NLP 실전 프로젝트 2020-2
Word Embedding

교육학과 15 유정아

워드 임베딩

| 컴퓨터가 이해할 수 있도록 자연어를 적절히 변환하는 것

| 단어를 밀집 벡터의 형태로 표현하는 방법

동의어: 임베딩 벡터

Word2Vec, GloVe, LSA, FastText 등 방법론

Word2Vec과 GloVe 둘 다 사용해보고 성능 더 좋은 것 선택해야

희소 표현(Sparse representation)

| 벡터 또는 행렬의 값이 대부분 0으로 표현되는 방법

Ex) 원-핫 벡터 (표현하고자 하는 단어 인덱스 값만 1)

문제점

단어의 개수가 늘어나면 벡터의 차원이 한 없이 커짐

공간적 낭비 야기

단어의 의미를 포함하지 못함

Ex) 가지고 있는 말뭉치에 단어가 10,000개라면 벡터의 차원은?

밀집 표현(Dense Representation)

| 사용자가 설정한 값으로 모든 단어의 벡터 표현 차원 맞춤

Ex) 밀집표현 차원 == 128 이라면 모든 단어 벡터 표현 차원이 128로 바뀌면서 모든 값이 실수가 됨

벡터의 차원이 조밀해져서 밀집 벡터라고 함

분산 표현(distributed representation)

| 단어의 의미를 다차원 공간에 벡터화하는 방법

분포 가설 가정 == ‘비슷한 위치에서 등장하는 단어들은 비슷한 의미를 가진다’

분포 가설을 이용해 단어들의 집합을 학습하고 벡터에 단어의 의미를 여러 차원에 분산해서 표현

저차원에 단어의 의미를 여러 차원에다가 분산해서 표현 -> 단어 간 유사도 계산 가능

Ex) 강아지 = [0 0 0 1 0 0 0 0 0 0 … , … 0]

Ex) 강아지 = [0.2 0.3 0.6 … , … 0.3]

분산 표현을 이용해서 단어의 유사도를 벡터화한 벡터 또한 임베딩 벡터, 밀집 벡터

Word2Vec

| 단어 간 유사도를 반영할 수 있도록 단어의 의미를 벡터화
Cf) 원-핫 벡터

각 단어 벡터가 단어 간 유사도를 반영한 값을 가지고 있음
CBOW(Continuous Bag of Words), Skip-Gram 방식

전반적으로 Skip-gram 이 CBOW보다 성능 좋다고 알려져 있음

CBOW(Continuous Bag of Words) vs. Skip-Gram

CBOW(Continuous Bag of Words)

