딥러닝 스터디

밑바닥부터 시작하는 딥러닝2

김제우

딥러닝스터디

목차

4. word2vec 속도 개선

word2vec 속도 개선

딥러닝스터디

CBOW 모델의 추론 처리 – 3장

- 입력 : 주변단어
- 출력 : 가운데 단어
- 주변단어로 가운데 단어 추론

word2vec 속도 개선

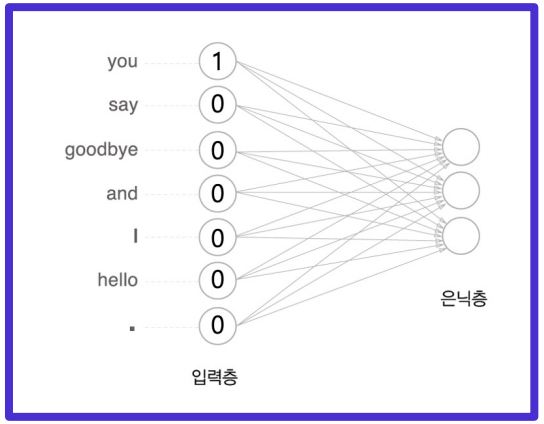
- 앞에서 배운 것은 word2vec의 구조와 CBOW의 구현!
- word2vec의 문제점
  - 말뭉치가 커지면 계산량도 커진다
  - 어휘 수가 어느 정도를 넘어서면 계산 시간이 너무 오래걸림

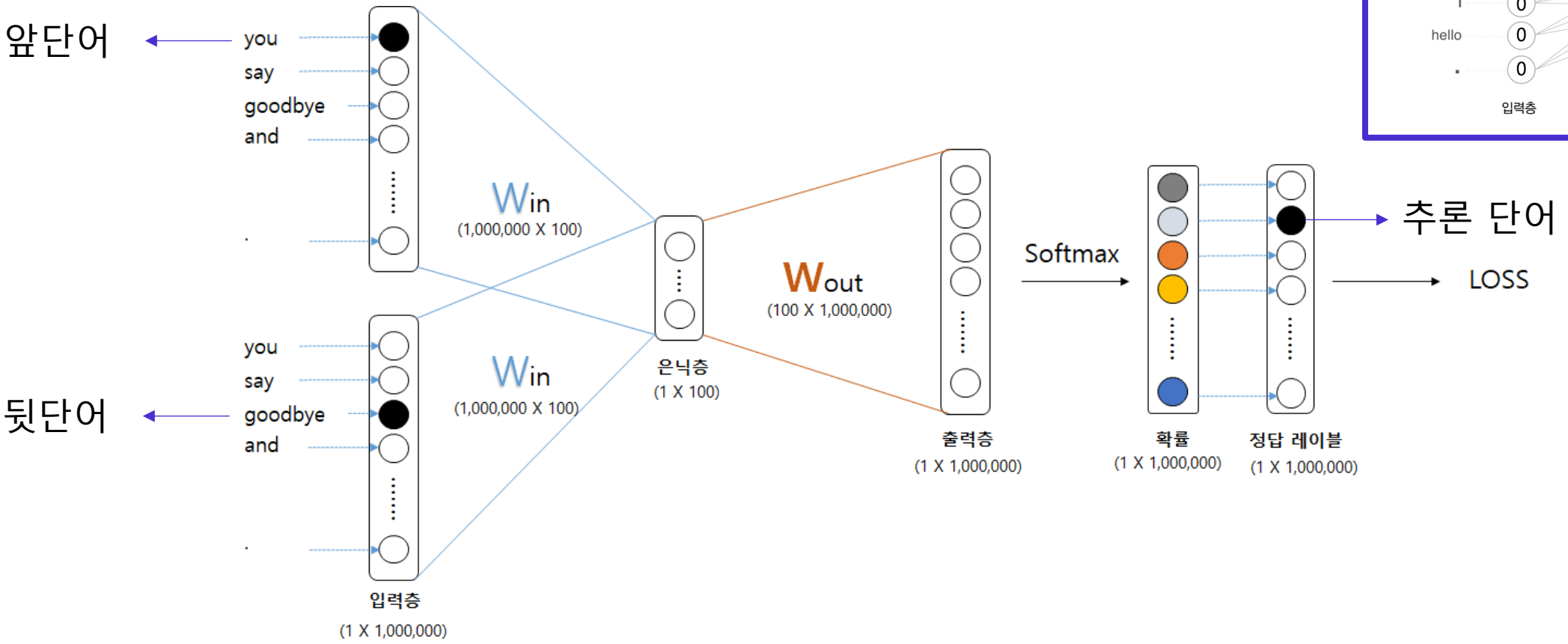
- 2가지 방법으로 개선
  1. Embedding 이라는 계층 도입
  2. 네거티브 샘플링이라는 새로운 손실함수

