토큰화 및 임베딩

딥러닝스터디 이아현

토큰화 - 단어 토큰화

보통 인간은 눈으로 띄어쓰기, 문장부호 등을 통해 문장에서 단어를 구별한다.

이처럼 컴퓨터의 단어 토큰화도 띄어쓰기, 문장부호, 대소문자 등의 특정 구분자로 구별한다.

예시에서 '최고!' 와 '최고!!'가 다른 토큰으로 분류되는 것처럼 cg라는 토큰이 단어 사전에 있어도 'cg.' 'cg는' 'cg도' 등은 다른 토큰(OOV)가 된다.

이처럼 단어 토큰화는 한국어 접사, 문장 부호, 오타 혹은 띄어쓰기 등에 취약하여 잘 쓰지 않는다.

토큰화 - 글자 토큰화

글자 토큰화는 단어 토큰화와 다르게 띄어쓰기뿐 아니라 글자 단위로 문장을 나눠서 비교적 작은 단어 사전을 구성한다. 주로 다음 단어를 예측하는 언어 모델링에 사용된다.

list 형태로 바꾸면 쉽게 글자로 토큰화된다. 공백도 토큰에 포함된다.

영어의 경우 글자 토큰화를 진행하면 각 알파벳으로 나뉘지만 한글의 경우 하나의 글자는 여러 자음과 모음의 조합으로 이루어져 있다.

토큰화 - 글자 단위 토큰화

조합된 글자 단위로 처리하는 완성형 한글을 글자를 자모 단위로 처리하는 조합형 한글로 바꾸는 h2j 함수로 바꾼 후 조합형 한글을 자소 단위로 나누어 반환해주는 h2hcj함수로 자소 단위를 반환해준다.

토큰화 - 글자 단위 토큰화

장점 : 단어 단위 토큰화에 비해 비교적 작은 크기의 단어 사전을 구축할 수 있다.

단점 : 개별 토큰은 아무런 의미가 없으므로 자연어 모델이 각 토큰의 의미를 조합해 결과를 도출해야 한다.

모델 입력 시퀀스(sequence)의 길이가 길어질수록 연산량이 증가한다.

->시퀀스(sequence)데이터는 순서가 의미를 가지는 데이터로 자연어 데이터는 거의 모든 데이터의 순서가 의미를 가지므로 시퀀스(sequence) 데이터라고 부른다.

토큰화 - 형태소 단위 토큰화

단어를 의미 있는 단위인 형태소로 분리하는 것을 말한다.

대부분의 언어는 띄어쓰기로 의미가 구분되지만 한국어는 어근에 다양한 조사와 접사가 결합되어 하나의 낱말을 이루는 교착어(agglutinative language)이다.

예_ 그는 나에게 인사를 했다.

‘그는’ -> ‘그’ + ‘는’

‘나에게’ -> ‘나’ + ‘에게’

자립형태소 : 그, 나, 인사

의존형태소 : -는, -에게, -를, 했-. -다

토큰화 - 형태소 단위 토큰화

형태소 어휘 사전 : 자연어 처리에서 토큰화 후 나온 단어의 집합을 모은 어휘 사전 중 각 단어의 형태소 정보를 포함하는 사전

예_ ‘그는 나에게 인사를 했다’

‘나는 그에게 인사를 했다’

띄어쓰기를 이용한 토큰화(단어 토큰화)를 이용한 어휘사전

[‘그는’, ‘나에게’, ‘인사를’, ‘했다’, ‘나는’, ‘그에게’]

‘그녀는 그에게 인사를 했다’ 와 같은 문장이 들어오면 ‘그녀는’과 ‘그는’, ‘나는’을 다른 의미 단위로 생각함

형태소 어휘 사전으로 만들면 [‘는’, ‘-에게’, ‘나’, ‘그’]에 ‘그녀’ 정보만 추가하면 됨

토큰화 - 형태소 단위 토큰화

형태소 어휘 사전에서는 각 형태소가 어떤 품사에 속하는지와 해당 품사의 뜻 정보도 제공

품사 태깅 : 텍스트 데이터를 텍스트 데이터를 형태소 분석하여 각 형태소에 해당하는 품사(Part Of Speech, POS)를 태깅해주는 작업

예_ ‘그는 나에게 인사를 했다’

