Lecture 9: 워드 임베딩(Word Embedding)

희소 표현 (Sparse Representation)

희소 표현이란?

- `원-핫 인코딩` 을 통해서 나온 `원-핫 벡터` 들은 표현하고자 하는 단어의 인덱스값만 1이고, 나머지 인덱스에는 전부 0으로 표현되는 벡터 표현 방법
- 이렇게 벡터 또는 행렬(matrix)의 값이 대부분이 0으로 표현되는 방법을 `희소 표현(sparse representation)` 이라고 한다.
- `원-핫 벡터` 는 `희소 벡터(sparse vector)` 이다.

희소 표현의 문제점

- 희소 벡터의 문제점은 단어의 개수가 늘어나면 벡터의 차원이 한없이 커진다는 점
- 원-핫 벡터로 표현할 때는 갖고 있는 코퍼스에 단어가 10,000개였다면 벡터의 차원은 10,000
- 그 중에서 단어의 인덱스에 해당되는 부분만 1이고 나머지는 0
- 단어 집합이 클수록 고차원의 벡터가 된다.
- 단어가 10,000개 있고 강아지란 단어의 인덱스가 5였다면 원-핫 벡터는 다음과 같이 표현

> ex) 강아지 = `[ 0 0 0 0 1 0 0 0 0 ... 0]` # 이 때 1 뒤의 9의 수는 9995개

- 희소 표현의 일종인 DTM의 경우에도 특정 문서에 여러 단어가 다수 등장하였으나, 다른 많은 문서에서는 해당 특정 문서에 등장했던 단어들이 전부 등장하지 않는다면 같은 문제 발생
- 원-핫 벡터는 단어의 의미를 담지 못한다.

밀집 표현 (Dense Representation)

- 희소 표현과 반대되는 표현인 `밀집 표현(dense representation)`이 있다.
- 밀집 표현은 벡터의 차원을 단어 집합의 크기로 상정하지 않는다.
- 사용자가 설정한 값으로 모든 단어의 벡터 표현의 차원을 맞춘다.
- 이 과정에서 더 이상 0과 1만 가진 값이 아니라 실수값을 가지게 된다.

> ex) 강아지 = `[ 0 0 0 0 1 0 0 0 0 ... 0]` # 벡터의 차원은 10,000

- 밀집 표현을 사용하고, 사용자가 밀집 표현의 차원을 128로 설정한다면, 모든 단어의 벡터 표현의 차원은 128로 바뀌면서 모든 값이 실수가 된다.

> ex) 강아지 = `[0.2 1.8 1.1 -2.1 1.1 2.8 ...]` # 벡터의 차원은 128

- 이 경우 벡터의 차원이 조밀해졌다고 하여 밀집 벡터(dense vector)라고 한다.

워드 임베딩 (Word Embedding)

- 단어를 밀집 벡터(dense vector)의 형태로 표현하는 방법
- 밀집 벡터를 워드 임베딩 과정을 통해 나온 결과라고 하여 `임베딩 벡터(embedding vector)` 라고도 한다.

워드 임베딩 방법론

- LSA, Word2Vec, FastText, Glove 등이 있다.

원-핫 벡터 vs 임베딩 벡터

	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

Word2Vec

- 원-핫 벡터는 단어 간 유사도를 계산할 수 없는 단점이 있다.
- 그래서 단어 간 유사도를 반영할 수 있도록 단어의 의미를 벡터화 할 수 있는 방법이 필요하다.
- 그리고 이를 위해서 사용되는 대표적인 방법이 `워드투벡터(Word2Vec)`이다.
- 예를 들어 아래의 식에서 좌변을 집어 넣으면, 우변의 답들이 나온다.

> 고양이 + 애교 = 강아지
> 한국 - 서울 + 도쿄 = 일본
> 박찬호 - 야구 + 축구 = 호나우두

- 단어가 가지고 있는 어떤 의미들을 가지고 연산을 하고 있는 것처럼 보인다.
- 이런 연산이 가능한 이유는 각 단어 벡터가 단어 간 유사도를 반영한 값을 가지기 때문이다.

분산 표현 (Distributed Representation)

- 희소 표현(sparse representation) 방법은 각 단어간 유사성을 표현할 수 없다는 단점이 있다.
- 이를 위한 대안으로 단어의 `의미` 를 다차원 공간에 벡터화하는 방법을 찾게 된다.
- 이러한 표현 방법을 `분산 표현(distributed representation)`이라고 한다.
- 그리고 이렇게 분산 표현을 이용하여 단어의 유사도를 벡터화하는 작업은 워드 임베딩 (embedding) 작업에 속한다.
  → 이렇게 표현된 벡터 또한 `임베딩 벡터(embedding vector)`라고 한다.
- 또한 저차원을 가지므로 `밀집 벡터(dense vector)`에도 속한다.

분포 가설 (distributional hypothesis)

- 분산 표현(distributed representation) 방법은 기본적으로 `분포 가설(distributional hypothesis)`이라는 가정 하에 만들어진 표현 방법이다.
- 이 가정은 `비슷한 위치에서 등장하는 단어들은 비슷한 의미를 가진다`라는 가정이다.
- "강아지"란 단어는 "귀엽다", "예쁘다", "애교" 등의 단어가 주로 함께 등장한다.
- 이는 분포 가설에 따라서 저런 내용을 가진 텍스트를 벡터화한다면 저 단어들은 의미적으로 가까운 단어가 된다.
- 분산 표현은 분포 가설을 이용하여 단어들의 셋을 학습하고, 벡터에 단어의 의미를 여러 차원에 분산하여 표현한다.

벡터의 차원 감소

- 이렇게 표현된 벡터들은 원-핫 벡터처럼 벡터의 차원이 단어 집합(vocabulary)의 크기일 필요가 없다.
- 그렇기 때문에 벡터의 차원이 상대적으로 저차원으로 줄어든다.
- Word2Vec로 임베딩 된 벡터는 굳이 벡터의 차원이 단어 집합의 크기가 될 필요가 없다.
- 강아지란 단어를 표현하기 위해 사용자가 설정한 차원을 가지는 벡터가 되면서 각 차원은 실수형의 값을 가진다.

> Ex) 강아지 = `[0.2 0.3 0.5 0.7 0.2 ... 0.2]`

요약

- 희소 표현
    - 고차원에 각 차원이 분리된 표현 방법
- 분산 표현
    - 저차원에 `단어의 의미를 여러 차원에다가 분산` 하여 표현
    - 이런 표현 방법을 사용하면 `단어 간 유사도` 를 계산할 수 있다.

분산 표현 학습 방법

Word2Vec의 두 가지 방식

- CBOW (Continuous Bag of Words)
    - 주변에 있는 단어들을 가지고, 중간에 있는 단어들을 예측하는 방법
- Skip-Gram
    - 중간에 있는 단어로 주변 단어들을 예측하는 방법
- 위 두 가지 방법의 메커니즘 자체는 거의 동일
- CBOW를 이해하면 Skip-Gram도 손쉽게 이해 가능

중심 단어(center word)와 주변 단어(context word)

> 예문 : "The fat cat sat on the mat

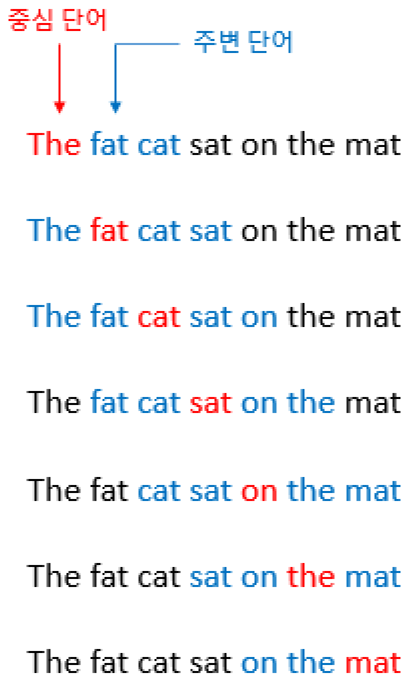
- 갖고 있는 코퍼스에 위와 같은 문장이 있다고 하자.
- 가운데 단어를 예측하는 것이 CBOW이다.
- 즉, `{"The", "fat", "cat", "on", "the", "mat}` 으로부터 "sat"을 예측하는 것이 CBOW가 하는 일이다.
- 이 때 예측해야 하는 단어 "sat"을 `중심 단어(center word)` 라고 한다.
- 예측에 사용되는 단어들을 `주변 단어(context word)` 라고 한다.

