Natural Language Processing with Pytorch

수학과 오서영

토큰화(Tokenization)

Tokenization
주어진 텍스트를 단어 또는 문자 단위로 자르는 것

1. spaCy 사용하기

```
import spacy
```

```
en_text = "A Dog Run back corner near spare bedrooms"
```

```
spacy_en = spacy.load('en')
```

```
def tokenize(en_text):
    return [tok.text for tok in spacy_en.tokenizer(en_text)]

print(tokenize(en_text))
```

```
['A', 'Dog', 'Run', 'back', 'corner', 'near', 'spare', 'bedrooms']
```

토큰화(Tokenization)

2. NLTK 사용하기

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:₩Users₩eilee₩AppData₩Roaming₩nltk_data...
[nltk_data]   Unzipping tokenizers₩punkt.zip.
```

```
True
```

```
from nltk.tokenize import word_tokenize
print(word_tokenize(en_text))
```

```
['A', 'Dog', 'Run', 'back', 'corner', 'near', 'spare', 'bedrooms']
```

```
print(en_text.split())
```

```
['A', 'Dog', 'Run', 'back', 'corner', 'near', 'spare', 'bedrooms']
```

단어집합(Vocabulary)

Vocabulary
중복을 제거한 텍스트의 총 단어의 집합

네이버 영화 리뷰 분류하기' 데이터를 다운로드

```
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ratings.txt", filename="ratings.txt")
data = pd.read_table('ratings.txt')
data[:10]
```

	id	document	label
0	8112052	어릴때보고 지금다시봐도 재밌어요ㅋㅋ	1
1	8132799	디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산...	1
2	4655635	폴리스스토리 시리즈는 1부터 뉴까지 버릴께 하나도 없음.. 최고.	1
3	9251303	와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이런...	1
4	10067386	안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화.	1

```
print('전체 샘플의 수 : {}'.format(len(data)))
```

```
전체 샘플의 수 : 200000
```

단어집합(Vocabulary)

한글과 공백을 제외하고 모두 제거

In [28]:
```
sample_data['document'] = sample_data['document'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
# 한글과 공백을 제외하고 모두 제거
sample_data[:10]
```

Out[28]:

	id	document	label
0	8112052	어릴때보고 지금다시봐도 재밌어요ㅋㅋ	1
1	8132799	디자인을 배우는 학생으로 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산업...	1
2	4655635	폴리스스토리 시리즈는 부터 뉴까지 버릴께 하나도 없음 최고	1

불용어 정의

```
stopwords=['의','가','이','은','들','는','좀','잘','걍','과','도','를','으로','자','에','와','한','하다']
```

단어집합(Vocabulary)

토큰화

```
tokenizer = Mecab()
```

```
tokenized=[]
for sentence in sample_data['document']:
    temp = []
    temp = tokenizer.morphs(sentence) # 토큰화
    temp = [word for word in temp if not word in stopwords] # 불용어 제거
    tokenized.append(temp)
```

```
print(tokenized[:10])
```

```
[['어릴', '때', '보', '고', '지금', '다시', '봐도', '재밌', '어요', 'ㅋㅋ'], ['디자인', '을', '배우', '학생', '외국', '디자이너',
'그', '일군', '전통', '을', '통해', '발전', '해', '문화', '산업', '부러웠', '는데', '사실', '우리', '나라', '에서', '그', '어려운',
'시절', '끝', '까지', '열정', '을', '지킨', '노라노', '같', '전통', '있', '어', '저', '같', '사람', '꿈', '을', '꾸', '고', '이뤄나
갈', '수', '있', '다는', '것', '감사', '합니다'], ['폴리스', '스토리', '시리즈', '부터', '뉴', '까지', '버릴', '께', '하나', '없',
'음', '최고'], ['연기', '진짜', '개', '쩔', '구나', '지루', '할거', '라고', '생각', '했', '는데', '몰입', '해서', '봤', '다', '그래',
'이런', '게', '진짜', '영화', '지'], ['안개', '자욱', '밤하늘', '떠', '있', '초승달', '같', '영화'], ['사랑', '을', '해', '본', '사
람', '라면', '처음', '부터', '끝', '까지', '웃', '을', '수', '있', '영화'], ['완전', '감동', '입니다', '다시', '봐도', '감동'],
['개', '전쟁', '나오', '나요', '나오', '면', '빠', '로', '보', '고', '싶', '음'], ['굿'], ['바보', '아니', '라', '병', '쉰', '인',
'듯']]
```

단어집합(Vocabulary)

각 단어에 고유한 정수 부여

