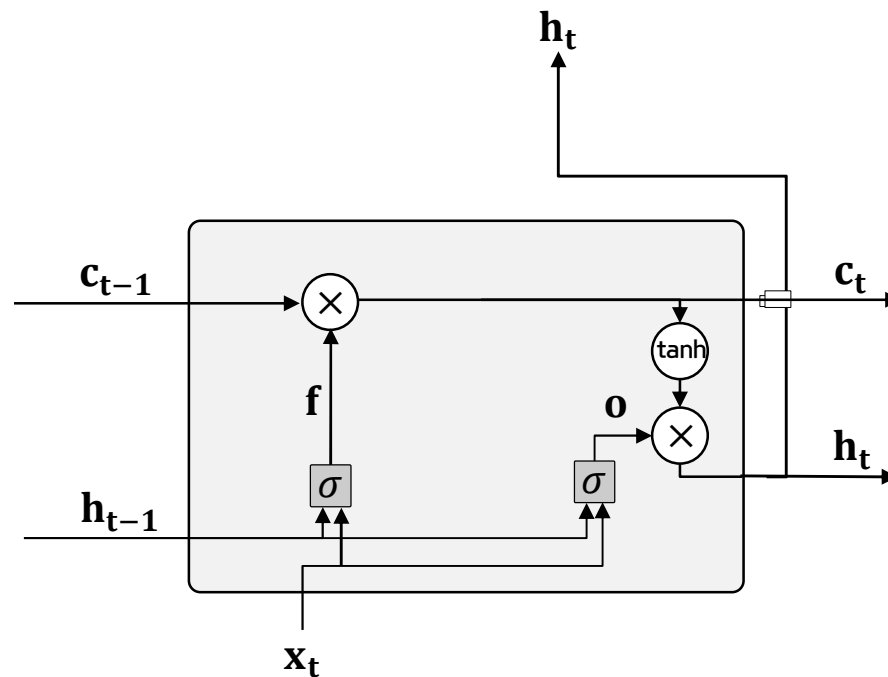


아다마르 곱(Hadamard product)

$$h_t = o \odot \tanh(c_t)$$

다음에 해야 할 일은 기억 셀에 '무엇을 잊을까'를 명확하게 지시하는 것이다.

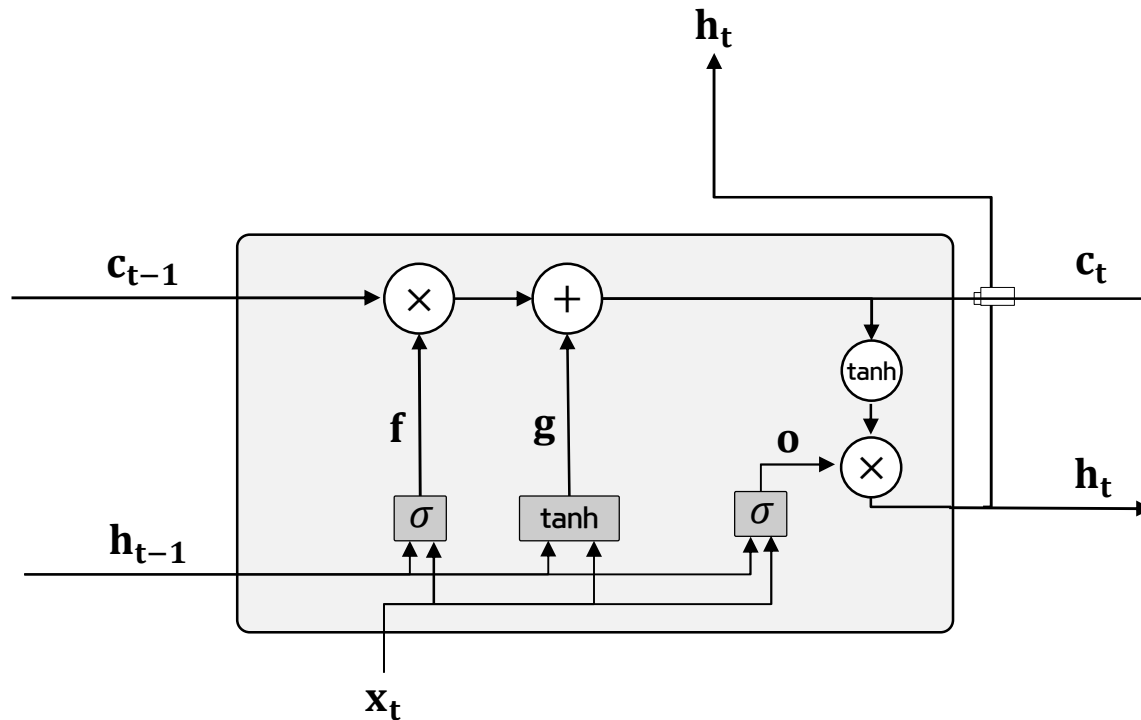
forget 게이트 추가



$$f = \sigma(x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)})$$

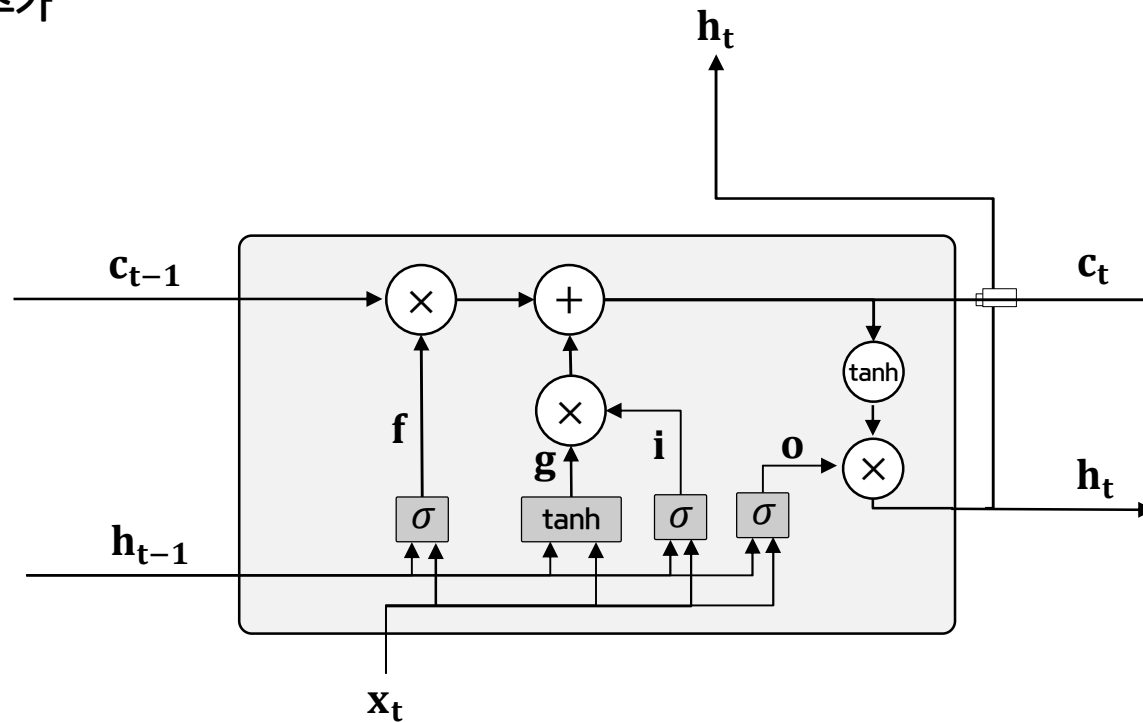
forget 게이트를 거치면서 이전 시각의 기억 셀로부터 잊어야 할 기억이 삭제되었다.