그(명사) + 는(조사) + 나(명사) + 에게(조사) + 인사(명사) + 를(조사) + 했다(동사)

KoNLPy, NLTK, spaCy 라이브러리를 이용해 토큰화

토큰화 - 형태소 단위 토큰화

konlpy : 자바 언어 기반 형태소 분석기

konlpy에서 제공하는 여러 객체 : okt, kkma,komoran, hannanum, mecab

okt 객체 : sns텍스트 데이터를 기반으로 만들어졌으며

명사추출, 구문추출, 형태소추출, 품사태깅(입력된 문장에서 각 단어에 대한 품사 정보를 추출하여 (형태소, 품사) 형태의 튜플(tuple)로 구성된 리스트(list)를 반환)

의 작업을 수행한다.

토큰화 - 형태소 단위 토큰화

NLTK : 자연어 처리를 위해 개발된 라이브러리로 토큰화, 형태소 분석, 개체명 인식, 감정 분석 등과 같은 기능을 제공한다.

주로 영어 자연어 처리를 위해 수행된다.

트리뱅크(TreeBank)라는 대규모 영어 말뭉치로 학습된 Punkt모델과 Averaged Perceptron Tagger모델을 사용하여 토큰화 및 품사 태깅 작업을 수행한다.

단어 토크나이저(word_tokenize)

문장 토크나이저(sent_tokenize)

토큰화 - 형태소 단위 토큰화

spaCy : 사이썬(Cython)기반으로 개발된 오픈소스 라이브러리로 빠른 속도와 높은 정확성을 목표로 하는 머신러닝 기반의 자연어처리 라이브러리다.

NLTK에서 사용되는 모델보다 더 크고 복잡하며 사용하고자 하는 언어에 맞는 사전학습 모델을 사용한다.

GPU가속 및 다른 언어 사용을 위해 로컬에서 파이프라인을 설치할 수 있다.

https://spacy.io/usage

토큰화 - 형태소 단위 토큰화

spaCy : 사이썬(Cython)기반으로 개발된 오픈소스 라이브러리로 빠른 속도와 높은 정확성을 목표로 하는 머신러닝 기반의 자연어처리 라이브러리다.

NLTK에서 사용되는 모델보다 더 크고 복잡하며 사용하고자 하는 언어에 맞는 사전학습 모델을 사용한다.

GPU가속 및 다른 언어 사용을 위해 로컬에서 파이프라인을 설치할 수 있다.

https://spacy.io/usage

토큰화 - 형태소 단위 토큰화

-> 장점 : 형태소는 자연어의 최소 의미 단위로 대부분의 자연어가 형태소 조합으로 이루어져 있어 컴퓨터가 자연어를 인간이 이해하는 방식으로 처리하게 하려면 이 방법이 제일 효과적이다.

-> 단점 : 형태소 분석기는 모르는 단어 (외래어, 띄어쓰기 오류, 오탈자) 를 적절한 단위로 나누는 것에 취약하며 이는 잠재적으로 어휘 사전의 크기를 크게 만들며 OOV에 대응하기 어렵게 한다.

토큰화 - 하위 단어 토큰화

하나의 단어를 빈번하게 사용되는 하위 단어(subword)의 조합으로 나누어 토큰화하는 방법

ex) Reinforcement = Rein + force + ment

단어의 길이를 줄일 수 있어 처리 속도가 빨라지고 OOV문제, 신조어, 은어, 고유어 등의 문제를 완화할 수 있다.

바이트페어인코딩, 워드피스, 유니그램 모델 등이 있다.

