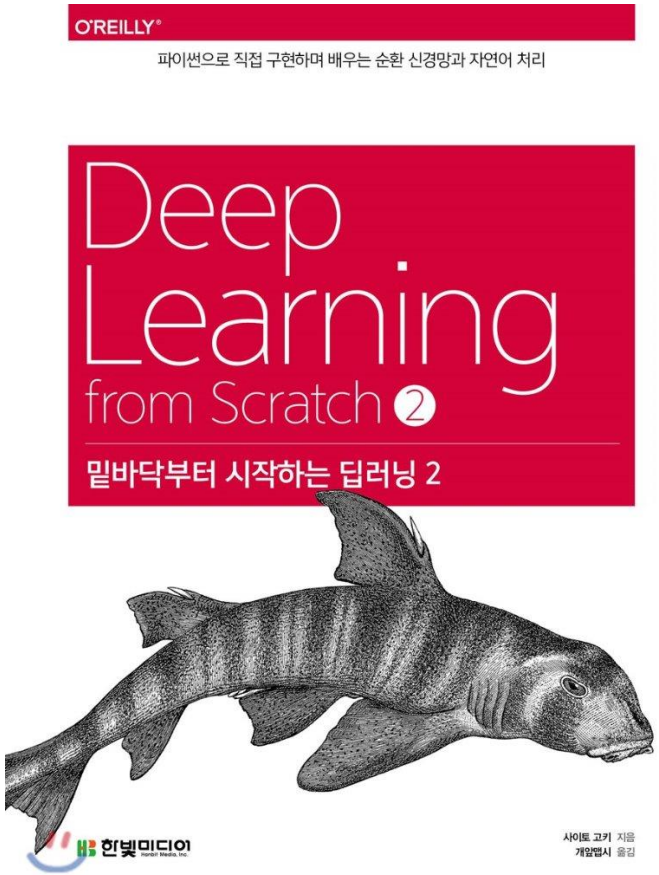
밑바닥부터 시작하는 딥러닝 ②

파이썬으로 직접 구현하며 배우는 순환신경망과 자연어처리



4장 word2vec 속도 개선

CHAPTER 4

Word2vec 속도 개선

모든 것을 알려고 애쓰지 마라.

그러다 보면 아무것도 기억할 수 없다.

_ 데모크리토스(고대 그리스 철학자)

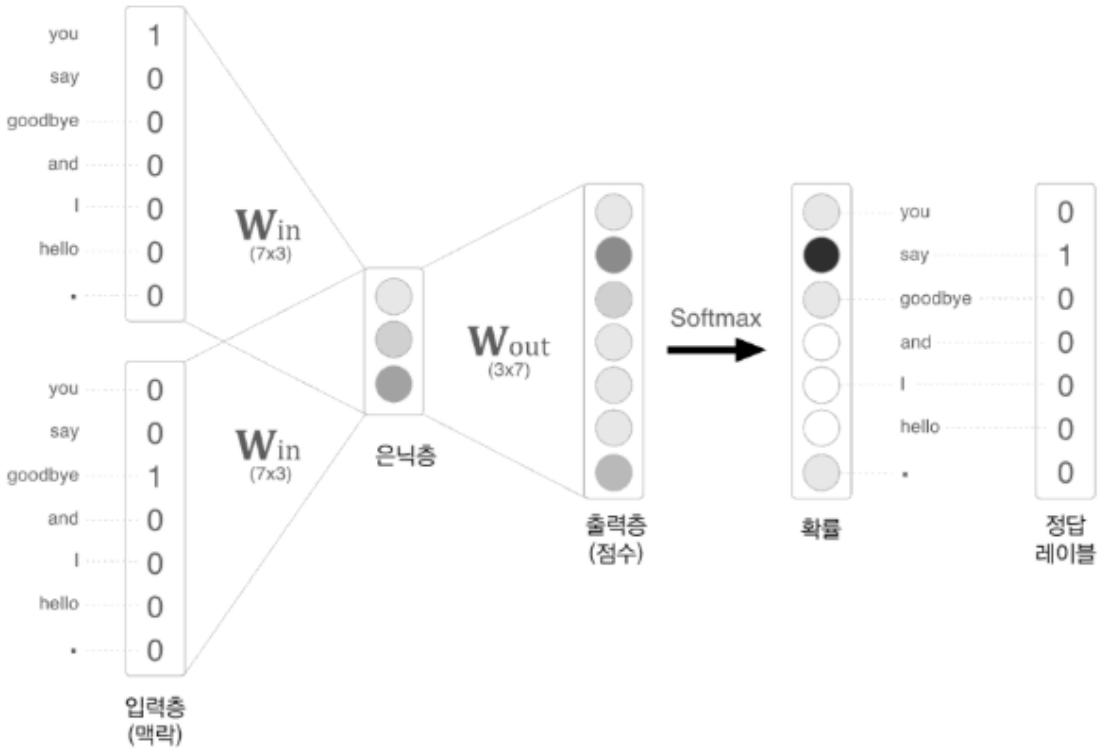
앞의장 3장에서 word2vec의 구조를 배우고 CBOW 모델을 구현해봤다.
CBOW 모델은 단순한 2층 신경망이라서 간단하게 구현 가능했지만, 그 구현에는 몇 가지 문제가 있었다.
가장 큰 문제는 말뭉치에 포함된 어휘 수가 많아지면 계산량도 커진다는 점이다.
실제로 어휘 수가 어느 정도를 넘어서면 앞 장의 CBOW 모델은 계산 시간이 너무 오래 걸린다.

