1. 희소표현(Sparse vector)란?

희소벡터란 원-핫 벡터처럼 표현하고자 하는 단어의 인덱스만 1이고 나머지는 모두 0인 형태의 벡터이다. 희소벡터는 공간낭비가 심하고 단어의 의미를 표현하지 못한다는 단점이 있다. 이에 반해, 정해진 차원의 벡터로 단어를 표현하는 밀집 벡터로 임베딩 벡터가 있다. 값은 1과 0뿐만 아니라 실수로 구성되어 있다. 정해진 차원의 벡터로 표현하기 때문에 단어의 수가 늘어나도 공간적인 낭비를 막을 수 있고 단어의 의미를 담을 수도 있다.

2. 분산 표현(Distributed Representation)이란?

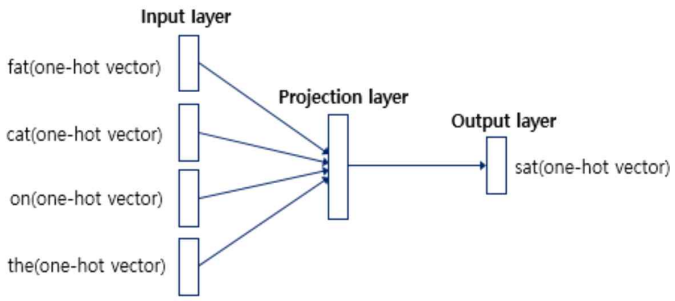
이렇게 단어의 의미를 다차원 공간에 벡터화하기 위한 방법으로 분산 표현이라는 것이 있다. 분산 표현은 기본적으로 분포 가설(distributed hypothesis)이라는 가정 하에 이뤄지는데, 이 가정은 **'비슷한 위치에 있는 단어는 비슷한 의미를 갖는다'** 는 가정이다. 분산 표현은 단어의 개수보다 **작은 차원에 걸쳐 단어의 의미를 분산하여 표현**한다고 볼 수 있다. 이렇게 표현된 임베딩 벡터는 단어의 의미를 내포하기 때문에 단어간의 유사도를 구할 수 있다.
분산 표현을 이용하여 단어를 임베딩 벡터로 표현하는 방법은 NNLM, RNNLM 등이 사용되었지만 요새는 주로 Word2Vec을 사용하고 있다.

3. Word2Vec

Word2Vec은 중심단어와 주변단어 벡터의 내적이 코사인 유사도가 되도록 단어벡터를 벡터공간에 임베딩하는 것이다. (CBOW를 예로 들면, 주변단어벡터와 중심단어벡터의 코사인 유사도가 증가하면, 소프트맥스 함수의 해당 인덱스의 값이 커지기 때문이다. 해당 인덱스의 값은 해당 인덱스의 단어가 중심단어가 될 확률이고, 그 값이 커지도록 학습되기 때문에 이렇게 말할 수 있다.) Word2Vec에는 CBOW와 Skip-gram이 있다. CBOW는 주변 단어들을 이용하여 중간 단어를 예측하는 방법이고, Skip-gram은 중간 단어를 이용하여 주변 단어를 예측하는 방식이다.

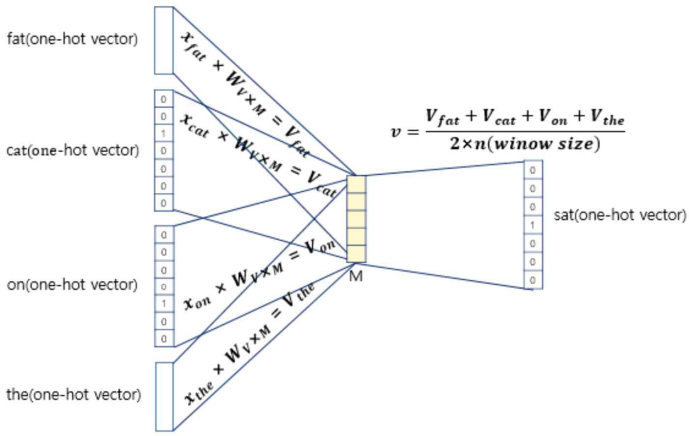
3-1 CBOW

"The fat cat sat on the mat"라는 예문이 있다고 하자. ["The", "fat", "cat", "on", "the", "mat"]로부터 "sat"을 **예측**하는 것이 CBOW가 하는 일이다. 중심 단어를 예측하기 위해 앞, 뒤로 몇개의 단어를 볼 지를 결정해야 하는데 이 범위를 윈도우라고 한다. 윈도우의 크기가 2개이면, 앞 뒤로 2개, 총 4개의 Context word를 참고해야 한다. 다시 말해, 윈도우의 크기가 n이면, 2*n 의 context word를 참고해야 한다. 윈도우가 결정되면, 윈도우 슬라이딩을 통해 중심 단어를 바꿔가면서 학습을 위한 데이터 셋을 만들어 간다.