```
word_to_index = {word[0] : index + 2 for index, word in enumerate(vocab)}
word_to_index['pad'] = 1
word_to_index['unk'] = 0
```

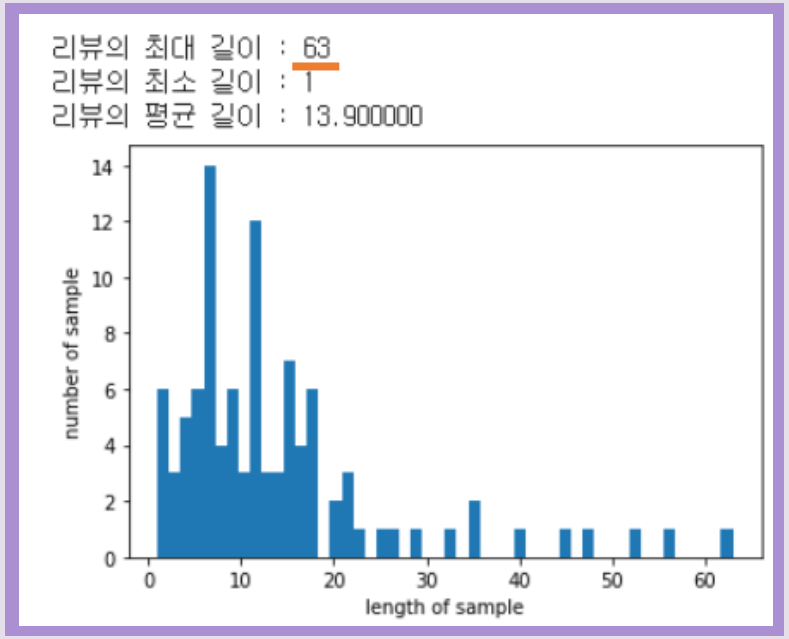
```
encoded = []
for line in tokenized: #입력 데이터에서 1줄씩 문장을 읽음
    temp = []
    for w in line: #각 줄에서 1개씩 글자를 읽음
      try:
        temp.append(word_to_index[w]) # 글자를 해당되는 정수로 변환
      except KeyError: # 단어 집합에 없는 단어일 경우 unk로 대체된다.
        temp.append(word_to_index['unk']) # unk의 인덱스로 변환

    encoded.append(temp)
```

```
print(encoded[:10])
```

```
[[78, 27, 9, 4, 50, 41, 79, 16, 28, 29], [188, 5, 80, 189, 190, 191, 42, 192, 114, 5, 193, 194, 21, 115, 195, 196, 13, 51, 81,
116, 30, 42, 197, 117, 118, 31, 198, 5, 199, 200, 17, 114, 7, 82, 52, 17, 43, 201, 5, 202, 4, 203, 14, 7, 83, 32, 204, 84], [20
5, 119, 206, 53, 207, 31, 208, 209, 54, 10, 25, 11], [44, 33, 120, 210, 211, 212, 213, 68, 45, 34, 13, 214, 121, 15, 2, 215, 6
9, 8, 33, 3, 35], [216, 217, 218, 219, 7, 220, 17, 3], [122, 5, 21, 36, 43, 123, 124, 53, 118, 31, 85, 5, 14, 7, 3], [125, 37,
221, 41, 79, 37], [120, 222, 55, 223, 55, 86, 224, 46, 9, 4, 47, 25], [56], [225, 87, 88, 226, 227, 57, 89]]
```

단어집합(Vocabulary)



패딩 -> 모든 리뷰의 길이를 63으로 통일

```
for line in encoded:
    if len(line) < max_len: # 현재 샘플이 정해준 길이보다 짧으면
        line += [word_to_index['pad']] * (max_len - len(line)) # 나머지는 전부 'pad' 토큰으로 채운다.
```

```
print('리뷰의 최대 길이 : %d' % max(len(l) for l in encoded))
print('리뷰의 최소 길이 : %d' % min(len(l) for l in encoded))
print('리뷰의 평균 길이 : %f' % (sum(map(len, encoded))/len(encoded)))
```