word2vec 속도 개선

딥러닝스터디

4.1 word2vec 개선 – 1





1,000,000 x 100 x 2번

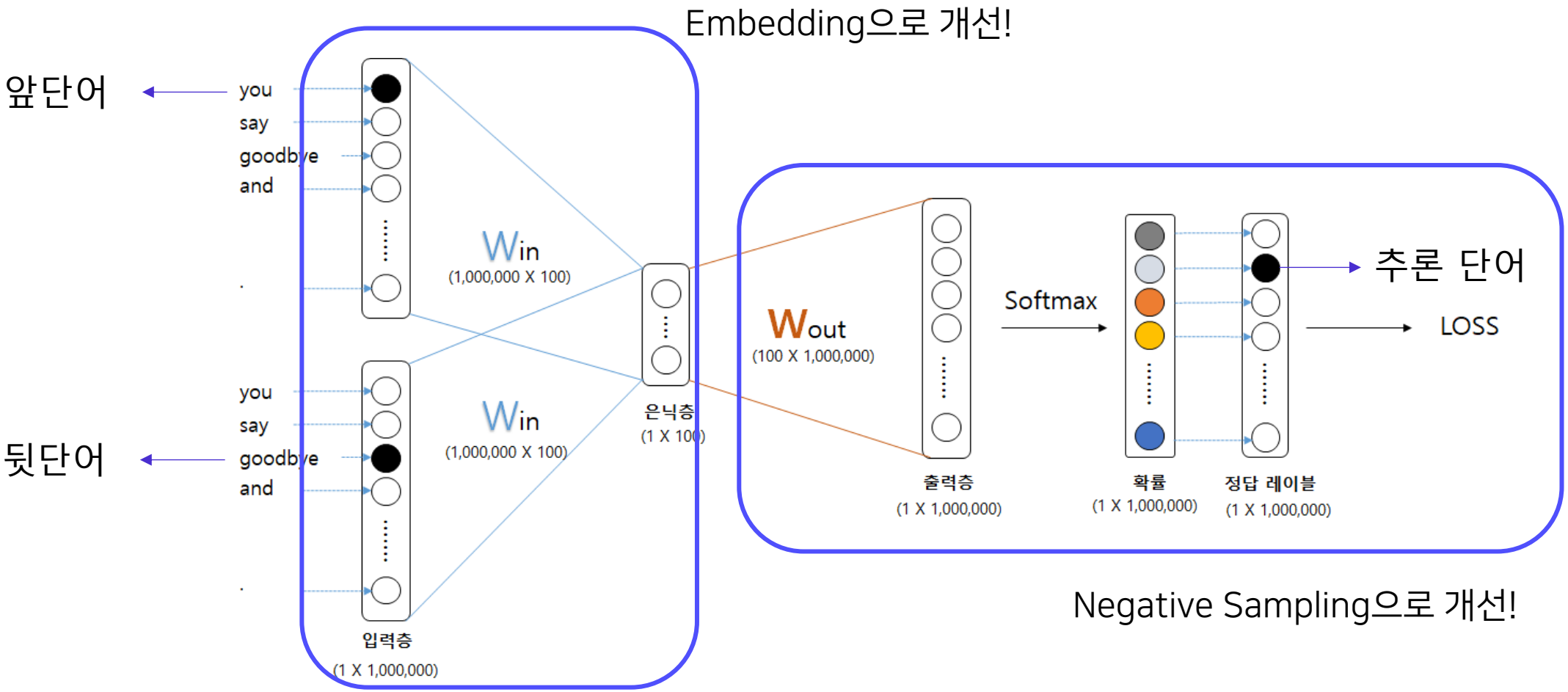
word2vec 속도 개선

연산량이 너무 많다!
메모리를 한번에 많이 차지함
-> 큰 사이즈의 램을 필요로 함

word2vec 속도 개선

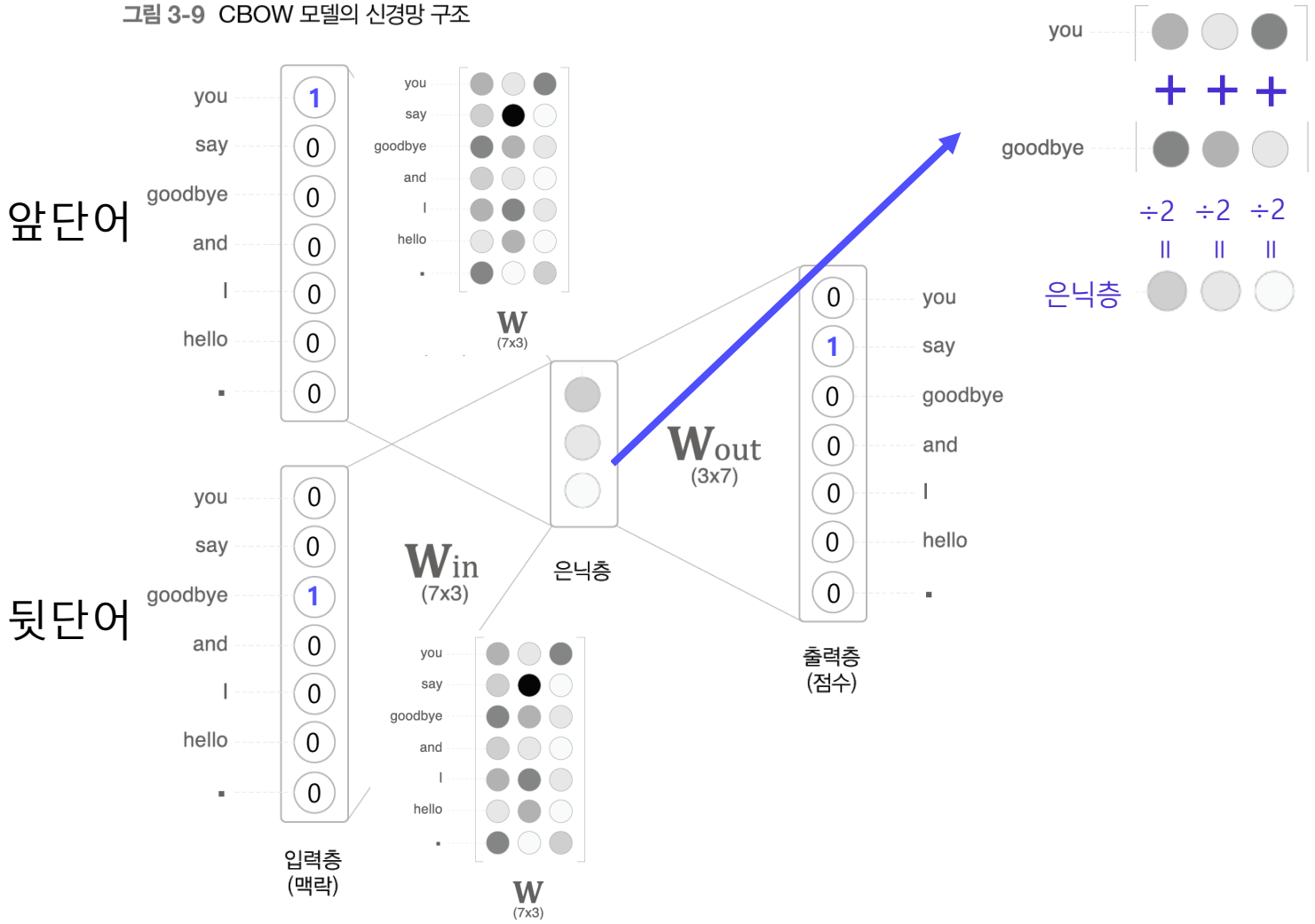
딥러닝스터디

4.1 word2vec 개선 – 1



word2vec 속도 개선

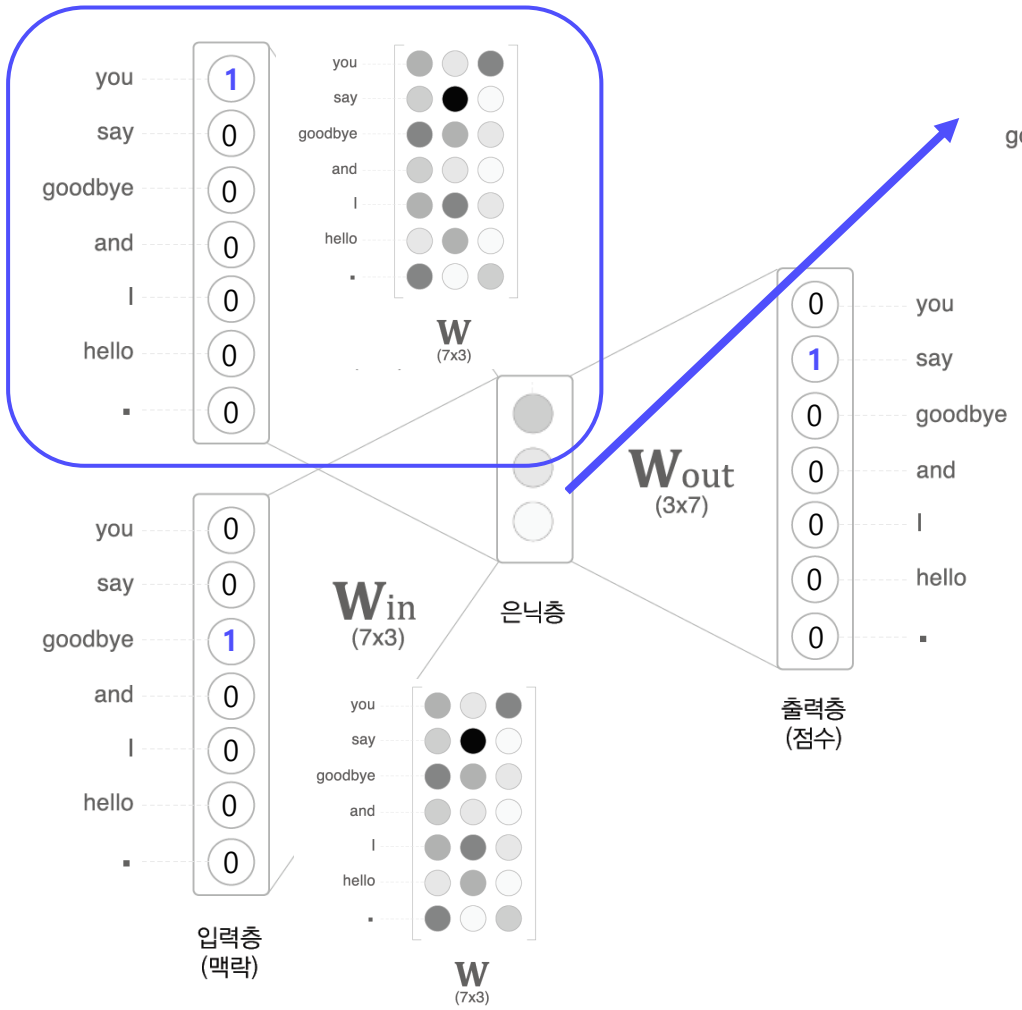
그림 3-9 CBOW 모델의 신경망 구조

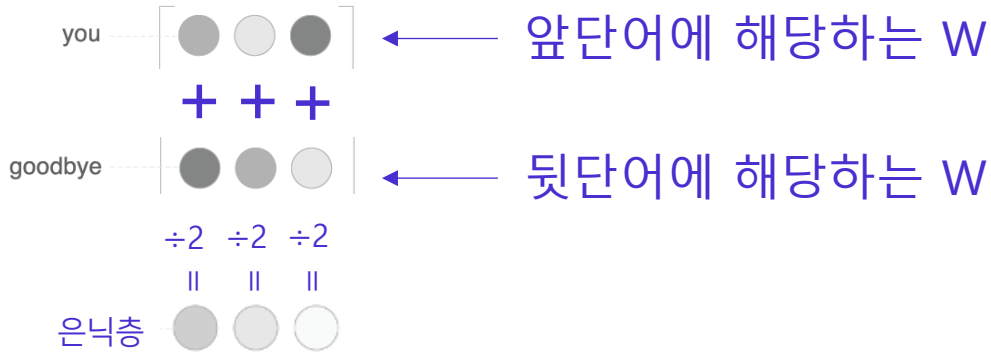


word2vec 속도 개선

딥러닝스터디

그림 3-9 CBOW 모델의 신경망 구조





원하는 한 줄 씩만 추출하면 되는데 1과 0을 꼭 다 곱해야할까??