윈도우 (window)

- 중심 단어를 예측하기 위해서 앞, 뒤로 몇 개의 단어를 볼 지를 결정했다면 이 범위를 `윈도우 (window)` 라고 한다.
- 예를 들어 윈도우 크기가 2이고, 예측하고자 하는 중심 단어가 "sat"이라고 한다면 앞의 두 단어인 "fat", "cat", 그리고 뒤의 두 단어인 "on", "the"를 참고한다.
- 윈도우 크기가 n이라고 한다면, 실제 중심 단어를 예측하기 위해 참고하려고 하는 주변 단어의 개수는 2n이 될 것이다.

슬라이딩 윈도우 (sliding window)

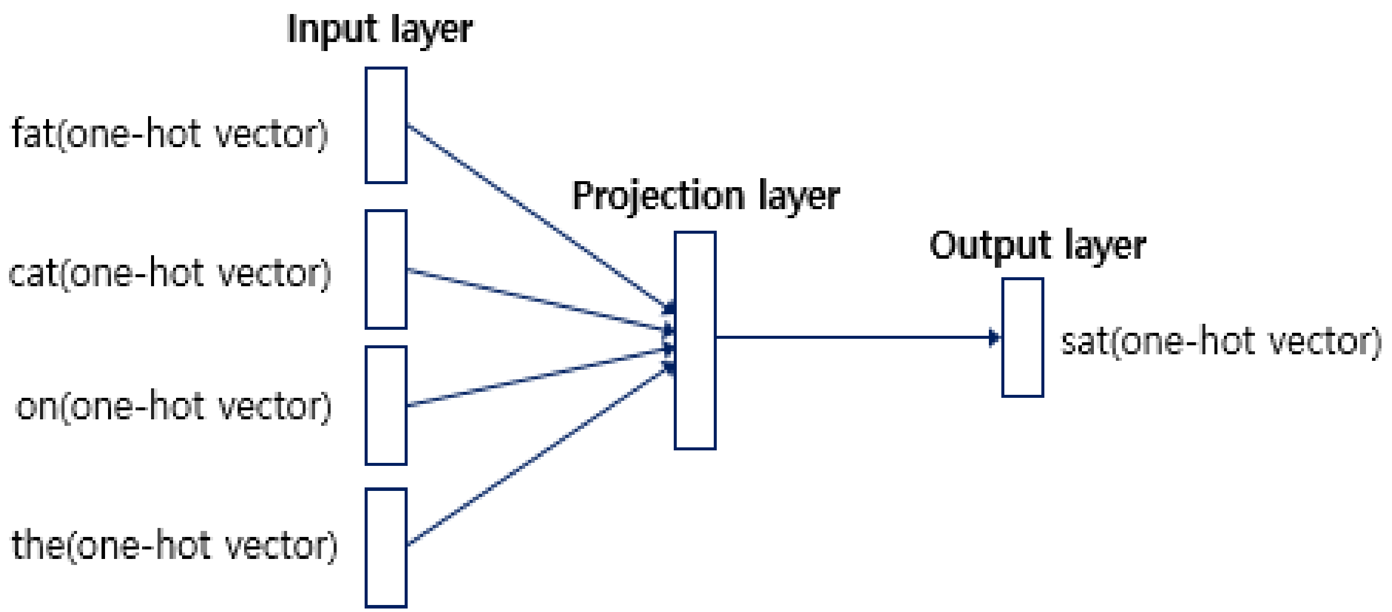
- 윈도우 크기를 정했다면, 윈도우를 계속 움직여서 주변 단어와 중심 단어 선택을 바꿔가며 학습을 위한 데이터 셋을 만들 수 있다.
- 이 방법을 `슬라이딩 윈도우(sliding window)` 라고 한다.



중심 단어	주변 단어
[1, 0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0]	[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0]	[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]

- 위 그림에서 좌측의 중심 단어와 주변 단어의 변화는 윈도우 크기가 2일 때, 슬라이딩 윈도우가 어떤 식으로 이루어지면서 데이터 셋을 만드는 지 보여준다.
- 우측 그림은 중심 단어와 주변 단어를 어떻게 선택했을 때에 따라서 각각 어떤 원-핫 벡터가 되는 지를 보여준다.

CBOW의 인공 신경망 도식화

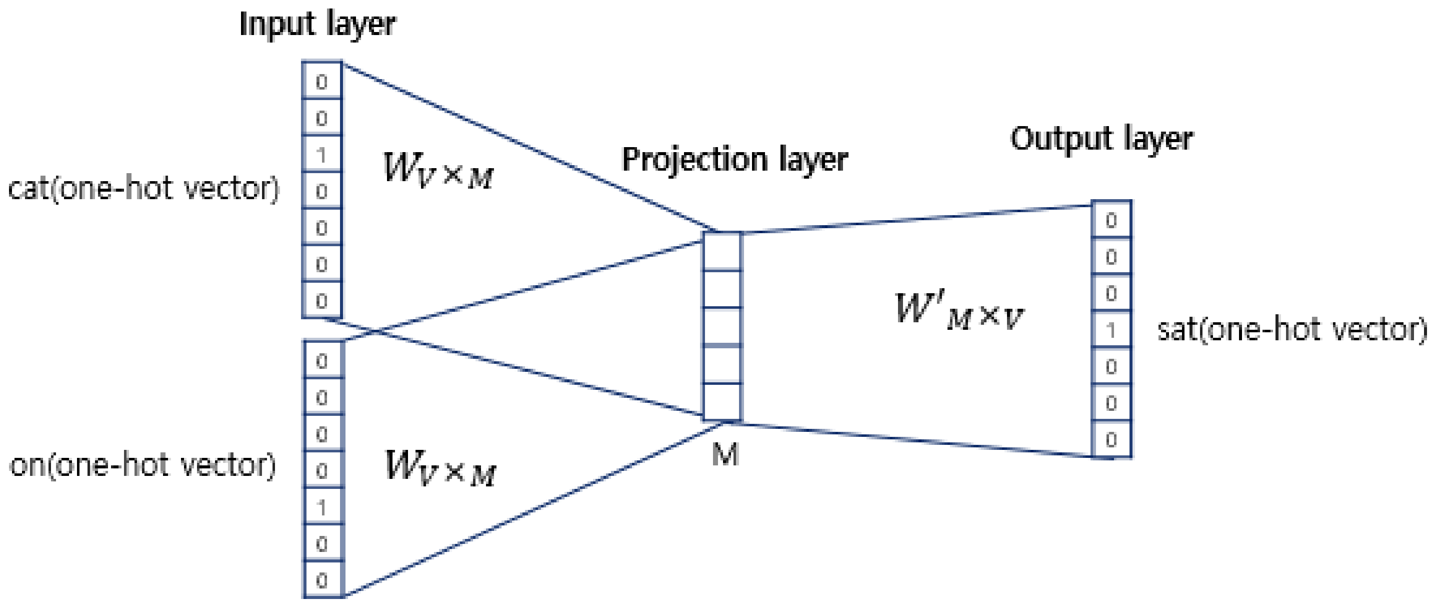


- 입력층(Input layer)의 입력으로서 앞, 뒤 사용자가 정한 윈도우 크기 범위 안에 있는 주변 단어들의 원-핫 벡터가 들어간다.
- 출력층(Output layer)에서 예측하고자 하는 중간 단어의 원-핫 벡터가 필요하다.
- Word2Vec의 학습을 위해서 이 중간 단어의 원-핫 벡터가 필요하다.

Word2Vec의 은닉층

- Word2Vec은 딥 러닝 모델(Deep Learning Model)은 아니다.
- 보통 딥 러닝이라 함은, 입력층과 출력층 사이의 은닉층의 개수가 충분히 쌓인 신경망을 학습할 때를 말한다.
- Word2Vec은 입력층과 출력층 사이에 하나의 은닉층만이 존재한다.
- 이렇게 은닉층(hidden Layer)이 1개인 경우에는 일반적으로 심층신경망(Deep Neural Network)이 아니라 `얕은신경망(Shallow Neural Network)` 이라고 부른다.
- 또한 Word2Vec의 은닉층은 일반적인 은닉층과는 달리 `활성화 함수가 존재하지 않는다.`
- 그 대신 룩업 테이블이라는 연산을 담당하는 층으로 일반적인 은닉층과 구분하기 위해 `투사층(projection layer)` 이라고 부르기도 한다.