토큰화 - 하위 단어 토큰화

바이트 페어 인코딩(Byte Pair Encoding) : 텍스트 데이터에서 가장 빈번하게 등장하는 글자 쌍의 조합을 찾아 부호화하는 압축 알고리즘

연속된 글자 쌍이 더이상 나타나지 않거나 정해진 어휘 사전 크기에 도달할 때까지 조합 탐지와 부호화를 반복하여 이 과정에서 자주 등장하는 단어는 하나의 토큰으로 토큰화하고 덜 등장하는 단어는 여러 토큰의 조합으로 토큰화한다.

ex) abracadabra 어휘 사전 : [‘a’,’b’,’r’,’c’,’d’]

1	AracadAra	ab =>A	[‘a’,’b’,’r’,’c’,’d’, ‘ab’]
2	ABcadAB	ra =>B	[‘a’,’b’,’r’,’c’,’d’, ‘ab’, ‘ra’]
3	CcadC	AB =>C	[‘a’,’b’,’r’,’c’,’d’, ‘ab’, ‘ra’,’abra’]

토큰화 - 하위 단어 토큰화

센텐피스(SentencePiece)라이브러리 : 구글에서 개발한 오픈소스 하위 단어 토크나이저 라이브러리

바이트 페어 인코딩과 유사한 알고리즘을 이용하여 입력 데이터를 토큰화하고 단어 사전을 생성한다.

코포라(Korpora)라이브러리 : 국립국어원이나 AI Hub에서 제공하는 말뭉치 데이터를 쉽게 이용할 수 있도록 제공하는 오픈소스 라이브러리

파이썬에서 쉽게 사용할 수 있게 API가 제공된다.

토큰화 - 하위 단어 토큰화

워드피스(Wordpiece) 토크나이저 : 바이트 페어 토크나이저와 유사한 방법으로 학습되지만, 빈도 기반이 아닌 확률 기반으로 글자 쌍을 병합한다.

f는 빈도(frequence)를 나타내는 함수이며, x와y는 병합하려는 하위 단어를 의미한다.

수식 5.1 글자 쌍 병합 점수 수식

$$score = \frac{f(x,y)}{f(x),f(y)}$$

f(x,y) : x와 y가 조합된 글자 쌍의 빈도 즉, xy 글자 쌍의 빈도

f(x) : 글자 x의 빈도

f(y) : 글자 y의 빈도

임베딩 - 텍스트 벡터화

문장을 컴퓨터가 이해할 수 있는 단위인 토큰으로 나누었으면 텍스트를 숫자로 변환하는 텍스트 벡터화(Text Vectorization)과정이 필요하다.

원-핫 인코딩(One-Hot Encoding), 빈도 벡터화(Count Vectorization) 등이 있다.

쉽고 간단하지만, 벡터의 희소성(Sparsity)가 크다는 단점이 있다. 위의 방법은 말뭉치 내에 존재하는 토큰의 개수만큼의 벡터 차원을 가져야 하지만 입력 문장 내의 토큰의 수는 적기 때문이다.

또한 텍스트의 벡터가 입력 텍스트의 의미를 내포하고 있지 않으므로 잘못 벡터화될 수도 있다.

임베딩 - 텍스트 벡터화

워드 임베딩(Word Embedding) : 희소성 및 의미 부재 문제를 해결하기 위해 단어의 의미를 학습해 표현하는 방법

단어를 고정된 길이의 실수 벡터로 표현하는 방법으로, 단어의 의미를 벡터 공간에서 다른 단어와의 상대적 위치로 표현해 단어 간의 관계를 추론한다.

워드 투 벡터(Word2Vec)나 패스트 텍스트(FastText)가 이에 해당한다.

동적 임베딩(Dynamic Embedding) : 워드 임베딩이 고정된 길이의 임베딩을 학습하기 때문에 다의어나 문맥 정보를 다루기 어렵다는 단점을 극복하기 위해 인공신경망을 사용한다.

임베딩 - 텍스트 벡터화

희소 표현(sparse representation) : 원-핫 인코딩, TF-IDF 와 같이 대부분의 벡터 요소가 0으로 표현되는 방법

단어 사전의 크기가 5000개라고 가정하고 10개의 토큰으로 이루어진 입력 텍스트를 원-핫 인코딩으로 표현하면 4,990개의 0이 포함된 벡터가 된다.