4.1.1 Embedding 계층

- 단어와 W를 곱하는 과정은 특정 행을 추출하는 것 뿐이므로 MatMul 연산은 불필요하다!
- 가중치 매개변수로부터 '단어 ID에 해당하는 행'을 추출하는 계층이 필요
- 이 계층을 Embedding 계층이라고 한다!

자연어 처리에서
통계 기반 기법으로 얻은 단어 벡터 distributional representation
신경망 추론을 통한 단어 벡터 distributed representation

4.1.2 Embedding 계층 구현

- 구현은 너무 쉬움

```
W = np.arange(21).reshape(7,3)
W


array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17],
       [18, 19, 20]])
```

```
W[2] :  [6 7 8]
W[5] :  [15 16 17]
W[[1,0,3,0]] = W[idx] :
 [[ 3  4  5]
 [ 0  1  2]
 [ 9 10 11]
 [ 0  1  2]]
```

word2vec 속도 개선

4.1.2 Embedding 계층 구현

- forward 구현은 너무 쉬움

```
class Embedding:
    def __init__(self, W):
        self.params = [W]
        self.grads = [np.zeros_like(W)]
        self.idx = None

    def forward(self, idx):
        W, = self.params
        self.idx = idx
        out = W[idx]
        return out
```

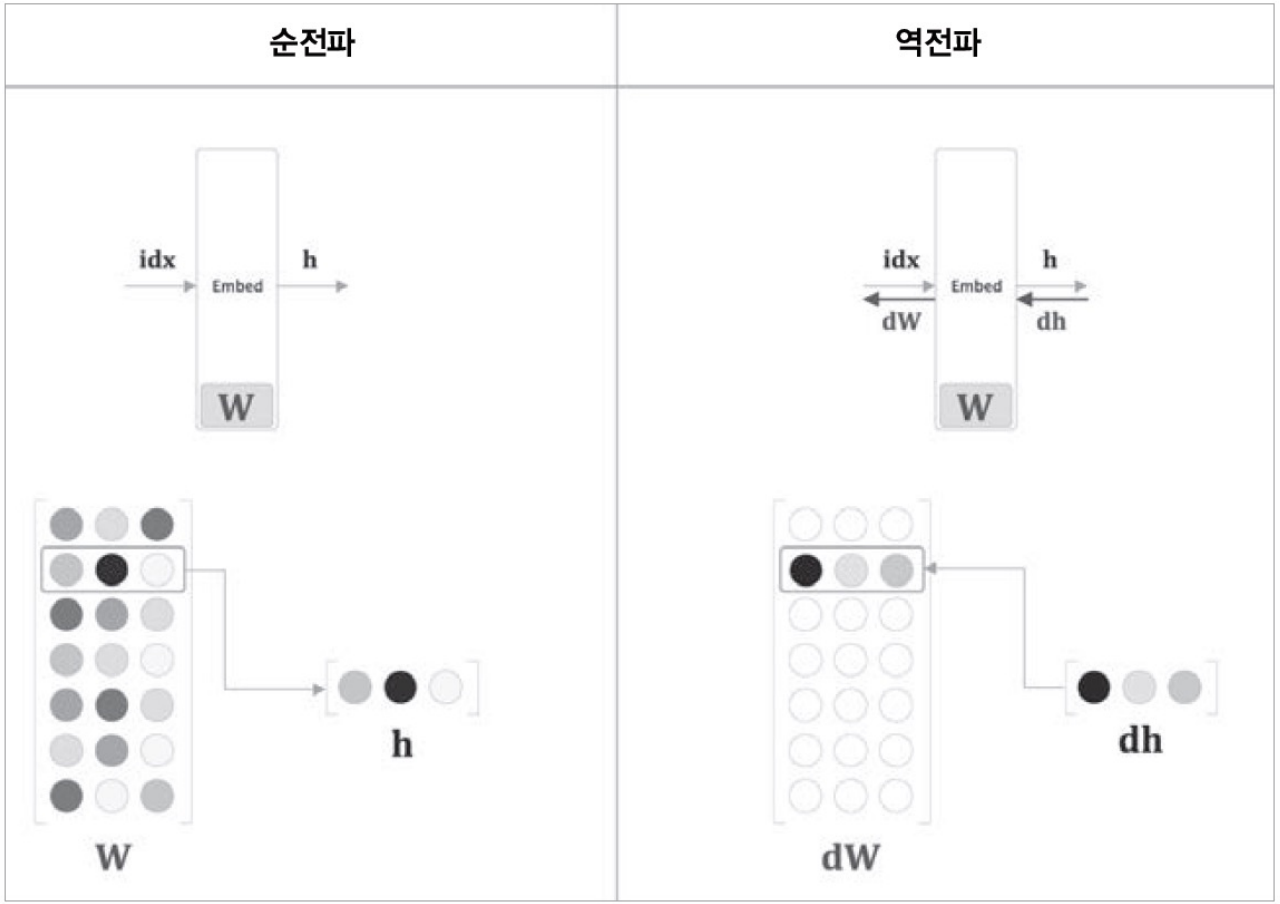
W에서 index를
골라주기만 하면 끝

word2vec 속도 개선

4.1.2 Embedding 계층 구현

- Backward 구현
  - 원래 위치를 찾아주면 된다.
  - 값을 바꾼 건 아니니 값은 그대로

그림 4-4 Embedding 계층의 forward와 backward 처리(Embedding 계층은 Embed로 표기)



4.1.2 Embedding 계층 구현

- Backward 구현
  - 원래 위치를 찾아주면 된다.
  - 값을 바꾼 건 아니니 값은 그대로

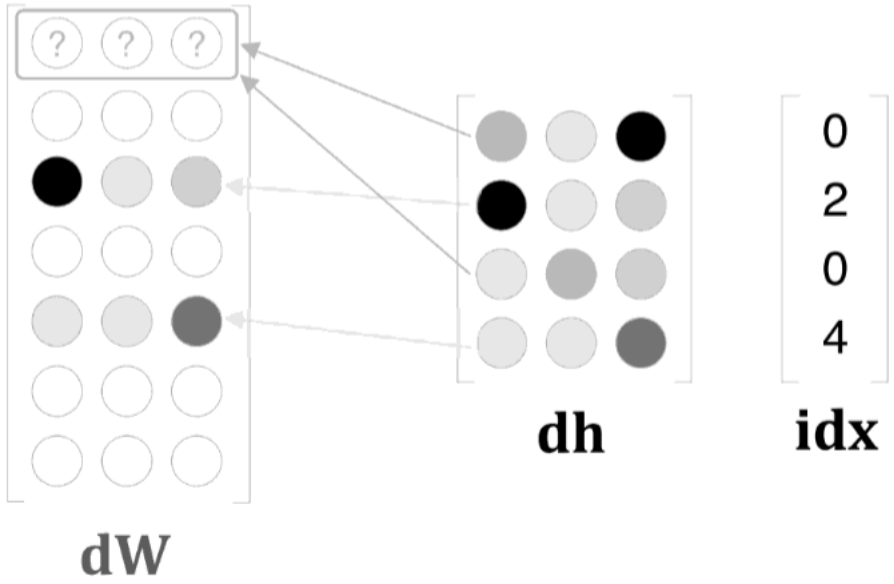
dW에서 index를
골라주기만 하면 끝

```
def backward(self, dout):
    dW, = self.grads
    dW[...] = 0
    dW[self.idx] = dout
    return None
```

4.1.2 Embedding 계층 구현

- Backward 구현

그림 4-5 idx 배열의 원소 중 값(행 번호)이 같은 원소가 있다면, dh를 해당 행에 할당할 때 문제가 생긴다.



4.1.2 Embedding 계층 구현

- Backward 구현

```
def backward(self, dout):
  dW, = self.grads
  dW[...] = 0
  dW[self.idx] = dout
  return None
```

```
def backward(self, dout):
  dW, = self.grads
  dW[...] = 0

  for i, word_id in enumerate(self.idx):
    dW[word_id] += dout[i]
  return None
```

중복되는 dh를 더해주는 형태로 변경

word2vec 속도 개선

4.2 word2vec 개선 -2

- 네거티브 샘플링(부정적 샘플링)
- 은닉층 이후의 처리 (행렬 곱과 Softmax 계층의 계산)에 적용
- 손실함수의 한 종류!
- Softmax 대신 사용하면 어휘가 아무리 많아져도 계산량을 낮은 수준에서 일정하게 억제할 수 있다.

word2vec 속도 개선

4.2.1 은닉층 이후 계산의 문제점

- 은닉층이 100
- 출력 뉴런 100만개
- forward & backward 모두에서 연산량이 많아짐 -> 각 100 x 100만
- softmax에서도 마찬가지로 연산량이 많아짐 -> exp 연산을 100만 번;;

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

word2vec 속도 개선

4.2.2 다중 분류에서 이진 분류로

- 다중 클래스 분류를 이진 분류로 근사하는 것!!
- 지금까지는 맥락이 주어질때 정답이 되는 단어의 확률을 높히도록 학습함.