CBOW의 동작 메커니즘



- 투사층의 크기 (M)
- 가중치 행렬 (W)

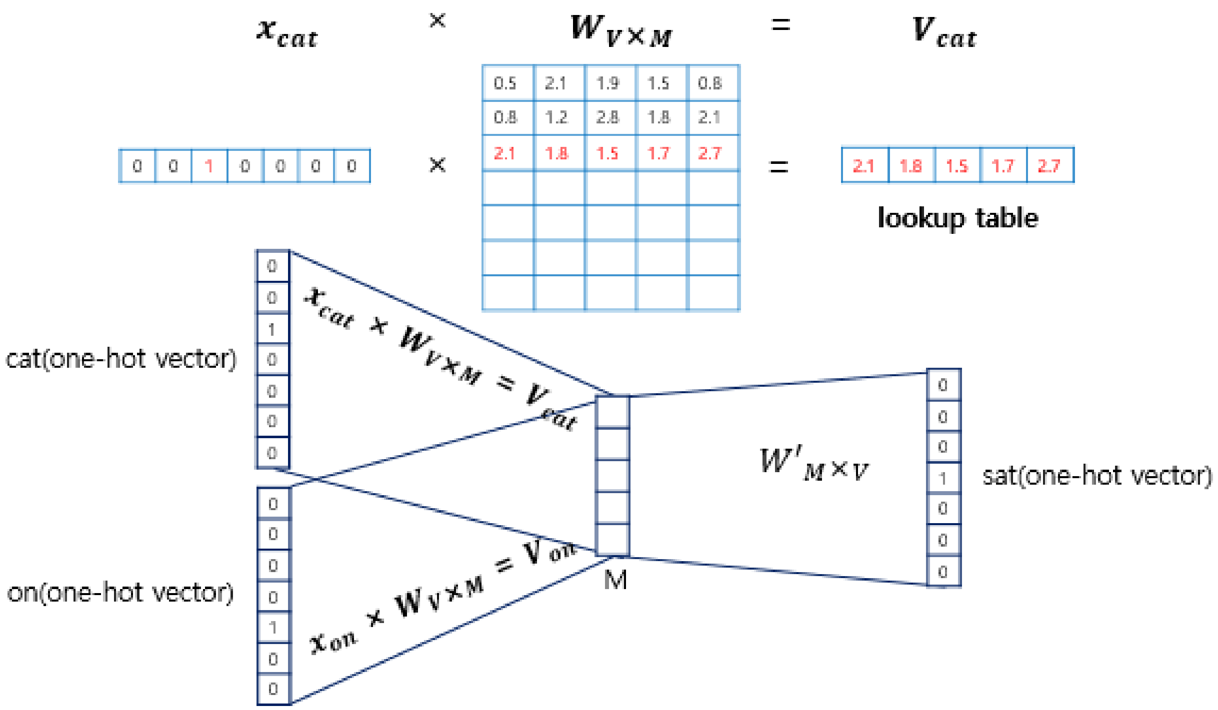
투사층의 크기

- 위 그림에서 투사층의 크기는 M이다.
- CBOW에서 투사층의 크기 M은 임베딩하고 난 벡터의 차원이 된다.
- 다시 말해, 위의 그림에서 투사층의 크기는 M=5이기 때문에 CBOW를 수행하고 나서 얻는 각 단어의 임베딩 벡터의 차원은 5가 될 것이다.

가중치 행렬

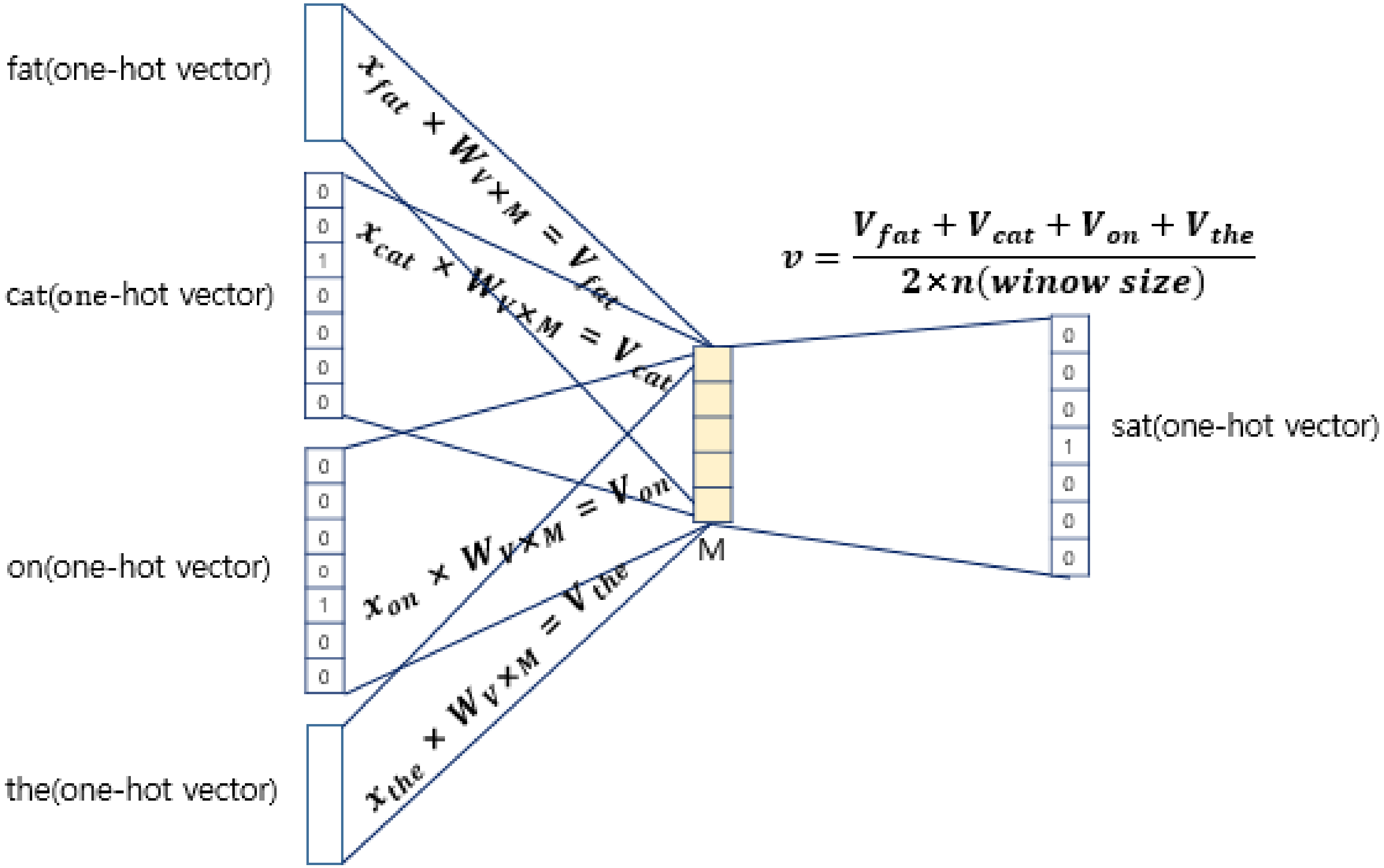
- 입력층과 투사층 사이의 가중치 W는 V x M 행렬이다.
- 투사층과 출력층 사이의 가중치 W'는 M x V 행렬이다.
- 여기서 V는 단어 집합의 크기를 의미한다.
- 즉, 위의 그림처럼 원-핫 벡터의 차원은 7이고, M은 5라고 하면
  $\rightarrow$ 가중치 W는 7 x 5 행렬이다.
  $\rightarrow$ 가중치 W'는 5 x 7 행렬이다.
- 주의할 점은 `이 두 행렬은 동일한 행렬을 전치(transpose)한 것이 아니라, 서로 다른 행렬이라는 점`이다.
- 인공 신경망의 훈련 전에 이 가중치 행렬 W와 W'는 굉장히 작은 랜덤 값을 가지게 된다.
- CBOW는 주변 단어로 중심 단어를 더 정확히 맞추기 위해 계속해서 이 W와 W'를 학습해가는 구조이다.