| 주변에 있는 단어들을 가지고 중간에 있는 단어들을 예측하는 방법

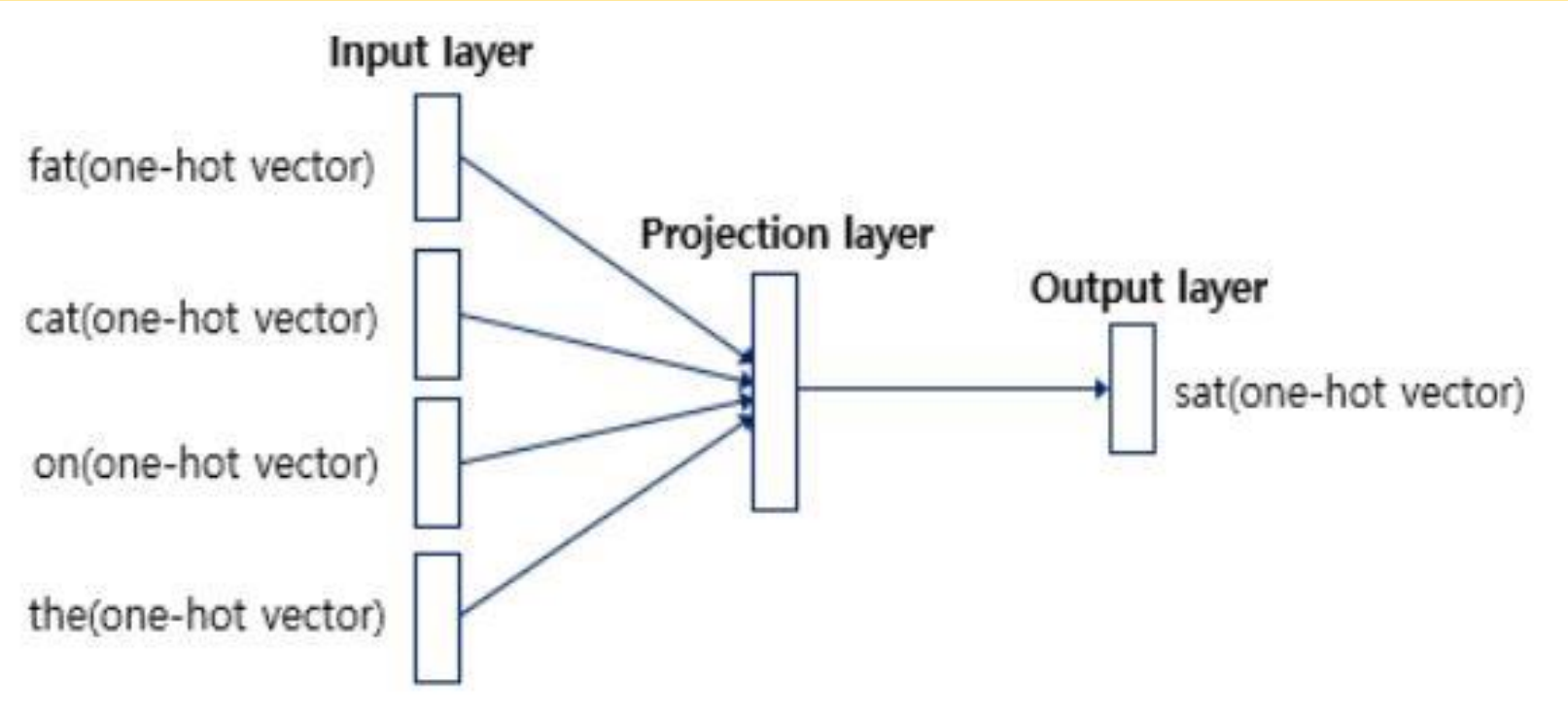
- 예측해야 하는 단어 == center word
- 예측에 사용되는 단어 == context word
- 중심 단어 예측을 위해 앞 뒤로 볼 단어의 범위 == window