- 다중 분류 방식 질문
- ‘맥락이 ‘you’랑 ‘goodbye’일 때, 타깃 단어는?’

- 이진 분류 방식 질문
- ‘맥락이 ‘you’와 ‘goodbye’일 때, 타깃 단어는 ‘say’입니까?’

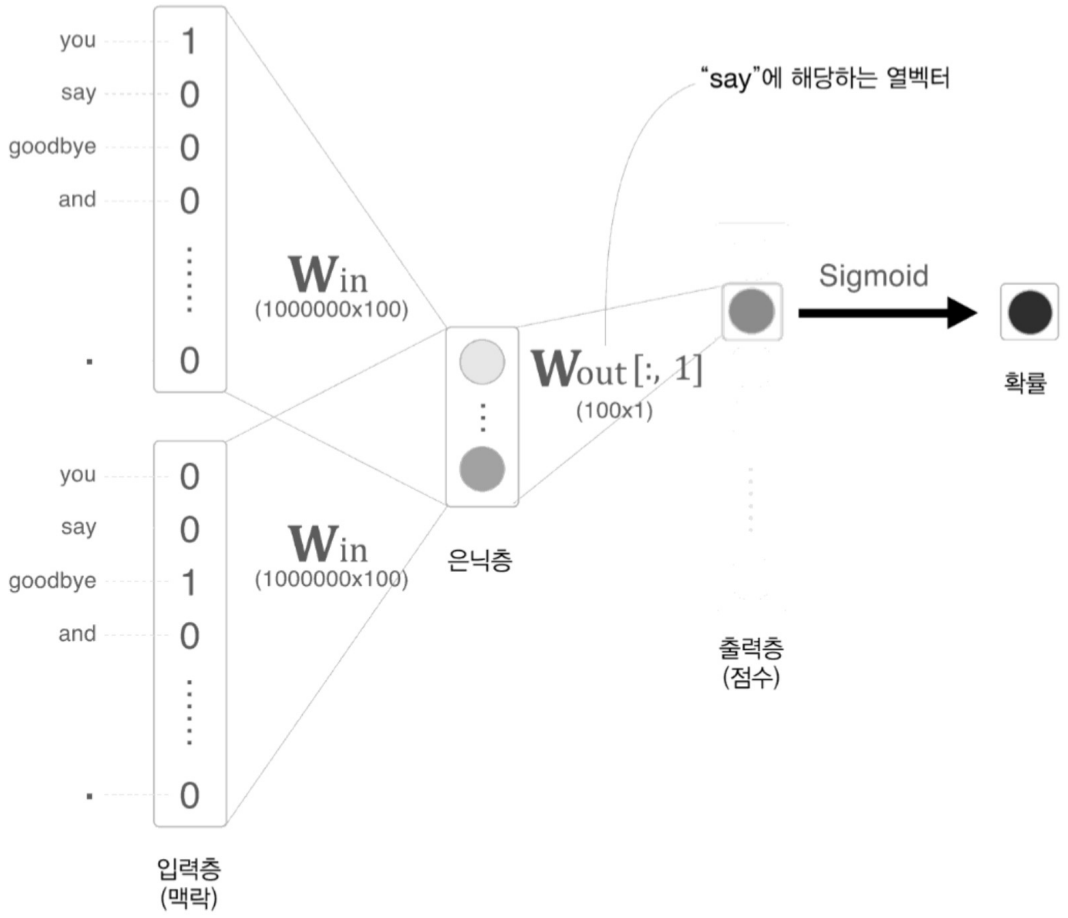
word2vec 속도 개선

딥러닝스터디

4.2.2 다중 분류에서 이진 분류로

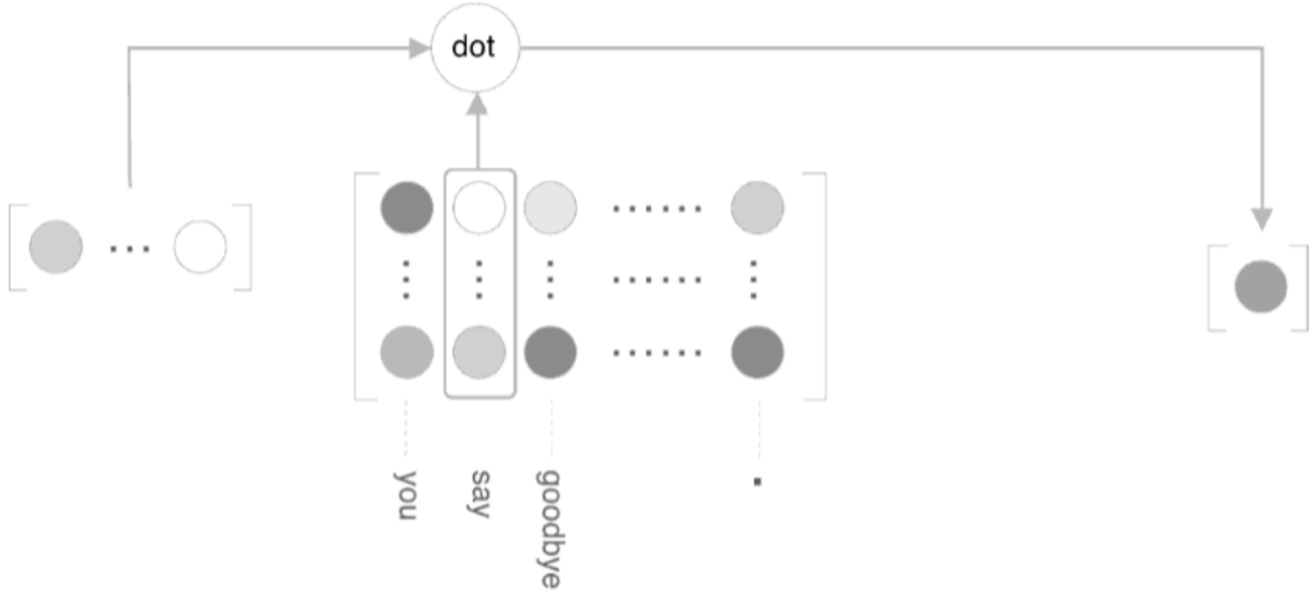
- 출력층의 뉴런이 하나면 된다.
- 추출된 벡터와 은닉층 뉴런과의 내적을 계산하면 끝