위의 그림은 CBOW의 인공 신경망을 도식화 한 것이다. CBOW의 입력과 출력은 모두 원-핫 벡터이다. 위 그림으로부터 알 수 있는 것은 Word2Vec은 은닉층이 하나이기 때문에 딥 러닝 모델은 아니라는 것이다. 또, Word2Vec의 은닉층은 다른 신경망과는 다르게, 투사층이라고 부른다.
CBOW의 projection layer의 크기가 M이라는 것은, 투사층을 지나고 난 후, 즉 임베딩 벡터의 차원이 M이라는 것이다. 따라서, 입력층과 투사층 사이의 가중치 W는 VxM이고, 투사층과 출력층 사이의 가중치 W'는 MxV이다. 이때 V는 단어 집합의 크기이면서 동시에 원-핫 벡터의 크기이다.

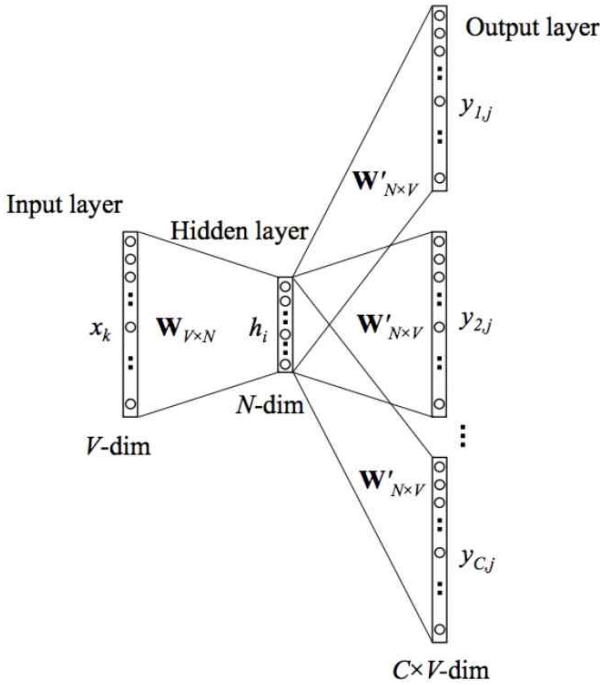
입력층과 projection layer사이에서 가중치 행렬 W가 어떻게 곱해지는지를 보자. 입력 벡터는 원-핫 벡터이므로 i번째 인덱스만 1의 값을 갖고 있다. 이때 입력 벡터와 가중치 행렬 W의 곱은 사실 W행렬의 i번째 행을 그대로 읽어 오는 것(Look-up)이다. 따라서 이런 작업을 look-up table이라고 한다. **CBOW의 가중치 행렬 W는 사실 Word2Vec을 수행한 후의 M차원의 임베딩 벡터들 그 자체이다.**



이렇게 look-up table을 통해 주변 단어(Context word)들의 임베딩 벡터가 구해지면, 이들의 평균이 바로 중심단어(Center word)의 임베딩 벡터가 된다. 위의 예시에서 n=2이기 때문에 4개의 주변 벡터의 평균 값이 'sat'의 임베딩 벡터가 된다.
중심단어의 임베딩 벡터가 구해지면, projection layer에서 출력층으로의 가중치 행렬 W'와 곱해지고, 그 결과로 입력 벡터인 원-핫 벡터와 동일한 차원의 벡터가 나온다.
CBOW에서는 이 벡터에 소프트맥스 함수를 취하는데, 이를 스코어 벡터라고 한다. 스코어 벡터의 i번째 인덱스의 값이 의미하는 바는 i번째 단어가 중심 단어일 확률을 의미한다.
이 스코어 벡터와 중심 단어의 실제 원-핫 벡터에 대해 cross entropy loss를 구하고 역전파를 통해 W와 W'를 학습한다.

