발표자: 박성은

발표일자: 2023년 7월 11일

목차

1. Word Embedding?

: 각 단어를 인공 신경망 학습을 통해 벡터화하는 방법으로 현재 가장 많이 사용되고 있는 방법

∵ 문자(텍스트)를 컴퓨터가 이해 및 효율적으로 처리할 수 있도록 숫자로 변환하기 위함, 단어 표현 방법에 따라서 자연어 처리 성능이 변함

- 단어를 벡터로 표현하는 방법
- 단어를 밀집 표현으로 변환하는 것

  ① 희소 표현(Sparse Representation)

  :벡터 또는 행렬(matrix)의 값이 대부분이 0으로 표현되는 방법

  ex) 원-핫 벡터(One-Hot vector)는 희소 벡터(sparse vector), DTM은 희소 행렬

  - **문제점**: 메모리 낭비
    - 원-핫 벡터의 경우, 단어집합이 클수록 고차원 벡터가 됨
    - DTM에서 등장하지 않는 단어가 많은 경우, 0이 행렬의 많은 부분을 차지

  ② 밀집 표현(Dense Representation)
  - 사용자가 모든 단어에 대한 벡터 표현의 차원을 설정
  - 벡터의 값은 실수값
- 단어를 밀집 벡터(dense vector)의 형태로 표현하는 방법

  → 단어를 실수값의 벡터로 변환하는데 벡터의 차원을 사용자가 설정하는 것
- 워드 임베딩을 통해 표현된 밀집 벡터를 **임베딩 벡터(embedding vector)**라고도 함
- 방법: LSA, Word2Vec, FastText, Glove

2. 원-핫 인코딩(One-Hot Encoding)

- 텍스트 전처리 방법 중 하나
- 단어 집합의 크기를 벡터의 차원으로 하고, 표현하고 싶은 단어의 인덱스에 1의 값을 부여하고, 다른 인덱스에는 0을 부여하는 단어의 벡터 표현 방식
    - 이렇게 생성된 벡터를 원-핫 벡터(One-Hot vector)라 칭함

단어 집합(vocabulary)

: 서로 다른 단어들의 집합, 텍스트의 모든 단어 중복 없이 모은 것, 단어의 변형 형태도 다른 단어로 간주
ex) book과 books

과정

1. 단어 집합 생성
    - 단어의 개수로 단어집합의 크기 결정
2. 정수 인코딩

    :단어 집합의 각 단어에 고유한 정수를 부여하는 것
    - 단어 집합의 단어들마다 인덱스를 부여함

        ex) 5000개의 단어 존재 → 단어 집합 단어들에 1부터 5000번까지 인덱스 부여 → 150번째의 단어인 book은 150을 부여받음
3. 원-핫 인코딩
    - 표현하고 싶은 단어의 고유한 정수를 인덱스로 간주하고 해당 위치에 1을 부여하고, 다른 단어의 인덱스의 위치에는 0을 부여

예제) **문장 : 나는 자연어 처리를 배운다**

해당 문장에 대해 ① 문장 토큰화 수행(단어 집합 생성); ② 토큰 정수 인코딩; ③ 인덱스별 0과 1 부여;하여 원-핫 벡터를 생성해본다.

1. 문장 토큰화 수행

- Okt 형태소 분석기를 이용

```
['나', '는', '자연어', '처리', '를', '배운다']
```

2. 각 토큰에 정수 인코딩

```
단어 집합: {'나': 0, '는': 1, '자연어': 2, '처리': 3, '를': 4, '배운다': 5}
```

3. 표현하고 싶은 단어의 고유 정수 위치 인덱스에 1을 부여하고 다른 단어의 인덱스는 0을 부여함

   '자연어'는 인코딩한 정수가 2이므로 원-핫 벡터는 인덱스 2의 값이 1이며, 나머지 값은 0인 벡터임 → 자연어 = **[0, 0, 1, 0, 0, 0]**

+) 원-핫 인코딩 - 케라스(Keras)를 이용

- 원-핫 인코딩을 수행하는 방법 중 하나
- 케라스(Keras)가 지원하는 to_categorical()로 원-핫 인코딩 수행

예제) **문장: "나랑 점심 먹으러 갈래 점심 메뉴는 햄버거 갈래 갈래 햄버거 최고야"**

1. 단어 집합 생성
    - 케라스 토크나이저를 이용

```
단어 집합 : {'갈래': 1, '점심': 2, '햄버거': 3, '나랑': 4, '먹으러': 5, '메뉴는': 6, '최고야': 7}
```