```
리뷰의 최대 길이 : 63
리뷰의 최소 길이 : 63
리뷰의 평균 길이 : 63.000000
```

토치텍스트 튜토리얼(Torchtext tutorial)

토치텍스트(Torchtext)
텍스트에 대한 여러 추상화 기능을 제공하는 자연어 처리 라이브러리

훈련 데이터와 테스트 데이터로 분리

```
df = pd.read_csv('IMDb_Reviews.csv', encoding='latin1')
df.head()
```

	review	sentiment
0	My family and I normally do not watch local mo...	1
1	Believe it or not, this was at one time the wo...	0
2	After some internet surfing, I found the "Home...	0
3	One of the most unheralded great works of anim...	1
4	It was the Sixties, and anyone with long hair ...	0

```
print('전체 샘플의 개수 : {}'.format(len(df)))
```

```
전체 샘플의 개수 : 50000
```

```
train_df = df[:25000]
test_df = df[25000:]
```

```
train_df.to_csv("train_data.csv", index=False)
test_df.to_csv("test_data.csv", index=False)
```

토치텍스트 튜토리얼(Torchtext tutorial)

필드 정의하기(torchtext.data)

```
TEXT = data.Field(sequential=True,
                  use_vocab=True,
                  tokenize=str.split,
                  lower=True,
                  batch_first=True,
                  fix_length=20)

LABEL = data.Field(sequential=False,
                   use_vocab=False,
                   batch_first=False,
                   is_target=True)
```

- sequential : 시퀀스 데이터 여부.
- use_vocab : 단어 집합을 만들 것인지 여부.
- tokenize : 어떤 토큰화 함수를 사용할 것인지 지정.
- lower : 영어 데이터를 전부 소문자화한다.
- batch_first : 미니 배치 차원을 맨 앞으로 하여 데이터를 불러올 것인지 여부.
- is_target : 레이블 데이터 여부.
- fix_length : 최대 허용 길이. 이 길이에 맞춰서 패딩 작업(Padding)이 진행된다.

토치텍스트 튜토리얼(Torchtext tutorial)

데이터셋 만들기

```
train_data, test_data = TabularDataset.splits(
        path='.', train='train_data.csv', test='test_data.csv', format='csv',
        fields=[('text', TEXT), ('label', LABEL)], skip_header=True)
```

```
print('훈련 샘플의 개수 : {}'.format(len(train_data)))
print('테스트 샘플의 개수 : {}'.format(len(test_data)))
```

```
훈련 샘플의 개수 : 25000
테스트 샘플의 개수 : 25000
```

```
{'text': ['my', 'family', 'and', 'i', 'normally', 'do', 'not', 'watch',
'they', 'are', 'poorly', 'made,', 'they', 'lack', 'the', 'depth,', 'and',
e', 'trailer', 'of', '"nasaan', 'ka', 'man"', 'caught', 'my', 'attention,
```

토치텍스트 튜토리얼(Torchtext tutorial)

단어 집합(Vocabulary) 만들기

```
TEXT.build_vocab(train_data, min_freq=10, max_size=10000)
```

```
print('단어 집합의 크기 : {}'.format(len(TEXT.vocab)))
```

```
단어 집합의 크기 : 10002
```

```
print(TEXT.vocab.stoi)
```

```
defaultdict(<bound method Vocab._default_unk_index of <torchtext.
1, 'the': 2, 'a': 3, 'and': 4, 'of': 5, 'to': 6, 'is': 7, 'in': 8
4, 'as': 15, 'for': 16, 'with': 17, 'but': 18, 'on': 19, 'movie':
e': 26, 'he': 27, 'be': 28, 'at': 29, 'one': 30, 'by': 31, 'an':
o': 38, 'just': 39, 'or': 40, 'has': 41, 'about': 42, "it's": 43,
9, 'when': 50, 'more': 51, 'there': 52, 'even': 53, 'would': 54,
```

토치텍스트 튜토리얼(Torchtext tutorial)

토치텍스트의 데이터로더 만들기

```
train_loader = Iterator(dataset=train_data, batch_size = batch_size)
test_loader = Iterator(dataset=test_data, batch_size = batch_size)
```

```
print('훈련 데이터의 미니 배치 수 : {}'.format(len(train_loader)))
print('테스트 데이터의 미니 배치 수 : {}'.format(len(test_loader)))
```

