Natural Language Processing 2

자연어 처리 2

01 자연어 처리 2

© Elice. All Rights Reserved

목차 자연어 처리 2

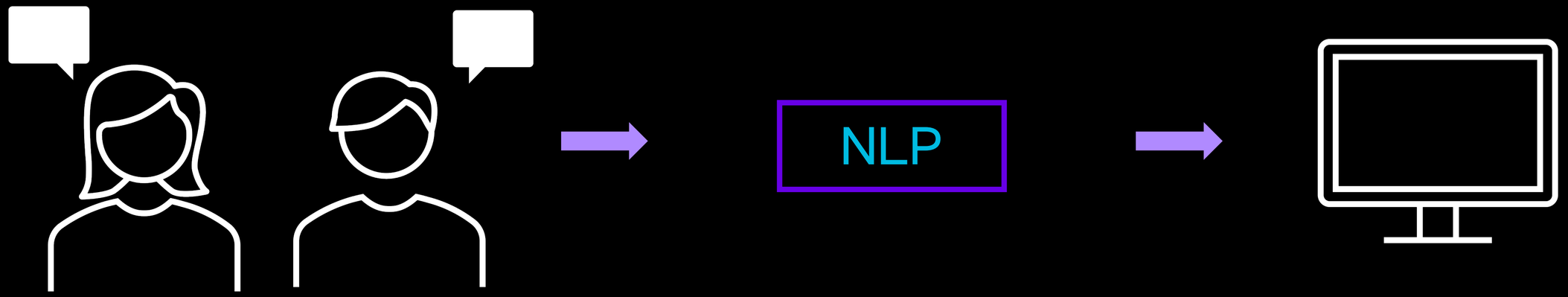
© Elice. All Rights Reserved

1. 자연어 처리 개요 및 텍스트 벡터화

© Elice. All Rights Reserved

자연어 처리(NLP)란?

- Natural Language Processing(NLP)은 컴퓨터가 인간의 언어를 이해하고 처리하며 생성할 수 있도록 하는 기술



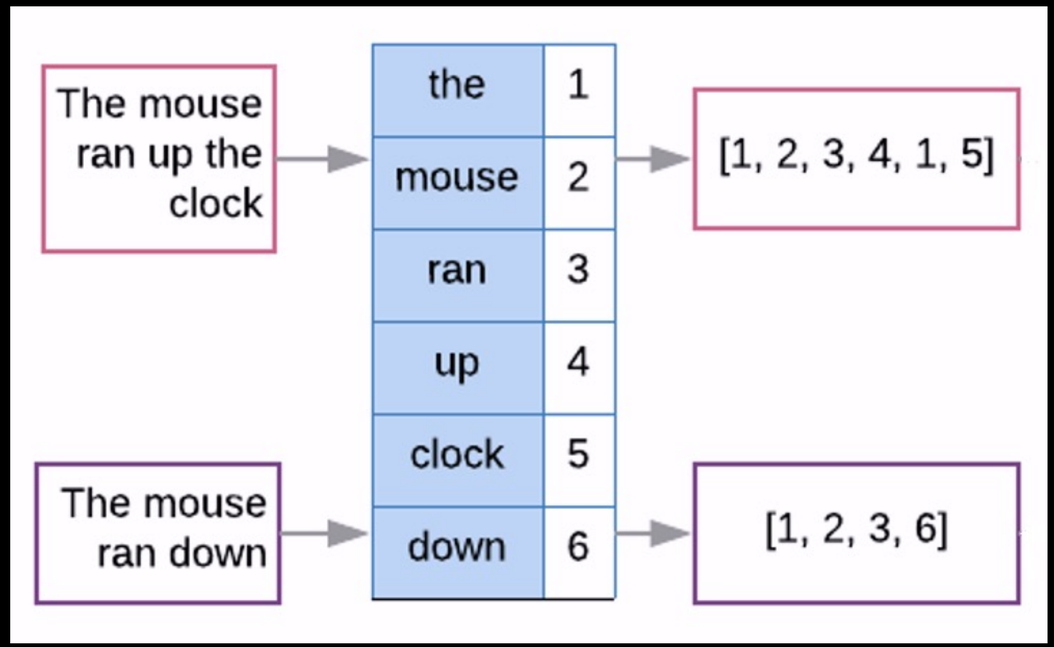
자연어 처리(NLP) 분야

자연어 처리

- 텍스트 분류 : “이 텍스트의 주제는?”
- 내용 필터링 : “욕설 사용? "
- 감성 분석 : “그래서 내용이 긍정이야 부정이야?
- 언어 모델링 : “이 문장 다음에 어떤 단어가 와야 할까?”
- 번역 : “이거를 한국어로 어떻게 말해?”
- 요약 : “이 기사를 한 줄로 요약해 볼래?”

텍스트 벡터화

- Text vectorization은 딥러닝 모델 사용을 위해 텍스트를 수치형 tensor로 변환하는 과정
- 일반적으로 세 단계로 진행
    1. Text standardization
    2. Tokenization
    3. Vocabulary indexing



출처: https://developers.google.com/machine-learning/guides/text-classification/step-3?hl=ko

Text Standardization

- Text standardization은 텍스트의 형식을 맞춰주는 단계
- 예시

1. 모든 문자를 소문자로 변환
2. 특수 기호 제거(., ;, ?, ‘ 등)
3. 동사/명사의 기본형 활용(cats=>

[cat], was staring or stared => [stare]

“The mouse ran up the clock!”

↓

“the mouse ran up the clock!”

“the mouse ran up the clock!”

↓

“the mouse ran up the clock”

출처: https://developers.google.com/machine-learning/guides/text-classification/step-3?hl=ko

Tokenization

- Tokenization은 텍스트를 단어 또는 문장 단위의 작은 token으로 나누는 단계
- 예시

“the mouse ran up the clock”

the	1
mouse	2
ran	3
up	4
clock	5

출처: https://developers.google.com/machine-learning/guides/text-classification/step-3?hl=ko

Tokenization

- 단어 기준 토큰화
  - 공백으로 쪼개기
  - 경우에 따라 동사 어근과 어미를 구분 : star+ing, call+ed 등
- N-그램 토큰화
  - N-그램 토큰 : 연속으로 위치한 N개(이하)의 단어 묶음
  - ex) the cat, he was 등은 2-그램(바이그램) 토큰
- 문자 기준 토큰화
  - 하나의 문자를 하나의 토큰으로 지정(텍스트 생성, 음성 인식 등에서 활용)

Tokenization

- 2-그램(바이그램) 주머니

```
{"the", "the cat", "cat", "cat sat", "sat",
"sat on", "on", "on the", "the mat", "mat"}
```