새로운 기억 셀



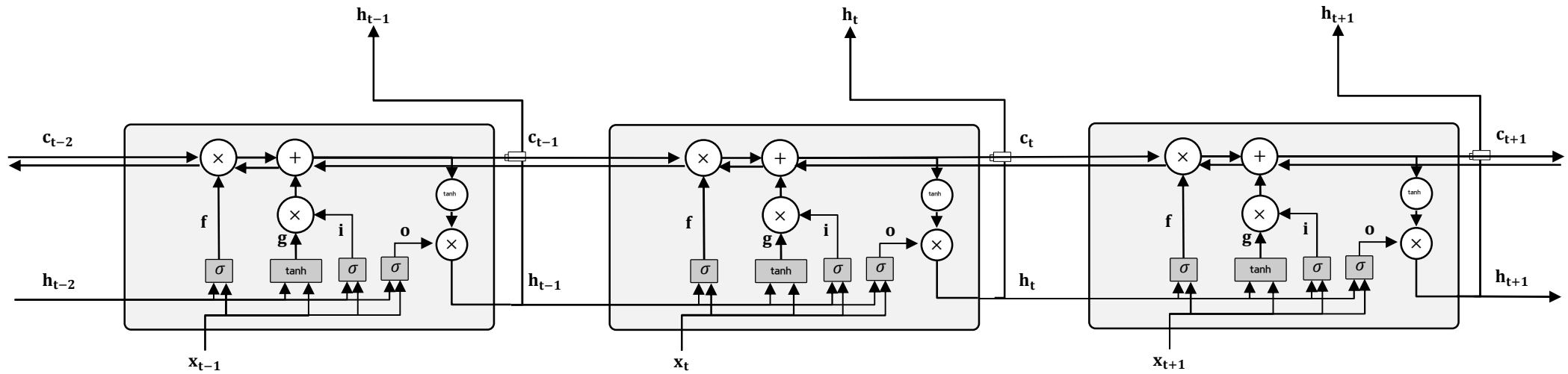
$$g = \tanh (x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)})$$

input 게이트 추가



$$i = \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)})$$

기억 셀의 역전파



6. 게이트가 추가된 RNN

6.1 RNN의 문제점

6.2 기울기 소실과 LSTM

6.3 LSTM 구현

6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

$$f = \sigma(x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)})$$

$$g = \tanh(x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)})$$

$$i = \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)})$$

$$o = \sigma(x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)})$$

$$c_t = f \odot c_{t-1} + g \odot i$$

$$h_t = o \odot \tanh(c_t)$$

각 식의 가중치들을 모아 4개의 식을 단 한 번의 아핀 변환으로 계산

$$x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)}$$

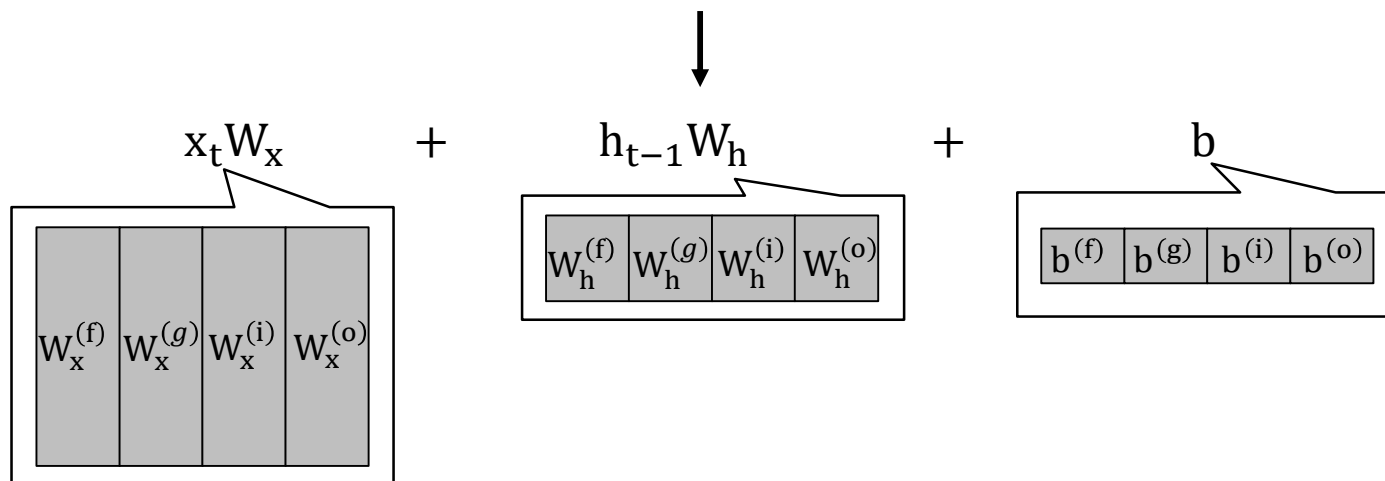
$$x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)}$$