그림 4-7 타깃 단어만의 점수를 구하는 신경망



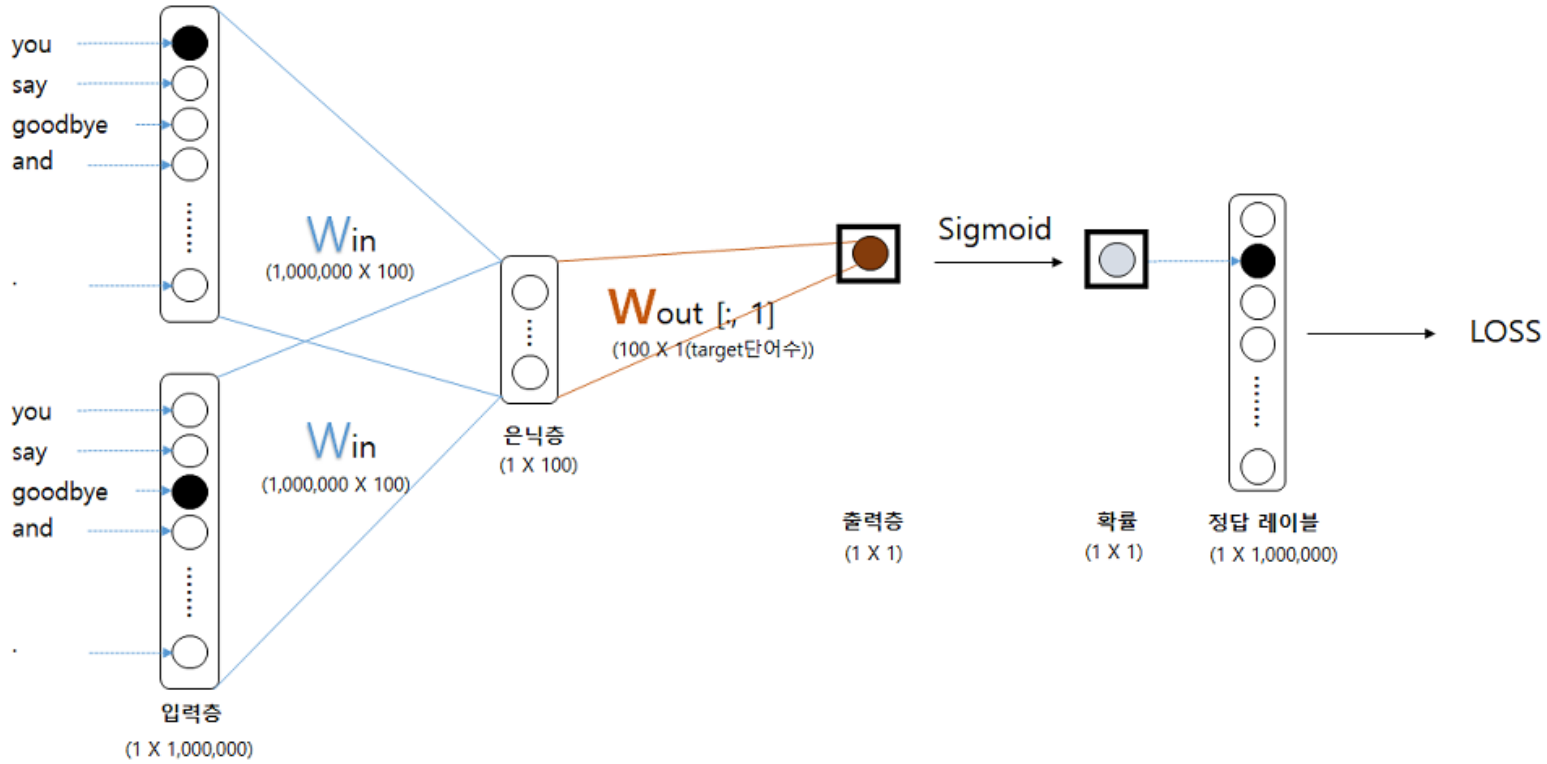
4.2.2 다중 분류에서 이진 분류로

그림 4-8 "say"에 해당하는 열벡터와 은닉층 뉴런의 내적을 계산한다('dot' 노드가 내적을 계산함).



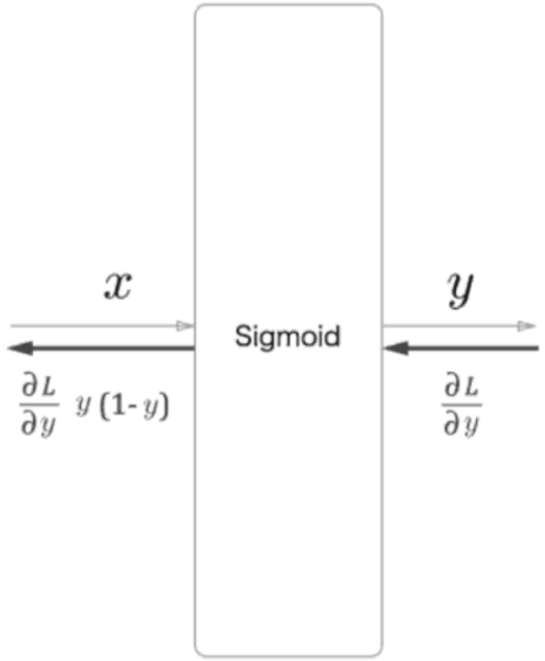
	$\mathbf{h}$	$\mathbf{W}_{\text{out}}$	$\mathbf{s}$
형상 :	(1 × 100)	(100 × 1000000)	(1 × 1)

4.2.3 시그모이드 함수와 교차 엔트로피 오차

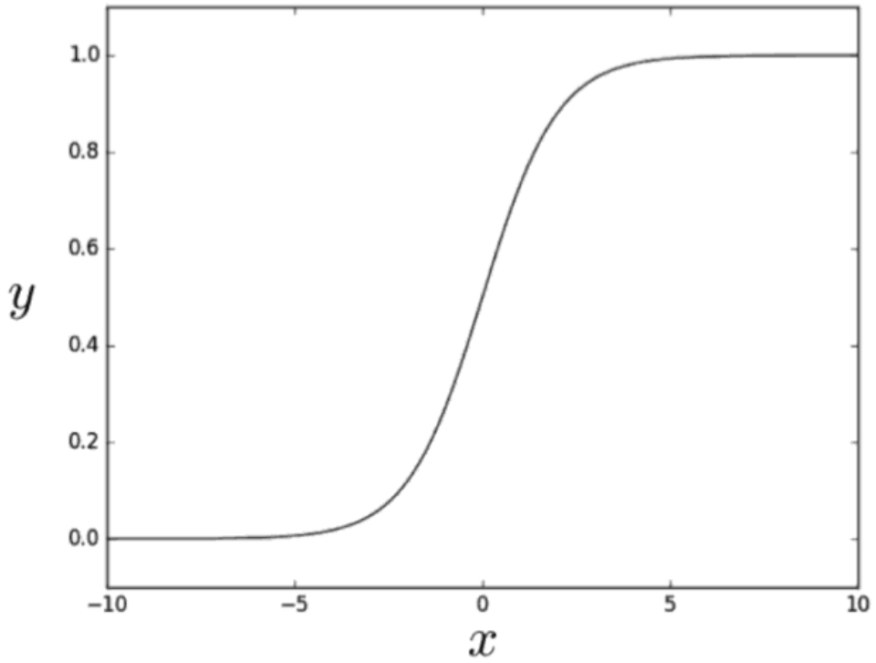


4.2.3 시그모이드 함수와 교차 엔트로피 오차

- 시그모이드 함수



0과 1사이의 실수로 변환



0~1이기 때문에 '확률'로 볼 수 있다.

4.2.3 시그모이드 함수와 교차 엔트로피 오차

- 교차 엔트로피 오차

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

- y는 시그모이드의 출력
- t는 정답 레이블
- t가 정답이면 1, 오답이면 0
- t가 1이면 $-\log y$
- t가 0이면 $\log(1-y)$

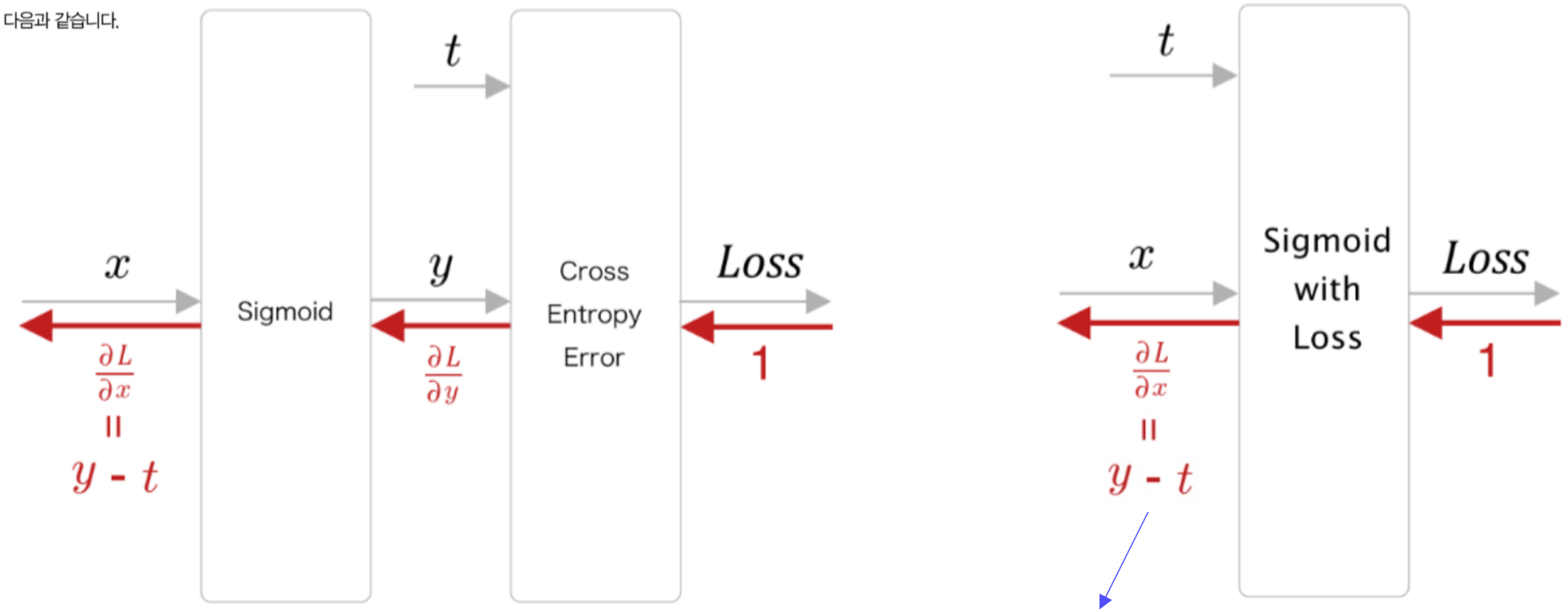
4.2.3 시그모이드 함수와 교차 엔트로피 오차

그림 4-10 Sigmoid 계층과 Cross Entropy Error 계층의 계산 그래프(오른쪽은 Sigmoid with Loss 계층으로 통합한 모습)

* 옮긴이_ $\frac{\partial L}{\partial x}$ 이 $y-t$로 유도되는 과정은 다음과 같습니다.