- 3-그램 주머니

```
{"the", "the cat", "cat", "cat sat", "the cat sat",
"sat", "sat on", "on", "cat sat on", "on the",
"sat on the", "the mat", "mat", "on the mat"}
```

Vocabulary Indexing

- Vocabulary indexing은 각 token을 고유한 숫자 인덱스로 변환하여 벡터로 표현
- 예시

"the mouse ran up the clock"

[1, 2, 3, 4, 1, 5]

the	1
mouse	2
ran	3
up	4
clock	5

출처: https://developers.google.com/machine-learning/guides/text-classification/step-3?hl=ko

Vocabulary Indexing

- 훈련셋에 포함된 모든 토큰들의 색인(인덱스) 생성
- 보통 사용 빈도가 높은 1만, 2만, 3만 개의 단어만 대상

```
from tensorflow.keras.datasets import imdb

imdb.load_data(num_words=10000)
```

- 사용 빈도수가 상위 1만 등 안에 들지 않는 단어는 무시

Vocabulary Indexing

- 텍스트를 단어 색인으로 구성된 리스트로 변환

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 0, 0, 0, 0, 0]
```

- 0과 1은 특별한 기능 수행
  - 1 : 미등록 어휘 인덱스, 어휘 색인에 미등록된(out-of-vocabulary, OOV) 단어는 모두 1로 처리, 일반 텍스트로 재번역되는 경우 “[UNK]”, unknown 단어로 표현
  - 0 : 텍스트 길이 통일을 위해 패딩으로 사용, 마스크 토큰이라 부르기도 함

TextVectorization 층

- 표준화 : 소문자화, 마침표 제거, 토큰화 : 단어 기준 쪼개기, n-그램 미사용
- 출력모드 : 정수 색인 인코딩, 텍스트 길이 제한 : 지정

```
from tensorflow.keras.layers import TextVectorization

text_vectorization = TextVectorization(
    standardize='lower_and_strip_punctuation',  # 기본값
    split='whitespace',                         # 기본값
    ngrams=None,                                # 기본값
    output_mode='int',                          # 기본값
)
```

TextVectorization 층

- 어휘집 생성 adapt() 메서드

```
>>> dataset = [
...         "I write, erase, rewrite",
...         "Erase again, and then",
...         "A poppy blooms.",
... ]
```

어휘 색인은 `adapt()` 메서드를 이용하여 만든다.

```
>>> text_vectorization.adapt(dataset)
```

텍스트벡터화

- 생성된 어휘 색인은 다음과 같다.

```
>>> vocabulary = text_vectorization.get_vocabulary()
>>> vocabulary
['',
 '[UNK]',
 'erase',
 'write',
 'then',
 'rewrite',
 'poppy',
 'i',
 'blooms',
 'and',
 'again',
 'a']
```

텍스트벡터화

- 생성된 어휘 색인을 활용하여 새로운 텍스트 벡터화

```
>>> test_sentence = "I write, rewrite, and still rewrite again"
>>> encoded_sentence = text_vectorization(test_sentence)
>>> print(encoded_sentence)
tf.Tensor([ 7  3  5  9  1  5 10], shape=(7,), dtype=int64)
```

- ‘still’ 단어는 1로 변환되며 길이 설정은 없기에 0, 마스크 토큰 패딩은 사용되지 않음

텍스트벡터화

- 벡터화된 텐서로부터 텍스트를 복원하면 표준화된 텍스트 생성

```
>>> inverse_vocab = dict(enumerate(vocabulary))
>>> decoded_sentence = " ".join(inverse_vocab[int(i)] for i in
encoded_sentence)
>>> print(decoded_sentence)
i write rewrite and [UNK] rewrite again
```

- 1로 변환된 'still'은 [UNK]로 복원

텍스트벡터화

- 벡터화된 텐서로부터 텍스트를 복원하면 표준화된 텍스트 생성