3-2 Skip-gram

Skip-gram은 CBOW와 다르게 중심단어로부터 주변 단어를 **예측**한다. Skip-gram의 인공 신경망을 도식화 해보면 다음과 같다.



이때 V는 단어 집합의 크기이고, N은 임베딩 벡터의 차원이다. Skip-gram은 아래 식을 최대화하는 방향으로 학습을 진행한다.

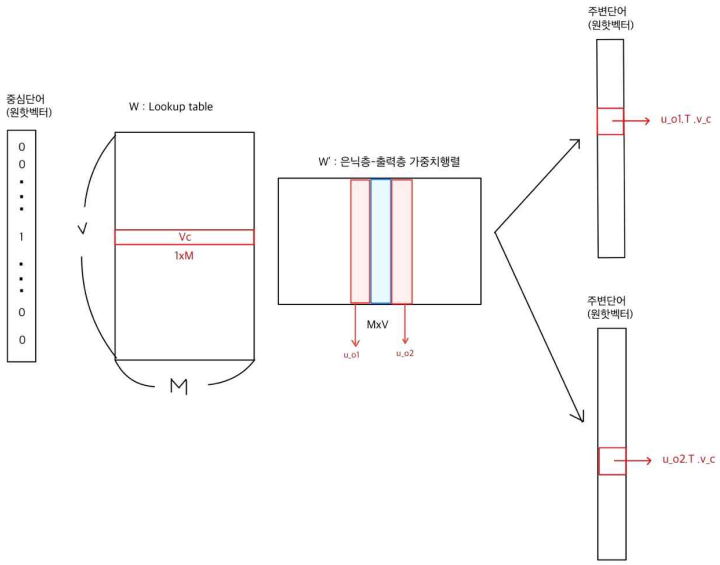
$$P(o_i|c) = \frac{\exp(u_{o_i}{}^T v_c)}{\sum_{w=1}^{W} \exp(u_w{}^T v_c)}$$

$\sum_{w=1}^{W} u_w^T v_c$: 가중치 행렬 W'와 중심단어 간의 가중치 합(내적)

$u_{o^i}^T v_c$: 중심단어와 가중치 행렬 W'의 가중치 합의 결과(원핫벡터) 중 주변단어 i에 해당하는값
(밑의 그림 참조)

(위의 식은 소프트맥스 함수, c : center word, o : context word)

우변의 v는 입력층과 은닉층을 잇는 가중치 행렬 W의 행벡터(중심단어의 임베딩 벡터), u는 은닉층과 출력층을 잇는 가중치 행렬 W'의 열벡터이다. 분자의 지수를 높이는 것은 중심단어에 해당하는 벡터와 주변단어에 해당하는 벡터의 내적값을 높인다는 뜻이다. **벡터 내적은 cosine 값이므로 내적값을 높이는 것은 단어 벡터간의 유사도를 높인다는 뜻**으로 해석할 수 있다.



***Negative sampling : Word2Vec의 학습트릭**

Word2Vec은 출력층이 내놓은 스코어 값에 소프트맥스 함수를 적용해 확률값으로 변환한 후 이를 정답(원핫벡터)과 비교해 역전파하는 구조이다. 그런데 소프트맥스의 분모를 구하려면 W'와 중심단어의 벡터의 내적을 해줘야 하는데 W'의 shape은 MxV이므로 단어 집합의 크기가 커지면 계산량이 많아진다.

따라서 소프트맥스의 분모를 구할 때, 전체 단어를 대상으로 하지 않고, 일부 단어만 랜덤으로 뽑아서 계산량을 줄이는 것이 Negative sampling이다.

Negative sampling의 과정은 다음과 같다.

(1) 윈도우 내에 등장하지 않는 단어를 5~20개 뽑는다.

(2) 이렇게 뽑은 단어와 정답 단어를 합쳐 전체 단어로 간주하고 소프트맥스 함수를 구한다.

(3) 이 단어들에 대해서만 가중치 업데이트를 한다.