그래서 이번 장의 목표는 word2vec의 속도 개선이다.
구체적으로는 앞 장의 단순한 word2vec에 두 가지 개선을 추가할 것이다.
첫 번째 개선으로는 Embedding이라는 새로운 계층을 도입하고,
두 번째 개선으로 네거티브 샘플링이라는 새로운 손실 함수를 도입한다.

word2vec 개선 ①

- **복습**

그림 4-1 앞 장에서 구현한 CBOW 모델



[그림 4-1]과 같이 앞 장의 CBOW 모델은 단어 2개를 맥락으로 사용해, 이를 바탕으로 하나의 단어 (타깃)를 추측한다.

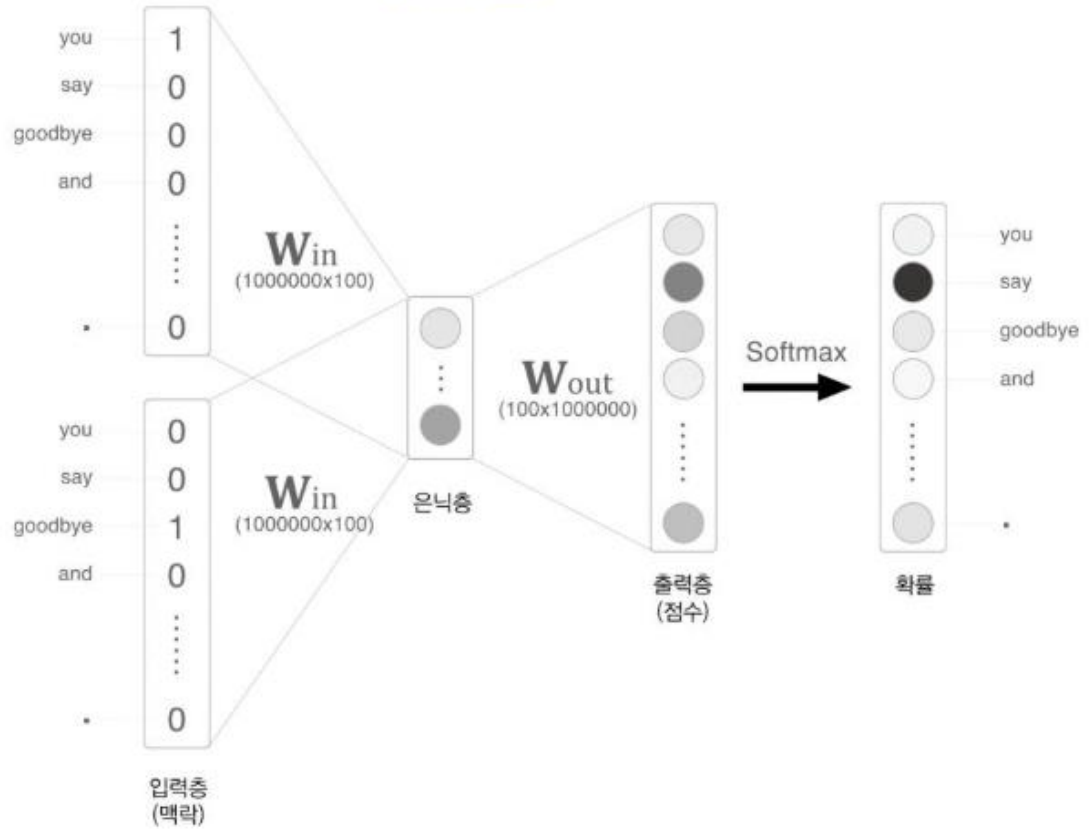
이때 입력 측 가중치 (W_{in})와의 행렬 곱으로 은닉층이 계산되고, 다시 출력 측 가중치(W_{out})와의 행렬 곱으로 각 단어의 점수를 구한다.