또한, 단어간의 유사성을 반영하지 못한다.

밀집 표현(dense representation) : 단어를 고정된 크기의 실수 벡터로 표현하기 때문에 단어 사전의 크기가 커지더라도 벡터의 크기가 커지지 않는다.

학습을 통해 단어를 벡터화하므로 단어의 의미를 비교할 수 있다.

밀집 표현된 벡터를 단어 임베딩 벡터(Word Embedding Vector)라고 하며 Word2vec, fastText는 대표적인 임베딩 기법이다.

임베딩 - 텍스트 벡터화

희소 표현

표 6.3 단어의 희소 표현

소	0	1	0	0	0
잃고	1	0	0	0	0
외양간	0	0	0	1	0
고친다	0	0	0	0	1

밀집 표현(dense representation) :

표 6.4 단어의 밀집 표현

소	0.3914	−0.1749	…	0.5912	0.1321
잃고	−0.2893	0.3814	…	−0.1492	−0.2814
외양간	0.4812	0.1214	…	−0.2745	0.0132
고친다	−0.1314	−0.2809	…	0.2014	0.3016

임베딩 - 밀집 표현

CBoW(Continuous Bag of Words) : 주변 단어(Context word)로 중심 단어(Center word)를 예측하는 기법

윈도(Window) : 중심 단어를 맞추기 위해 고려할 주변 단어의 개수

슬라이딩 윈도(Sliding Window) : 학습을 위해 윈도를 이동해가며 학습하는 방법

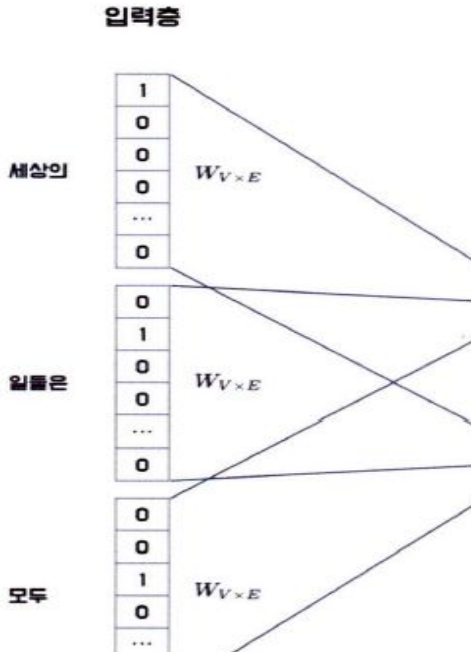
한 번의 학습으로 여러 개의 중심 단어와 그에 대한 주변 단어를 학습할 수 있다.

임베딩 - 밀집 표현

(주변 단어 | 중심 단어)로 구성된 학습 데이터로 대량의 말뭉치에서 효과적으로 단어의 분산 표현을 학습할 수 있다.

입력 문장	학습 데이터
(세상의 재미있는 일들은)모두 밤에 일어난다	(재미있는, 일들은 | 세상의)
(세상의 재미있는 일들은 모두)밤에 일어난다	(세상의, 일들은, 모두 | 재미있는)
(세상의 재미있는 일들은 모두 밤에)일어난다	(세상의, 재미있는, 모두, 밤에 | 일들은)
세상의(재미있는 일들은 모두 밤에 일어난다)	(재미있는, 일들은, 밤에, 일어난다 | 모두)
세상의 재미있는(일들은 모두 밤에 일어난다)	(일들은, 모두, 일어난다 | 밤에)
세상의 재미있는 일들은(모두 밤에 일어난다)	(모두, 밤에 | 일어난다)

그림 6.4 CBoW의 학습 데이터 구성



임베딩 - 밀집 표현

입력층 : 입력 단어의 위치에 1을 주고 나머지 단어 사전은 0인 원-핫 벡터들

투사층 : 단어 사전 크기의 원-핫 벡터에

(단어 사전 크기, 임베딩 벡터 크기)의

가중치 W(vxe)를 곱해줘서 임베딩 벡터를 만든다.