```
>>> inverse_vocab = dict(enumerate(vocabulary))
>>> decoded_sentence = " ".join(inverse_vocab[int(i)] for i in
encoded_sentence)
>>> print(decoded_sentence)
i write rewrite and [UNK] rewrite again
```

텍스트벡터화

- TextVectorization 층은 GPU or TPU에서 지원되지 않음
- 모델 구성에 직접 사용하는 방식은 모델 훈련을 늦출 수 있기에 권장되지 않음
- 전처리는 모델 구성과 독립적으로 처리
- 훈련이 끝난 모델을 실전 배치할 경우 TextVectorization층을 완성된 모델에 추가해서 사용

Embedding : bag of words

- 문장속 단어 빈도

	학교에	가서	수업을	들었다	온건	오랜만이다	친구	얘기를	내일	뭐	먹지
학교에 가서 수업을 들었다. 학교에 온건 오랜만이다.	2	1	1	1	1	1	0	0	0	0	0
학교에 가서 친구 얘기를 들었다.	1	1	0	1	0	0	1	1	0	0	0
내일 가서 뭐 먹지?	0	0	0	0	0	0	0	0	1	1	1

- 학교에 가서 친구 얘기를 들었다 => [1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0]
- 숫자 분포를 통해 1번 문장은 3번 문장보다 2번 문장과 유사함을 알 수 있음

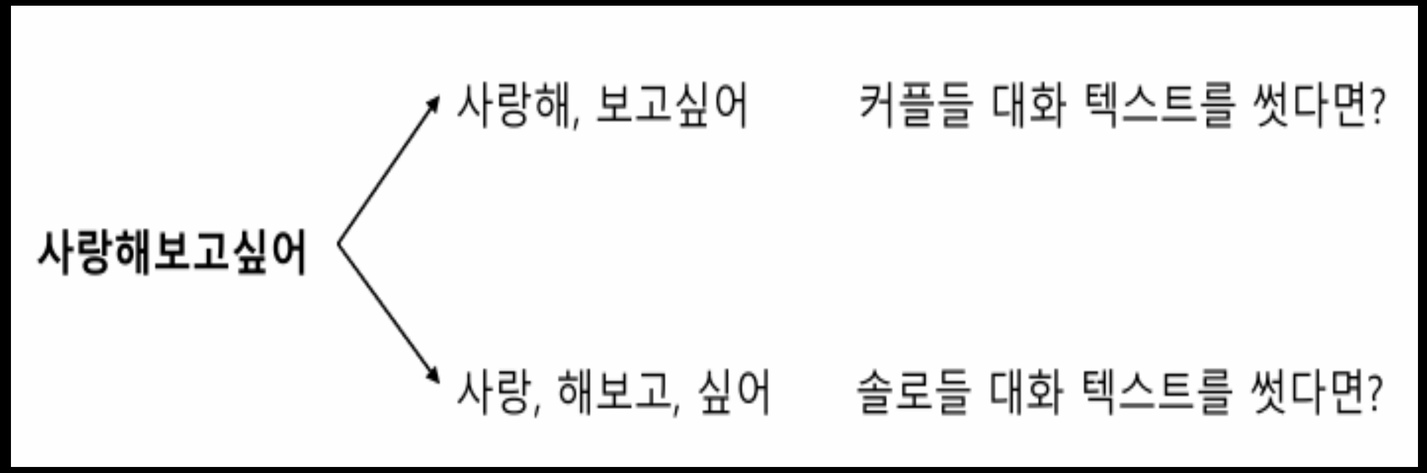
Embedding : bag of words

- 빈도만 고려할 뿐 순서 고려 X
- 학교에 가서 친구 얘기를 들었다. → [1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] → [학교에, 가서, 들었다, 친구, 얘기를]
- 단어 개수가 늘어나면 배열의 크기가 10만개~100만개로 엄청나게 커지지만 실제로 대부분의 값은 0

Tokenizer

- 띄어쓰기 적용 => [오늘, 저녁은, 맛있었다]
  - 명료하고 쉬우며 '맛있다/맛있어요/맛있었다'가 모두 다르게 인식(단어 사진이 커짐)
- 문자 단위 => [오, 늘, 저, 녁, 은, 맛, 있, 었, 다]
  - 각 token이 의미를 담지 못함
- Subword 단위 => [오늘, 저녁, 은, 맛있, 었, 다]
  - 형태소로 나누기(컴퓨터가 알아서 할 수 없음. Ex. '학교' + '에')
  - BPE(byte-pair encoding) : 전체 문서를 문자 단위로 쪼갠 뒤 빈번하게 나오는 문자들을 묶어 단어 수를 줄임. Ex. '학', '교'를 따로 쓰는 것보다 '학교 ' 가 많으면 후자 선택

Tokenizer



- Subword tokenizer도 학습해야 하며, 학습한 텍스트 데이터셋에 따라 결과가 다름
  - 띄어쓰기나 문자단위 tokenizer는 학습 필요 X, tokenizer 학습은 인공지능 모델 학습과 별도로 이루어짐
  - 모든 자연어처리 과정은 tokenizer로 부터 시작

Embedding: TFIDF

- TFIDF (Term Frequency-Inverse Document Frequency)
  - 오늘은 기분이 좋다 →오늘, 은, 기분, 이, 좋, 다 the book is on the desk
  - 내일은 수업이 없다 →내일, 은, 수업, 이, 없, 다 the sky and the see
  - 기분이 최고 →기분, 이, 최고 book and pencil
- 빈번하지만 문장의 특징을 나타내지 않는 단어(a, the, 조사 등)의 가중치 낮추기
- 단어가 나온 문장의 수로 나누기
- 다른 문장에는 자주 나오지 않지만 특정 문장에서는 빈번하게 등장하는 단어 => 키워드

Embedding: TFIDF

TFIDF ~ log(해당 문장의 단어 수 / 단어가 나오는 문장 수)

Bag of words에 해당

여러 문장에 나올 수록 단어의 가중치를 낮춤

- Term frequency (원래 bag of words): 숫자 배열은 문장 내 단어의 빈도수
- TFIDF(업그레이드된 bag of words): 숫자 배열은 문장 내 단어의 중요도

2. 텍스트 벡터화 예제

© Elice. All Rights Reserved

텍스트벡터화 예제

- IMDB 영화 후기 데이터셋(5만 개의 영화 후기) 다운로드 후 압축 풀기

```
aclImdb/
...test/
......pos/
......neg/
...train/
......pos/
......neg/
```

텍스트벡터화 예제

- 검증셋 준비

```
aclImdb/
...test/
......pos/
......neg/
...train/
......pos/
......neg/
...val/
......pos/
......neg/
```

- train/val/test폴더의 각 pos와 neg 서브디렉토리는 긍정과 부정 후기를 포함

텍스트벡터화 예제

- 텐서 데이터셋 준비

```
from tensorflow.keras.utils import text_dataset_from_directory

batch_size = 32

train_ds = text_dataset_from_directory("aclImdb/train",
                                       batch_size=batch_size)

val_ds = text_dataset_from_directory("aclImdb/val",
                                     batch_size=batch_size)

test_ds = text_dataset_from_directory("aclImdb/test",
                                      batch_size=batch_size)
```

텍스트벡터화 예제

- 텐서 데이터셋 준비

```
for inputs, targets in train_ds:
    print("inputs.shape:", inputs.shape)
    print("inputs.dtype:", inputs.dtype)
    print("targets.shape:", targets.shape)
    print("targets.dtype:", targets.dtype)

    # 예제: 첫째 배치의 첫째 후기
    print("inputs[0]:", inputs[0])
    print("targets[0]:", targets[0])
    break
```

- tf.string은 문자열 텐서와 타깃은 int32 정수 텐서
- tf.string은 파이썬 기본 자료형은 str과 다름 주의

텍스트벡터화 예제


```
inputs.shape: (32,)
inputs.dtype: <dtype: 'string'>
targets.shape: (32,)
targets.dtype: <dtype: 'int32'>

inputs[0]: tf.Tensor(b'The film begins with a bunch of kids in reform school and
focuses on a kid named \'Gabe\', who has apparently worked hard to earn his parole.
Gabe and his sister move to a new neighborhood to make a fresh start and soon Gabe
meets up with the Dead End Kids. The Kids in this film are little punks, but they
are much less antisocial than they\'d been in other previous films and down deep,
they are well-meaning punks. However, in this neighborhood there are also some
criminals who are perpetrating insurance fraud through arson and see Gabe as a
convenient scapegoat--after all, he\'d been to reform school and no one would
believe he was innocent once he was framed. So, when Gabe is about ready to be sent
back to "The Big House", it\'s up to the rest of the gang to save him and expose
the real crooks.<br /><br />The "Dead End Kids" appeared in several Warner Brothers
films in the late 1930s and the films were generally very good (particularly ANGELS
WITH DIRTY FACES). However, after the boys\' contracts expired, they went on to
Monogram Studios and the films, to put it charitably, were very weak and formulaic
--with Huntz Hall and Leo Gorcey being pretty much the whole show and the group
being renamed "The Bowery Boys". Because ANGELS WASH THEIR FACES had the excellent
writing and production values AND Hall and Gorcey were not constantly mugging for
the camera, it\'s a pretty good film--and almost earns a score of 7 (it\'s REAL
close). In fact, while this isn\'t a great film aesthetically, it\'s sure a lot of
fun to watch, so I will give it a 7! Sure, it was a tad hokey-particularly towards
the end when the kids take the law into their own hands and Reagan ignores the Bill
of Rights--but it was also quite entertaining. The Dead End Kids are doing their
best performances and Ronald Reagan and Ann Sheridan provided excellent support.
Sure, this part of the film was illogical and impossible but somehow it was still
funny and rather charming--so if you can suspend disbelief, it works well.',
shape=(), dtype=string)
targets[0]: tf.Tensor(1, shape=(), dtype=int32)
```

텍스트벡터화 예제

- 텍스트 벡터화