*SGNS(Skip-gram with Negative Sampling)

일반적인 Skip-gram에서는 중심단어를 입력하면 주변 단어를 예측하는 방식으로 학습한다. 하지만 SGNS에서는 중심단어와 두 종류의 입력단어(이웃단어, 이웃이 아닌 단어=negative sample)를 입력했을 때, 두 단어가 윈도우 내에 존재할 확률(서로 이웃일 확률)을 구하는 방식으로 학습한다. 과정은 다음과 같다.

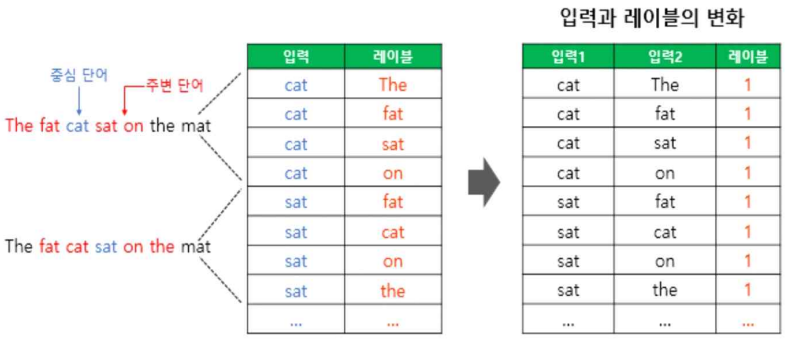
(1) 레이블이 1 (이웃관계)인 데이터 준비

(2) 단어 집합에서 이웃관계가 아닌 단어들을 랜덤으로 선택하여 레이블 0의 샘플로 추가

(3) 이때 기준이 되는 입력(중심단어)를 입력1로, 주변단어(이웃이 아닌 단어도 포함)를 입력2로 설정하여 각각의 Look-up table 만들기

(4) 입력1의 단어 벡터(중심단어)와 입력2의 단어벡터(주변단어) 중 샘플링 된 단어들의 벡터를 내적하여 예측값을 구하고 손실함수를 통해 임베딩 벡터를 업데이트한다.

1. 이웃 관계인 단어들을 레이블 1로 설정하여 데이터셋 준비

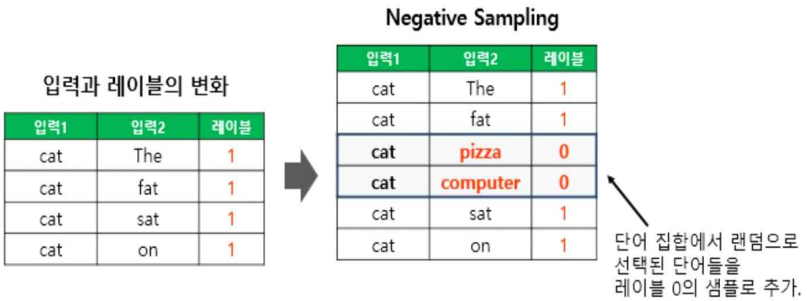


입력	레이블
cat	The
cat	fat
cat	sat
cat	on
sat	fat
sat	cat
sat	on
sat	the
...	...

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	sat	1
cat	on	1
sat	fat	1
sat	cat	1
sat	on	1
sat	the	1
...	...	...

↓

2. 단어집합에서 이웃관계가 아닌 단어를 랜덤으로 선택해서 레이블 0으로 샘플에 추가



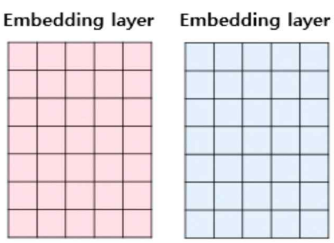
입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	sat	1
cat	on	1

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	**pizza**	**0**
cat	**computer**	**0**
cat	sat	1
cat	on	1