```
훈련 데이터의 미니 배치 수 : 5000
테스트 데이터의 미니 배치 수 : 5000
```

```
print(batch.text)
```

```
tensor([[   9,   39,  202,    0,    0,    4,  388,   12,
          19,   10,  138,    0,    7,    3,  838, 3546],
```

원-핫 인코딩(One-hot encoding)

One-hot encoding

단어 집합의 크기를 벡터의 차원으로 하고,
표현하고 싶은 단어의 인덱스에 1의 값을 부여하고, 다른 인덱스에는 0을 부여하는 단어의 벡터 표현 방식

(1) 각 단어에 고유한 인덱스를 부여합니다. (정수 인코딩)
(2) 표현하고 싶은 단어의 인덱스의 위치에 1을 부여하고, 다른 단어의 인덱스의 위치에는 0을 부여합니다.

"나는 자연어 처리를 배운다 "

```
from konlpy.tag import Okt
okt = Okt()
token = okt.morphs("나는 자연어 처리를 배운다")
print(token)
```

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

토큰화

```
word2index = {}
for voca in token:
     if voca not in word2index.keys():
       word2index[voca] = len(word2index)
print(word2index)
```

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

각 토큰에 대한 인덱스 부여

원-핫 인코딩(One-hot encoding)

One-hot encoding

```
def one_hot_encoding(word, word2index):
        one_hot_vector = [0]*(len(word2index))
        index = word2index[word]
        one_hot_vector[index] = 1
        return one_hot_vector
```