```
max_length = 600     # 후기 길이 제한
max_tokens = 20000   # 단어 사용빈도 제한

text_vectorization = layers.TextVectorization(
                        max_tokens=max_tokens,
                        output_mode="int",
                        output_sequence_length=max_length,
                    )
```

- 평균적으로 233개의 단어 사용, 600개 단어 이상 후기는 전체의 5%
- 600개보다 적은 수의 단어라면 마스크 토큰 0을 패딩으로 사용

텍스트벡터화 예제

- 어휘 색인화

```
# 어휘 색인 생성 대상 훈련셋 후기 텍스트 데이터셋
text_only_train_ds = train_ds.map(lambda x, y: x)

# 어휘 색인
text_vectorization.adapt(text_only_train_ds)
```

텍스트벡터화 예제

- 데이터셋 벡터화

```
# 후기를 길이가 2만인 정수들의 리스트로 변환
int_train_ds = train_ds.map(lambda x, y: (text_vectorization(x), y))

int_val_ds = val_ds.map(lambda x, y: (text_vectorization(x), y))

int_test_ds = test_ds.map(lambda x, y: (text_vectorization(x), y))
```

텍스트벡터화 예제

- 첫째 배치의 첫째 샘플

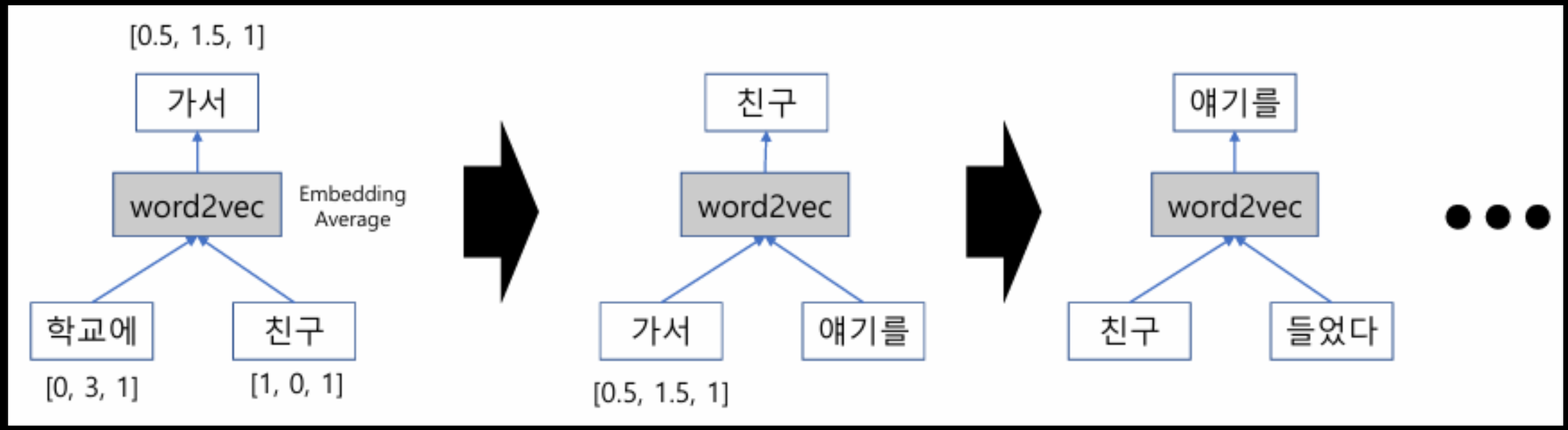
```
tf.Tensor(
[   11     7     4  8614    18    38     9   139   138   197   640    12
    30    22   167     6  3035     2    86  3146   664    19    12   291
    11    14  2400  2996    13    55   322   429    11    19   172     4
   337    35   116   230   172     4  1107     2   196  1562    14    12
    10   399     9   100     9    14   478    46  1368   162    31    47
   509    56     2  7585   645    66   733     5   239  1428     1    17
     2    86    18     3    56    47   645    12    23    66     6    28
   ...
   ...
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0],
shape=(600,), dtype=int64)
```

3. 단어 임베딩

© Elice. All Rights Reserved

Embedding: Word2Vec(2013)

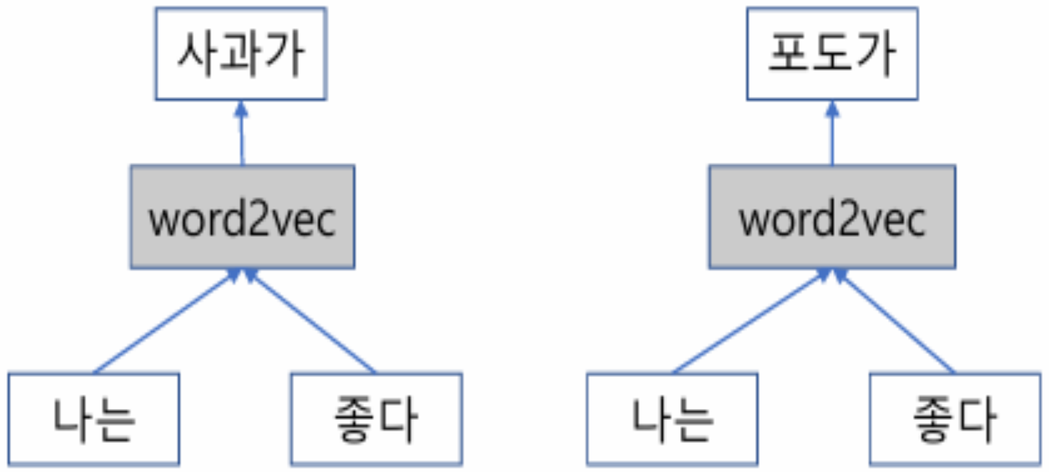
- 주변 맥락으로 단어를 표현(소수를 통해 더 적은 크기의 배열로 다양한 단어 표현)