Skip-Gram

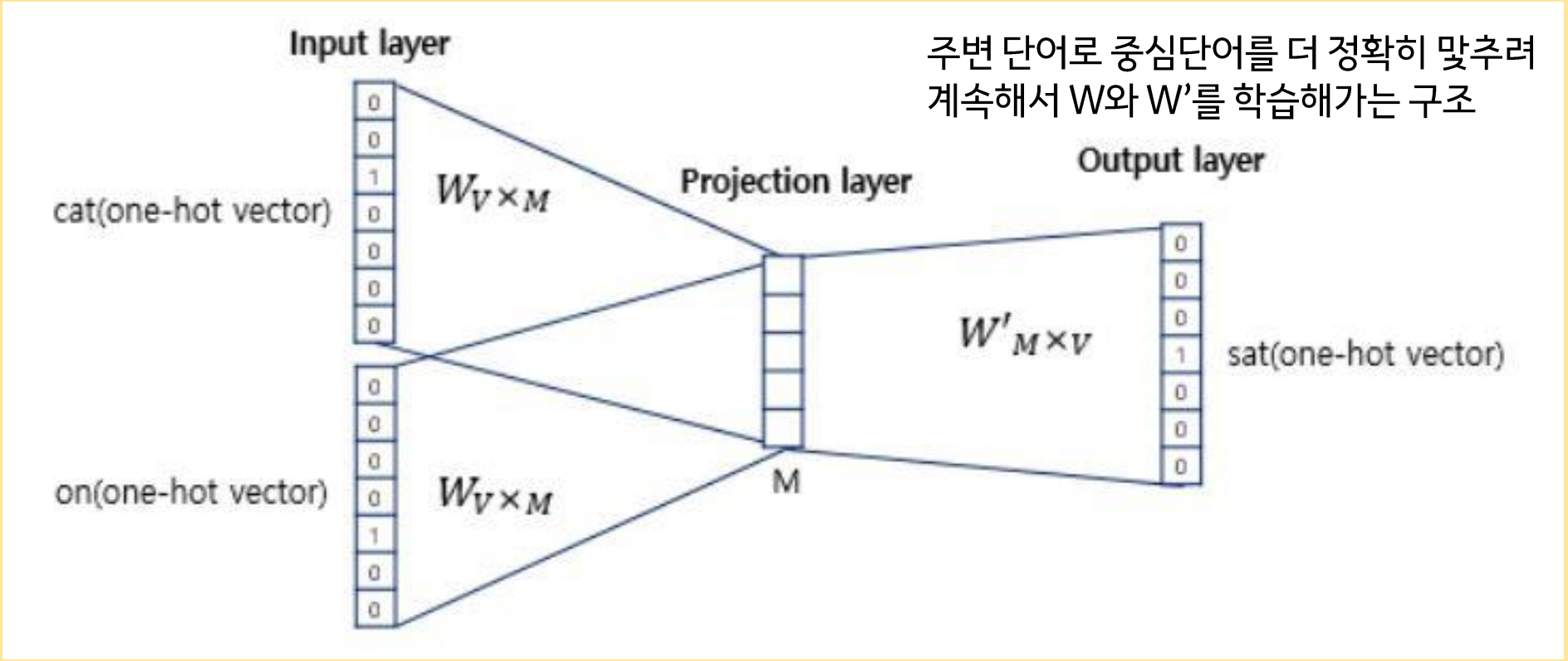
| 중간에 있는 단어로 주변 단어 예측하는 방법

CBOW(Continuous Bag of Words)

Sliding window를 통해 CBOW를 위한 전체 학습 데이터셋 만듦



입력층 = 사용자가 정한 윈도우 크기 범위 내에 있는 주변 단어들의 one-hot vector
투사층 = 룩업 테이블이라는 연산 담당하는 층(활성화 함수 없음)
출력층 = 예측하고자 하는 중간 단어의 one-hot vector