2. 정수 인코딩

- 단어 집합에 있는 단어들로만 구성된 텍스트(sub_text)가 있다면, texts_to_sequences()를 통해서 정수 인코딩 가능

sub_text = "점심 먹으러 갈래 메뉴는 햄버거 최고야"

```
[2, 5, 1, 6, 3, 7]
```

3. 원-핫 인코딩

- 케라스가 지원하는 to_categorical() 이용
- 정수 인코딩된 결과로부터 수행

```
[[0. 0. 1. 0. 0. 0. 0. 0.] # 인덱스 2의 원-핫 벡터
 [0. 0. 0. 0. 0. 1. 0. 0.] # 인덱스 5의 원-핫 벡터
 [0. 1. 0. 0. 0. 0. 0. 0.] # 인덱스 1의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 1. 0.] # 인덱스 6의 원-핫 벡터
```

```
  [0. 0. 0. 1. 0. 0. 0. 0.] # 인덱스 3의 원-핫 벡터
  [0. 0. 0. 0. 0. 0. 0. 1.]] # 인덱스 7의 원-핫 벡터
```

원-핫 인코딩의 한계

- 단어의 개수 증가에 따라 벡터 차원 증가 → 벡터 저장을 위한 공간 필요
- 단어의 유사도 표현 불가 → 검색의 어려움 발생

➜ 단어의 잠재 의미를 반영하여 다차원 공간에 벡터화 하는 기법 두가지: 카운트 기반 벡터화 방법, 예측 기반 벡터화 방법

1. 카운트 기반의 벡터화 방법: LSA(잠재 의미 분석), HAL
2. 예측 기반 벡터화 방법: NNLM, RNNLM, Word2Vec, FastText
3. 카운트 기반과 예측 기반 두 가지 방법을 모두 사용하는 방법: GloVe

3. 카운트 기반의 단어 표현(Count based word Representation)

- 정보 검색과 텍스트 마이닝 분야에서 주로 사용되는 카운트 기반의 텍스트 표현 방법
  - DTM(Document Term Matrix)
  - TF-IDF(Term Frequency-Inverse Document Frequency)
- 문서의 핵심어 추출, 검색 엔진에서 검색 결과의 순위 결정, 문서들 간의 유사도를 구하는 등의 용도로 사용 가능

3-1. Bag of Words

: 단어들의 가방(직역)

- 단어들의 등장 순서는 전혀 고려하지 않고, 단어들의 출현 빈도(frequency)에만 집중하는 텍스트 데이터의 수치화 표현 방법

  → 분류 문제나 여러 문서 간의 유사도를 구하는 문제에 주로 쓰임
- BoW 생성 과정
  1. 단어 집합 생성
  2. 각 단어에 고유한 정수 인덱스를 부여
  3. 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터 생성

예제1) **문서1 : 정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다.**

```
vocabulary : {'정부': 0, '가': 1, '발표': 2, '하는': 3, '물가상승률': 4, '과': 5, '소비자': 6, '느끼는': 7, '은': 8,
'다르다': 9}
```

> 입력된 문서에 대해서 단어 집합(vocaburary)을 만들고 각 단어에 정수 인덱스를 할당함

```
bag of words vector : [1, 2, 1, 1, 2, 1, 1, 1, 1, 1]
```

> 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록함

> 인덱스 4에 해당하는 물가상승률은 두 번 언급됨 → 인덱스 4에 해당하는 값이 2

- BoW는 여러 문서의 단어 집합을 합친 뒤에, 해당 단어 집합에 대한 각 문서의 BoW를 구하기도 함

예제3) **문서3: 정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다. 소비자는 주로 소비하는 상품을 기준으로 물가상승률을 느낀다.**

문서3 = 문서1+문서2

문서1 : 정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다.

문서2 : 소비자는 주로 소비하는 상품을 기준으로 물가상승률을 느낀다.

```
vocabulary : {'정부': 0, '가': 1, '발표': 2, '하는': 3, '물가상승률': 4, '과': 5, '소비자': 6, '느끼는': 7, '은': 8,
'다르다': 9, '는': 10, '주로': 11, '소비': 12, '상품': 13, '을': 14, '기준': 15, '으로': 16, '느낀다': 17}
```

> 문서3의 단어 집합은 문서1과 문서2의 단어들을 모두 포함하고 있음
>
> 문서3에 대한 단어 집합을 기준으로 문서1, 문서2의 BoW를 생성

```
문서3 단어 집합에 대한 문서1 BoW : [1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
문서3 단어 집합에 대한 문서2 BoW : [0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 2, 1, 1, 1]
```