세 단어의 임베딩 벡터 V세상의, V일들은, V모두

를 평균낸 평균 벡터를 가중치 행렬 W'(exv)와

곱해주면 V크기의 벡터를 얻고 여기에

소프트맥스함수를 취해 중심 단어를 예측한다.

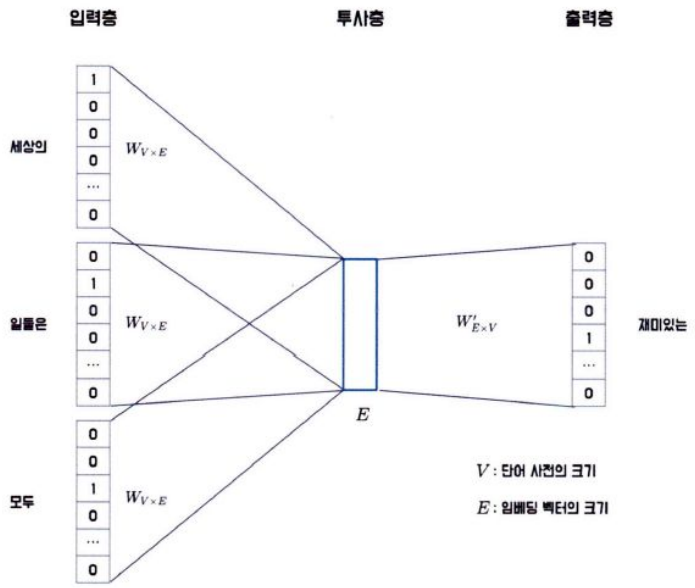


그림 6.5 CBoW 모델 구조

임베딩 - 밀집 표현

Skip-gram은 반대로 중심 단어를 입력으로 받아서 주변 단어를 예측하는 모델이다.

윈도(window) : 중심 단어를 기준으로 주변 단어로 삼을 크기

중심 단어와 각 주변 단어를 하나의 쌍으로 하여 모델을 학습시킨다.

입력 문장	학습 데이터
(세상의 재미있는 일들은)모두 밤에 일어난다	(세상의 | 재미있는), (세상의 | 일들은)
(세상의 재미있는 일들은 모두)밤에 일어난다	(재미있는 | 세상의), (재미있는 | 일들은), (재미있는 | 모두)
(세상의 재미있는 일들은 모두 밤에)일어난다	(일들은 | 세상의), (일들은 | 재미있는), (일들은 | 모두), (일들은 | 밤에)
세상의(재미있는 일들은 모두 밤에 일어난다)	(모두 | 재미있는), (모두 | 일들은), (모두 | 밤에), (모두 | 일어난다)
세상의 재미있는(일들은 모두 밤에 일어난다)	(밤에 | 일들은), (밤에 | 모두), (밤에 | 일어난다)
세상의 재미있는 일들은(모두 밤에 일어난다)	(일어난다 | 모두), (일어난다 | 밤에)

그림 6.6 Skip-gram의 학습 데이터 구성

임베딩 - 밀집 표현

CBoW와의 차이점 : CBoW는 하나의 윈도에서 하나의 학습 데이터가 만들어지는 반면, Skip-gram은 하나의 윈도에 있는 주변 단어들 중 하나의 주변 단어를 하나의 쌍으로 여러 학습 데이터가 만들어진다.

입력 문장	학습 데이터
(세상의 재미있는 일들은)모두 밤에 일어난다	(재미있는, 일들은 | 세상의)
(세상의 재미있는 일들은 모두)밤에 일어난다	(세상의, 일들은, 모두 | 재미있는)
(세상의 재미있는 일들은 모두 밤에)일어난다	(세상의, 재미있는, 모두, 밤에 | 일들은)
세상의(재미있는 일들은 모두 밤에 일어난다)	(재미있는, 일들은, 밤에, 일어난다 | 모두)
세상의 재미있는(일들은 모두 밤에 일어난다)	(일들은, 모두, 일어난다 | 밤에)
세상의 재미있는 일들은(모두 밤에 일어난다)	(모두, 밤에 | 일어난다)