$$x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)}$$

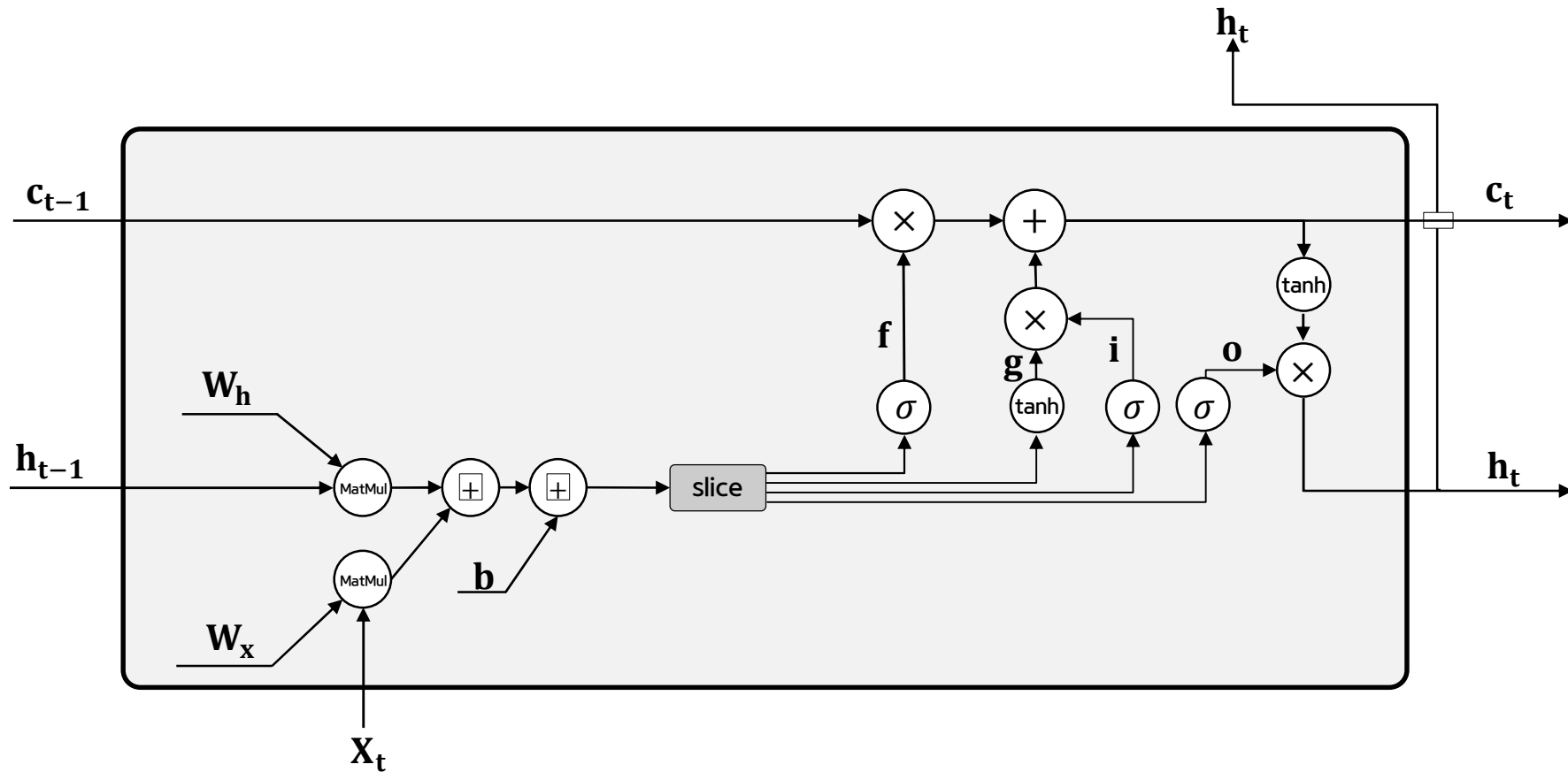
$$x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)}$$

$$\downarrow$$

$$x_t \begin{bmatrix} W_x^{(f)} & W_x^{(g)} & W_x^{(i)} & W_x^{(o)} \end{bmatrix} + h_{t-1} \begin{bmatrix} W_h^{(f)} & W_h^{(g)} & W_h^{(i)} & W_h^{(o)} \end{bmatrix} + \begin{bmatrix} b^{(f)} & b^{(g)} & b^{(i)} & b^{(o)} \end{bmatrix}$$