> 문서3 단어 집합에서 인덱스 4에 해당하는 "물가상승률"은 문서1에서는 2회 등장하며, 문서2에서는 1회 등장함

불용어를 제거한 BoW 만들기

- 불용어: 자연어 처리에서 별로 의미를 갖지 않는 단어들
- BoW를 만들때 불용어를 제거하여(전처리) 자연어 처리의 정확도를 높일 수 있음

  ∵ BoW: 텍스트 내의 중요 단어를 확인하려는 목적으로 문서에서 각 단어 빈도수 확인을 위해 사용 → 불용어 제거 필요

CountVectorizer: 영어 불용어 제거 기능 지원

- 영어의 BoW를 만들기 위해 사용하는 CountVectorizer는 지정한 불용어를 제외하고 BoW를 만들 수 있음

(1) 사용자가 직접 정의한 불용어 사용

```
text = ["Family is not an important thing. It's everything."]
vect = CountVectorizer(stop_words=["the", "a", "an", "is", "not"])
```

```
bag of words vector : [[1 1 1 1 1]]
vocabulary : {'family': 1, 'important': 2, 'thing': 4, 'it': 3, 'everything': 0}
```

(2) CountVectorizer에서 제공하는 자체 불용어 사용

```
text = ["Family is not an important thing. It's everything."]
vect = CountVectorizer(stop_words="english") #영어 불용어 지원
```

```
bag of words vector : [[1 1 1]]
vocabulary : {'family': 0, 'important': 1, 'thing': 2}
```

3-2. DTM(Document-Term Matrix, 문서 단어 행렬)

- 서로 다른 문서들의 BoW를 하나의 행렬로 결합한 표현 방법 즉, BoW 표현을 다수의 문서에 대해서 행렬로 표현하고 부르는 용어
- 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것 → 문서간 비교 가능
- 행과 열을 반대로 선택하면 TDM

예제) 4개의 문서에 대해 띄어쓰기 단위 토큰화를 수행한다 가정하고, 문서 단어 행렬로 표현

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

각 문서에서 등장한 단어의 빈도를 행렬의 값으로 표기

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

문서 단어 행렬의 한계

- 많은 양의 저장 공간과 높은 계산 복잡도를 요구

  ∵ 희소 표현(Sparse representation): 대부분의 값이 0인 표현
- 단순 빈도 수 기반 접근

  ex) 'the'의 빈도수가 높다고 문서가 유사항 문서라고 판단할 수 없음

→ DTM에 불용어와 중요한 단어에 대해서 가중치를 주는 아이디어를 적용한 TF-IDF

3-3. TF-IDF(Term Frequency-Inverse Document Frequency)

: 단어 빈도 역 문서 빈도(직역)

- 단어의 빈도와 역 문서 빈도(문서의 빈도에 특정 식을 취함)를 사용하여 DTM 내의 각 단어들마다 중요한 정도를 가중치로 주는 방법

- 기존의 DTM을 사용하는 것보다 많은 정보를 고려하여 문서를 비교할 수 있음 → 대부분 DTM보다 고성능
- 문서의 유사도를 구하는 작업, 검색 시스템에서 검색 결과의 중요도를 정하는 작업, 문서 내에서 특정 단어의 중요도를 구하는 작업 등에 사용
- 과정
  1. DTM 생성
  2. TF-IDF 가중치를 부여
- TF-IDF = TF x IDF
  - 문서: d, 단어: t, 문서의 총 개수: n

  ① TF

  - tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수
  - DTM의 예제에서 각 단어들이 가진 값(해당 단어의 등장 빈도를 나타내는 값)들과 동일

  ② DF

  - df(t) : 특정 단어 t가 등장한 문서의 수
  - 문서에서 특정 단어가 등장한 빈도 수와는 관련 없음

> 문서1 : 먹고 싶은 사과
> 문서2 : 먹고 싶은 바나나
> 문서3 : 길고 노란 바나나 바나나
> 문서4 : 저는 과일이 좋아요
>
> | | 과일이 | 길고 | 노란 | 먹고 | 바나나 | 사과 | 싶은 | 저는 | 좋아요 |
> |---|---|---|---|---|---|---|---|---|---|
> | 문서1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
> | 문서2 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
> | 문서3 | 0 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 |

> | | 과일이 | 길고 | 노란 | 먹고 | 바나나 | 사과 | 싶은 | 저는 | 좋아요 |
> |---|---|---|---|---|---|---|---|---|---|
> | 문서4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
>
> ∴ df(바나나) = 2
>
> ∵ 문서2와 문서3에서 바나나가 등장, 문서3에 바나나가 두 번 등장했으나 무관함

③ IDF

- idf(d, t) : df(t)에 반비례하는 수

  ![image-20230711090941909](/Users/seongeun/Library/Application Support/typora-user-images/image-20230711090941909.png)

- 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단하며, 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단
- TF-IDF 값이 크면 중요도가 큼
  - the, a와 같은 불용어의 TF-IDF의 값은 다른 단어의 TF-IDF에 비해 상대적으로 낮음

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

> - TF: DTM을 그대로 사용, 각 문서에서 각 단어의 TF
> - IDF
>   - 로그의 밑은 TF-IDF를 사용하는 사용자가 임의로 정할 수 있음(주로 밑이 e인 자연로그 사용)
>   - 문서의 총 수: 4