- 초기 단어를 임의의 배열로 셋팅
- 주변 단어의 배열로 관심 단어의 배열을 만듬
- 내가 가진 모든 텍스트에 대해 반복 학습

Embedding: Word2Vec(2013)

- 유사단어끼리 유사한 배열을 가지게 됨



문맥상 '사과가', '포도가' 이 두개가 비슷한 것을 학습 가능

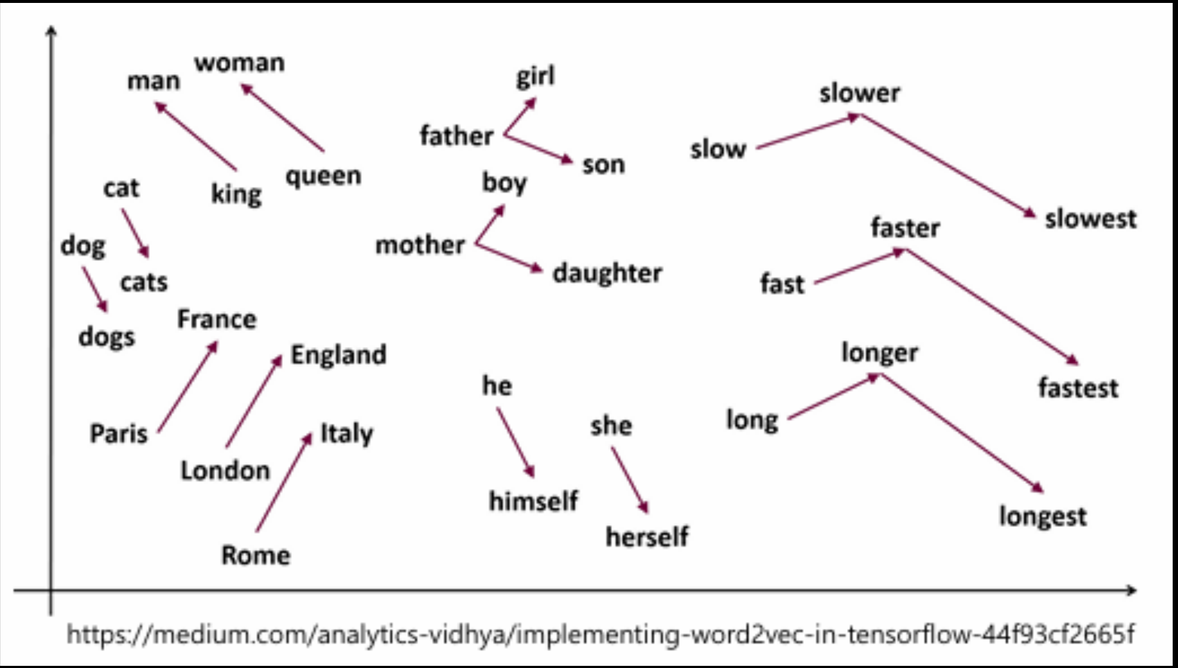
Word2vec을 이용한 단어 유사도 평가 예시

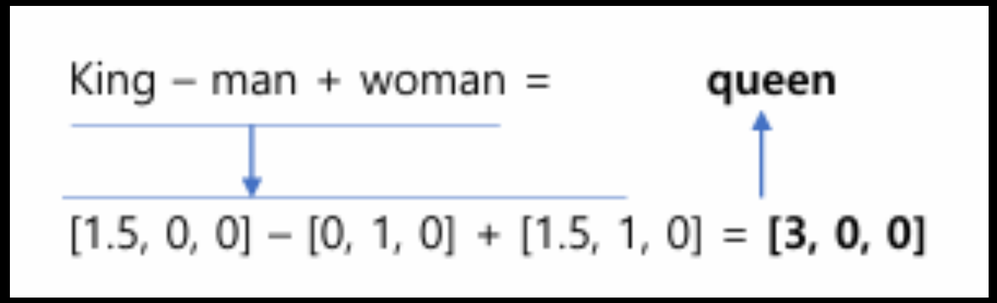
감동	고등학교	이제까지	명작
웃음	초등학교	10년동안	명작.
그자체	중학교	3년간	명작이다.
소름	이제까지	여태껏	불후의
재미와	근	재난영화중	명작중의
영상	2학년때	머리털나고	걸작

26MB의 네이버 영화 텍스트 데이터로 1분간 학습한 결과

Embedding: Word2Vec(2013)

- 배열은 단어의 주변 맥락을 표현. 이를 더하고 뺌으로써 단어 사이의 관계 파악

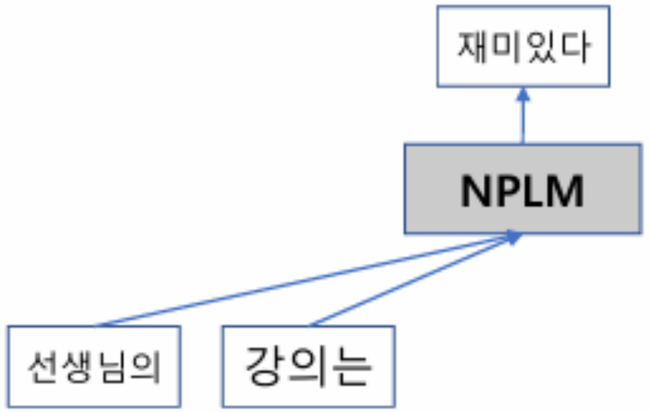




- 매우 빠른 속도와 합리적인 성능으로 현재에도 속도가 중요한 상용시스템에 사용됨

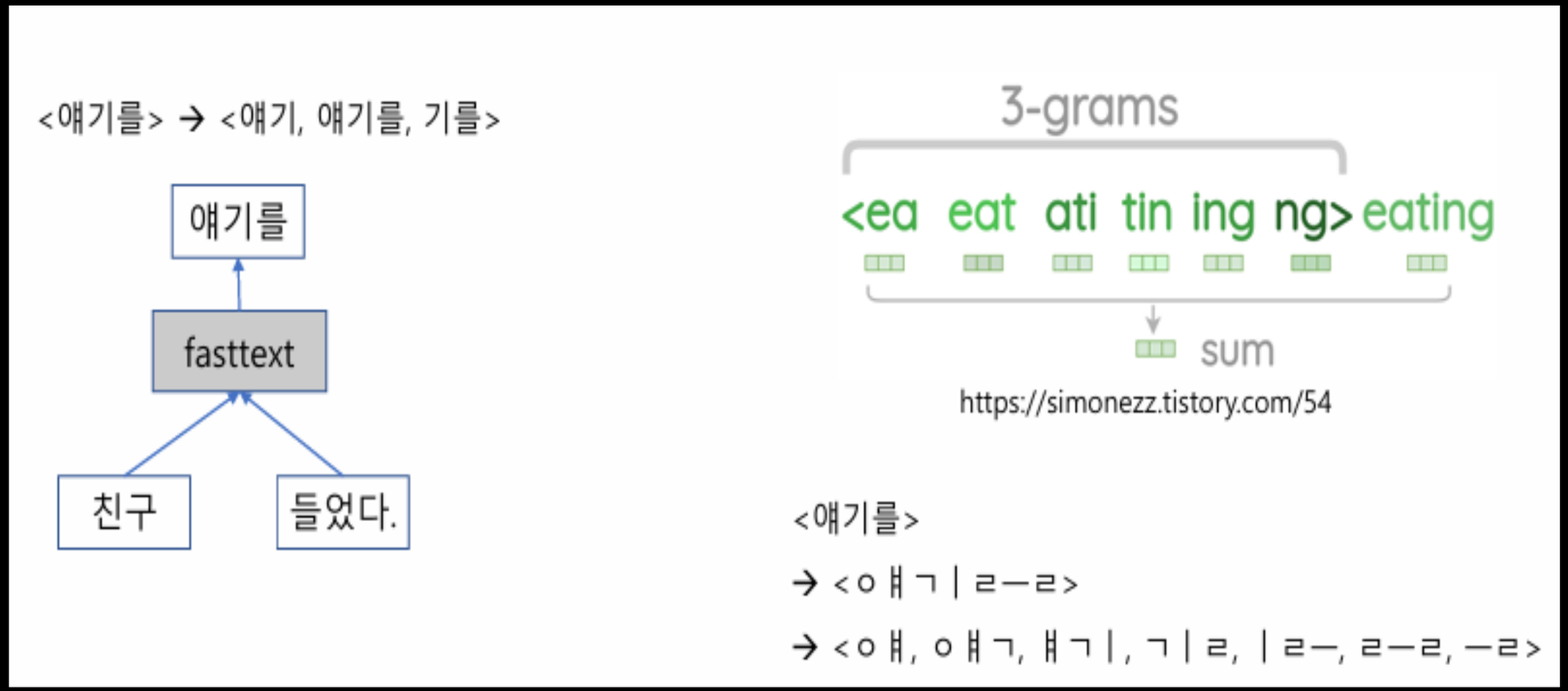
Word2vec의 변형

학습 문장: 선생님의 강의는 재미있다



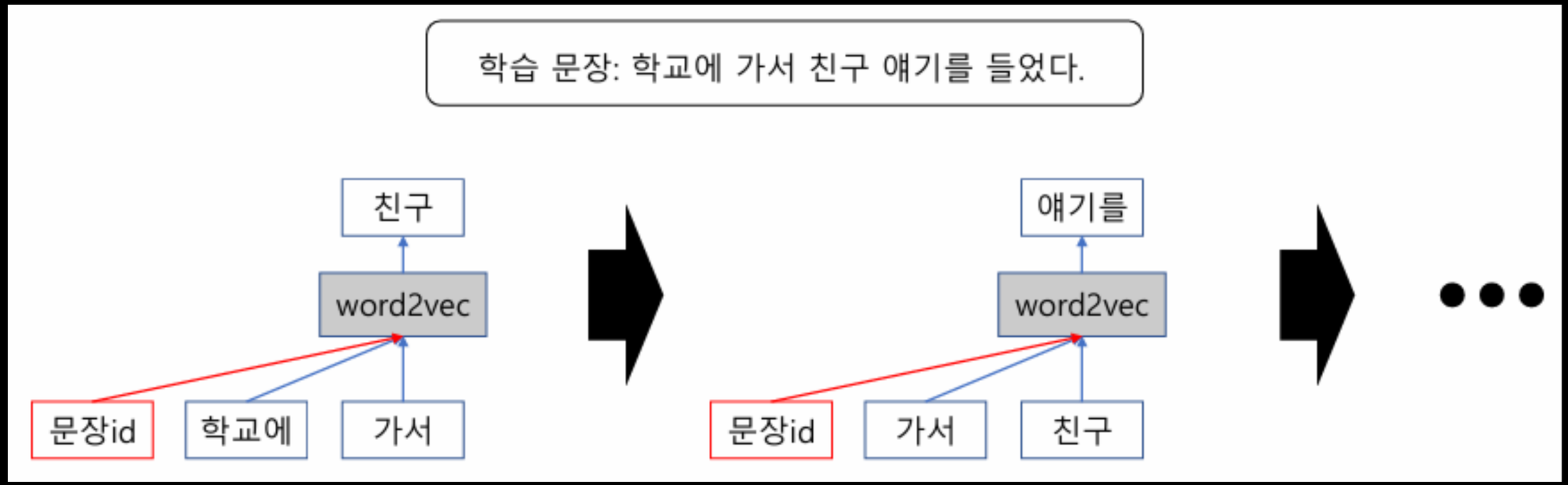
Autoencoding model	Autoregressive model
양방향 정보를 이용	한쪽방향 정보만 이용
문장 이해도가 좋음	문장 이해도는 떨어짐
문장 생성 불가능	**문장 생성** 가능

Embedding: Fasttext