```
one_hot_encoding("자연어",word2index)
```

```
[0, 0, 1, 0, 0, 0]
```

인덱스 : 2

One-hot encoding 한계

(1) 단어의 개수가 늘어날 수록, 벡터를 저장하기 위해 필요한 공간이 계속 늘어난다
i.e 단어 집합의 크기 = 벡터의 차원 수
(2) 단어의 유사도를 표현하지 못한다
강아지, 개, 냉장고 -> [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]
-> 강아지라는 단어가 개와 냉장고라는 단어 중 어떤 단어와 더 유사한지도 알 수 없다.
-> **검색시스템** : 단어간 유사성을 계산할 수 없으면, 연관검색어를 보여줄 수 없다.

워드 임베딩(Word Embedding)

Word Embedding
: 단어를 벡터로 표현하는 것

(1) 희소 표현 (Sparse Representation)
: 벡터 또는 행렬(matrix)의 값이 대부분이 0으로 표현되는 방법 -> 공간낭비, 유사도x

(2) 밀집 표현 (Dense Representation)
: 벡터의 차원을 단어 집합의 크기로 상정하지 않습니다.
사용자가 설정한 값으로 모든 단어의 벡터 표현의 차원을 맞춥니다. 실수값을 가지게 됩니다.
Ex) 강아지 = [0.2 1.8 1.1 -2.1 1.1 2.8 ... # 차원 : 128

(3) 워드 임베딩 (Word Embedding)
: 단어를 밀집 벡터의 형태로 표현하는 방법

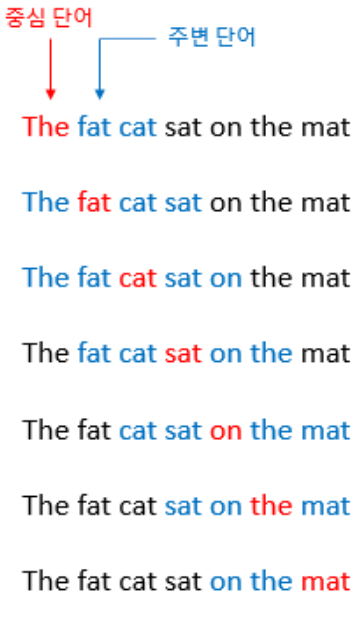
-	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

워드투벡터(Word2Vec)

Word2Vec
: 단어 간 유사도를 반영할 수 있도록 단어의 의미를 벡터화 할 수 있는 방법

(1) 분산 표현 (Distributed Representation)
: **비슷한 위치에서 등장하는 단어들은 비슷한 의미를 가진다**
벡터의 차원이 단어 집합(vocabulary)의 크기일 필요가 없으므로, 상대적으로 저차원