룩업 테이블 (lookup table)



- i번 째 인덱스에 1이라는 값을 가지고 그 이외의 0의 값을 가지는 입력 벡터와 가중치 W 행렬의 곱은 W 행렬의 i번째 행을 그대로 읽어오는 것(lookup)과 동일하다.
- 그래서 이 작업을 룩업 테이블(lookup table)이라고 부른다.
- 여기서 W의 각 행벡터가 Word2Vec을 수행한 후 각 단어의 M 차원의 크기를 갖는 임베딩 벡터

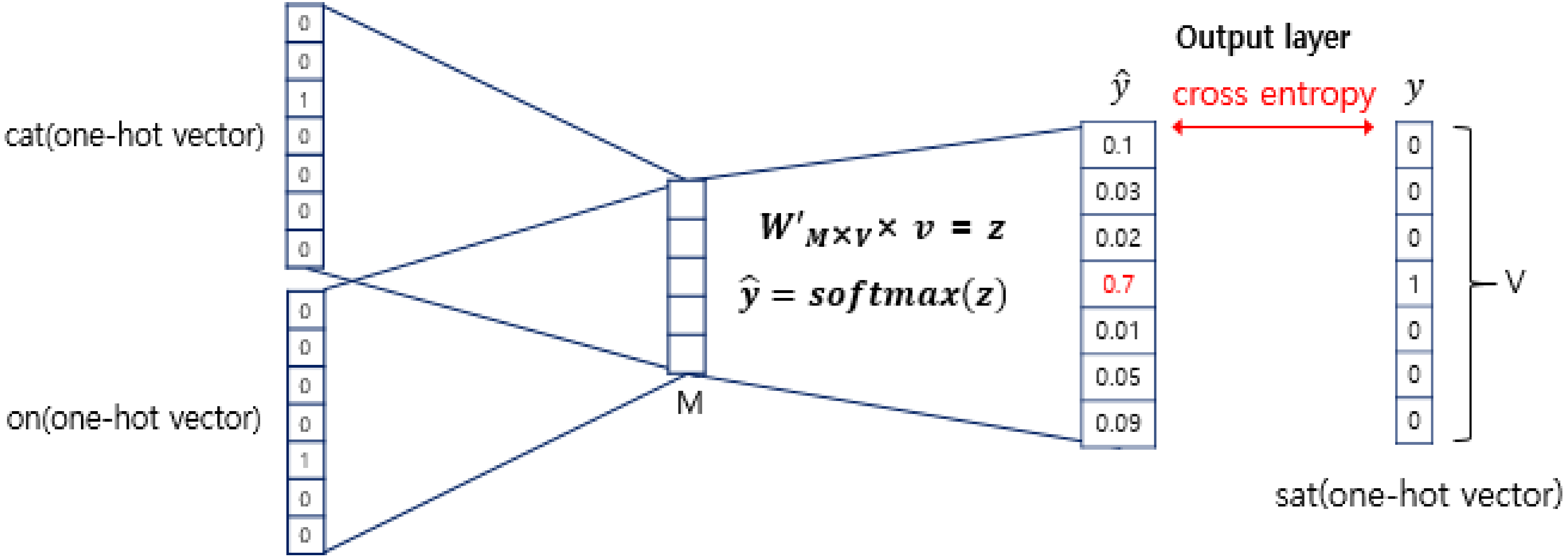
투사층에서 벡터의 평균 계산



투사층에서 벡터의 평균 계산

- 각 주변 단어의 원-핫 벡터에 대해서 가중치 W가 곱해서 생겨진 결과 벡터들은 투사층에서 만나 이 벡터들의 평균인 벡터를 구한다.
- 윈도우 크기가 2 $\longrightarrow$ 입력 벡터의 총 개수 = 2n
- 그러므로 중간 단어를 예측하기 위해서는 총 4개가 입력 벡터로 사용된다.
- 그렇기 때문에 평균을 구할 때는 4개의 결과 벡터에 대해서 평균을 구하게 된다.
- 투사층에서 벡터의 평균을 구하는 부분은 CBOW가 Skip-Gram과 다른 차이점이기도 하다. (Skip-Gram은 입력이 중심 단어 하나이기때문에 투사층에서 벡터의 평균을 구하지 않는다.)

스코어 벡터(score vector) 생성



스코어 벡터(score vector) 생성

- 이렇게 구해진 평균 벡터는 두 번째 가중치 행렬 W'와 곱해진다.
- 곱셈의 결과로는 원-핫 벡터들과 차원이 V로 동일한 벡터가 나온다.
(만약 입력 벡터의 차원이 7이었다면 여기서 나오는 벡터도 마찬가지이다.)
- 이 벡터에 CBOW는 `소프트맥스(softmax) 함수`를 취한다.
- 소프트맥스 함수로 인한 출력값은 0과 1 사이의 실수로, 각 원소의 총 합은 1이 되는 상태로 바뀐다.
- 이렇게 나온 벡터를 `스코어 벡터(score vector)`라고 한다.
- 스코어 벡터의 j번 째 인덱스가 가진 0과 1 사이의 값 ⇒ `j번 째 단어가 중심 단어일 확률`
- 스코어 벡터는 중심 단어 원-핫 벡터의 값에 가까워져야 한다.

손실 함수 : 크로스 엔트로피 함수

- 스코어 벡터를 $\hat{y}$라고 하자.
- 중심 단어를 y로 했을 때, 이 두 벡터값의 오차를 줄이기 위해 CBOW는 손실 함수(loss function)로 **cross-entropy 함수**를 사용한다.
- cross-entropy 함수에 실제 중심 단어인 원-핫 벡터와 스코어 벡터를 입력값으로 넣고, 이를 식으로 표현하면 다음과 같다.

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \, log(\hat{y_j})$$

- y가 원-핫 벡터라는 점을 고려하면, 이 식은 다음과 같이 간소화시킬 수 있다.

$$H(\hat{y}, y) = -y_j \, log(\hat{y_j})$$

- c : 중심 단어에서 1을 가진 차원의 값의 인덱스
- $\hat{y_c} = 1$: $\hat{y}$가 y를 정확하게 예측한 경우
- 이를 식에 대입해보면 $-1\,log(1) = 0$이 되기 때문에, 결과적으로 $\hat{y}$가 y를 정확하게 예측한 경우의 cross-entropy의 값은 0이 된다.
- 다음 값을 최소화하는 방향으로 학습해야 한다.

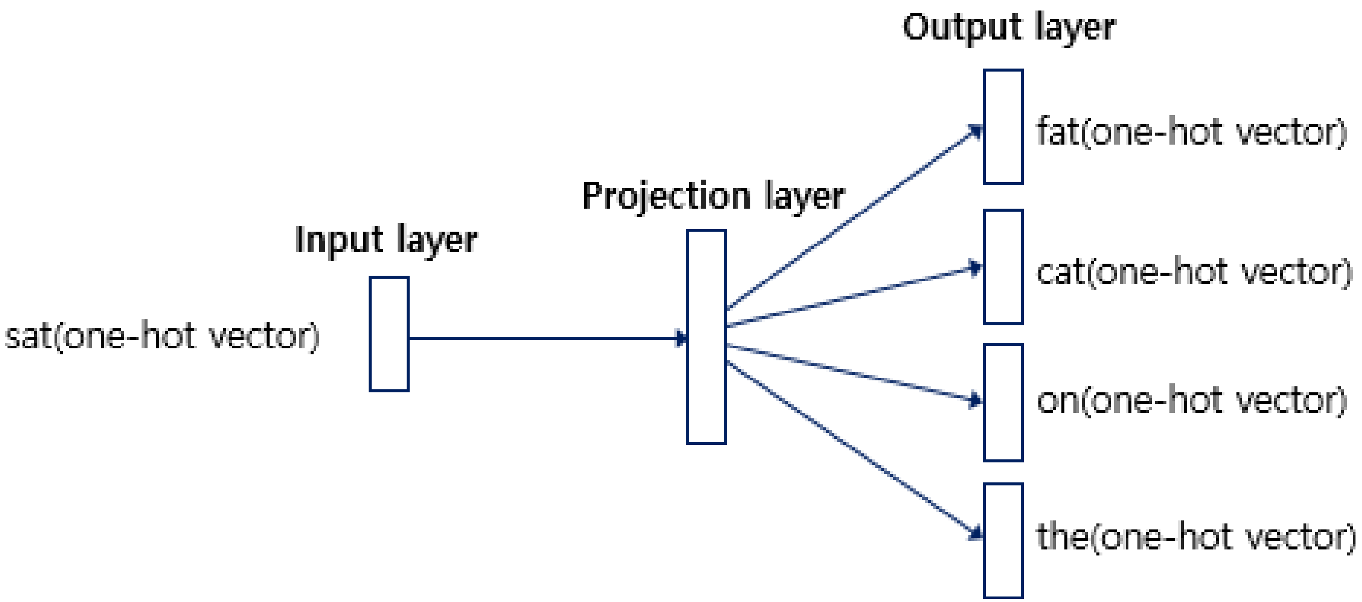
$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j\, log(\hat{y_j})$$