투사층 크기 M == 임베딩 후 벡터의 차원

단어 집합의 크기 V

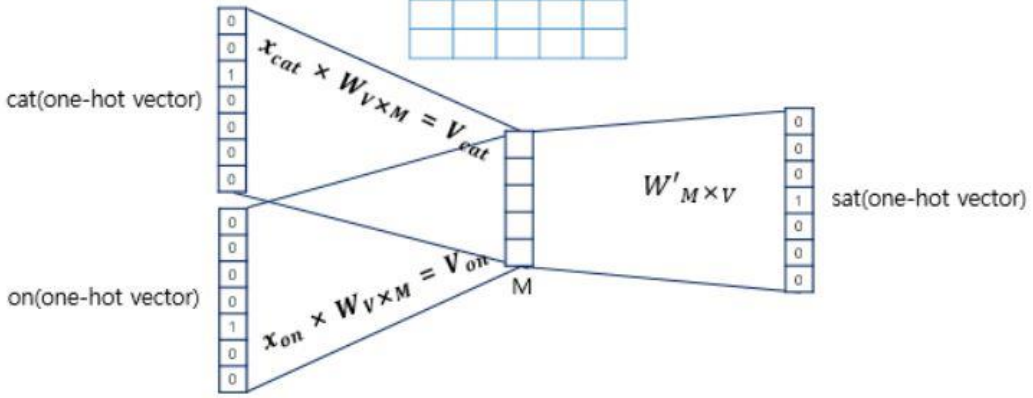
가중치 W == V x M 행렬

W와 W'는 서로 다른 행렬

인공 신경망 훈련 전 이 행렬들은 굉장히 작은 랜덤 값을 가짐

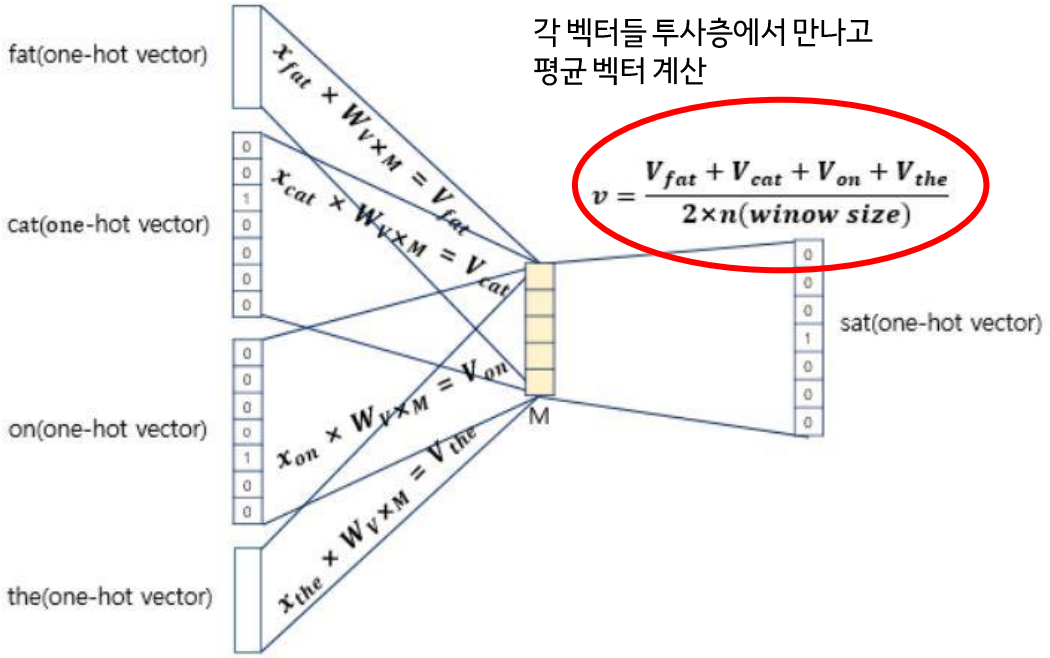
주변 단어 cat
원-핫 벡터 Xcat

W행렬의 i번째 행을
그대로 읽음



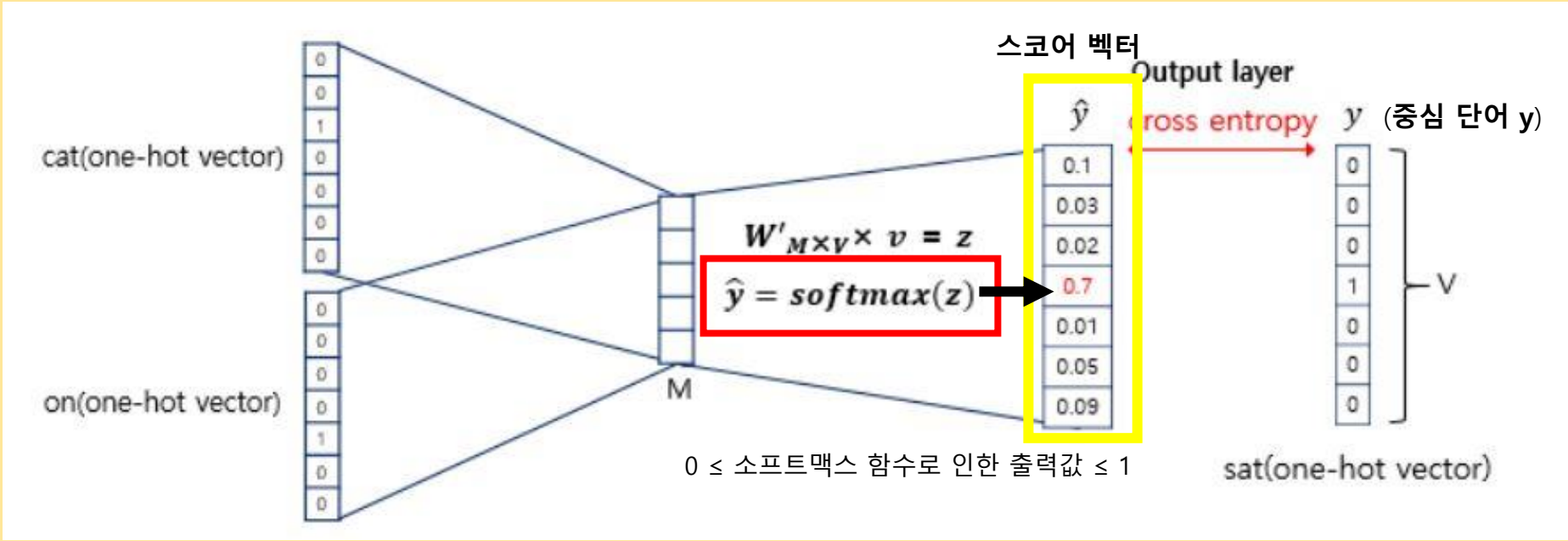
[1 x **V**] * [V x M] = [1 x M]