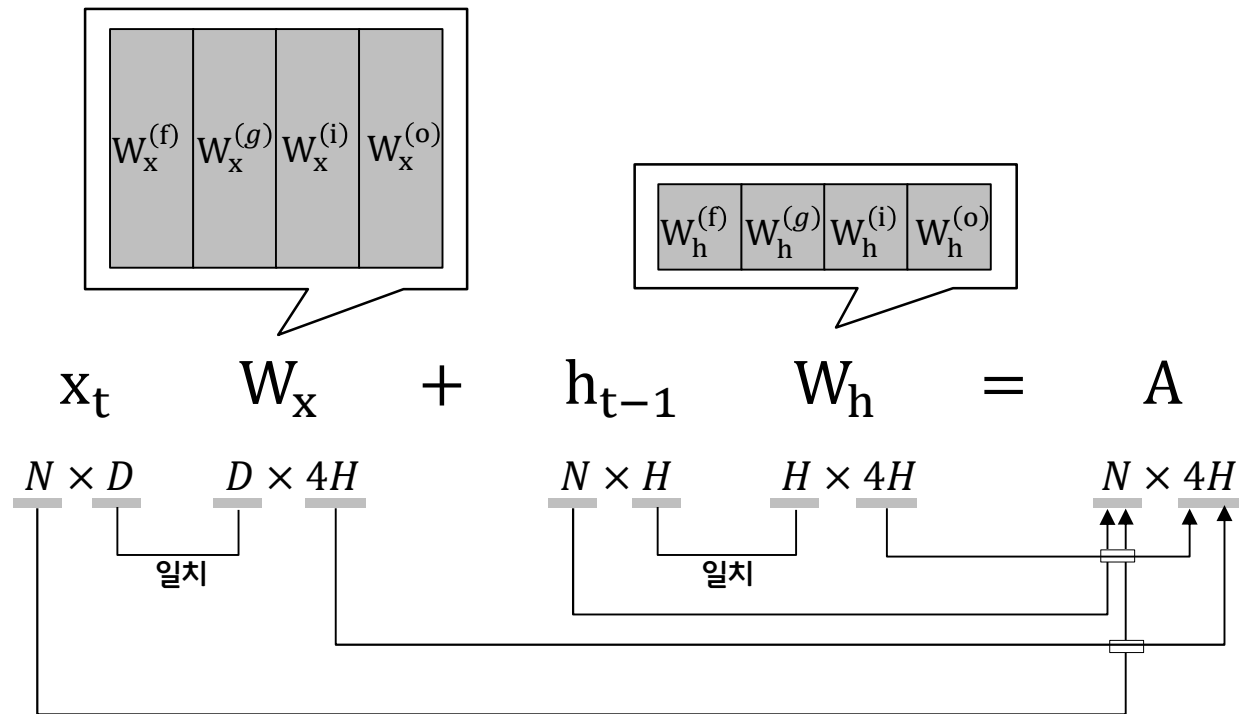


4개분의 가중치를 모아 아핀 변환을 수행하는 LSTM의 계산 그래프

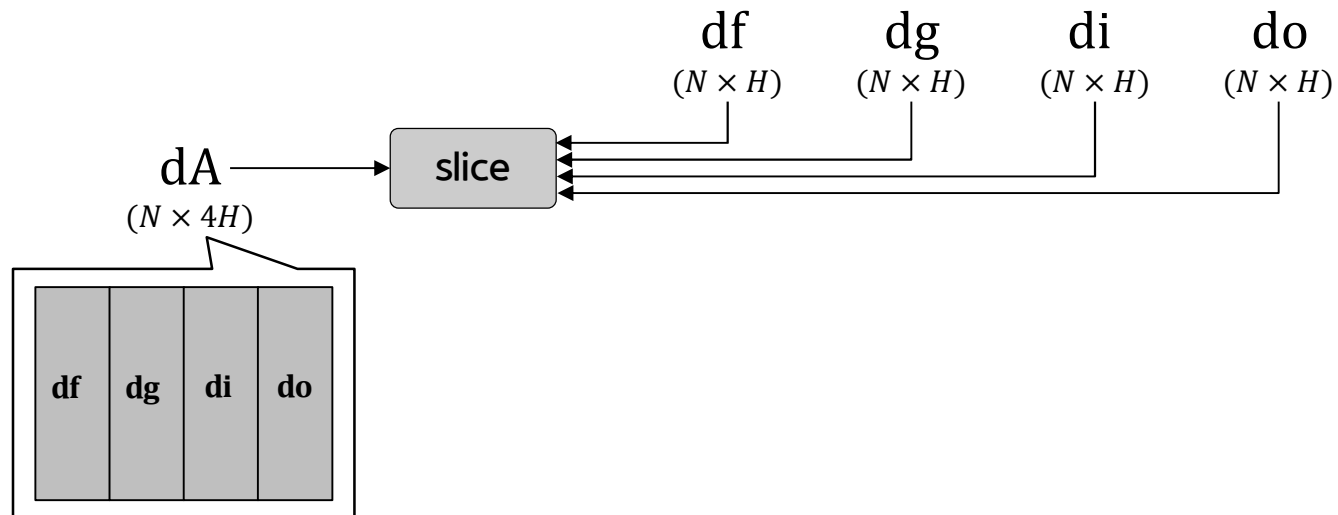
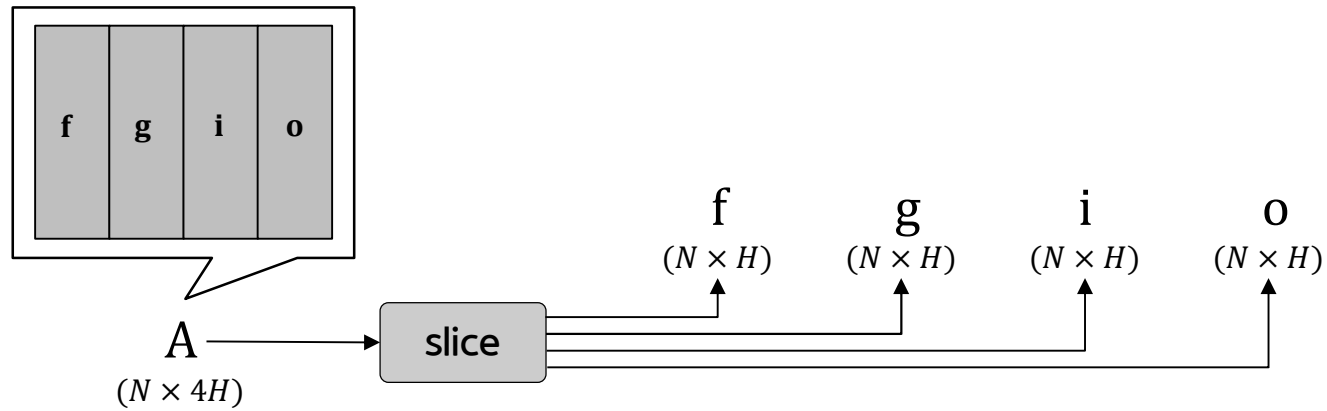


만약 W_x, W_h, b 각각에 4개분의 가중치가 포함되어 있다고 가정하면 위의 그림처럼 그래프가 그려진다.

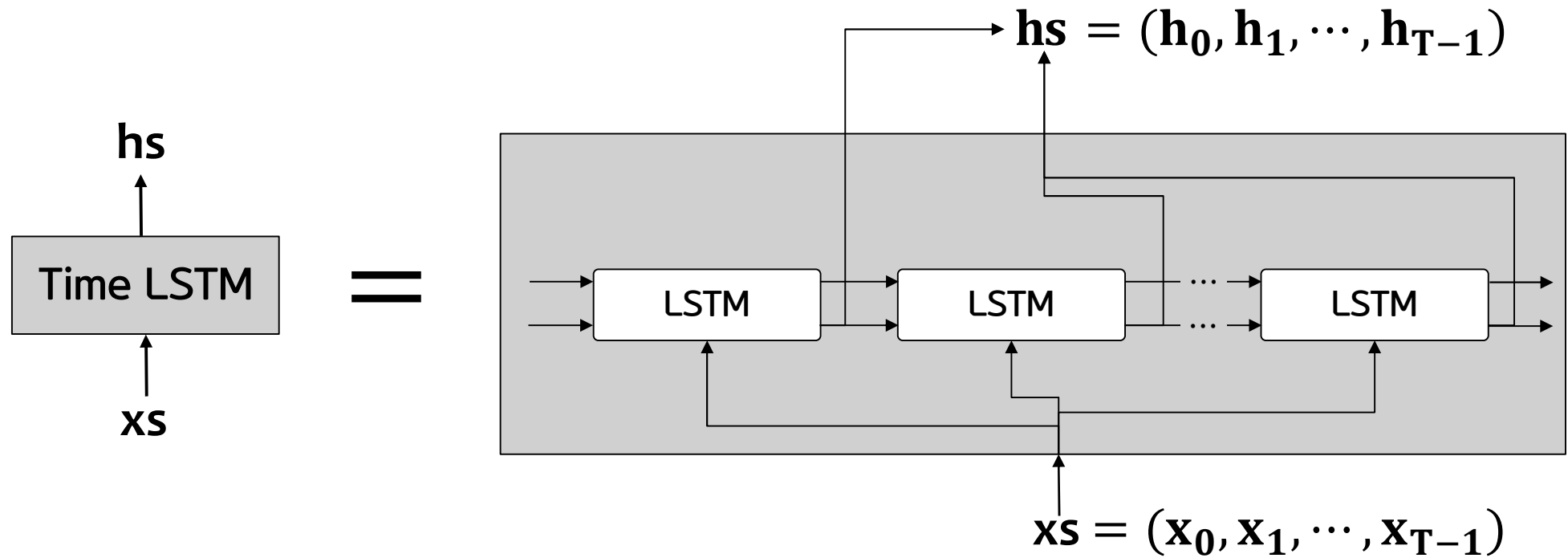
아핀 변환 시의 형상 추이



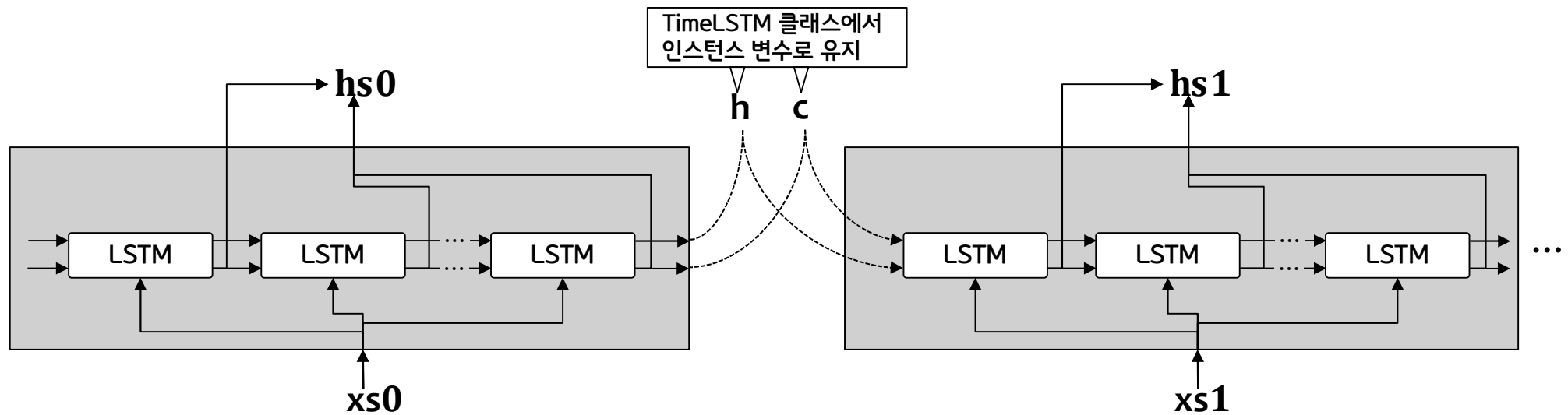
slice 노드의 순전파(위)와 역전파(아래)