> - DF: 각 단어가 등장한 문서의 수

단어	**IDF(역 문서 빈도)**
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

> - DF('먹고')= 2
>   - 총 2개의 문서(문서1, 문서2)에 등장
> - IDF는 여러 문서에서 등장한 단어의 가중치를 낮추는 역할
>   - 문서 1개에만 등장하는 단어인 '사과'와 문서 2개에 등장하는 단어 '먹고'의 IDF 값 차이가 큼

- TF-IDF = TF x IDF

각 단어의 TF는 DTM에서의 각 단어의 값과 동일 → DTM에서 단어 별로 위의 IDF값을 곱하여 TF-IDF 도출

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

> - 문서2에서의 바나나의 TF-IDF 가중치와 문서3에서의 바나나의 TF-IDF 가중치가 다름
> - 문서3에서 '바나나'의 중요도가 큰 것

원-핫 벡터와 임베딩 벡터의 차이

	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

4-1. 워드투벡터(Word2Vec)

- 예측 기반 단어 표현
- 단어 벡터 간 유의미한 유사도 반영 가능
  - 한국 - 서울 + 도쿄 = 일본

    → 각 단어 벡터가 단어 벡터 간 유사도를 반영한 값을 가지고 있으므로 단어의 의미로 연산됨
- 은닉층이 1개인 얕은 신경망(shallow neural network)
  - 일반적인 은닉층과는 달리 활성화 함수가 존재하지 않음
  - 룩업 테이블이라는 연산을 담당함
  - 투사층(projection layer)이라 칭하기도 함
- 분산표현방법을 위한 대표적인 학습 방법임

  : 그 단어를 표현하고자 주변을 참고하여 단어를 표현하는 방법
  - 저차원에 **단어의 의미를 여러 차원에다가 분산** 하여 표현
  - **단어 벡터 간 유의미한 유사도** 계산 가능
- Word2Vec의 학습 방식

  ① CBOW(Continuous Bag of Words): 주변에 있는 단어들을 입력으로 중간에 있는 단어들을 예측하는 방법

  ② Skip-Gram: 중간에 있는 단어들을 입력으로 주변 단어들을 예측하는 방법

① CBOW(Continuous Bag of Words)

- 주변에 있는 단어들을 입력으로 중간에 있는 단어들을 예측하는 방법
  - 중심 단어(center word): 예측해야하는 단어
  - 주변 단어(context word): 예측에 사용되는 단어
- 윈도우: 중심 단어를 예측하기 위해 볼 주변 단어의 범위

- 윈도우 크기가 n이면, 실제 중심 단어를 예측하기 위해 참고하려고 하는 주변 단어의 개수는 2n

예문 : "The fat cat sat on the mat"

윈도우 크기가 2일 때, ['The', 'fat', 'cat', 'on', 'the', 'mat']으로부터 sat을 예측

→ 'sat': 중심단어

→ 입력: 'fat', 'cat', 'on', 'the'

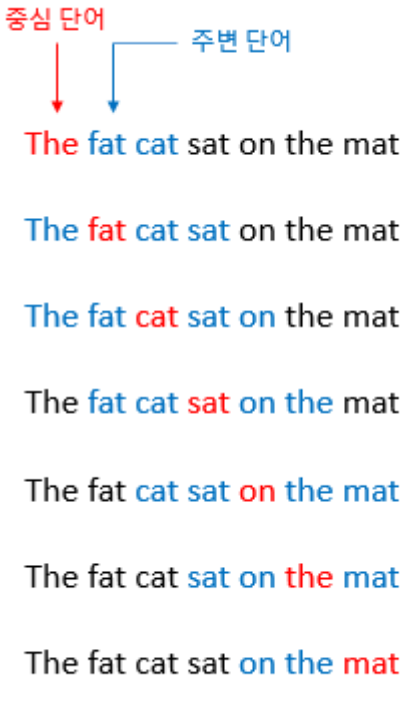
슬라이딩 윈도우(sliding window)