- 단어를 문자단위로 쪼개, 조각마다 word2vec를 수행(오타가 있어도 합리적인 결과)
- 한글의 경우 자모로 쪼개서 할 경우 성능이 향상됨

Sentence embedding: Doc2vec (2014)



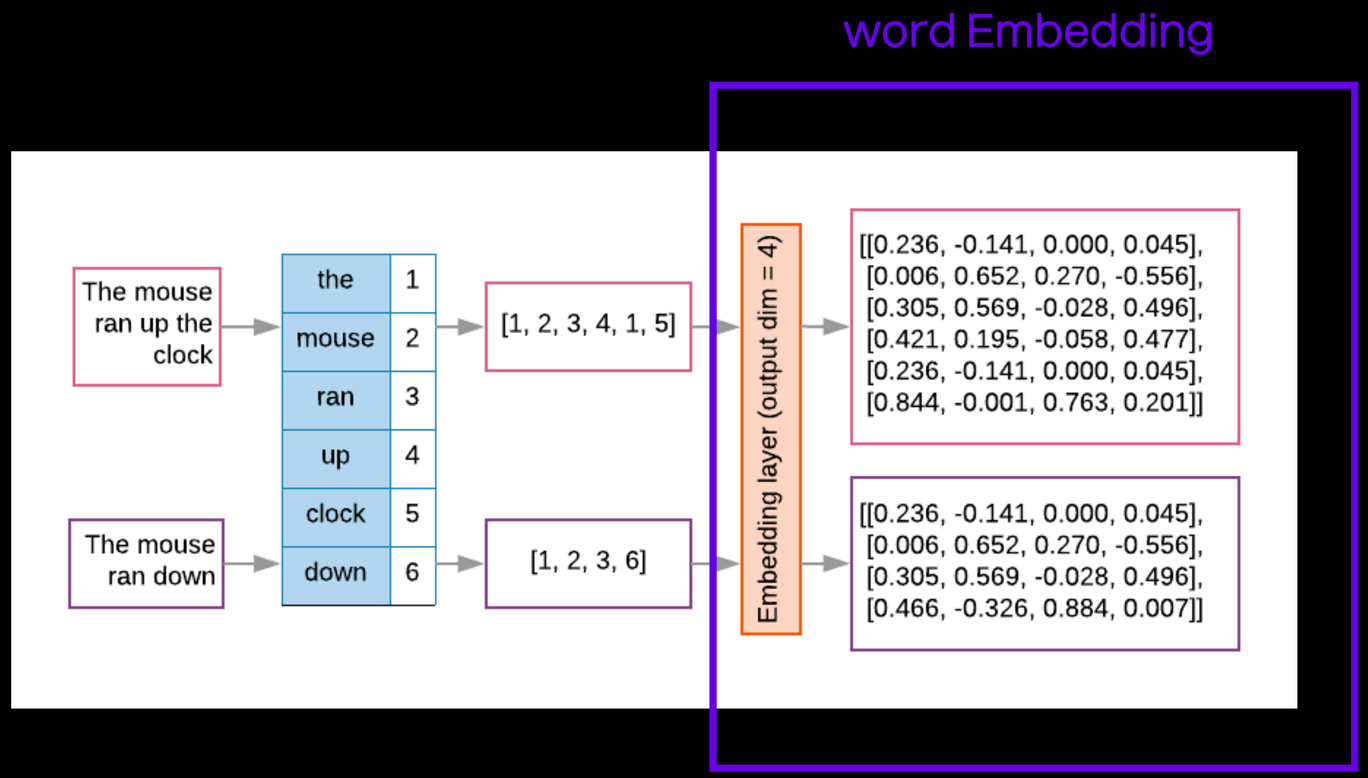
- 문장마다 id를 부여하고 word2vec와 유사한 형태로 학습하고 문장 id 배열을 업데이트
- 비슷한 단어로 이루어진 문장은 비슷한 배열을 가짐

Embedding: Word2Vec 한계점

- 동음이의어를 구분할 수 없음
  - 우리는 말을 보며 말을 나눴다 => horse와 mouth 맥락이 혼재
- 전혀 관계 없더라도 주변 단어가 비슷하면 비슷하게 임베딩
  - 오늘 xxx 갔다 => '학교, 서울, 그곳에' 등 무관한 단어가 비슷하게 표현됨
- 단어 기반의 학습이기에 문장 단위의 맥락을 이해하지 못함
  - 어제 식당에서 친구를 만났다. 거기서 함께 밥을 먹었다 => 거기서가 무엇을 가리키는지 알지 못함

단어 임베딩

- Word Embedding은 고차원 벡터 공간에서 단어의 의미적 연관성을 벡터로 표현
- 연관성 정보는 부동소수점으로 구성된 벡터



출처: https://developers.google.com/machine-learning/guides/text-classification/step-3?hl=ko

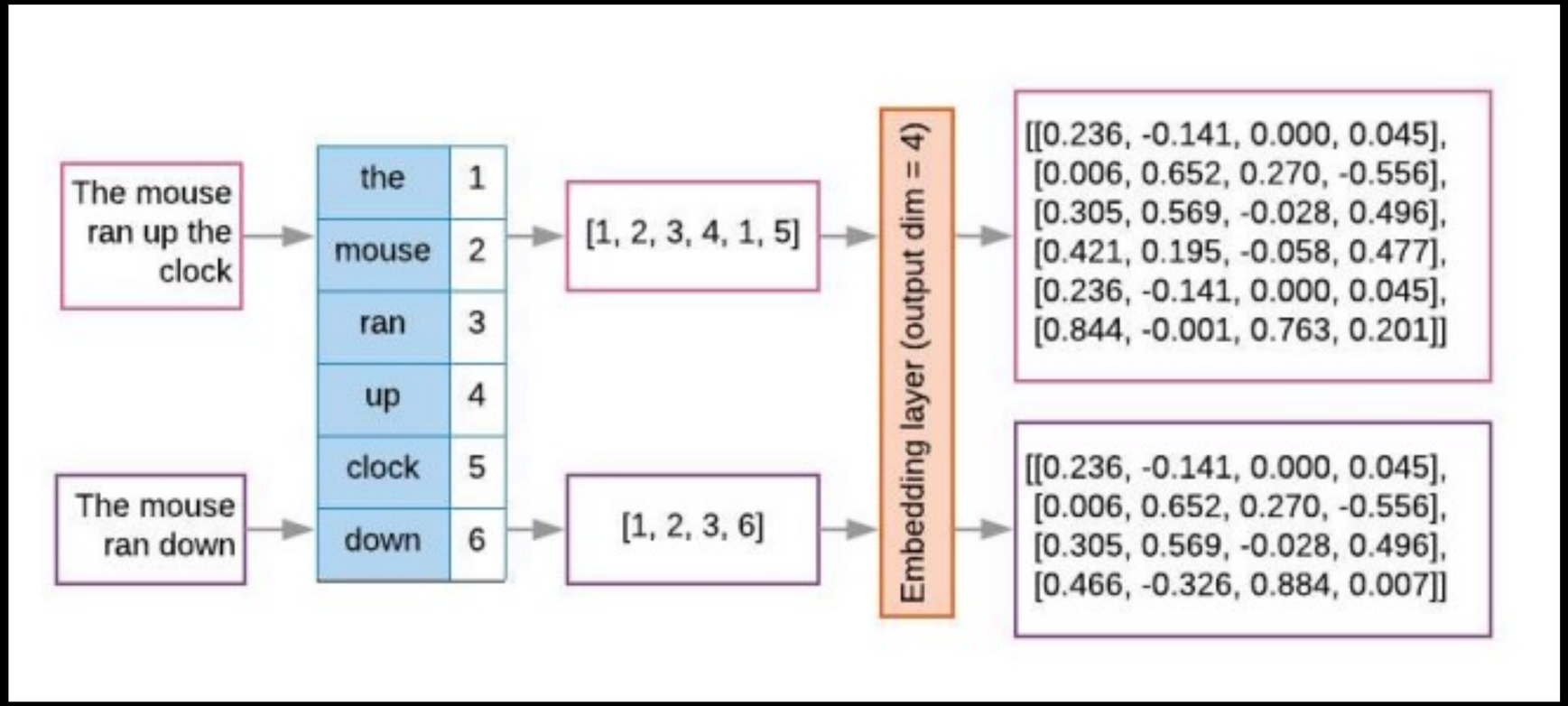
단어 임베딩

- 임베딩층은 (7, 4) 모양의 2차원 텐서로 구성, 훈련이 시작될 때 균등 분포를 이용하여 초기화
- 7x4 개의 항목이 훈련 파라미터로써 학습