각 벡터들 투사층에서 만나고
평균 벡터 계산



[1 x M] * [M x V] = [1 x **V**]

입력 벡터의 차원과 동일



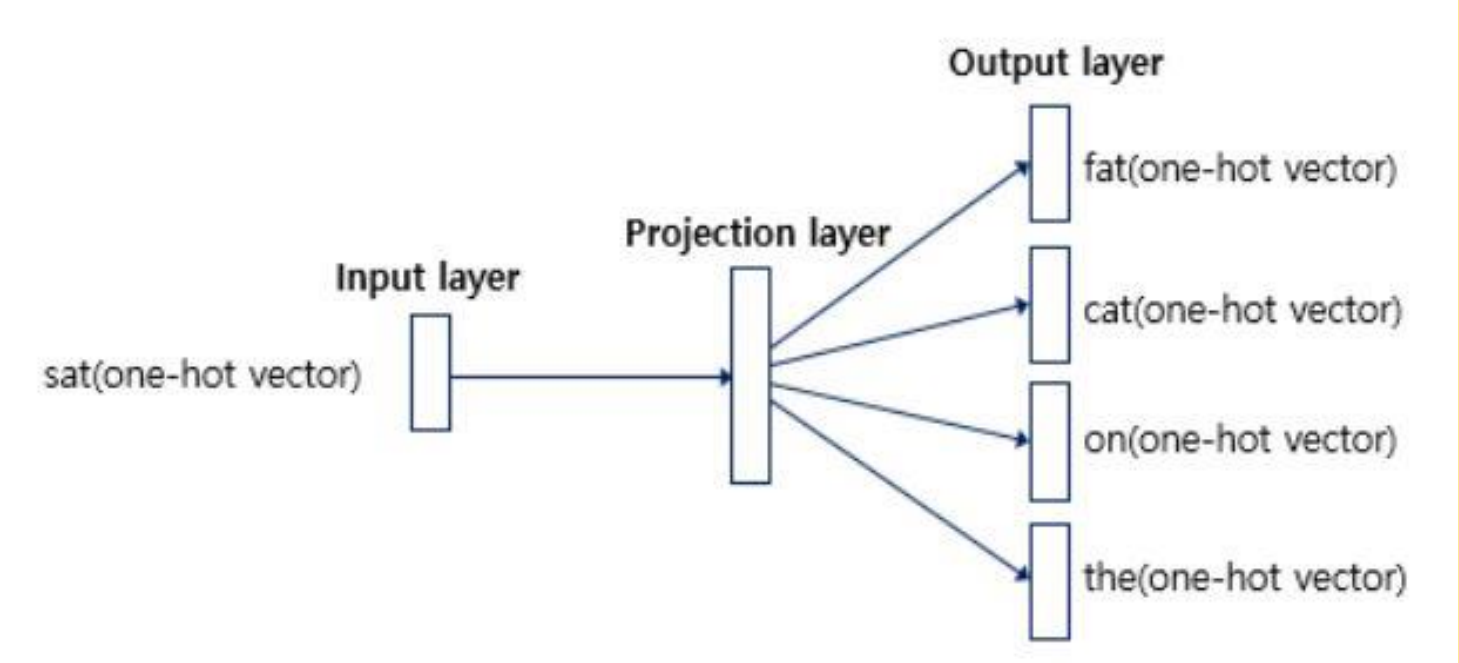
스코어 벡터의 j번째 index의 값은 j번째 단어가 중심 단어일 확률
이 값은 ground truth 벡터인 중심 단어 원-핫 벡터 값에 가까워져야 함
이 두 벡터값의 오차 줄이기 위해 CBOW는 손실 함수로 cross-entropy 사용

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

이 값을 최소화 하는 방향으로 학습 -> 역전파를 수행하면서 W, W' 학습

Skip-gram

| 중심 단어에서 주변 단어 예측



SGNS(Skip-Gram with Negative Sampling)