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

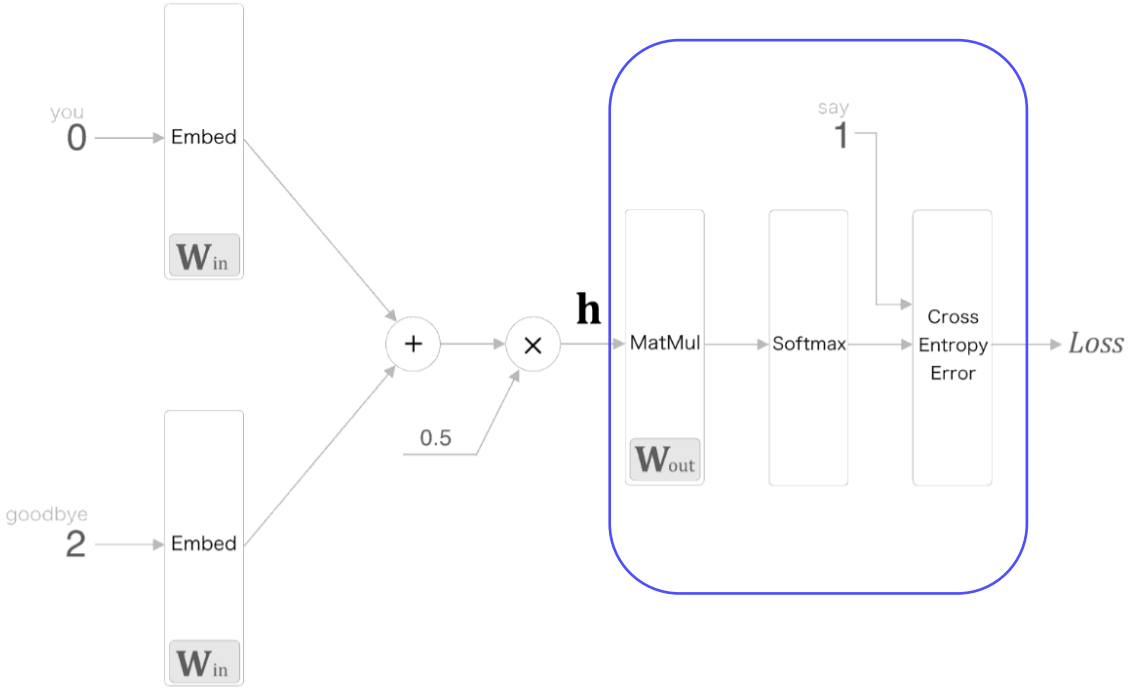
따라서 $\frac{\partial L}{\partial x}=y-t$가 됩니다.



오차가 크면 크게 학습하고 작으면 작게 학습한다.

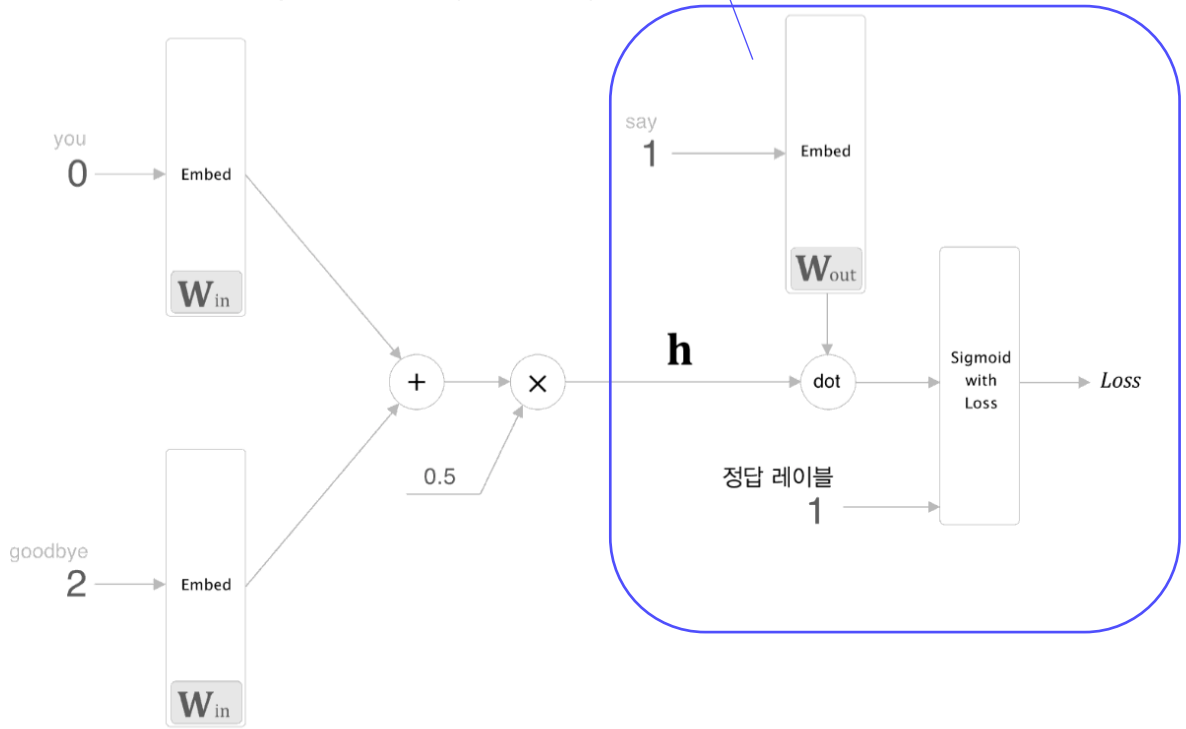
4.2.4 다중 분류에서 이진 분류로 (구현)

그림 4-11 다중 분류를 수행하는 CBOW 모델의 전체 그림(Embedding 계층은 Embed로 표기)



say에 해당하는 것만 고름

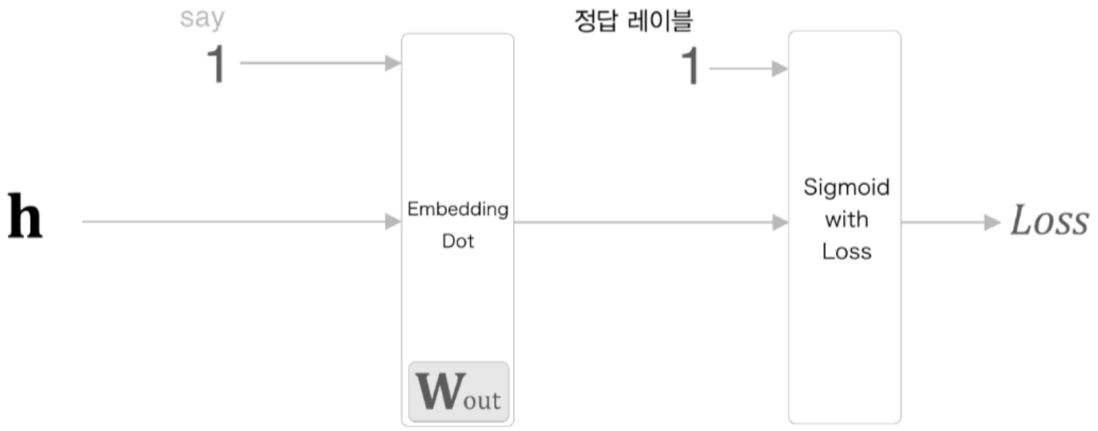
그림 4-12 이진 분류를 수행하는 word2vec(CBOW 모델)의 전체 그림



4.2.4 다중 분류에서 이진 분류로 (구현)

도식 단순화

그림 4-13 [그림 4-12]의 은닉층 이후 처리(Embedding Dot 계층을 사용하여 Embedding 계층과 내적 계산을 한 번에 수행)




```
class EmbeddingDot:
    def __init__(self, W):
        self.embed = Embedding(W)
        self.params = self.embed.params
        self.grads = self.embed.grads
        self.cache = None

    def forward(self, h, idx):
        target_W = self.embed.forward(idx)
        out = np.sum(target_W * h, axis=1)

        self.cache = (h, target_W)
        return out

    def backward(self, dout):
        h, target_W = self.cache
        dout = dout.reshape(dout.shape[0], 1)

        dtarget_W = dout * h
        self.embed.backward(dtarget_W)
        dh = dout * target_W
        return dh
```

