인공지능

딥러닝 중급_자연어처리

강성관 (silicon1@hanmail.net)

자연어 처리

자연어 (natural language) 처리

- 자연어 : 일상생활에서 사용하는 언어
- 자연어 처리 : 자연어의 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 과정
- 자연어 처리 분야 : 음성 인식, 내용 요약, 번역, 사용자의 감정 분석, 텍스트 분류 작업(스팸 메일 분류, 뉴스 기사 카테고리 분류), 질의 응답 시스템, 챗봇 등

- 코퍼스 (Corpus) : 자연어 처리에 사용되는 많은 데이터의 모음 (텍스트 모음)

- 자연어 처리는 기계에게 인간의 언어를 이해시키는 것 → 인공지능에 있어서 가장 중요한 연구 분야

- 문자열 : 가장 흔한 시퀀스 형태의 데이터
- 시퀀스 처리용 딥러닝 모델 : 문서 분류, 감성 분석, 저자 식별, (제한된 범위의) 질문 응답 (QA) 등에 활용 → 일종의 패턴 인식

자연어 (natural language) 처리

- 텍스트 전처리 (Text Preprocessing) : 목적에 맞게 텍스트를 사전에 분류하는 과정
- 전처리 방법

(1) 토큰화 (Tokenization) : 코퍼스를 토큰으로 나누는 작업 (문자, 단어, n-gram)

(2) 정규화 (Normalization) : 코퍼스를 용도에 맞게 토큰을 분류하는 작업 (유사단어 통합, 대소문자 통합, 불필요한 단어 제거 (작은 빈도 단어, 짧은 길이의 단어))

(3) 어간 추출 (Stemming) : 정규화의 한 방법으로 단어의 핵심 부분만 추출하는 것

표제어 추출 (Lemmatization) : 정규화의 한 방법으로 유사한 단어들에서 대표 단어를 추출

formalize → formal allowance → allow Electricical → electirc

어간추출

am, are, is → be

표제어 추출

(4) 불용어 (Stopword) 제거 : 의미없는 데이터를 제거하는 작업

자연어 (natural language) 처리

(5) 정규 표현식 (Regular Expression)

(6) 단어 분리 (Subword Segamentation)

(7) 정수 인코딩 (Integer Encoding) : 각 단어를 정수에 맵핑하는 작업

토큰화 (Tokenization)

- 텍스트 벡터화 (Vectorizing Text) : 텍스트를 수치형 텐서로 변환하는 과정
  (1) 텍스트를 단어로 나누고 각 단어를 하나의 벡터로 변환
  (2) 텍스트를 문자로 나누고 각 문자를 하나의 벡터로 변환
  (3) 텍스트에서 단어나 문자의 n-gram을 추출하여 n-gram을 하나의 벡터로 변환
- n-gram : 연속된 단어나 문자의 그룹으로 텍스트에서 단어나 문자를 하나씩 이동하면서 추출
- token : 텍스트를 나누는 단위 (단어, 문자, n-gram)
- 토큰화 (tokenization) : 텍스트를 토큰으로 나누는 작업
- 토큰과 벡터를 연결하는 방법
  (1) one-hot encoding : 0과 1로 이루어진 숫자로 변환
  (2) token embedding : 실수로 변환, 단어에만 사용되므로 word embedding이라고도 함

n-gram

- 최대 n 개의 토큰 단위로 문자 시퀀스를 구분

AI is too difficult

- 2-gram으로 분해

{“AI”, “AI is”, “is”, “is too”, “too”, “too difficult”, “difficult”}

- 3-gram으로 분해

{“AI”, “AI is”, “is”, “is too”, “AI is too”, “too”, “too difficult”, “difficult”, “is too difficult”}

단어의 원-핫 인코딩

- 사용될 문장 설정 / 변수 초기화

```
1  import numpy as np
2
3  samples = ['AI is too difficult', 'AI is very complicated']
4  token_index = {}
5  max_length = 8;    #최대 토큰 수 (최대 단어 수)
```