단어의 의미를 여러 차원에다가 분산 -> 단어 간 유사도를 계산
Ex) 강아지 = [0.2 0.3 0.5 0.7 0.2 ... 0.2]

(2) CBOW (Continuous Bag of Words)
: 주변에 있는 단어로 중간 단어들을 예측하는 방법
예문 : "The fat cat sat on the mat"
- {"The", "fat", "cat", "on", "the", "mat"}으로부터 sat을 예측
- 윈도우 (window)
: 중심 단어를 예측하기 위해서 앞, 뒤로 볼 단어 수
- **슬라이딩 윈도우 (sliding window)**
: 윈도우를 계속 움직여서 주변 단어와 중심 단어 선택을 바꿔가며 학습을 위한 데이터 셋 제작



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, 1]
[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, 1]	[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0]

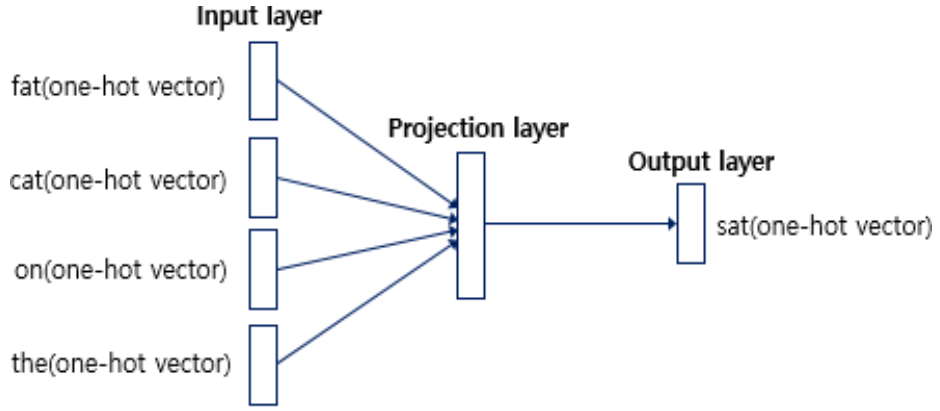
워드투벡터(Word2Vec)

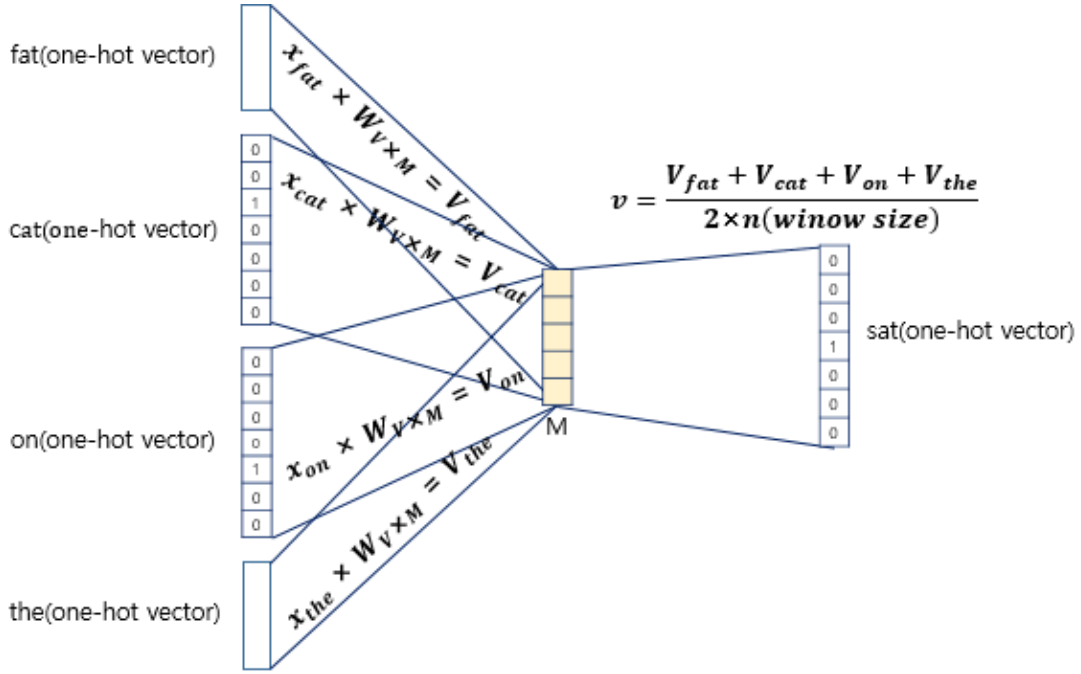
- CBOW의 인공 신경망

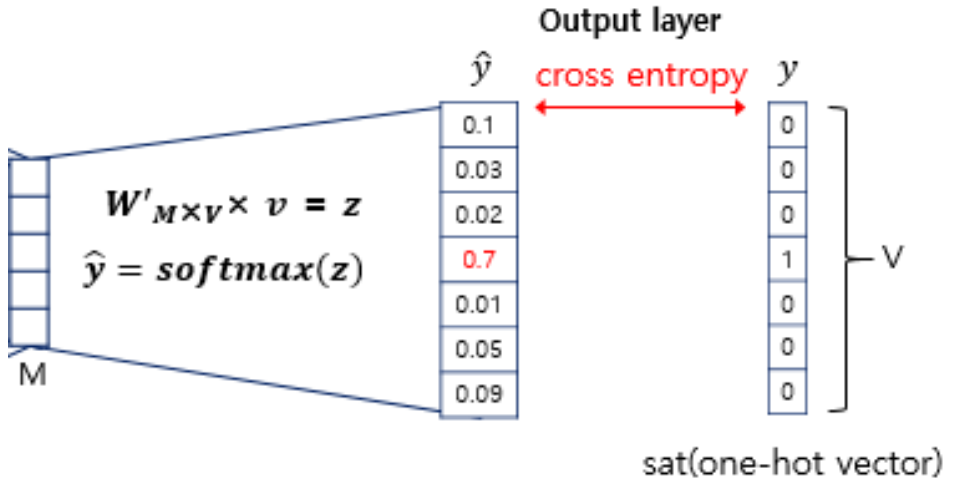
입력층 : 앞, 뒤로 사용자가 정한 윈도우 크기 범위 안 주변 단어들의 원-핫 벡터

출력층 : 예측하고자 하는 중간 단어의 원-핫 벡터

-> 주변 단어로 중심 단어를 더 정확히 맞추기 위해 계속해서 이 W와 W'를 학습해가는 구조

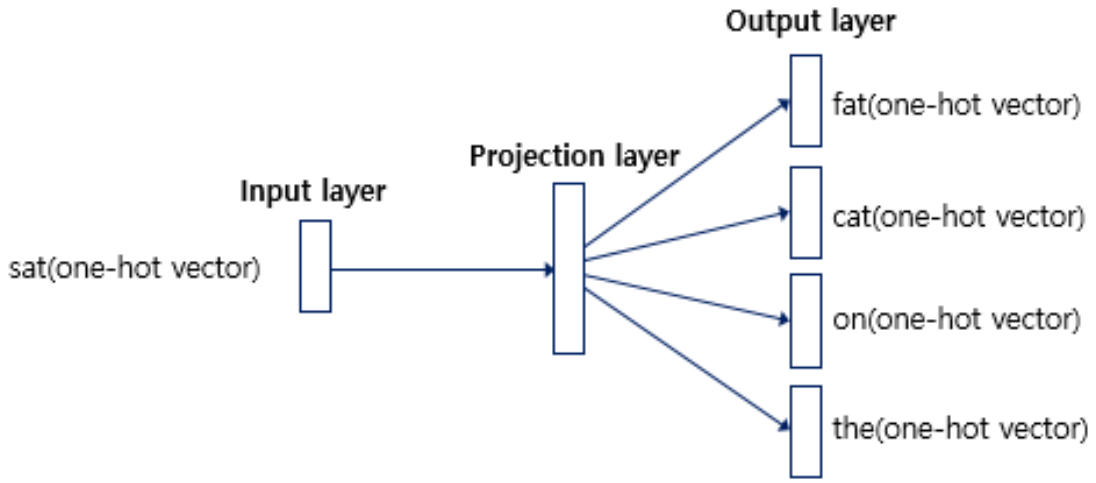






워드투벡터(Word2Vec)

(3) Skip-gram
: 중심 단어에서 주변 단어를 예측



(4) 네거티브 샘플링(Negative Sampling)
: 기존 Word2Vec 보다 연산량에 있어서 훨씬 효율적