모든 단어 집합에 대해서 소프트맥스 함수, 역전파 수행

주변 단어와 상관없는 모든 단어까지 워드 임베딩 조정 작업함

Ex) 강아지, 고양이와 같은 단어에 집중해야한다면 이와 연관 관계 없는 수많은 단어의 임베딩 조정할 필요 없다

효율적으로 개선한 것 : SGNS

| 전체 단어 집합보다 훨씬 작은 단어 집합 만들어놓고 마지막 단계를 이진 분류 문제로 바꿈

주변 단어들을 positive로 두고 랜덤으로 샘플링 된 단어들을 negative로 둔 다음 이진 분류 문제 수행

GloVe

| LSA의 메커니즘인 카운트 기반 방법(유추 작업 성능 떨어짐) + Word2Vec의 메커니즘인 예측 기반 방법론(코퍼스의 전체 통계 정보 반영 못함) hybrid

임베딩 된 중심 단어와 주변 단어 벡터의 내적이 전체 말뭉치에서의 동시 등장 확률이 되도록 만드는 것. 이를 만족하도록 임베딩 벡터를 만드는 것이 목표.

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

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

| 행과 열을 전체 단어 집합의 단어들로 구성하고

i 단어의 윈도우 크기 내에서 k 단어가 등장한 횟수를 i행 k열에 기재한 행렬

Ex) I Like Deep Learning / I Like NLP / I Enjoy Flying (윈도우 크기 1일 때)

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

동시 등장 확률

| P(*k* | *i*)

동시 등장 행렬로부터 특정 단어 i의 전체 등장 횟수 카운트하고, 특정 단어 i가 등장했을 때 어떤 단어 k가 등장한 횟수를 카운트해서 계산한 조건부 확률 (중심단어 i, 주변단어 k)

Ex) i가 deep 이고 k가 learning 일 때 P(k|i) =0.5

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

동시 등장 확률과 관계 비(ratio)

ratio	k = solid(단단한)	k = gas	k = water	k = fashion
P(k | ice)	큰 값	작은 값	큰 값	작은 값
P(k | steam)	작은 값	큰 값	큰 값	작은 값
P(k | ice)/P(k | steam)	큰 값	작은 값	1에 근접	1에 근접

손실 함수(단어 관계를 잘 표현하는 함수여야 함)

- 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(solid I ice) = 단어 ice가 등장했을 때 단어 solid가 등장할 확률
- $\frac{P_{ik}}{P_{jk}}$: P_{ik}를 P_{jk}로 나눠준 값

  Ex) P(solid I ice) / P(solid I steam) = 8.9
- w_i : 중심 단어 i의 임베딩 벡터
- $\tilde{w}_k$: 주변 단어 k의 임베딩 벡터

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

임베딩 된 중심 단어와 주변 단어 벡터의 내적

전체 코퍼스에서의 동시 등장 확률

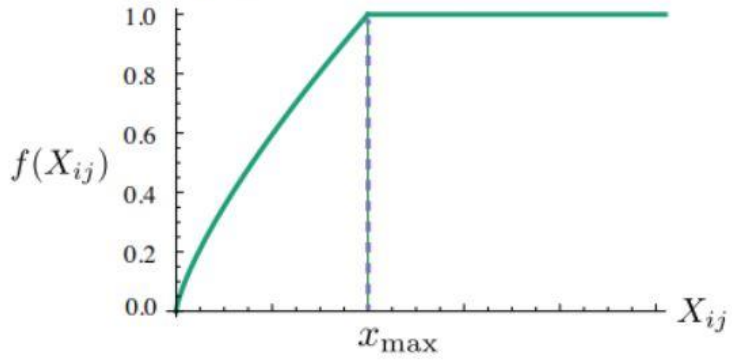
$$dot\ product(w_i\ \tilde{w}_k) \approx log\ P(k \mid i) = log\ P_{ik}$$

함수 최적화

| 함수 F는 두 단어 사이의 동시 등장 확률의 크기 관계 비율 정보를 벡터 공간에 인코딩하는 것이 목적

$$Loss\ function = \sum_{m,n=1}^{V} f(X_{mn})(w_m^T \tilde{w}_n + b_m + \tilde{b_n} - logX_{mn})^2$$

GloVe에 도입되는 $f(X_{ik})$의 그래프를 보겠습니다.