- 토큰을 분리하고 인덱스를 할당 (인덱스를 1부터 시작)

```
6  for sample in samples:
7      for word in sample.split():
8          # 만약 token_index에 word가 없다면 인덱스를 할당 (0은 사용하지 않음)
9          if word not in token_index:
10             token_index[word] = len(token_index) + 1
```

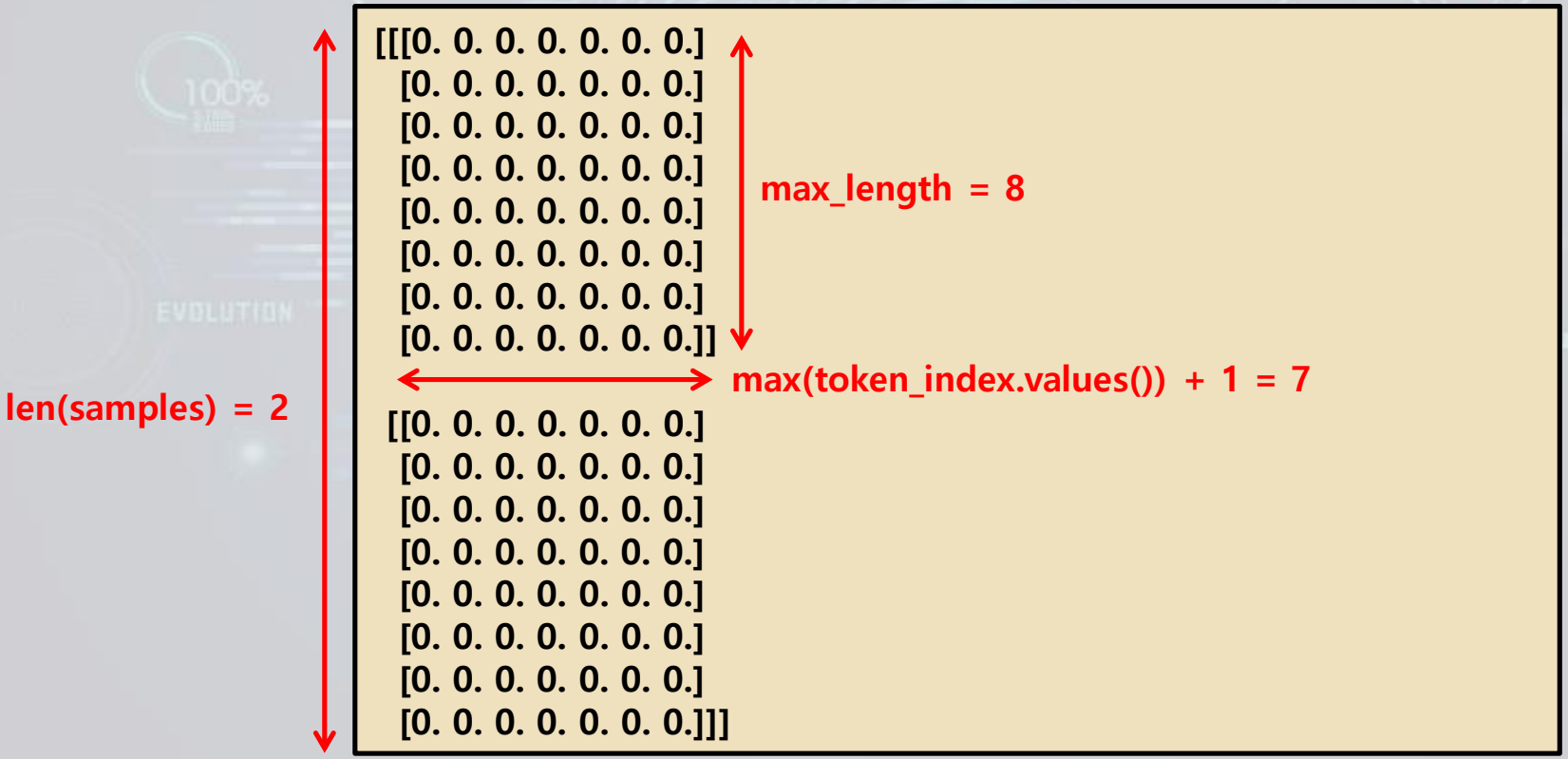
단어의 원-핫 인코딩

for sample in samples: print(sample)	AI is too difficult AI is very complicated
for word in sample.split(): print(word)	AI is too difficult AI is very complicated
token_index[word] = len(token_index) + 1 print(token_index[word])	1 2 3 4 5 6