그림 6.4 CBoW의 학습 데이터 구성

입력 문장	학습 데이터
(세상의 재미있는 일들은)모두 밤에 일어난다	(세상의 | 재미있는), (세상의 | 일들은)
(세상의 재미있는 일들은 모두)밤에 일어난다	(재미있는 | 세상의), (재미있는 | 일들은), (재미있는 | 모두)
(세상의 재미있는 일들은 모두 밤에)일어난다	(일들은 | 세상의), (일들은 | 재미있는), (일들은 | 모두), (일들은 | 밤에)
세상의(재미있는 일들은 모두 밤에 일어난다)	(모두 | 재미있는), (모두 | 일들은), (모두 | 밤에), (모두 | 일어난다)
세상의 재미있는(일들은 모두 밤에 일어난다)	(밤에 | 일들은), (밤에 | 모두), (밤에 | 일어난다)
세상의 재미있는 일들은(모두 밤에 일어난다)	(일어난다 | 모두), (일어난다 | 밤에)

그림 6.6 Skip-gram의 학습 데이터 구성

임베딩 - 밀집 표현

Skip-gram 모델

입력층 : 단어 사전 크기를 가지고 중심 단어만 1인 원 - 핫 벡터

투사층 : 입력 단어의 원-핫 벡터에 (단어 사전 크기 x 임베딩 벡터 크기) 의 W(vxe) 를 곱해서 해당 단어의 임베딩 벡터를 가져온다.

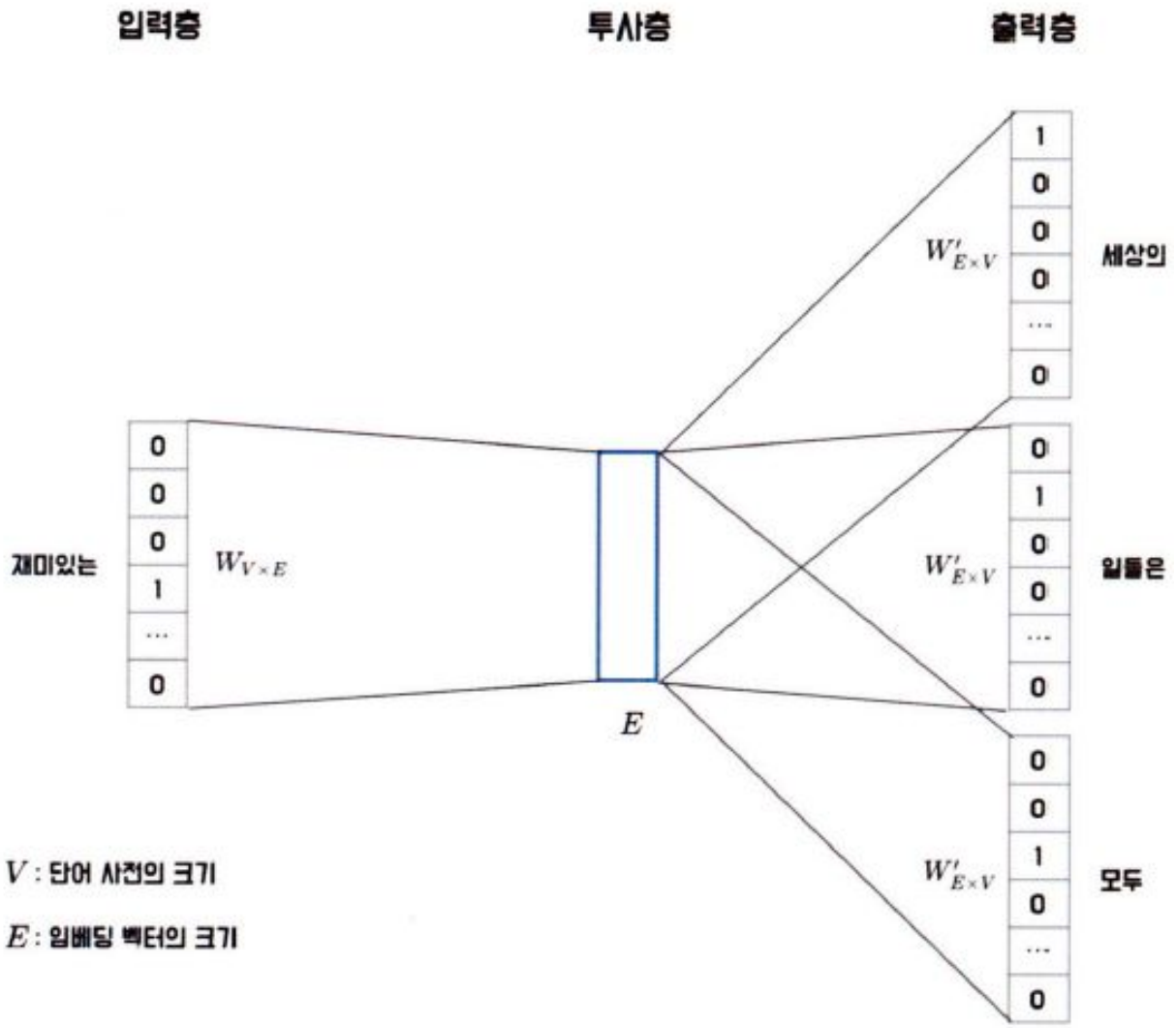
출력층 : 입력 단어의 임베딩 벡터에 주변 단어의 개수만큼 (임베딩 벡터 크기 x 단어 사전의 크기)의 W’(exv) 가중치 행렬을 곱해서 V 크기의 벡터들을 얻고 각 벡터에 소프트맥스 연산을 취해 주변 단어를 예측한다.