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

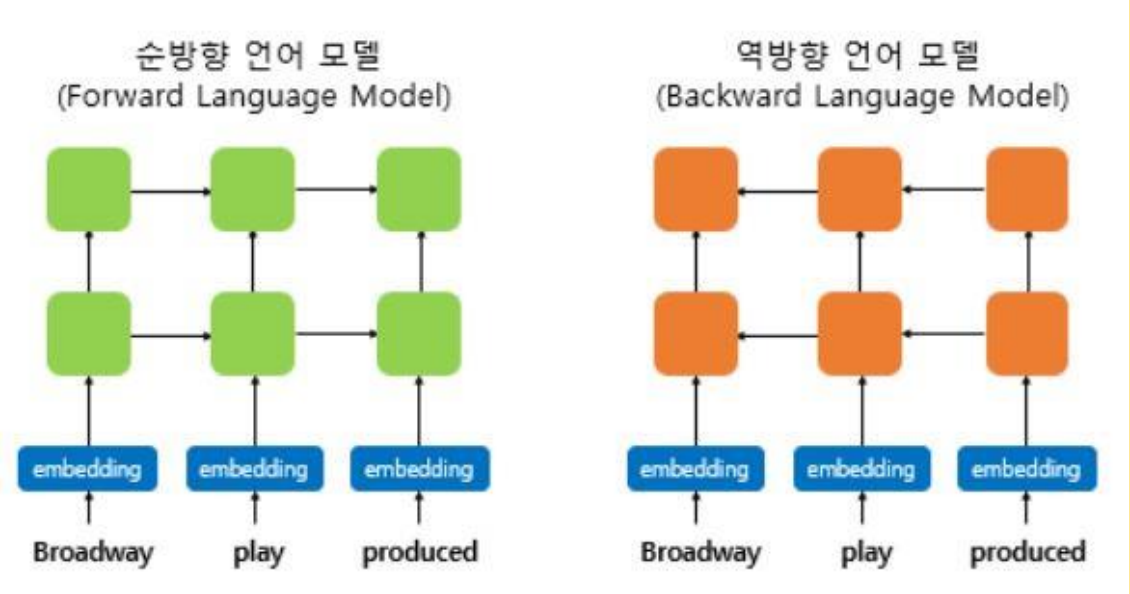
엘모(Embeddings from Language Model, ELMo)

| 2018년 등장한 new word embedding 기법: 언어 모델로 하는 임베딩
사전 훈련된 언어 모델(Pre-trained language model)사용
순방향 RNN 과 역방향 RNN 활용(biLM: Bidirectional Language Model)

단어 임베딩 전에 전체 문장을 고려해서 임베딩 함
(Contextualized Word Embedding)

Word2Vec 이나 GloVe 로 표현된 임베딩 벡터들은 문맥을 고려하지 못함
같은 표기의 단어라도 문맥에 따라서 다르게 워드 임베딩 할 수 있으면 자연어 처리 성능 올라감

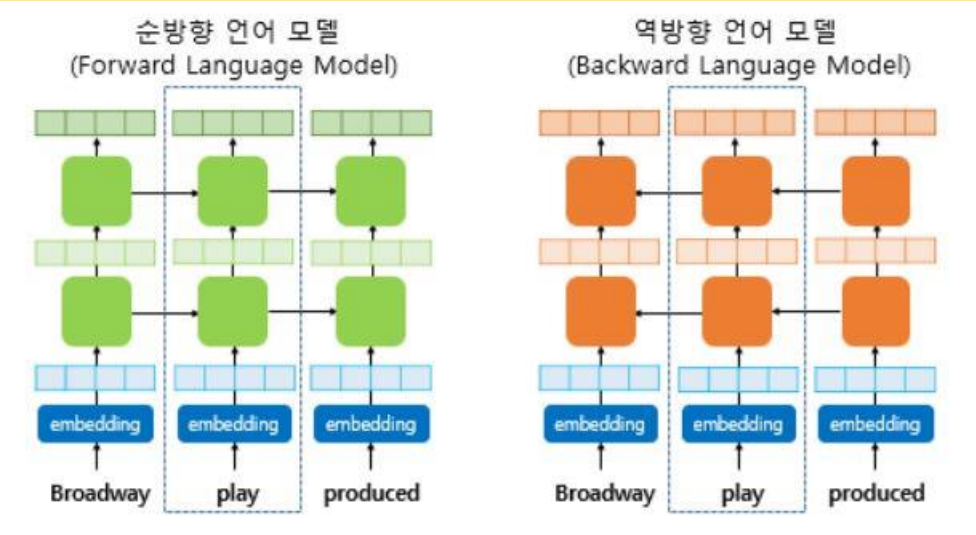
biLM 사전 훈련



기본적으로 다층 구조 전제 == 은닉층 최소 2개 이상
biLM의 입력이 되는 워드 임베딩 방법으로는 char CNN이라는 방법 사용
char CNN은 글자(character)단위로 계산됨, 문맥과 상관없이 단어들의 연관성 찾을 수 있음

biLM 순방향 언어모델과 역방향 언어모델은 각각의 은닉 상태만을 다음 은닉층으로 보내며
훈련시킨 후에 ELMo 표현으로 사용하기 위해 은닉 상태 concatenate 시킴