임베딩 벡터 결정

- 역전파(Back Propagation)를 수행하면 W와 W'가 학습된다.
- 학습이 다 되었다면 M 차원의 크기를 갖는 W의 행이나 W'의 열로부터 어떤 것을 임베딩 벡터로 사용할 지 결정하면 된다.
- 때로는 W와 W'의 평균치를 가지고 임베딩 벡터를 선택하기도 한다.

Skip-gram

- 앞서 CBOW에서는 주변 단어를 통해 중심 단어를 예측했다.
- Skip-gram은 중심 단어에서 주변 단어를 예측하려고 한다.



- 중심 단어에 대해서 주변 단어를 예측하므로 `투사층에서 벡터들의 평균을 구하는 과정은 없다.`
- 전반적으로 skip-gram이 CBOW보다 성능이 좋다고 알려져 있다.

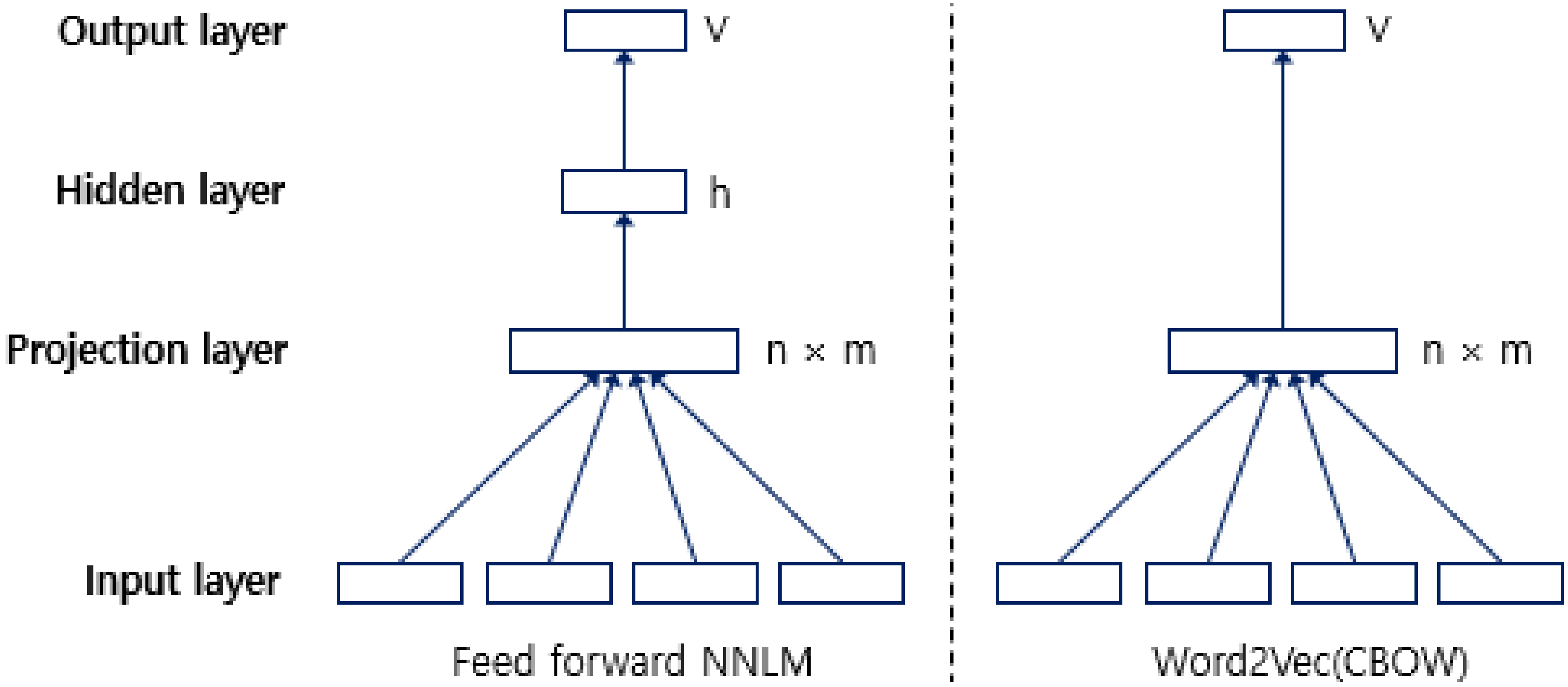
NNLM vs Word2Vec

- NNLM은 단어 간 유사도를 구할 수 있도록 워드 임베딩의 개념을 도입했고, NNLM의 느린 학습 속도와 정확도를 개선하여 탄생한 것이 Word2Vec이다.

예측하는 대상의 차이

- NNLM
  - 언어 모델이므로 다음 단어를 예측한다.
  - 예측 단어의 이전 단어들만을 참고한다.
- Word2Vec(CBOW)
  - 워드 임베딩 자체가 목적이므로 다음 단어가 아닌 중심 단어를 예측하게 하여 학습한다.
  - 중심 단어를 예측하게 하므로서 예측 단어의 전, 후 단어들을 모두 참고한다.

NNLM vs Word2Vec: 구조의 차이



NNLM vs Word2Vec: 구조의 차이

- 위의 그림은 각각의 변수를 다음과 같이 정의할 때 NNLM과 Word2Vec의 차이를 보여준다.
  - n : 학습에 사용하는 단어의 수
  - m : 임베딩 벡터의 차원
  - h : 은닉층의 크기
  - V : 단어 집합의 크기
- Word2Vec은 우선 NNLM에 존재하던 활성화 함수가 있는 은닉층을 제거하였다.
- 이에 따라 투사층 다음에 바로 출력층으로 연결되는 구조이다.

Word2Vec이 학습 속도에서 강점을 가지는 이유

- Word2Vec이 NNLM보다 학습 속도에서 강점을 가지는 이유는 은닉층을 제거한 것뿐만 아니라 추가적으로 사용되는 기법들 덕분이다.
- 대표적인 기법
  - `계층적 소프트맥스(hierarchical softmax)`
  - `네거티브 샘플링(negative sampling)`

NNLM의 연산량

- 입력층에서 투사층, 투사층에서 은닉층, 은닉층에서 출력층으로 향하며 발생하는 NNLM의 연산량은 다음과 같다.

$$\text{NNLM} : (n \times m) + (n \times m \times h) + (h \times V)$$

Word2Vec의 연산량

- 추가적인 기법들까지 사용하였을 때 Word2Vec은 출력층에서의 연산에서 V를 $log(V)$로 바꿀 수 있다.

$$\text{Word2Vec} : (n \times m) + (m \times log(V))$$

네거티브 샘플링 (Negative Sampling)

SGNS (Skip-Gram with Negative Sampling)

- 대체적으로 Word2Vec를 사용한다고 하면 SGNS(Skip-Gram with Negative Sampling)을 사용한다.
- Skip-gram을 사용하는 데, 네거티브 샘플링(Negative Sampling)이란 방법까지 추가로 사용한다는 것이다.
- Skip-gram을 전제로 네거티브 샘플링에 대해서 알아보자.