: 윈도우 크기에 따라 윈도우를 움직여 주변 단어와 중심 단어의 선택을 변경해가며 학습을 위한 데이터 셋을 만드는 방법

Word2Vec에서 입력은 모두 원-핫 벡터가 되어야 하는데,

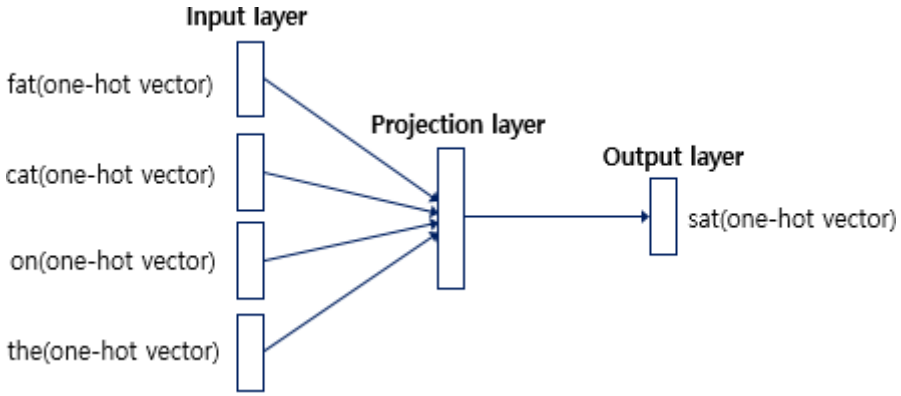
- 윈도우 크기에 따른 주변 단어의 변화
- 중심 단어와 주변 단어에 따른 원-핫 벡터 결과

전체 데이터 셋:



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

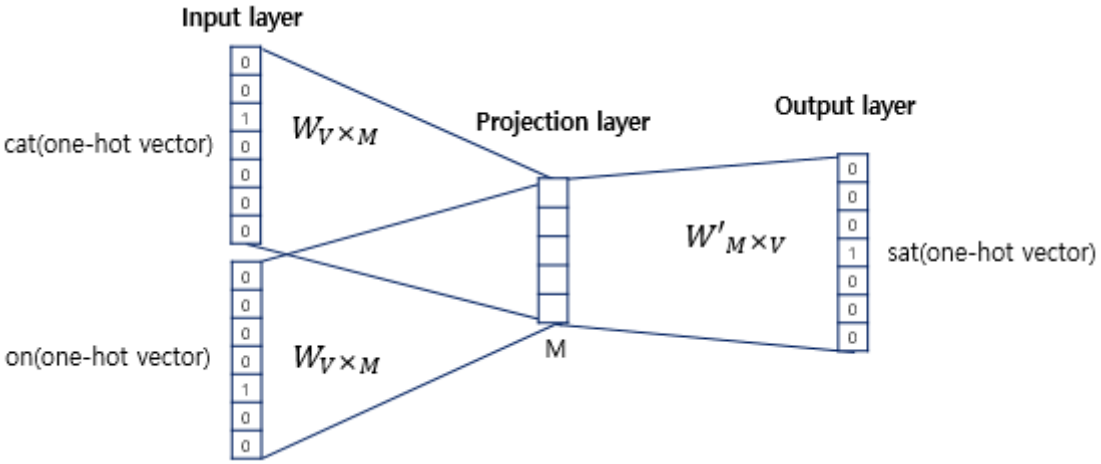
CBOW의 인공 신경망 도식화



- 입력층(Input layer)에 주변 단어들의 원-핫 벡터가 입력됨

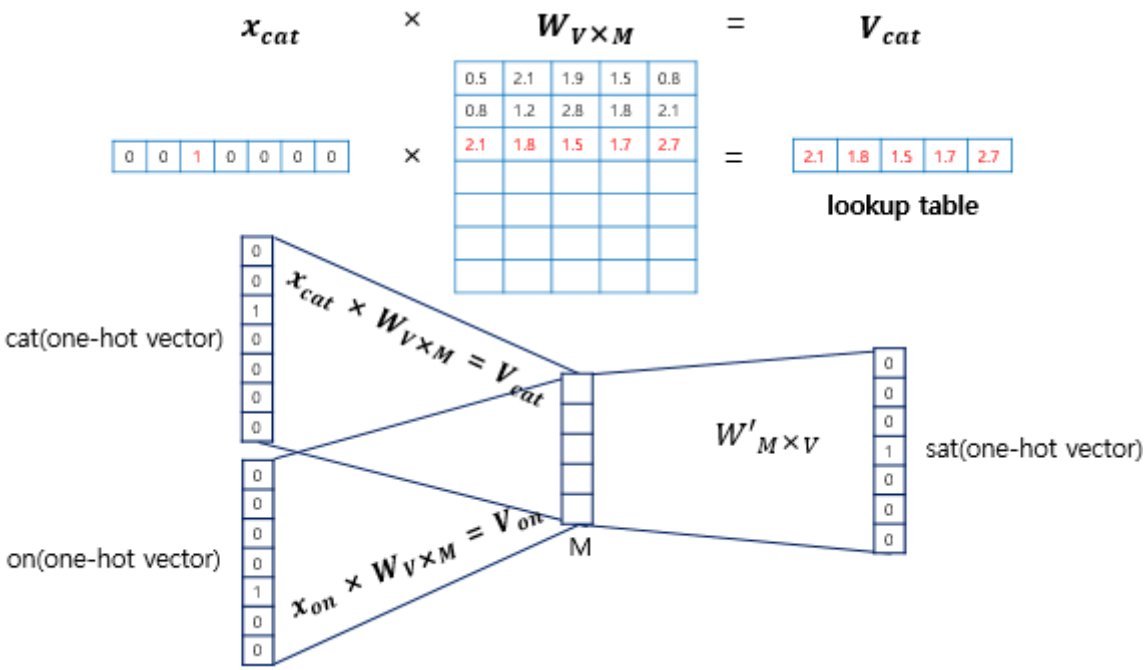
- 출력층(Output layer)에서 예측하고자 하는 중간 단어의 원-핫 벡터가 레이블로서 필요