Time LSTM의 입출력



Time LSTM 역전파의 입출력



6. 게이트가 추가된 RNN

6.1 RNN의 문제점

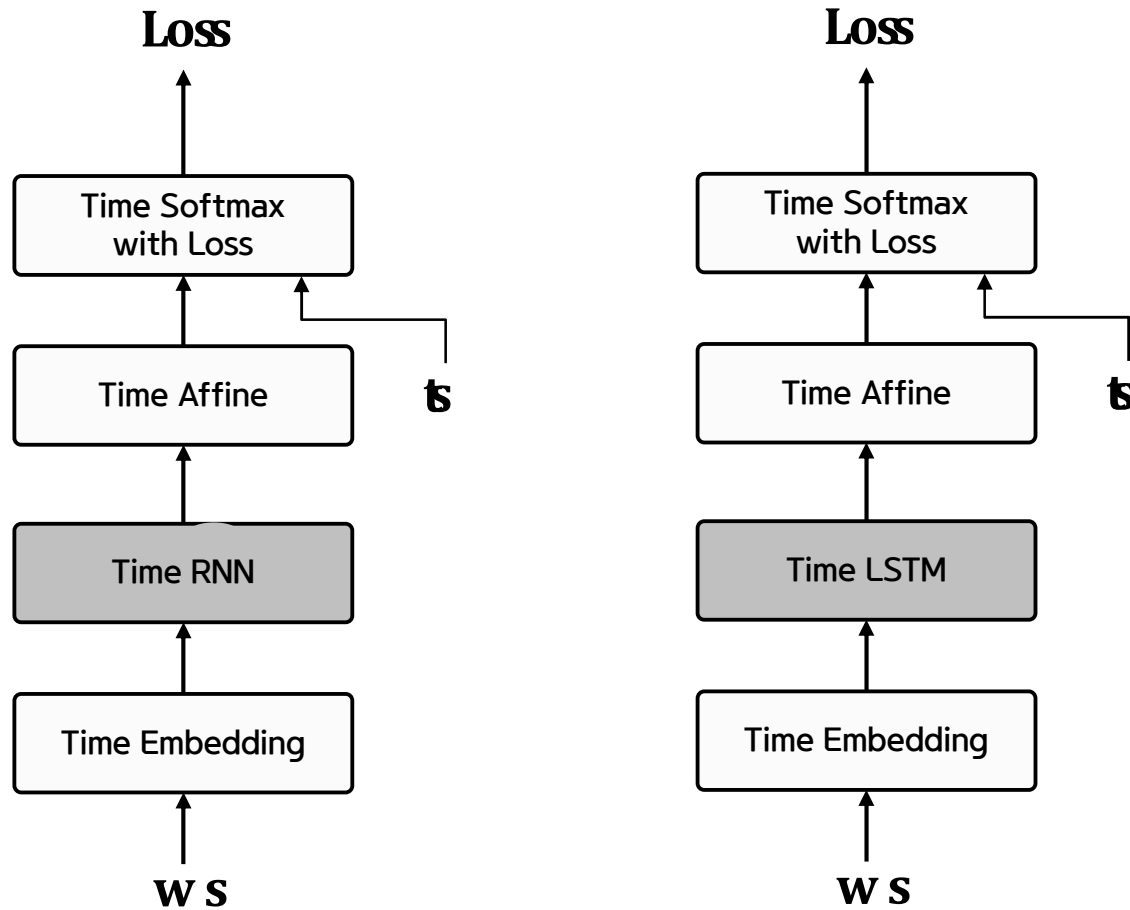
6.2 기울기 소실과 LSTM

6.3 LSTM 구현

6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

언어 모델의 신경망 구성



6. 게이트가 추가된 RNN

6.1 RNN의 문제점

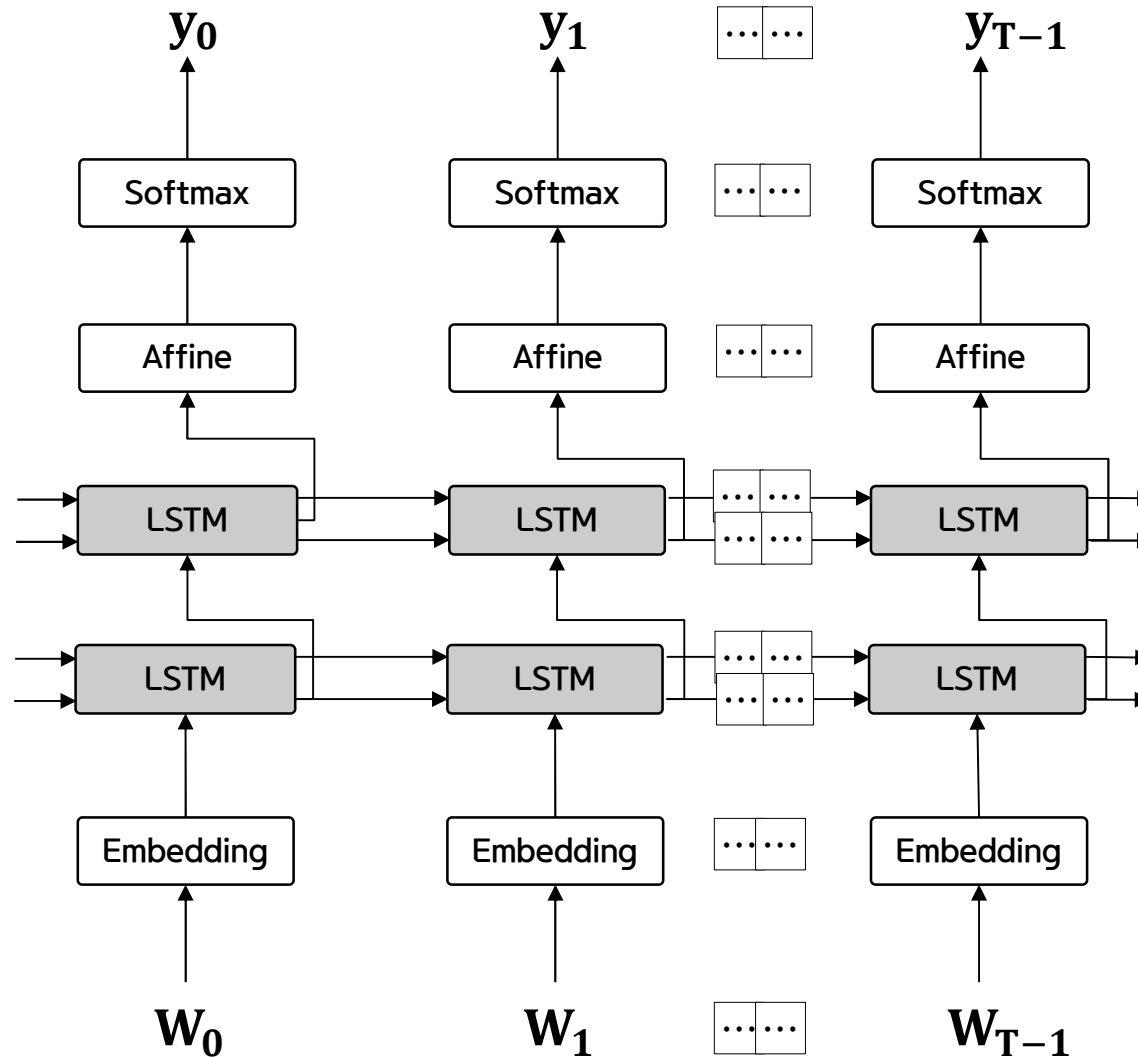
6.2 기울기 소실과 LSTM

6.3 LSTM 구현