→ say의 idx를 W로 만듦

→ W, 입력h dot연산

→ 순전파 결과 저장

4.2.4 다중 분류에서 이진 분류로 (구현)

그림 4-14 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] [3 4 5] [6 7 8] [9 10 11] [12 13 14] [15 16 17] [18 19 20]]	[0 3 1]	[[0 1 2] [9 10 11] [3 4 5]]	[[0 1 2] [3 4 5] [6 7 8]]	[[0 1 4] [27 40 55] [18 28 40]]	[5 122 86]

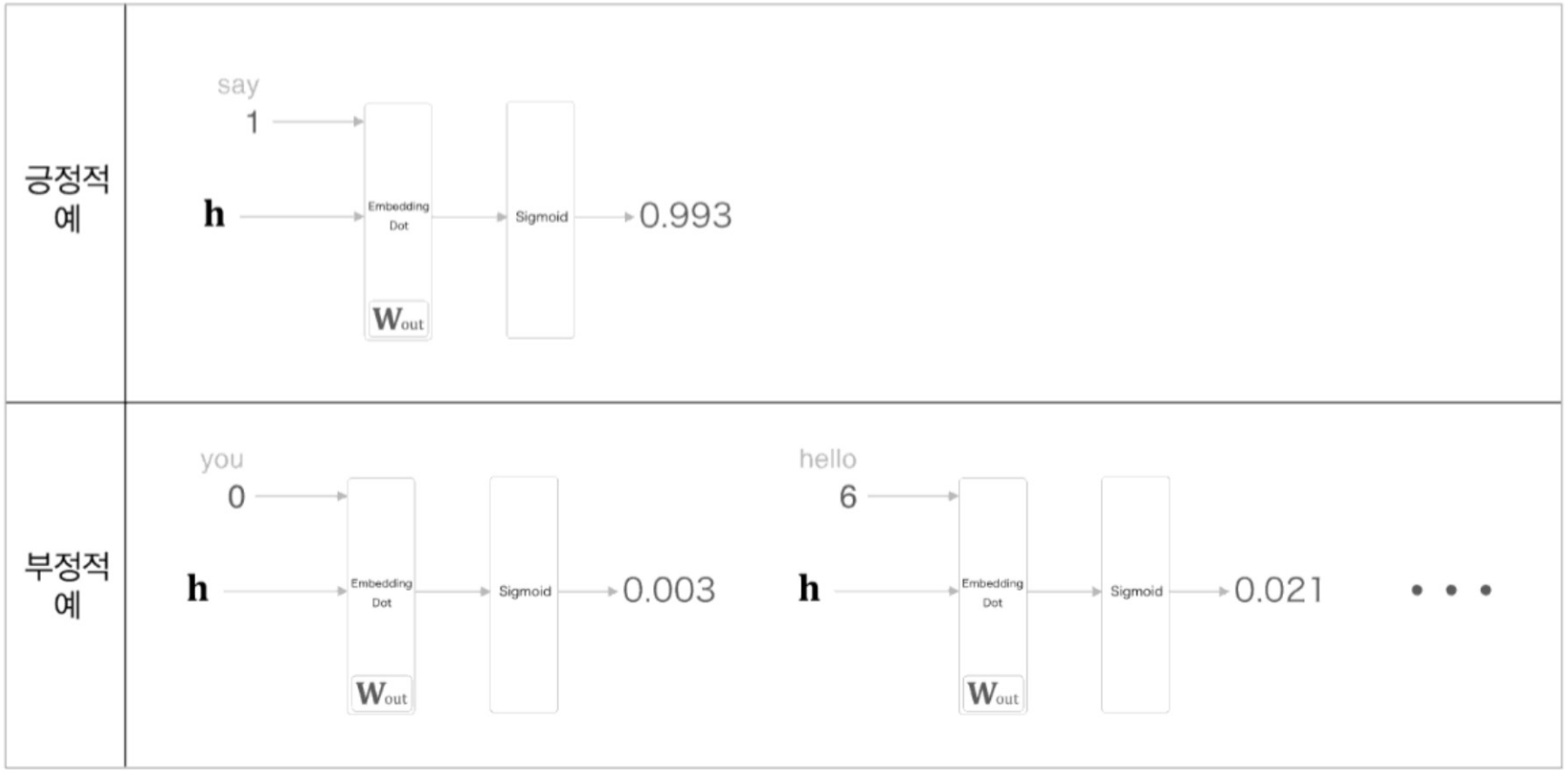
word2vec 속도 개선

4.2.5 네거티브 샘플링

- 긍정적인 예에 대해서만 학습했기 때문에 부정적인 예를 입력하면 어떤 결과가 나오는지 확인해야함.
- 이대로 한다면 정답일 때(say일때)만 학습하고 나머지는 학습하지 못하게된다.
- 부정적인 예를 맞췄을 때도 loss가 낮아져야한다!

4.2.5 네거티브 샘플링

그림 4-16 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.



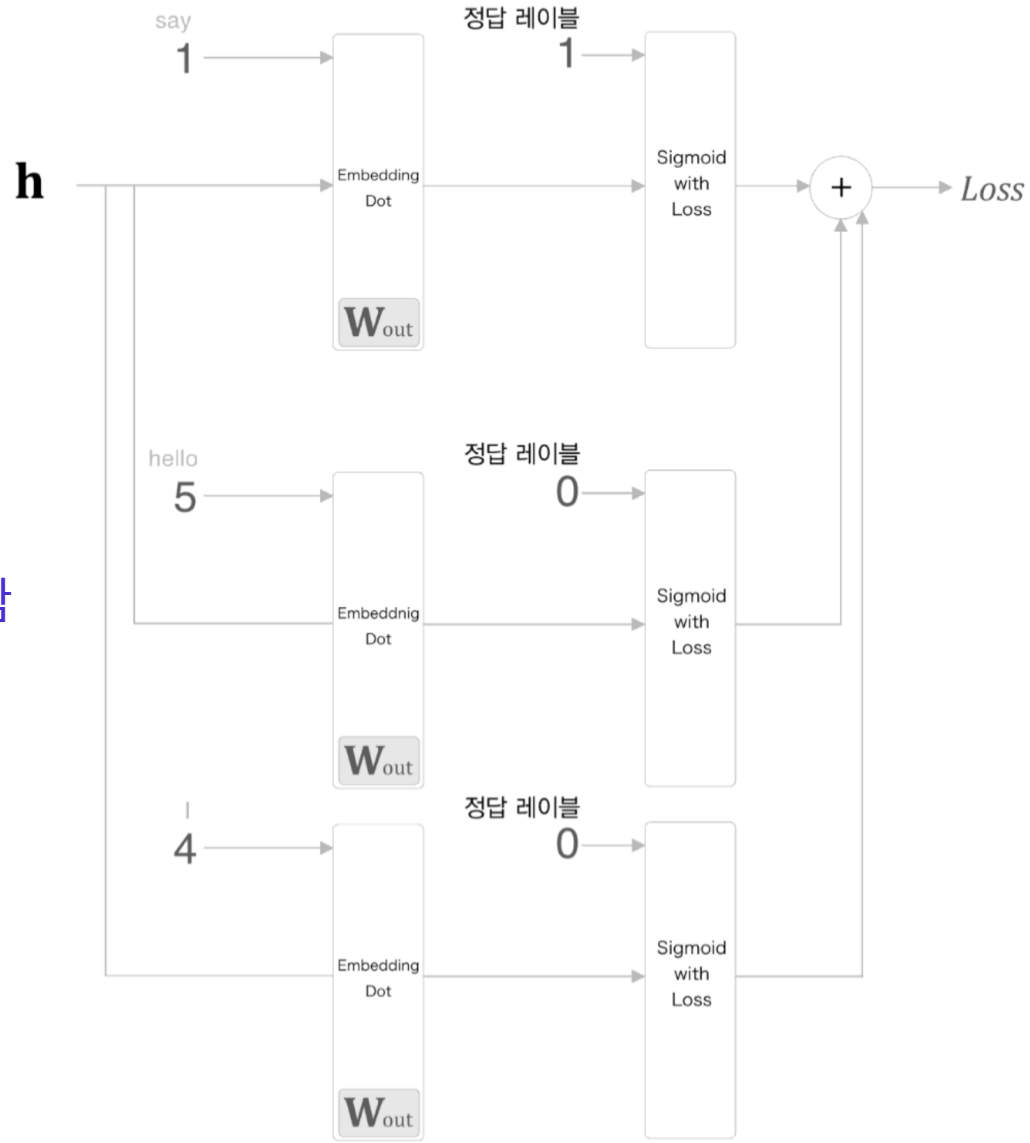
4.2.5 네거티브 샘플링

- 이렇게 했을때의 문제점
- 정답에 대한 학습은 1단어 오답에 대한 학습은 무수히 많다.
- 전부 오답으로만 예측해도 높은 정답률을 얻을 수 있게 될 수 있음.
- 이걸 해결하기 위해서 부정적인 예를 몇 개를 랜덤하게 선택(샘플링)해서 사용
- 이것 '네거티브 샘플링'이라고 한다.