CBOW의 인공 신경망

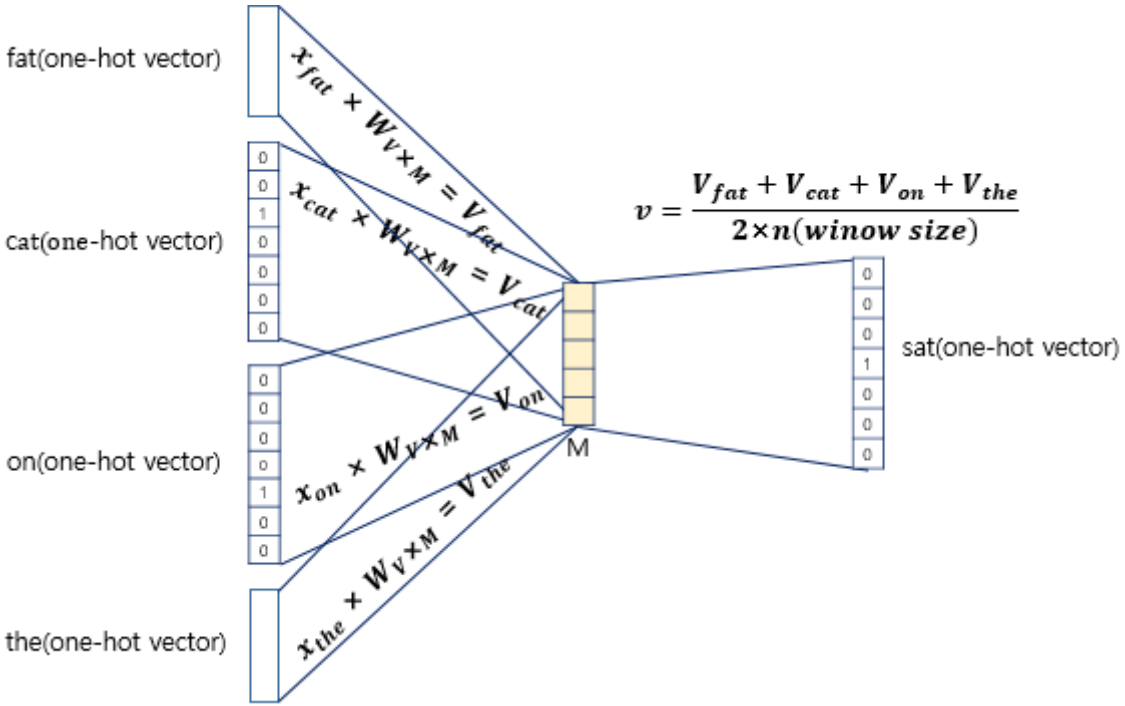


- 투사층의 크기: M

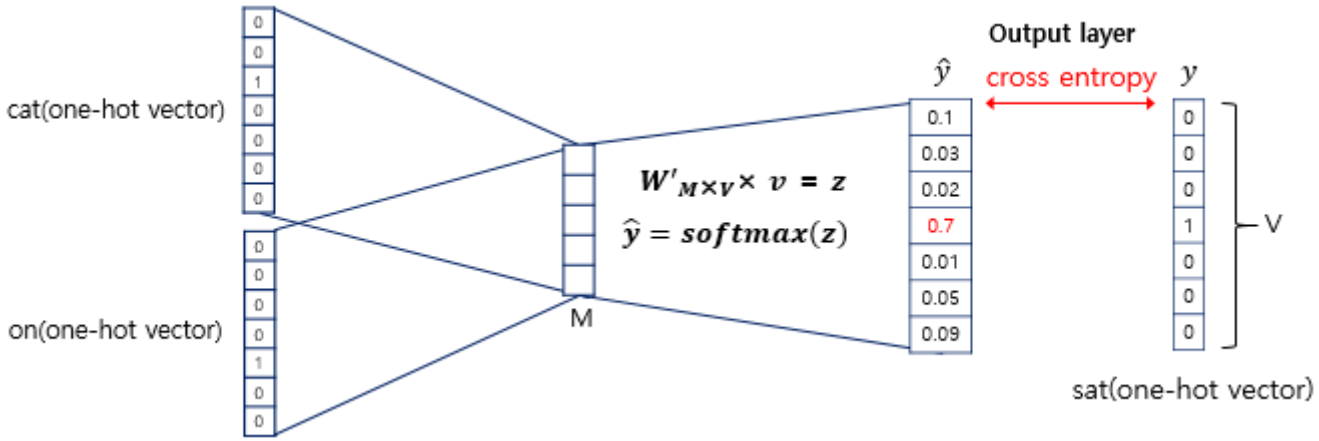
  CBOW에서 투사층의 크기 M은 임베딩하고 난 벡터의 차원이 됨
- 입력층과 투사층 사이의 가중치 W는 V × M 행렬, 투사층에서 출력층사이의 가중치 W'는 M × V 행렬
  - V: 단어 집합의 크기(원-핫 벡터의 차원)
  - M이 5라면 가중치 W는 7 × 5 행렬, W'는 5 × 7 행렬
  - W와 W'는 동일한 행렬을 전치(transpose)한 것이 아닌 서로 다른 행렬