단어의 원-핫 인코딩

- resutls 배열을 생성하고 0으로 채움

```
12  results = np.zeros(shape=(len(samples), max_length, max(token_index.values()) + 1))
```



단어의 원-핫 인코딩

- 원-핫 인코딩을 수행하고 결과를 출력

```
14  for i, sample in enumerate(samples):
15      for j, word in list(enumerate(sample.split()))[:max_length]:
16          index = token_index.get(word)
17          results[i, j, index] = 1.
18
19  print(results)
```

단어의 원-핫 인코딩

- enumerate() : 리스트, 튜플, 문자열의 자료형을 입력받아 인덱스 값을 포함하는 자료형으로 반환

```
for i, sample in enumerate(samples):
```

i	sample
0	AI is too difficult
1	AI is very complicated

단어의 원-핫 인코딩

- token_index.get(word) : token_index 에서 word의 인덱스를 가져온다

```
1
2
3
4
1
2
5
6
```

단어의 원-핫 인코딩

- results[i, j, index] = 1. : word에 일치하는 위치를 1. 으로 설정

```
[[[0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0.]
  [0. 0. 0. 0. 1. 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. 1. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0.]
  [0. 0. 0. 0. 0. 0. 1.]
  [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.]]]
```

문자의 원-핫 인코딩

- 함수 로딩 및 사용할 문장 및 변수 (문자 수) 설정

```
1  import numpy as np
2  import string
3
4  samples = ['AI is too difficult', 'AI is very complicated']
5  max_length = 50
```

문자의 원-핫 인코딩

- 출력 가능한 모든 ASCII 문자를 저장

```
7  characters = string.printable
```

```
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-
./:;<=>?@[₩]^_`{|}~
```

문자의 원-핫 인코딩

- characters를 1부터 개수+1까지 인덱스를 부여하여 dict 자료형으로 변환

```
8  token_index = dict(zip(characters, range(1, len(characters) + 1)))
```

```
{'0': 1, '1': 2, '2': 3, '3': 4, '4': 5, '5': 6, '6': 7, '7': 8, '8': 9, '9': 10, 'a': 11, 'b': 12, 'c': 13, 'd': 14, 'e': 15, 'f': 16, 'g': 17, 'h': 18, 'i': 19, 'j': 20, 'k': 21, 'l': 22, 'm': 23, 'n': 24, 'o': 25, 'p': 26, 'q': 27, 'r': 28, 's': 29, 't': 30, 'u': 31, 'v': 32, 'w': 33, 'x': 34, 'y': 35, 'z': 36, 'A': 37, 'B': 38, 'C': 39, 'D': 40, 'E': 41, 'F': 42, 'G': 43, 'H': 44, 'I': 45, 'J': 46, 'K': 47, 'L': 48, 'M': 49, 'N': 50, 'O': 51, 'P': 52, 'Q': 53, 'R': 54, 'S': 55, 'T': 56, 'U': 57, 'V': 58, 'W': 59, 'X': 60, 'Y': 61, 'Z': 62, '!': 63, '"': 64, '#': 65, '$': 66, '%': 67, '&': 68, "'": 69, '(': 70, ')': 71, '*': 72, '+': 73, ',': 74, '-': 75, '.': 76, '/': 77, ':': 78, ';': 79, '<': 80, '=': 81, '>': 82, '?': 83, '@': 84, '[': 85, '₩₩': 86, ']': 87, '^': 88, '_': 89, '`': 90, '{': 91, '|': 92, '}': 93, '~': 94, ' ': 95, '₩t': 96, '₩n': 97, '₩r': 98, '₩x0b': 99, '₩x0c': 100}
```