word2vec 속도 개선

4.2.5 네거티브 샘플링

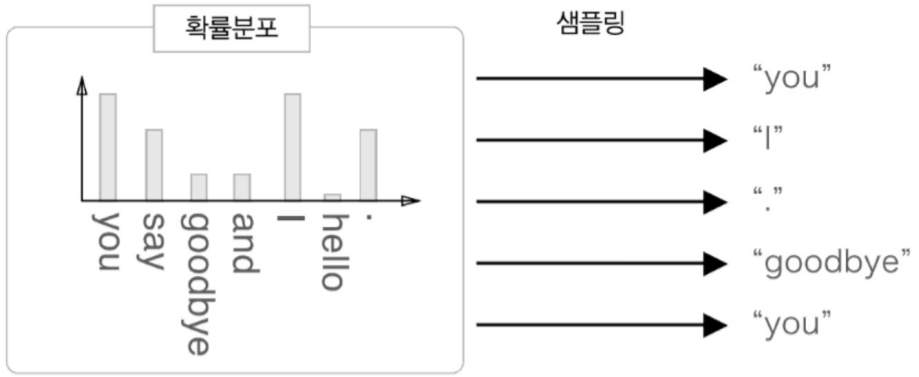
정답인 경우 1개와 오답인 경우 몇개를 학습함



4.2.6 네거티브 샘플링의 샘플링 기법

- 부정적 예를 샘플링하는 방식(단순 무작위가 아님)
- 말뭉치 통계 데이터를 기초로 샘플링하는 방식
- 말뭉치에서 자주 등장하는 단어를 많이 추출하고 드물게 등장하는 단어를 적게 추출함.
- 이를 위해 단어 출현 횟수를 구해서 확률분포로 만들어야한다.

그림 4-18 확률분포에 따라 샘플링을 여러 번 수행한다.



4.2.7 네거티브 샘플링 구현

```
class NegativeSamplingLoss:
    def __init__(self, W, corpus, power=0.75, sample_size=5):
        self.sample_size = sample_size
        self.sampler = UnigramSampler(corpus, power, sample_size)
        self.loss_layers = [SigmoidWithLoss() for _ in range(sample_size + 1)]
        self.embed_dot_layers = [EmbeddingDot(W) for _ in range(sample_size + 1)]
        self.params, self.grads = [], []
        for layer in self.embed_dot_layers:
            self.params += layer.params
            self.grads += layer.grads
```

4.2.7 네거티브 샘플링 구현

```
def forward(self, h, target):
    batch_size = target.shape[0]
    negative_sample = self.sampler.get_negative_sample(target)

    # 긍정적 예 순전파
    score = self.embed_dot_layers[0].forward(h, target)
    correct_label = np.ones(batch_size, dtype=np.int32)
    loss = self.loss_layers[0].forward(score, correct_label)

    # 부정적 예 순전파
    negative_label = np.zeros(batch_size, dtype=np.int32)
    for i in range(self.sample_size):
        negative_target = negative_sample[:, i]
        score = self.embed_dot_layers[1 + i].forward(h, negative_target)
        loss += self.loss_layers[1 + i].forward(score, negative_label)

    return loss
```

4.2.7 네거티브 샘플링 구현

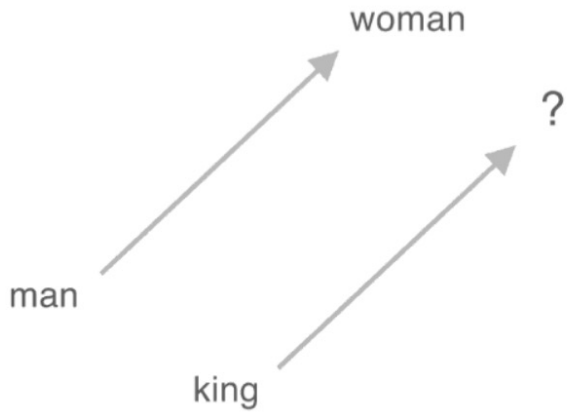
```
def backward(self, dout=1):
    dh = 0
    for l0, l1 in zip(self.loss_layers, self.embed_dot_layers):
        dscore = l0.backward(dout)
        dh += l1.backward(dscore)

    return dh
```

4.3 개선판 word2vec 전체 학습

```
[query] you
 we: 0.610597074032
 someone: 0.591710150242
 i: 0.554366409779
 something: 0.490028560162
 anyone: 0.473472118378

[query] year
 month: 0.718261063099
 week: 0.652263045311
 spring: 0.62699586153
 summer: 0.625829637051
 decade: 0.603022158146

[query] car
 luxury: 0.497202396393
 arabia: 0.478033810854
 auto: 0.471043765545
 disk-drive: 0.450782179832
 travel: 0.40902107954

[query] toyota
 ford: 0.550541639328
 instrumentation: 0.510020911694
 mazda: 0.49361255765
 bethlehem: 0.474817842245
 nissan: 0.474622786045
```

"king − man + woman = queen"

그림 4-20 "man : woman = king : ?" 유추 문제 풀기(단어 벡터 공간에서 각 단어의 관계성)



4.3 개선판 word2vec 전체 학습

```
[analogy] king:man = queen:?
 woman: 5.161407947540283
 veto: 4.928170680999756
 ounce: 4.689689636230469
 earthquake: 4.633471488952637
 successor: 4.6089653968811035

[analogy] take:took = go:?
 went: 4.548568248748779
 points: 4.248863220214844
 began: 4.090967178344727
 comes: 3.9805688858032227
 oct.: 3.9044761657714844
```

word2vec 속도 개선

4.4 word2vec 남은 주제

4.4.1 word2vec을 사용한 애플리케이션의 예

- transfer learning (전이 학습)
- 한 분야에서 배운 지식을 다른 분야에 활용함
- 처음부터 word2vec의 단어 분산 표현을 학습하지는 않는다.
- 대신 초대형 말뭉치(위키백과나 구글 뉴스) 같은 데이터로 학습한 후 분산 표현만을 자신의 작업에 사용함.

그림 4-21 단어의 분산 표현을 이용한 시스템의 처리 흐름

word2vec 속도 개선

딥러닝스터디

4.4.2 단어 벡터 평가 방법

- 'king : queen = man : ?'와 같은 유추 문제들을 출제하고 정답률 측정
- 이걸 평가하면 단어의 의미나 문법적인 문제를 이해하는지를 볼 수 있음

그림 4-23 유추 문제에 의한 단어 벡터의 평가 결과(논문 [27]에서 발췌)

모델	차수	말뭉치 크기	의미(semantics)	구문(syntax)	종합
CBOW	300	16억	16.1	52.6	36.1
skip_gram	300	10억	61	61	61
CBOW	300	60억	63.6	67.4	65.7
skip_gram	300	60억	73.0	66.0	69.1
CBOW	1000	60억	57.3	68.9	63.7
skip_gram	1000	60억	66.1	65.1	65.6

1)모델에 따라 정확도가 다름
2)말뭉치는 클수록 결과가 좋음
3)단어 벡터 차원 수는 적당한 크기가 좋음

4.5 정리

- Embedding 계층은 단어의 분산 표현을 담고 있으며, 순전파 시 지정한 단어 ID의 벡터를 추출한다.
- word2vec은 어휘 수의 증가에 비례하여 계산량도 증가하므로, 근사치로 계산하는 빠른 기법을 사용하면 좋다.
- 네거티브 샘플링은 부정적 예를 몇 개 샘플링하는 기법으로, 이를 이용하면 다중 분류를 이진 분류처럼 취급할 수 있다.
- word2vec으로 얻은 단어의 분산 표현에는 단어의 의미가 녹아들어 있으며, 비슷한 맥락에서 사용되는 단어는 단어 벡터 공간에서 가까이 위치한다.
- word2vec의 단어의 분산 표현을 이용하면 유추 문제를 벡터의 덧셈과 뺄셈으로 풀 수 있게 된다.
- word2vec은 전이 학습 측면에서 특히 중요하며, 그 단어의 분산 표현은 다양한 자연어 처리 작업에 이용할 수 있다.

word2vec은 자연어처리 분야뿐만 아니라 음성, 이미지, 동영상 등에도 응용되고 있다!

word2vec을 제대로 이해했다면, 그 지식은 다양한 분야에서 큰 도움이 될 것입니다.

다음 주는 순환 신경망 RNN

참고 논문 : Distributed Representations of Words and Phrases and their Compositionality