```
vocab_size = 7 # 총 어휘 수
embed_dim = 4  # 단어별로 4 개의 특성 파악.

embedding_layer = ayers.Embedding(input_dim=vocab_size,
                                  output_dim=embed_dim)
```

단어 임베딩



- 정 수 색인으로 길이가 6인 벡터 [1, 2, 3, 4, 1, 5]로 변환
- 단어 임베딩을 통해 4개의 무동소수점을 포함한 벡터 6개로 구성된 (6, 4) 텐서로 변환

단어 임베딩

- 단어 임베딩에 사용된 (7, 4) 모양의 2차원 텐서

```
embedding_matrix = [[-0.012, 0.005, 0.008, 0.001],
                    [0.236, -0.141, 0.000, 0.045],
                    [0.006, 0.652, 0.270, -0.556],
                    [0.305, 0.569, -0.028, 0.496],
                    [0.421, 0.195, -0.058, 0.477],
                    [0.844, -0.001, 0.763, 0.201],
                    [0.466, -0.326, 0.884, 0.007]]
```

단어 임베딩

- “The mouse ran up the clock” => [1, 2, 3, 4, 1, 5] => (6, 4)

```
'the'   => 1 => [0.236, -0.141, 0.000, 0.045]
'mouse' => 2 => [0.006, 0.652, 0.270, -0.556]
'ran'   => 3 => [0.305, 0.569, -0.028, 0.496]
'up'    => 4 => [0.421, 0.195, -0.058, 0.477]
'the'   => 1 => [0.236, -0.141, 0.000, 0.045]
'clock' => 5 => [0.844, -0.001, 0.763, 0.201]
```

단어 임베딩

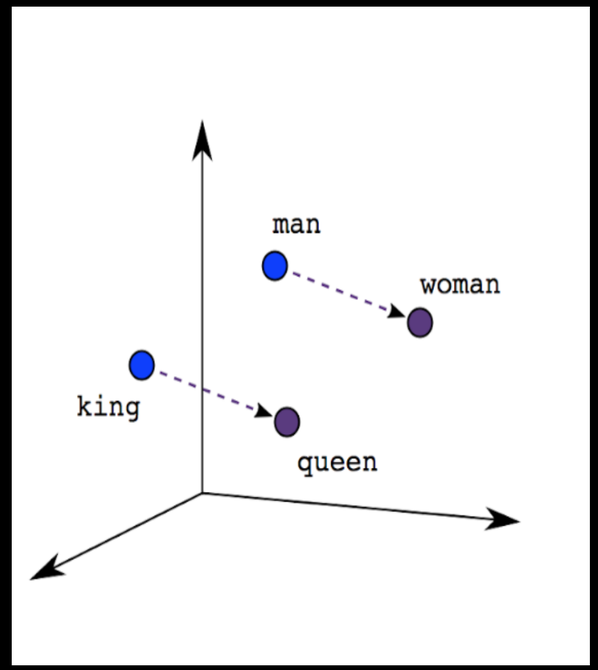
- “The mouse ran down” => [1, 2, 3, 6] => (4, 4)

- `'the'   => 1 => [0.236, -0.141, 0.000, 0.045]`
- `'mouse' => 2 => [0.006, 0.652, 0.270, -0.556]`
- `'ran'   => 3 => [0.305, 0.569, -0.028, 0.496]`
- `'down'  => 6 => [0.466, -0.326, 0.884, 0.007]`

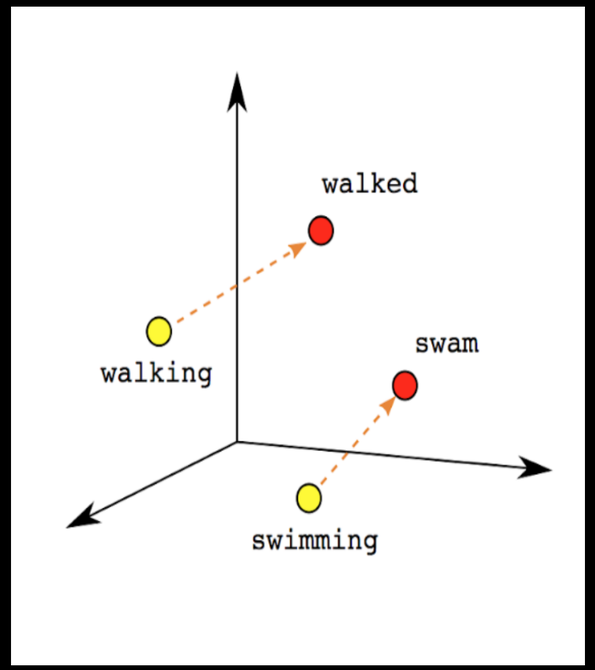
단어 임베딩의 의미

- 단어 사이의 연관성을 벡터 공간에 표현한 예시(해당 단어가 가질 수 있는 다양한 의미를 가리킴)

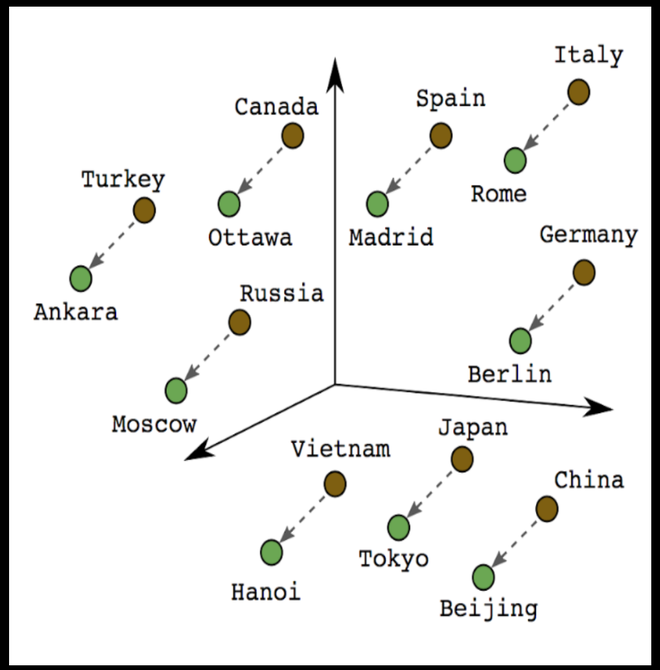
남성-여성



현재진행형-과거형



국가-수도



출처: https://developers.google.com/machine-learning/guides/text-classification/step-3?hl=ko

단어 임베딩 활용

- 단어 임베딩은 입력층 다음에 위치, Embedding층 다음 TransformerEncoder층이 텍스트의 내용을 파악하고 최종적으로 모델의 예측값을 Dense층을 통해 결정

```
vocab_size = 20000 # 총 어휘 수
embed_dim = 256    # 단어별로 256 개의 특성 파악.

# 입력층: 단어 벡터화된 배치 데이터셋
inputs = keras.Input(shape=(None,), dtype="int64")

# 단어 임베딩 실행
x = layers.Embedding(vocab_size, embed_dim)(inputs)

# 트랜스포머 인코더: 텍스트 내용 파악
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)

# 출력층
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

# 모델 선언
model = keras.Model(inputs, outputs)
```