- 입력 벡터: 원-핫 벡터
- 입력 벡터와 가중치 W의 곱인 V는 W행렬의 i(입력 벡터에서 1이 있는 인덱스 위치)번째 행
  - 룩업 테이블(lookup table)



- 투사층에서 입력의 원-핫 벡터와 가중치 W의 곱 결과 벡터들에 대해 평균인 벡터를 구함
  - 윈도우 크기 n=2일 때, 입력 벡터의 총 개수는 4개
  - 4개의 결과 벡터에 대한 평균 구함

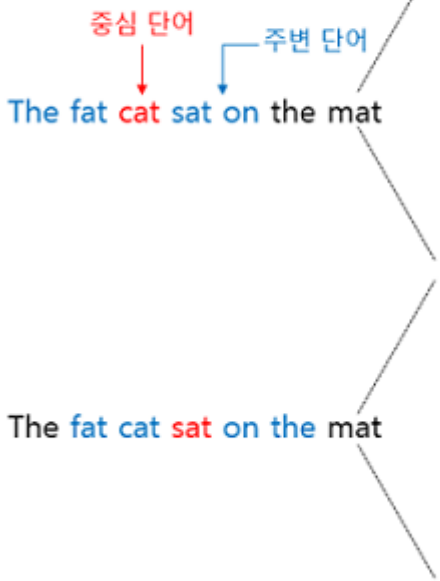


- 평균 벡터와 두번째 가중치 행렬 W'를 곱하여 얻은 z
  - 원-핫 벡터의 차원과 동일한 차원
- z는 소프트맥스(softmax) 함수를 지나면서 벡터의 각 원소들의 값은 0과 1사이의 실수로, 총 합은 1이 됨
  - 일종의 스코어 벡터(score vector)
  - 스코어 벡터의 j번째 인덱스가 가진 값은 j번째 단어가 중심 단어일 확률을 나타냄
  - 스코어 벡터의 값은 레이블에 해당하는 벡터인 중심 단어 원-핫 벡터의 값에 가까워져야 함
- CBOW는 인공 신경망의 훈련 전에 랜덤 값을 가지던 가중치 행렬 W와 W'를 학습해가는 구조

② Skip-gram

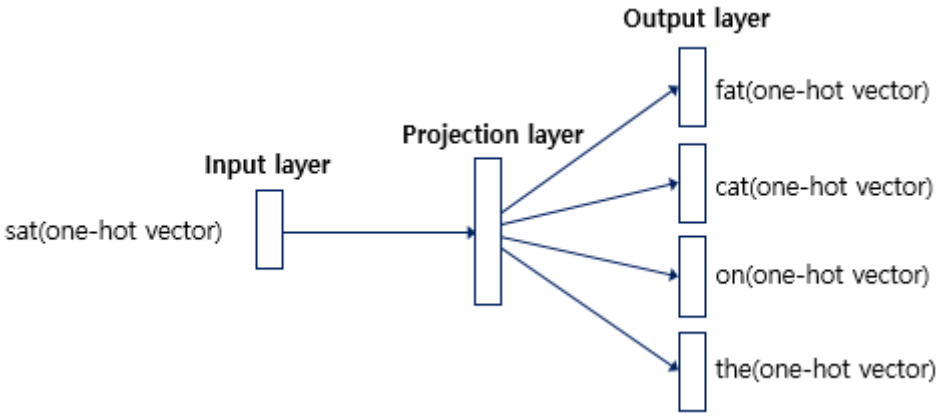
- 중심 단어에서 주변 단어를 예측하는 방법
- CBOW보다 성능이 좋음