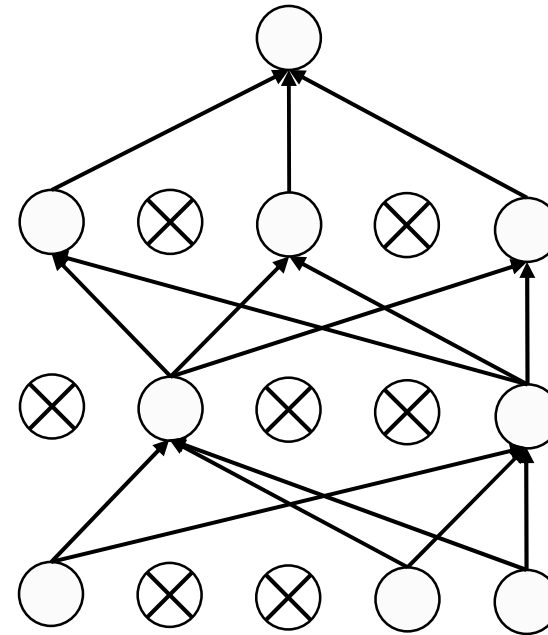
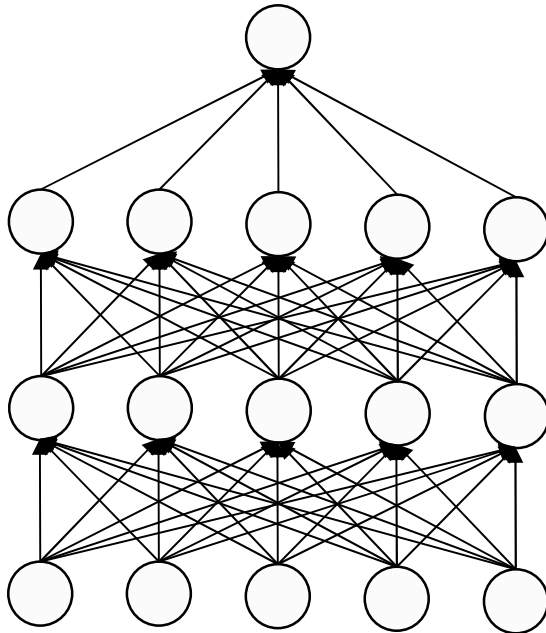
6.4 LSTM을 사용한 언어 모델

6.5 RNNLM 추가 개선

LSTM 계층을 2층으로 쌓은 RNNLM

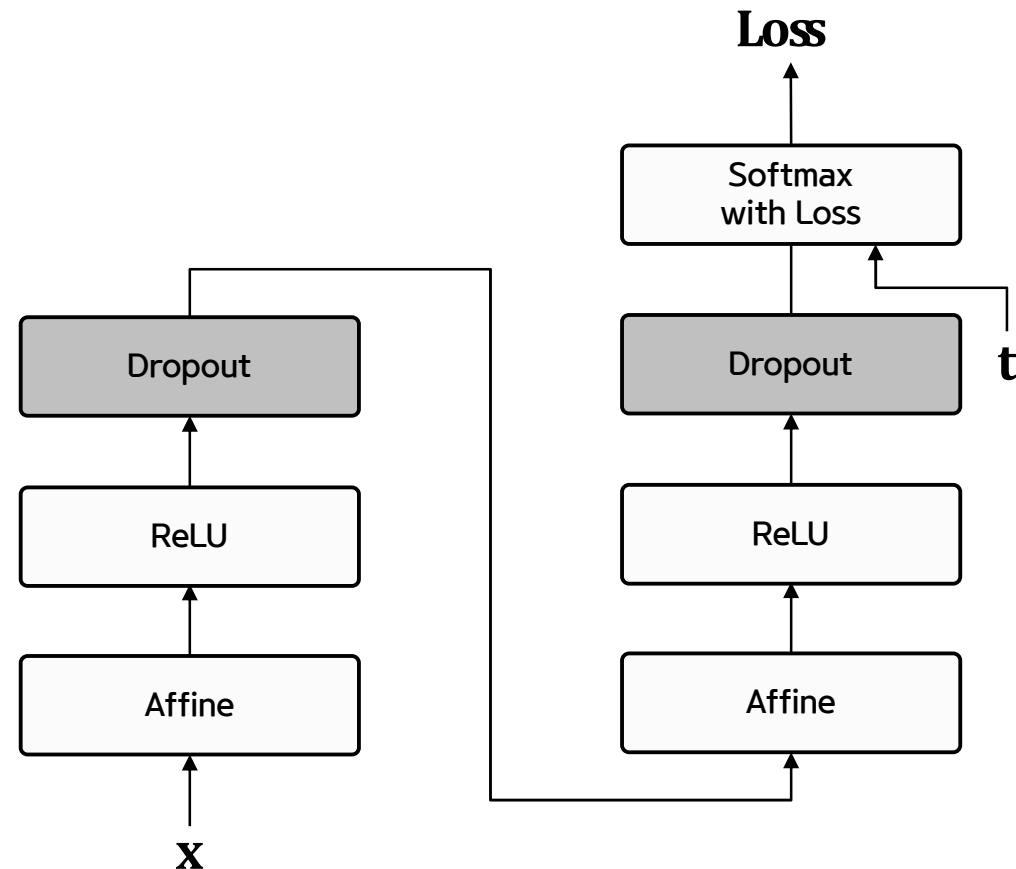


드롭아웃 개념도 : 왼쪽이 일반적인 신경망, 오른쪽이 드롭아웃을 적용한 신경망



드롭아웃은 무작위로 뉴런을 선택하여 선택한 뉴런을 무시한다.
무시한다는 말은 그 앞 계층으로부터의 신호 전달을 막는다는 뜻이다.
이 '무작위한 무시'가 제약이 되어 신경망의 일반화 성능을 개선하는 것이다.