Word2Vec 모델의 문제점

- 위에서 배운 Word2Vec 모델에는 한 가지 문제점이 있다.
- 바로 속도이다.
- 출력층에 있는 소프트맥스 함수는 단어 집합 크기의 벡터 내의 모든 값을 0과 1 사이의 값이면서 모두 더하면 1이 되도록 바꾸는 작업을 수행한다.
- 그리고 이에 대한 오차를 구하고 모든 단어에 대한 임베딩을 조정한다.
- 그 단어가 중심 단어나 주변 단어와 전혀 상관없는 단어라도 마찬가지이다.
- 그런데 만약 단어 집합의 크기가 수백만에 달한다면 이 작업은 굉장히 무거운 작업이다.

연관 관계가 없는 단어들에 대한 임베딩 조정의 불필요성

- 여기서 중요한 건 Word2Vec이 모든 단어 집합에 대해서 소프트맥스 함수를 수행하고, 역전파를 수행하므로 주변 단어와 상관 없는 모든 단어까지의 워드 임베딩 조정 작업을 수행한다는 것이다.
- 만약 마지막 단계에서 '강아지'와 '고양이'와 같은 단어에 집중하고 있다면, Word2Vec은 사실 '돈가스'나 '컴퓨터'와 같은 연관 관계가 없는 수많은 단어의 임베딩을 조정할 필요가 없다.

일부 단어 집합 사용하여 이진 분류 문제로 변경

- 전체 단어 집합이 아니라 일부 단어 집합에 대해서만 고려하면 안될까?
- 이렇게 일부 단어 집합을 만들어보자.
- '강아지', '고양이', '애교'와 같은 주변 단어들을 가져온다.
- 그리고 여기에 '돈가스', '컴퓨터', '회의실'과 같은 랜덤으로 선택된 주변 단어가 아닌 상관없는 단어들을 일부만 가져온다.
- 이렇게 전체 단어 집합보다 훨씬 작은 단어 집합을 만들어놓고 마지막 단계를 `이진 분류 문제`로 바꿔버린다.
- 즉, Word2Vec은 주변 단어들을 긍정(positive)으로 두고 랜덤으로 샘플링 된 단어들을 부정(negative)으로 둔 다음에 이진 분류 문제를 수행한다.
- 기존의 다중 클래스 분류 문제를 이진 분류 문제로 바꾸면서도 연산량에 있어서 훨씬 효율적이다.

글로브 (GloVe)

- 글로브(Global Vectors for Word Representation, GloVe)는 카운트 기반과 예측 기반을 모두 사용하는 방법론이다.
- 2014년에 미국 스탠포드대학에서 개발한 단어 임베딩 방법론이다.
- 앞서 학습하였던 기존의 카운트 기반의 LSA(Latent Semantic Analysis)와 예측 기반의 Word2Vec의 단점을 지적하며 이를 보완한다는 목적으로 나왔다.
- 실제로도 Word2Vec 만큼 뛰어난 성능을 보여준다.
- 현재까지의 연구에 따르면 단정적으로 Word2Vec와 GloVe 중에서 어떤 것이 더 뛰어나다고 말할 수는 없고, 이 두 가지 전부를 사용해보고 성능이 더 좋은 것을 사용하는 것이 바람직하다.

기존 방법론에 대한 비판

LSA

- LSA는 DTM이나 TF-IDF 행렬과 같이 각 문서에서의 각 단어의 빈도 수를 카운트한 행렬이라는 전체적인 통계 정보를 입력으로 받아 차원을 축소(Truncated SVD)하여 잠재된 의미를 끌어내는 방법론이다.

Word2Vec

- Word2Vec는 실제값과 예측값에 대한 오차를 손실 함수를 통해 줄여나가며 학습하는 예측 기반의 방법론이다.

LSA vs. Word2Vec vs. GloVe

- LSA
  - (장점) : 카운트 기반으로 코퍼스의 전체적인 통계 정보를 고려한다.
  - (단점) : 왕 : 남자 = 여왕 : ? (정답은 여자)와 같은 단어 의미의 유추 작업(Analogy task)에는 성능이 떨어진다.
- Word2Vec
  - (장점) : 예측 기반으로 단어 간 유추 작업에는 LSA보다 뛰어나다.
  - (단점) : 임베딩 벡터가 윈도우 크기 내에서만 주변 단어를 고려하기 때문에 코퍼스의 전체적인 통계 정보를 반영하지 못한다.
- GloVe
  - GloVe는 LSA의 메커니즘이었던 카운트 기반의 방법과 Word2Vec의 메커니즘이었던 예측 기반의 방법론 두 가지를 모두 사용한다.

윈도우 기반 동시 등장 행렬 (Window based Co-occurrence Matrix)

단어의 동시 등장 행렬

- 행과 열을 전체 단어 집합의 단어들로 구성
- i 단어의 윈도우 크기(Window Size) 내에서 k 단어가 등장한 횟수를 i행 k열에 기재한 행렬
- 아래와 같은 텍스트가 있다고 하자.

> I like deep learning
>
> I like NLP
>
> I enjoy flying

- 윈도우 크기가 N 일 때는 좌, 우에 존재하는 N개의 단어만 참고하게 된다.
- 윈도우 크기가 1일 때, 위의 텍스트를 가지고 구성한 동시 등장 행렬은 다음과 같다.

카운트	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

윈도우 기반 동시 등장 행렬의 특징

- 위 행렬은 행렬을 전치(Transpose)해도 동일한 행렬이 된다는 특징이 있다.
- 그 이유는 i 단어의 윈도우 크기 내에서 k 단어가 등장한 빈도는 반대로 k 단어의 윈도우 크기 내에서 i 단어가 등장한 빈도와 동일하기 때문이다.

동시 등장 확률 (Co-occurrence Probability)

동시 등장 확률 $P(k \mid i)$

- 동시 등장 행렬로부터 특정 단어 i의 전체 등장 횟수를 카운트한다.
- 특정 단어 i가 등장했을 때 어떤 단어 k가 등장한 횟수를 카운트하여 계산한 조건부 확률이다.
- $P(k \mid i)$에서 i를 중심 단어(Center Word), k를 주변 단어(Context Word)라고 할 때
  - 분모값 : 동시 등장 행렬에서 중심 단어 i의 행의 모든 값을 더한 값
  - 분자값 : i행 k열의 값

$$P(k \mid i) = \frac{\text{value of the i row and k column}}{\text{sum of all values in i row}}$$

표로 정리한 동시 등장 확률

- 다음은 GloVe의 제안 논문에서 가져온 동시 등장 확률을 표로 정리한 하나의 예이다.

동시 등장 확률과 크기 관계 비(ratio)	k=solid	k=gas	k=water	k=fasion
P(k l ice)	0.00019	0.000066	0.003	0.000017
P(k l steam)	0.000022	0.00078	0.0022	0.000018
P(k l ice) / P(k l steam)	8.9	0.085	1.36	0.96

k = solid

- `solid` 가 등장했을 때 `ice` 가 등장할 확률인 0.00019은 `solid` 가 등장했을 때 `steam` 이 등장할 확률인 0.000022보다 약 8.9배 더 크다.
- `solid` 는 '단단한'이라는 의미를 가졌으므로, '증기'라는 의미를 가지는 `steam` 보다는 당연히 '얼음'의 뜻을 가진 `ice` 라는 단어와 더 자주 등장할 것이다.
- k가 `solid` 일 때, $P(\text{solid} \mid \text{ice}) / P(\text{solid} \mid \text{steam})$를 계산한 값은 8.9가 나온다.
- 이 값은 1보다는 매우 큰 값이다. ($P(\text{solid} \mid \text{ice})$의 값은 크고, $P(\text{solid} \mid \text{steam})$의 값은 작기 때문)

k = gas

- `gas` 는 `ice` 보다는 `steam` 과 더 자주 등장한다.
- 그러므로 $P(\text{gas} \mid \text{ice})/P(\text{gas} \mid \text{steam})$를 계산한 값은 0.085가 나온다.
- 이 값은 1보다 훨씬 작은 값이다.

k = water

- k가 `water` 인 경우에는 `solid` 와 `steam` 두 단어 모두와 동시 등장하는 경우가 많으므로 1에 가까운 값(1.36)이 나온다.

k = fasion

- k가 `fasion` 인 경우에는 `solid` 와 `steam` 두 단어 모두와 동시 등장하는 경우가 적으므로 1에 가까운 값(0.96)에 가까운 값이 나온다.