문제는 모든 단어를 대상으로 내적 연산을 취하므로 단어 사전의 크기가 커지면 학습 속도가 느려진다.

이를 해결하기 위해 계층적 소프트맥스와 네거티브 샘플링 기법을 적용한다.

임베딩

Skip-gram



임베딩 - 밀집 표현

fastText : 메타에서 개발한 오픈소스 임베딩 모델로, Word2vec과 유사하지만 단어의 하위 단어를 고려하므로 더 높은 정확도와 성능을 제공한다.

하위 단어를 고려하기 위해 N-gram을 사용해 단어를 분해하고 벡터화한다. 각 하위 단어의 임베딩 벡터를 구하고, 이를 모두 합산하여 입력 단어의 최종 임베딩 벡터를 계산한다.

그림 6.11 fastText의 하위 단어 집합

임베딩 - 밀집 표현

일반적으로 fastText는 다양한 N-gram을 적용해 입력 토큰을 분해하고 하위 단어 벡터를 구성함으로써 단어의 부분 문자열을 고려하는 유연하고 정확한 하위 단어 집합을 생성한다.

장점 1 : 같은 하위 단어를 공유하는 단어끼리는 정보를 공유해 학습할 수 있다.

장점 2 : OOV단어도 하위 단어로 나누어 임베딩을 계산할 수 있게 된다.

EX) 말뭉치에 [‘개인택시’, ‘정보처리기사’, ‘임대차보호법’] 단어가 있을 때, 이 단어들의 하위 단어들로 ‘개인정보보호법’ 이라는 단어의 임베딩도 연산할 수 있다.

임베딩 - 밀집 표현

fastText와 Word2Vec 모델 모두 단어를 고정 길이 벡터 형태로 표현하기 위해 분산 표현 학습을 수행하고 주변 단어의 정보를 활용하여 단어의 의미를 파악한다.

이를 통해 단어 간의 유사성을 측정하고 비슷한 의미를 가진 단어를 유사한 벡터로 표현한다.

fastText 모델도 CBoW와 Skip-gram으로 구성되며 네거티브 샘플링 기법을 사용해 학습한다. 단지 단어의 기본 단위가 아닌 하위 단어로 학습한다는 차이점이 있어 모든 하위 단어 크기를 갖는 임베딩 계층이 필요하여 조금 더 복잡하다.