윈도우 크기 n = 2인 경우의 전체 데이터셋



중심 단어	주변 단어
cat	The
cat	Fat
cat	sat
cat	on
sat	fat
sat	cat
sat	on
sat	the

인공 신경망을 도식화



- 중심 단어에 대해서 주변 단어를 예측하므로 투사층에서 벡터들의 평균을 구하는 과정이 없음

4-2.패스트텍스트(FastText)

- facebook에서 개발
- 내부 단어를 고려하여 학습
- Word2Vec의 확장

Word2Vec	FastText
단어를 쪼개질 수 없는 단위로 간주	하나의 단어 안에도 여러 단어들이 존재하는 것으로 간주 내부 단어(서브워드, subword)를 고려하여 학습
단어집합에 존재하지 않는 단어에 대해 유사한 단어를 찾아낼 수 없음	단어집합에 존재하지 않는 단어의 유사한 단어를 찾을 수 있음

내부 단어(subword)의 학습

- 각 단어는 글자 단위 n-gram의 구성으로 취급
  - n에 따라서 단어들이 얼마나 분리되는지 결정됨

```
# n = 3인 경우
<ap, app, ppl, ple, le>
```

> - n = 3으로 트라이그램(tri-gram)
> - apple은 app, ppl, ple로 분리하고, 시작과 끝을 의미하는 <, >를 도입하여 아래의 5개 내부 단어(subword) 토큰을 벡터화

```
# 특별 토큰
<apple>
```

> - 기존의 단어에 <>를 붙인 특별 토큰 생성

```
# n = 3인 경우, FastText는 단어 apple에 대해서 다음의 6개의 토큰을 벡터화함
<ap, app, ppl, ple, le>, <apple>
```

- '내부 단어들을 벡터화한다' = '내부단어들에 대해서 Word2Vec을 수행한다'
- 단어 apple의 벡터값은 내부 단어들의 벡터값의 총 합으로 구성

```
apple = <ap + app + ppl + ppl + le> + <apple>
```

FastText의 강점

(1) 모르는 단어(Out Of Vocabulary, OOV)에 대한 대응

FastText의 인공 신경망을 학습한 후, 데이터 셋의 모든 단어는 각 n-gram에 대해서 워드 임베딩 됨

→ 데이터 셋만 충분한다면 내부 단어(Subword)를 통해 다른 단어와 OOV의 유사도 계산 가능

(2) 단어 집합 내 빈도 수가 적었던 단어(Rare Word)에 대한 대응

- Word2Vec의 경우, 등장 빈도 수가 적은 단어(rare word)에 대해서는 임베딩의 정확도가 높지 않음
  - 참고할 수 있는 경우의 수가 적어 정확하게 임베딩이 되지 않는 것
- 희귀 단어의 n-gram이 다른 단어의 n-gram과 겹치면 상대적으로 높은 임베딩 벡터값 얻음
  - 희귀 단어: 등장 빈도수가 매우 적은 단어 (∵ 오타, 틀린 맞춤법 etc.)

  → 노이즈가 많은 코퍼스에서 강점을 가진 것

5. 글로브(GloVe, Global Vectors for Word Representation)

- 카운트 기반과 예측 기반을 모두 사용하는 단어 임베딩 방법
- 기존의 카운트 기반 LSA(Latent Semantic Analysis)와 예측 기반 Word2Vec의 단점 보완 목적으로 생성됨

    - LSA: 코퍼스의 전체적 통계 정보 고려하나 단어 의미 유추 작업 성능이 좋지 않음
    - Word2Vec: 단어 간 유추 작업은 상대적으로 좋으나 임베딩벡터가 윈도우 크기 내의 단어만 고려하므로 코퍼스 전체의 통계 정보 반영 불가
- 동시 등장 행렬로부터 동시 등장 확률 구함

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

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

예시) 동시 등장 행렬

- window size: 1
- 3개의 문서 데이터
    - I like deep learning
    - I like NLP
    - I enjoy flying

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

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

동시 등장 확률(Co-occurrence Probability)

: P(k|i)

- 동시 등장 행렬로부터 특정 단어 i의 전체 등장 횟수를 카운트하고, 특정 단어 i가 등장했을 때 어떤 단어 k가 등장한 횟수를 카운트하여 계산한 조건부 확률
- 중심 단어 i의 행의 모든 값을 더한 값을 분모로 하고 i행 k열의 값을 분자로 한 값
  - i: 중심 단어(Center Word)
  - k: 주변 단어(Context Word)
  - P(like|deep) = 1/2

예시) 동시 등장 확률 표

- GloVe 제안 논문에서 발췌

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

> - P(solid I ice) = 0.00019, ice가 등장했을 때 solid가 등장할 확률
> - P(solid|steam) = 0.000022, steam이 등장했을 때 solid가 등장할 확률

- P(solid I ice) > P(solid|steam) → solid('단단한')은 steam('증기')보다 ice('얼음')라는 단어와 더 자주 등장하는 것이 자명