동시 등장 확률과 크기 관계 비(ratio)	k=solid	k=gas	k=water	k=fasion
P(k l ice)	큰 값	작은 값	큰 값	작은 값
P(k l steam)	작은 값	큰 값	큰 값	작은 값
P(k l ice) / P(k l steam)	큰 값	작은 값	1에 가까움	1에 가까움

손실 함수 (Loss function)

손실 함수 관련 용어 정리

- X
  - 동시 등장 행렬 (Co-occurrence Matrix)
- X_{ij}
  - 중심 단어 i가 등장했을 때 윈도우 내 주변 단어 j가 등장하는 횟수
- X_i $(=\sum_j X_{ij})$
  - 동시 등장 행렬에서 i행의 값을 모두 더한 값

- P_{ik} ($= P(k \mid i) \; = \; \frac{X_{ik}}{X_i}$)
    - 중심 단어 i가 등장했을 때, 윈도우 내 주변 단어 k가 등장할 확률
    - ex) $P(\text{solid} \mid \text{ice})$ = 단어 `ice` 가 등장했을 때, 단어 `solid` 가 등장할 확률
- $\frac{P_{ik}}{P_{jk}}$
    - P_{ik}를 P_{jk}로 나눠준 값
    - ex) $P(\text{solid} \mid \text{ice}) / P(\text{solid} \mid \text{steam}) = 8.9$
- w_i
    - 중심 단어 i의 임베딩 벡터
- $\tilde{w_k}$
    - 주변 단어 k의 임베딩 벡터

GloVe의 아이디어

> "임베딩된 중심 단어와 주변 단어 벡터의 내적이 전체 코퍼스에서의 동시 등장 확률이 되도록 만드는 것"

- 즉, 이를 만족하도록 임베딩 벡터를 만드는 것이 목표이다.
- 이를 식으로 표현하면 다음과 같다.

$$dot\ product\,(w_i\ \tilde{w}_k)\ \approx\ P(k\,|\,i) = P_{ik}$$

- 더 정확히는 GloVe는 아래와 같은 관계를 가지도록 임베딩 벡터를 설계한다.

$$dot\ product\,(w_i\ \tilde{w}_k)\ \approx\ log\ P(k\,|\,i) = log\ P_{ik}$$

임베딩 벡터들을 만들기 위한 손실 함수 설계

- 가장 중요한 것은 **단어 간의 관계를 잘 표현하는 함수여야 한다는 것**이다.
- 이를 위해 앞서 배운 개념인 P_{ik}/P_{jk}를 식에 사용한다.

초기 식

- GloVe의 연구진들은 벡터 $w_i, w_j, \tilde{w_k}$를 가지고 어떤 함수 F를 수행하면, P_{ik}/P_{jk}가 나온다는 초기 식으로부터 전개를 시작한다.

$$F(w_i,\ w_j,\ \tilde{w_k}) = \frac{P_{ik}}{P_{jk}}$$

- 아직 이 함수 F가 어떤 식을 가지고 있는 지는 정해진 것이 없다.
- 위의 목적에 맞게 근사할 수 있는 함수식은 무수히 많을 것이다.
- 선형 공간(Linear Space)에서 단어의 의미 관계를 표현하기 위해 `뺄셈`과 `내적`을 선택

뺄셈 적용

- 함수 F는 두 단어 사이의 동시 등장 확률의 크기 관계 비(ratio) 정보를 벡터 공간에 인코딩하는 것이 목적이다.
- 이를 위해 GloVe 연구진들은 w_i와 w_j라는 두 벡터의 차이를 함수 F의 입력으로 사용하는 것을 제안한다.

$$F(w_i - w_j,\ \tilde{w_k}) = \frac{P_{ik}}{P_{jk}}$$

내적 적용

- 그런데 위 식에서 우변($\frac{P_{ik}}{P_{jk}}$)은 스칼라 값이고, 좌변은 벡터값이다.
- 이를 성립하게 해주기 위해서 함수 F의 두 입력에 내적(Dot Product)을 수행한다.

$$F((w_i - w_j)^T \cdot \tilde{w_k}) = \frac{P_{ik}}{P_{jk}}$$

준동형(Homomorphism)

- 여기서 함수 F가 만족해야 할 필수 조건이 있다.
- 중심 단어 w와 주변 단어 $\tilde{w}$ 라는 선택 기준은 실제로는 **무작위 선택**이다.
- 그러므로 이 둘의 관계는 **자유롭게 교환될 수 있도록** 해야 한다.
- 이 것이 성립되게 하기 위해서 GloVe 연구진은 함수 F가 **실수의 덧셈**과 **양수의 곱셈**에 대해서 `준동형(Homomorphism)`을 만족하도록 한다.
- 정리하면 a와 b에 대해서 함수 F가 $F(a+b)$가 $F(a)F(b)$와 같도록 만족시켜야 한다는 의미이다.

$$F(a+b) = F(a)F(b) \quad \forall a,\, b \ \in \ \mathbb{R}$$

준동형식 변형 (1)

- 이제 이 준동형식을 현재 전개하던 GloVe 식에 적용할 수 있도록 조금씩 바꿔보자.
- 전개하던 GloVe 식에 따르면, 함수 F는 결과값으로 스칼라 값인 $\frac{P_{ik}}{P_{jk}}$이 나와야 한다.
- 준동형식에서
  - a와 b가 각각 벡터값인 경우 $\rightarrow$ 함수 F의 결과값으로는 스칼라 값이 나올 수 없다.
  - a와 b가 각각 **두 벡터의 내적값**인 경우 $\rightarrow$ 함수 F의 결과값으로 스칼라 값이 나올 수 있다.
- 그러므로 위의 준동형식을 아래와 같이 바꿔보자.
- 여기서 $v_1,\ v_2,\ v_3,\ v_4$는 각각 벡터값이다.
- 아래의 V는 벡터를 의미한다.

$$F\left(v_1^T\, v_2 + v_3^T\, v_4\right) = F(v_1^T\, v_2)\, F(v_3^T\, v_4) \quad \forall\, v_1,\ v_2,\ v_3,\ v_4\ \in\ V$$

준동형식 변형 (2)

- 그런데 앞서 작성한 GloVe 식에서는 w_i와 w_j라는 두 벡터의 차이를 함수 F의 입력으로 받았다.
- GloVe 식에 바로 적용을 위해 준동형식을 뺄셈에 대한 준동형식으로 변경한다.
- 그렇게 되면 곱셈도 나눗셈으로 바뀐다.

$$F\left(v_1^T\,v_2 - v_3^T\,v_4\right) = \frac{F(v_1^T\,v_2)}{F(v_3^T\,v_4)} \quad \forall\, v_1,\,v_2,\,v_3,\,v_4\ \in\ V$$

준동형식 적용

- 우리의 목적은 함수 F의 우변을 다음과 같이 바꿔야 한다.

$$F((w_i - w_j)^T \cdot \tilde{w_k}) = \frac{F(w_i^T \ \tilde{w_k})}{F(w_j^T \ \tilde{w_k})}$$

- 이전 식에 따르면 우변은 본래 $\frac{P_{ik}}{P_{jk}}$ 이였으므로, 결과적으로 다음과 같다.

$$\frac{P_{ik}}{P_{jk}} = \frac{F(w_i^T \ \tilde{w_k})}{F(w_j^T \ \tilde{w_k})}$$

$$F(w_i^T \ \tilde{w_k}) = P_{ik} = \frac{X_{ik}}{X_i}$$

- 좌변을 풀어쓰면 다음과 같다.

$$F(w_i^T \ \tilde{w_k} \ - \ w_j^T \ \tilde{w_k}) = \frac{F(w_i^T \ \tilde{w_k})}{F(w_j^T \ \tilde{w_k})}$$

- 이는 뺄셈에 대한 준동형식의 형태와 정확히 일치한다.