그리고 이 점수에 소프트맥스 함수를 적용해 각 단어의 출현 확률을 얻고, 이 확률을 정답 레이블과 비교하여 손실을 구한다.

word2vec 개선 ①

▪ 복습

그림 4-2 어휘가 100만 개일 때를 가정한 CBOW 모델



앞 장에서 구현한 CBOW 모델에서 어휘가 100만 개,
은닉층의 뉴런이 100개인 CBOW 모델을 생각해보자.

[그림 4-2]를 보면 입력층과 출력층에는 각 100만 개의 뉴런이 존재한다.
이 수많은 뉴련 대문에 중간 계산에 많은 시간이 소요된다.
정확히는 다음의 두 계산이 병목이 된다.

- **입력층의 원핫 표현과 가중치 행렬 W_{in}의 곱 계산 (4.1절)**
- **은닉층과 가중치 행렬 W_{out}의 곱 및 Softmax 계층의 계산 (4.2절)**

첫 번째는 원핫 표현과 관련한 문제이다.
예컨대 어휘가 100만 개라면 그 원핫 표현 하나만 해도 원소 수가
100만개인 벡터가 된다. 이는 상당한 메모리를 차지하게 된다.
이 원핫 벡터와 가중치 행렬 W_{in}을 곱해야 하는데, 이것만으로도
계산 자원을 상당히 사용하게 된다.

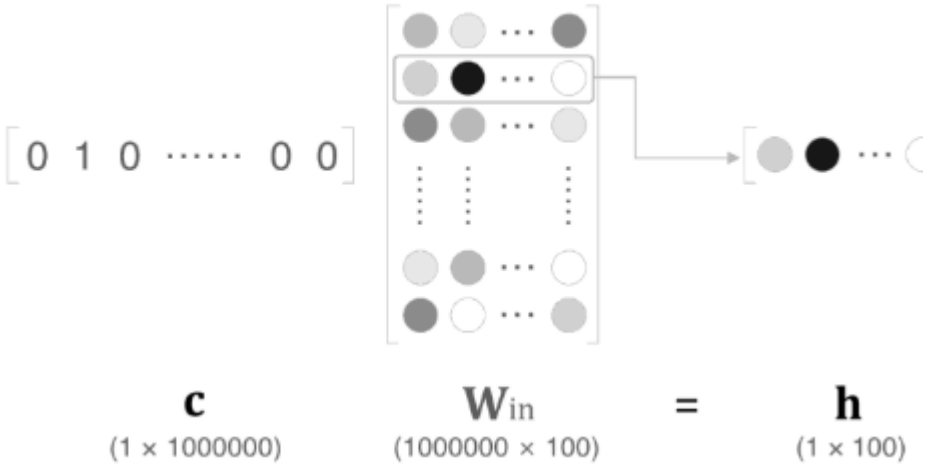
두 번째 문제는 은닉층 이후의 계산이다.
우선 은닉층과 가중치 행렬 W_{out}의 곱만 해도 계산량이 상당하다.
그리고 Softmax 계층에서도 다루는 어휘가 많아짐에 따라 계산량이
증가하는 문제가 있다.

word2vec 개선 ①

▪ Embedding 계층

앞 장의 word2vec 구현에서는 단어를 원핫 표현으로 바꾸고 Matmul 계층에서 가중치 행렬을 곱했다.
그럼 여기서 어휘 수가 100만개인 경우를 상상해보자. 이때 은닉층 뉴런이 100개라면, Matmul 계층의 행렬 곱은
[그림 4-3]처럼 된다.