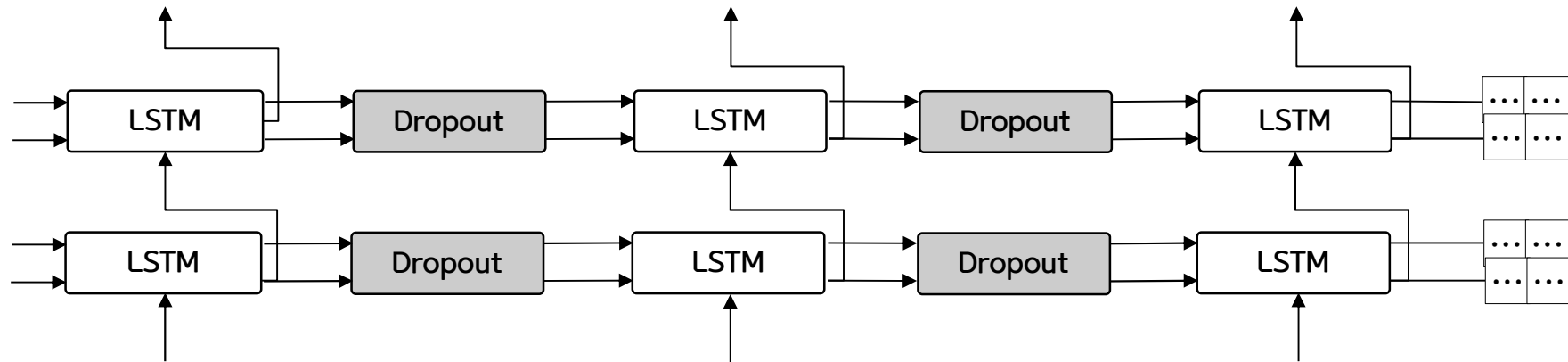
피드포워드 신경망에 드롭아웃 계층을 적용하는 예



이 그림은 드롭아웃 계층을 활성화 함수 뒤에 삽입하는 방법으로 과적합 억제에 기여하는 모습이다.

RNN을 사용한 모델에서 드롭아웃 계층을 LSTM 계층의 시계열 방향으로 삽이면 좋은 방법이 아니다.

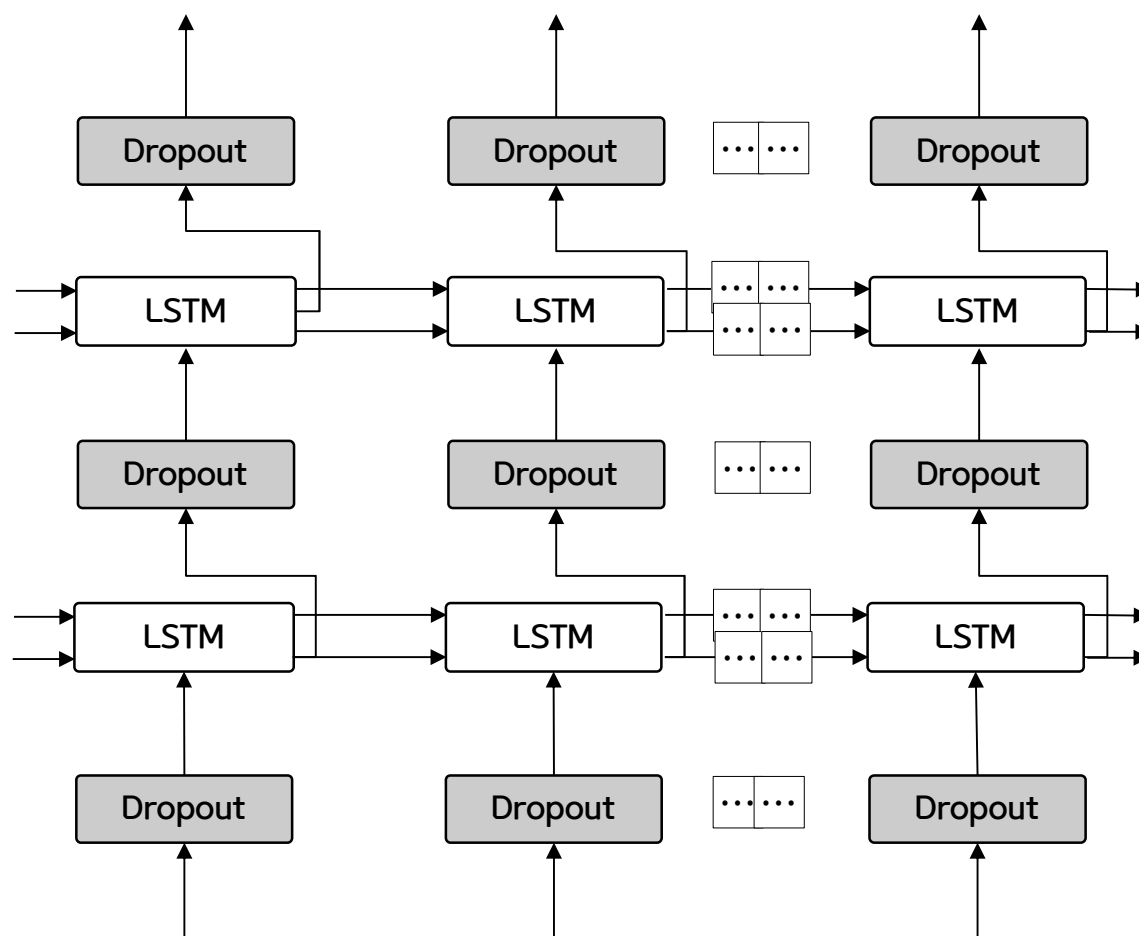
나쁜 예: 드롭아웃 계층을 시계열 방향으로 삽입



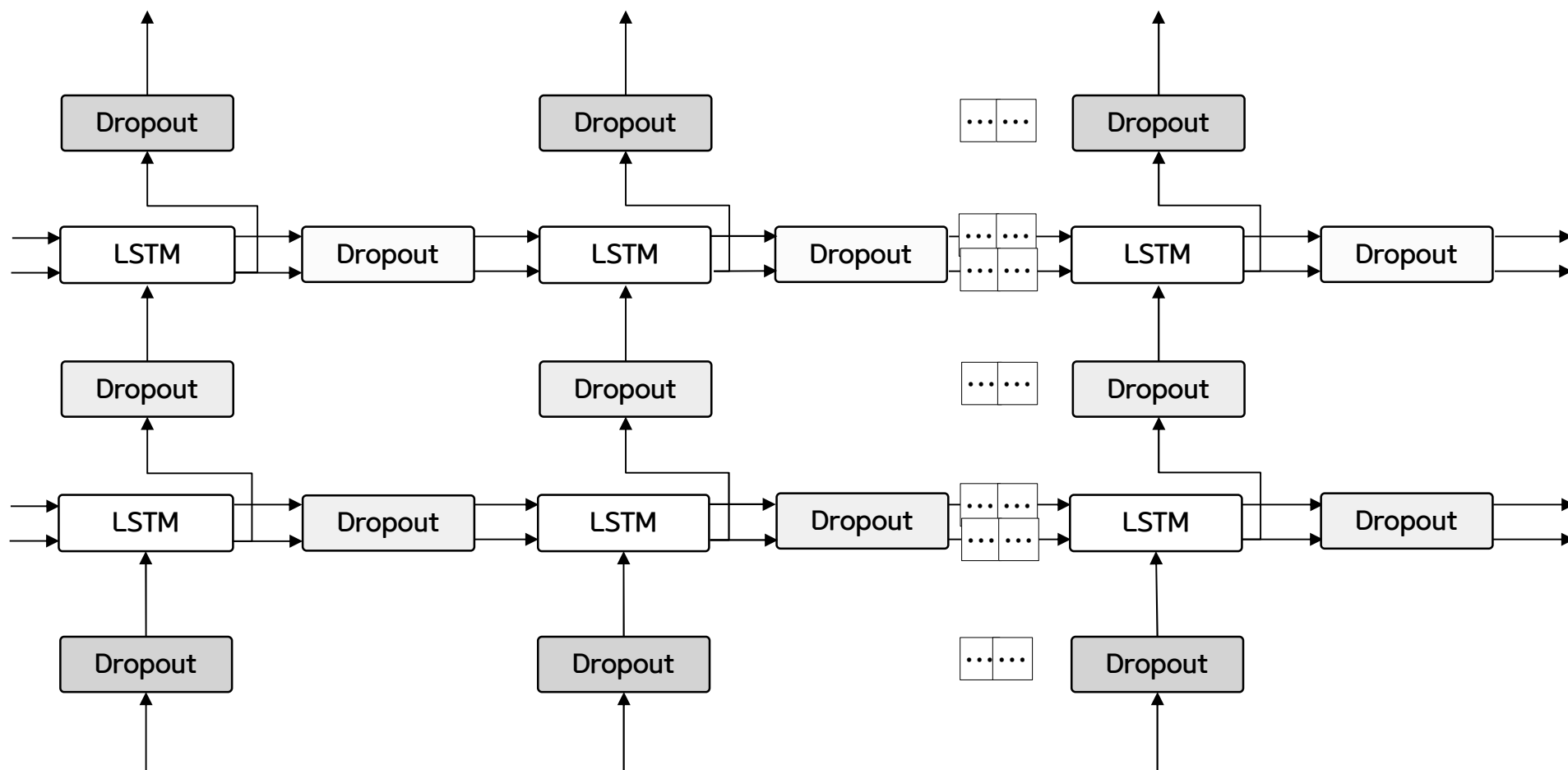
RNN에서 시계열 방향으로 드롭아웃을 학습 시 놓어버리면 시간이 흐름에 따라 정보가 사라질 수 있다. 즉, 흐르는 시간에 비례해 드롭아웃에 의한 노이즈가 축적된다.