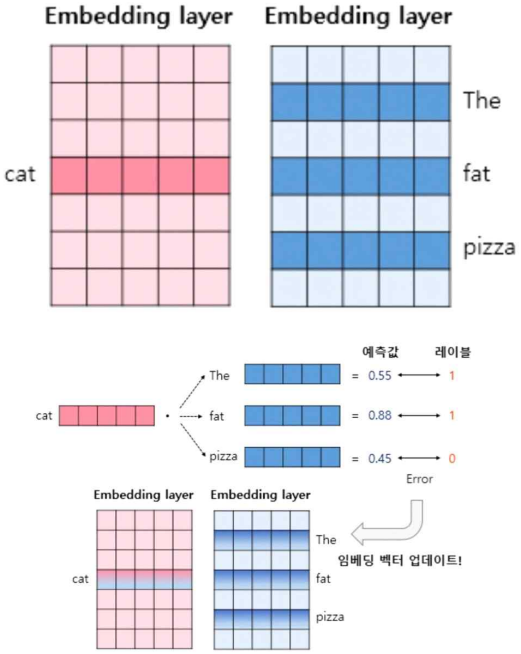
↓

3. 중심단어를 입력 1, 주변 단어(negative sample도 포함)을 입력2로 하고 각각의 projection layer를 설정

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	pizza	0
cat	computer	0
cat	sat	1
cat	on	1
cat	cute	1
cat	mighty	0
...	...	...



4. 입력1의 중심단어 벡터와 입력2의 주변단어 중 샘플링 된 단어들의 벡터를 내적하여 예측값을 구하고 손실함수를 통해 임베딩 벡터를 업데이트한다.



Summary :

Word2Vec의 main idea는 중심벡터와 주변벡터의 내적이 코사인 유사도가 되도록 단어 벡터를 벡터 공간에 임베딩 하는 알고리즘이다. Word2Vec은 CBOW, Skip-gram의 두 가지 방식이 있다. CBOW는 주변 단어를 통해 중심 단어를 예측하는 과정을 통해 단어를 임베딩 하는 것이고, Skip-gram은 중심 단어를 통해 주변 단어를 예측하는 과정에서 단어를 임베딩 하는 것이다. 이때 주변단어인지 아닌지를 결정하는 것을 윈도우라고 하고, 윈도우의 크기를 n이라고 하면, 중심 단어로부터 전, 후 n개의 단어까지가 윈도우의 범위에 있다.

CBOW는 윈도우 내의 주변단어들의 원 핫 벡터를 입력으로 받는다. 입력 벡터는 projection layer를 거치는데, 입력층에서 은닉층으로의 가중치 행렬 W(v x m)의 행 중에서 원 핫 벡터가 1인 값의 인덱스와 동일 인덱스를 그대로 읽어오는 look up table 방식으로 임베딩 벡터를 얻는다. 이렇게 얻은 주변 단어들의 임베딩 벡터의 평균이 중심 단어의 임베딩 벡터가 된다. 중심 단어의 임베딩 벡터는 은닉층에서 출력층으로의 가중치 행렬 W'(m x v)와 내적하여 단어 집합크기의 벡터가 된다. 이때 i번째 값은 단어집합의 i번째 단어가 중심단어가 될 확률을 의미한다. 중심단어의 원 핫 벡터와 cross entropy loss를 구하고 이를 토대로 업데이트 된다.

Skip-gram은 중심단어의 원 핫 벡터를 입력으로 받는다. 마찬가지로 입력층에서 은닉층으로의 가중치 행렬 W(v x m)의 행 중에서 중심 단어에 해당하는 인덱스의 행을 look up table 하는 방식으로 중심 단어의 임베딩 벡터를 얻는다. 은닉층에서 출력층으로의 가중치 행렬 W'(m x v)와 내적하여 단어 집합 크기의 벡터가 된다. 이때 i번째 값은 단어 집합의 i 번째

단어가 주변 단어가 될 확률을 의미한다. 실제로 해당되는 주변 단어의 원 핫 벡터와 cross entropy loss를 구하고 이를 토대로 업데이트 된다.
Word2Vec은 윈도우가 이동해 가면서 매번 단어 벡터와 W' 간의 연산, 업데이트가 이뤄지기 때문에 단어 집합의 크기가 커지면 많은 시간을 필요로 한다. 연산량을 줄이기 위해서 negative sampling을 도입한다. negative sampling은 중심단어와 이웃관계에 있는 주변 단어 샘플에 이웃관계가 아닌 임의의 단어샘플(negative sample)을 추가하여 샘플을 만든다. 중심단어를 입력 1, 주변 단어(negative sample 포함)를 입력 2로 하여 각각의 projection layer를 형성한다. 준비한 샘플에 대해서 입력 1의 중심단어 벡터와 입력 2의 주변 단어 벡터를 내적하여 예측값을 구하고 loss function을 계산하여 업데이트 한다.