그림 4-3 맥락(원핫 표현)과 MatMul 계층의 가중치를 곱한다.



이런 거대한 벡터와 가중치 행렬을 곱해야 하는 것이나, 결과적으로 수행하는 일은 단지 **행렬의 특정 행을 추출하는 것뿐**이다.
따라서 원핫 표현으로의 변환과 Matmul 계층의 행렬 곱 계산은 사실 필요가 없는 것이다.

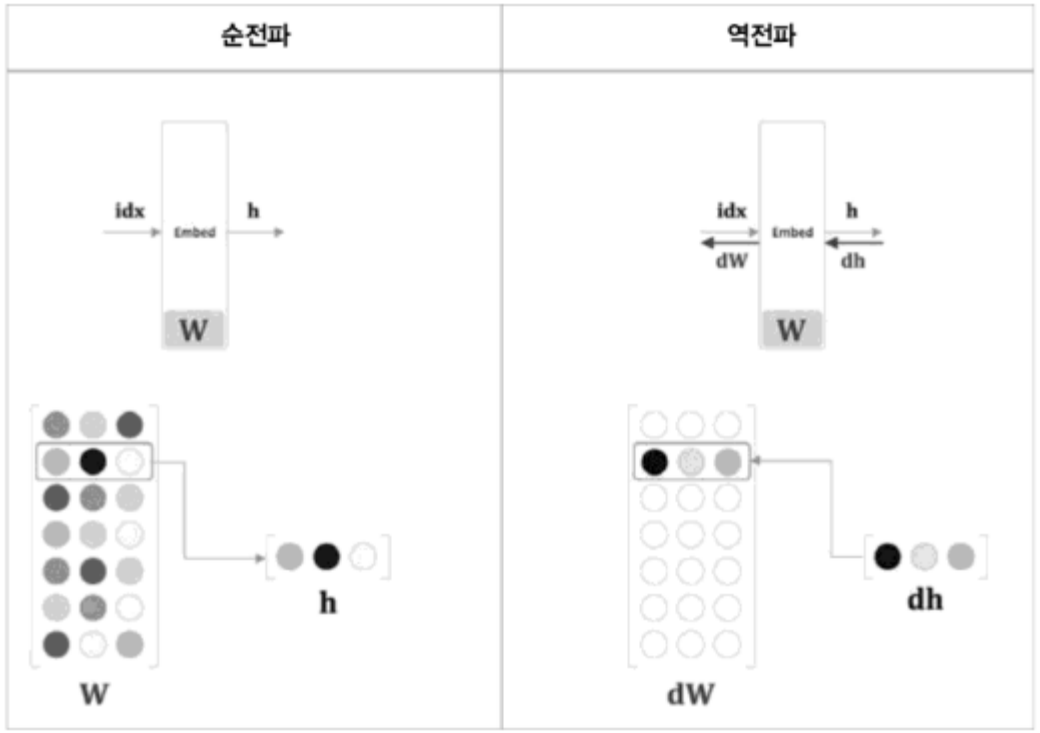
NOTE_ 자연어 처리 분야에서 단어의 밀집벡터 표현을 **단어 임베딩** 혹은 단어의 **분산 표현**이라 한다.

word2vec 개선 ①

▪ Embedding 계층 구현

행렬에서 특정 행만 추출하면 되므로 다음과 같이 간단하게 구현 가능하다.

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




```
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
```

역전파 같은 경우는 Embedding 계층의 순전파는 가중치 W의 특정행을 추출할 뿐이므로 역전파에서는 앞 층으로부터 전해진 기울기를 다음 층으로 그대로 흘려주면 된다.

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

        for i, word_id in enumerate(self.idx):
            dW[word_id] += dout[i]
        # 혹은
        # np.add.at(dW, self.idx, dout)

        return None
```