드롭아웃 계층을 깊이 방향(상하 방향)으로 삽입하는 방안을 생각해보자

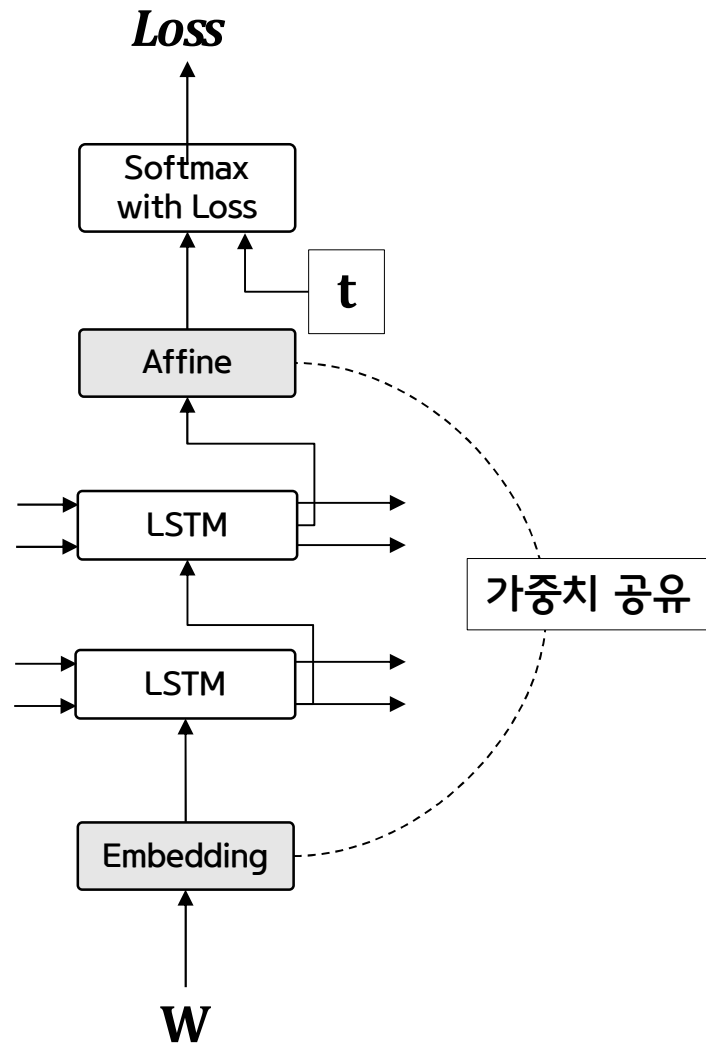
좋은 예: 드롭아웃 계층을 깊이 방향(상하 방향)으로 삽입



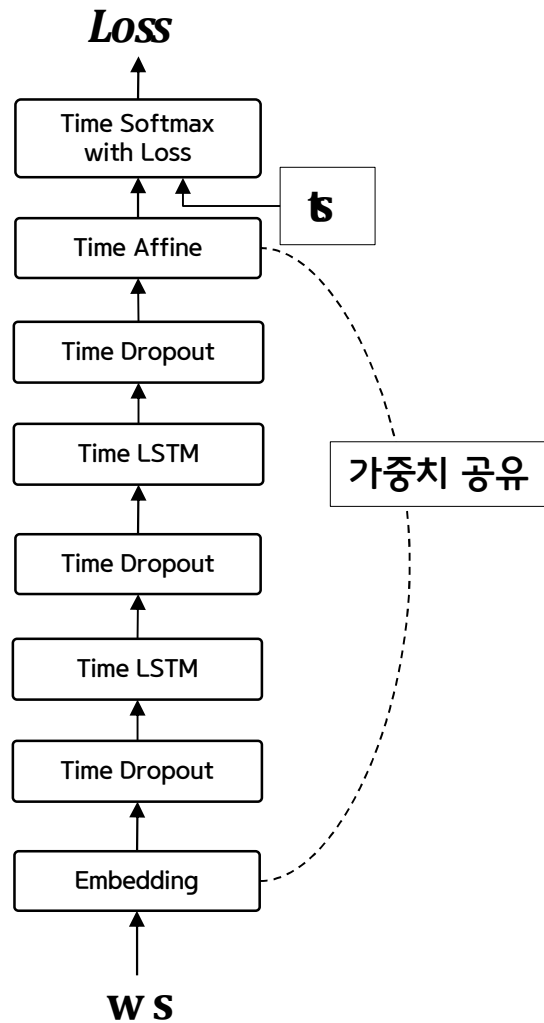
변형 드롭아웃의 예: 색이 같은 드롭아웃 끼리는 같은 마스크를 이용한다. 이처럼 같은 계층에 적용되는 드롭아웃 끼리는 공통의 마스크를 이용함으로써 시간 방향 드롭아웃도 효과적으로 작동할 수 있다.



언어 모델에서의 가중치 공유 예: Embedding 계층과 Sotmax 앞단의 Affine 계층이 가중치를 공유한다.



BetterRnnlm 클래스의 신경망 구성



- LSTM 계층의 다층화(여기서는 2층)
- 드롭아웃 사용(깊이 방향으로만 적용)
- 가중치 공유(Embedding 계층과 Affine 계층에서 가중치 공유)

※ 코드 참조