ELMo 표현 얻기



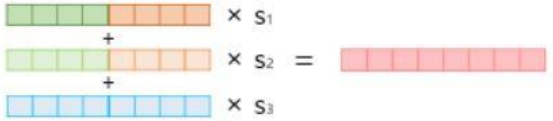
ELMo가 임베딩 벡터 얻는 과정

1) 각 층의 출력값을 연결(concatenate)한다.

2) 각 층의 출력값 별로 가중치를 준다.

이 가중치를 여기서는 s_1, s_2, s_3라고 합시다.

3) 각 층의 출력값을 모두 더한다.

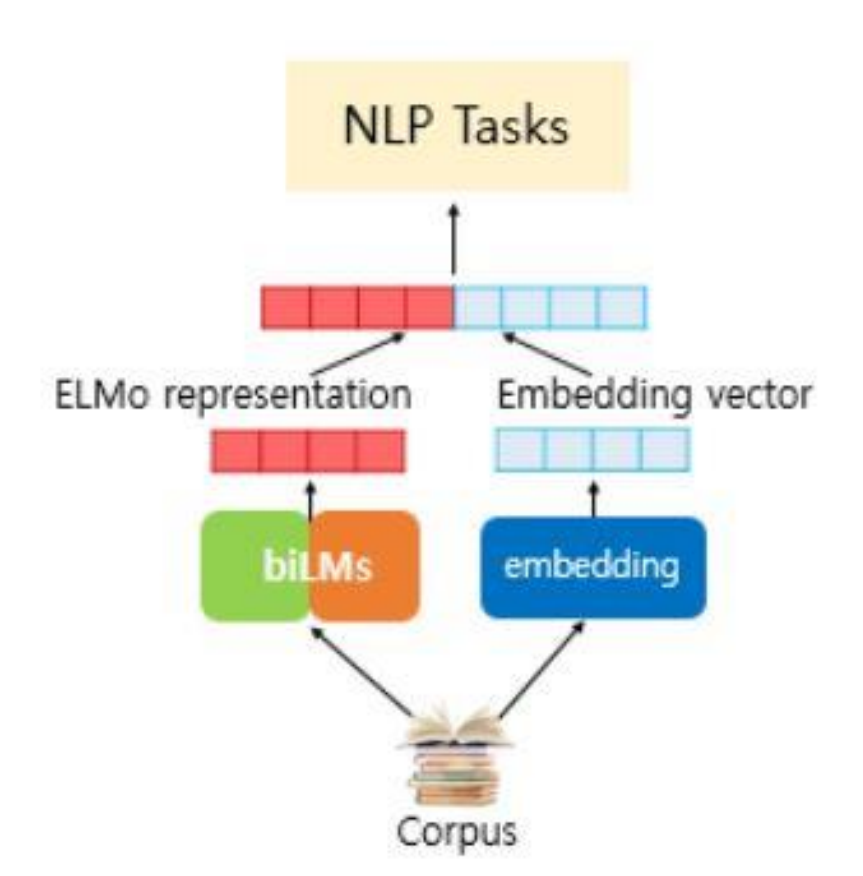


2)번과 3)번의 단계를 요약하여 가중합(Weighted Sum)을 한다고 할 수 있습니다.

4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.

이 스칼라 매개변수를 여기서는 γ이라고 합시다.

텍스트 분류 작업에 사용되는 ELMo 표현



기존의 임베딩 벡터와 함께 사용

준비된 ELMo 표현을 GloVe 임베딩 벡터와 연결해서 입력값으로 사용

ELMo 표현 만드는데 사용된 사전 훈련 언어 모델 가중치 고정
s1 s2 s3 r 은 훈련 과정에서 학습됨