함수 F

- 이제 이를 만족하는 함수 F를 찾아야 한다.
- 그리고 이를 정확하게 만족시키는 함수가 있는데 바로 `지수 함수(Exponential function)` 이다.
- F를 지수 함수 exp라고 해보자.

$$exp(w_i^T\,\tilde{w_k}\ -\ w_j^T\,\tilde{w_k}) = \frac{exp(w_i^T\,\tilde{w_k})}{exp(w_j^T\,\tilde{w_k})}$$

$$exp(w_i^T\,\tilde{w_k}) = P_{ik} = \frac{X_{ik}}{X_i}$$

- 위의 두 번째 식으로부터 다음과 같은 식을 얻을 수 있다.

$$w_i^T\,\tilde{w_k} = log\,P_{ik} = log\left(\frac{X_{ik}}{X_i}\right) = log\,X_{ik} - log\,X_i$$

편향 적용

- 그런데 여기서 상기해야 할 것은 앞서 언급했듯이, 사실 w_i와 $\tilde{w}$는 두 값의 위치를 서로 바꾸어도 식이 성립해야 한다.
- X_{ik}의 정의를 생각해보면 X_{ki}와도 같다.
- 그런데 이게 성립되려면 위의 식에서 $log\ X_i$ 항이 걸림돌이다.
- 이 부분만 없다면 이를 성립시킬 수 있다.
- 그래서 GloVe 연구팀은 $log\ X_i$ 항을 w_i에 대한 편향 b_i라는 상수항으로 대체하기로 한다.
- 같은 이유로 $\tilde{w_k}$에 대한 편향 $\tilde{b_k}$를 추가한다.

$$w_i^T\ \tilde{w_k} + b_i + \tilde{b_k} = log\ X_{ik}$$

손실 함수 일반화

- 위의 식이 손실 함수의 핵심이 되는 식이다.
- 우변의 값과의 차이를 최소화하는 방향으로 좌변의 4개의 항은 학습을 통해 값이 바뀌는 변수들이 된다.
- 즉, 손실 함수는 다음과 같이 일반화될 수 있다.

$$Loss\ function = \sum_{m,n=1}^{V} \left(w_m^T \tilde{w_n} + b_m + \tilde{b_n} - log\, X_{mn} \right)^2$$

- 여기서 V는 단어 집합의 크기를 의미한다.

손실 함수의 보완 필요성

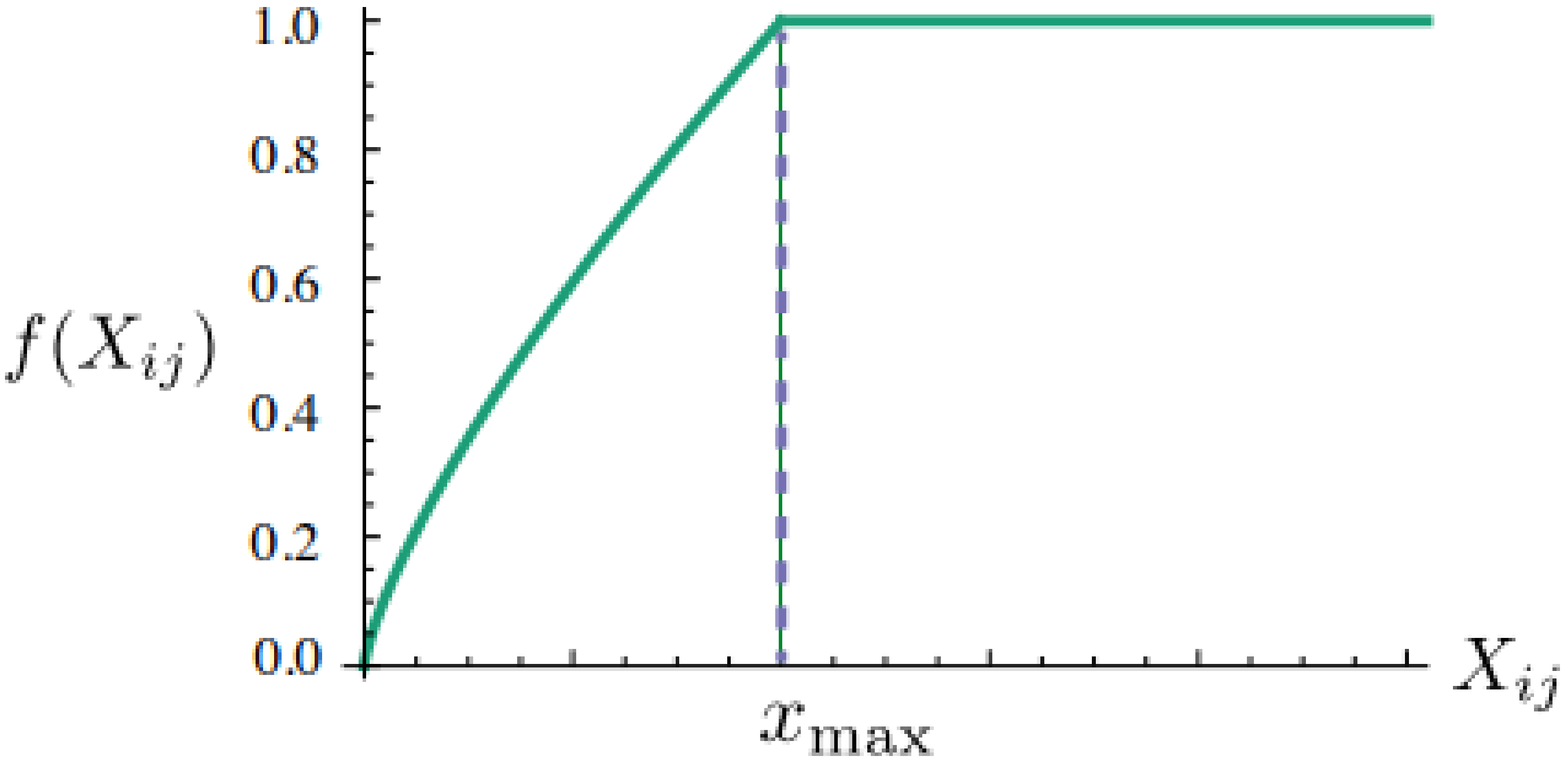
- 그런데 아직 최적의 손실 함수라기에는 부족하다.
- GloVe 연구진은 $log\, X_{ik}$에서 X_{ik} 값이 0이 될 수 있음을 지적한다.
- 대안 중 하나는 $log\, X_{ik}$ 항을 $log\,(1 + X_{ik})$로 변경하는 것이다.
- 하지만 이렇게 해도 여전히 해결되지 않는 문제가 있다.
- 바로 **동시 등장 행렬** X**는 마치 DTM처럼 희소 행렬(Sparse Matrix)**일 가능성이 다분하다는 점이다.

가중치 함수 도입

- 동시 등장 행렬 X에는 많은 값이 0 이거나, 동시 등장 빈도가 적어서 많은 값이 작은 수치를 가지는 경우가 많다.
- 앞서 빈도수를 가지고 가중치를 주는 고민을 하는 TF-IDF나 LSA와 같은 몇 가지 방법들을 본 적이 있다.
- GloVe의 연구진은 동시 등장 행렬에서 동시 등장 빈도의 값 X_{ik}이 굉장히 낮은 경우에는 정보에 거의 도움이 되지 않는다고 판단한다.
- 그래서 이에 대한 가중치를 주는 고민을 하게 된다.
- GloVe 연구팀이 선택한 것은 바로 **X_{ik}의 값에 영향을 받는 가중치 함수(Weighting function) $f\left(X_{ik}\right)$를 손실 함수에 도입하는 것**이다.

가중치 함수 도입

- GloVe에 도입되는 $f\left(X_{ik}\right)$의 그래프는 아래와 같다.



가중치 함수 도입

- X_{ik}의 값이 작다 $\rightarrow$ 상대적으로 함수의 값은 작도록 한다.
- X_{ik}의 값이 크다 $\rightarrow$ 상대적으로 함수의 값은 크도록 한다.
- 하지만 X_{ik}가 지나치게 높다고 해서 지나친 가중치를 주지 않기 위해서 또한 함수의 최대값이 정해져 있다. (최대값은 1)
  - ex) 'It is'와 같은 불용어의 동시 등장 빈도수가 높다고 해서 지나친 가중을 받아서는 안된다.
- 이 함수의 값을 손실 함수에 곱해주면 가중치의 역할을 할 수 있다.
- 이 함수 $f(x)$의 식은 다음과 같이 정의된다.

$$f(x) = min\left(1,\ (x/x_{max})^{3/4}\right)$$

최종 일반화된 손실 함수

- 최종적으로 다음과 같은 일반화된 손실 함수를 얻어낼 수 있다.

$$Loss\ function = \sum_{m,n=1}^{V} f\left(X_{mn}\right)\left(w_m^T \tilde{w_n} + b_m + \tilde{b_n} - log\ X_{mn}\right)^2$$