문자의 원-핫 인코딩

- 변수 초기화 (0으로 채움), 원-핫 인코딩 수행
- sample에 포함된 문자들의 ASCII 코드에 해당하는 위치를 1.으로 설정

```
 9  results = np.zeros((len(samples), max_length, max(token_index.values())+1))
10
11  for i, sample in enumerate(samples):
12      for j, character in enumerate(sample):
13          index = token_index.get(character)
14          results[i, j, index] = 1.
15
16  print(results[0, 1])
17  print(results[1, 1])
18  print(results[1, 3])
19  print(results[1, 16])
```

문자의 원-핫 인코딩

- 수행 결과

```
[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. 1. 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. 1. 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. 1. 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. 1. 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.]
```

results[0, 1]
AI is too difficult

results[1, 1]
AI is very complicated

results[1, 3]
AI is too difficult

results[1, 16]
AI is very complicated

keras를 이용한 단어의 원-핫 인코딩

- 함수 로딩, 사용할 문장 설정

```
1  from keras.preprocessing.text import Tokenizer
2
3  samples = ['AI is too difficult', 'AI is very complicated']
```

keras를 이용한 단어의 원-핫 인코딩

- 단어 분리, 빈수도 분석, 단어에 인덱스 할당

```
5  #가장 빈도가 높은 n-1개의 단어만 선택
6  tokenizer = Tokenizer(num_words=5)
7  #단어 인덱스 분석 (단어, 빈도수, 문서수, 인덱스)
8  tokenizer.fit_on_texts(samples)
9  # 단어의 빈도수를 저장하고 있는 dict 자료형
10 print(tokenizer.word_docs)
11 # 단어와 고유 인덱스를 저장하고 있는 dict 자료형 (빈도수 순서로 인덱싱)
12 print(tokenizer.word_index)
```

```
defaultdict(<class 'int'>, {'difficult': 1, 'is': 2, 'ai': 2, 'too': 1, 'very': 1, 'complicated': 1})
{'ai': 1, 'is': 2, 'too': 3, 'difficult': 4, 'very': 5, 'complicated': 6}
```

빈도수가 높은 순으로 인덱싱

keras를 이용한 단어의 원-핫 인코딩

- 문자열을 정수 인덱스의 리스트로 변환 (설정 빈도수에 해당하는 문자만 변환)

```
14 sequences = tokenizer.texts_to_sequences(samples)
15 print(sequences)
```

```
[[1, 2, 3, 4], [1, 2]]
```

very와 complicated는 빈도수가 5-6번째이므로 제외

keras를 이용한 단어의 원-핫 인코딩

- 이진 벡터 행렬로 변환 (binary : 단어 존재 여부, count : 단어 수, tfidf : 단어 빈도, freq : 단어 비율)

```
16 on_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')
17 print(on_hot_results)
```

```
[[0. 1. 1. 1. 1.]
 [0. 1. 1. 0. 0.]]
```

첫 번째 인덱스는 사용하지 않음

AI	is	too	difficult	AI	is	very	complicated
1	1	1	1	1	1	0	0

해싱 기법을 이용한 단어의 원-핫 인코딩

- 사전에 있는 고유한 토큰의 수가 너무 커서 모두 다루기 힘들 때 사용
- 각 단어에 명시적으로 인덱스를 할당하고 인덱스를 딕셔너리가 아닌 단어를 해싱하여 고정된 크기의 벡터로 변환 → 메모리 절약, 온라인 방식으로 데이터 인코딩 가능
- 문제점 : 해시 충돌 (Hash collision) – 2개의 단어가 동일한 해시 값을 생성하는 문제 → 해시 공간 차원을 해싱될 고유 토큰의 수보다 크게 하면 감소

해싱 기법을 이용한 단어의 원-핫 인코딩

- 함수 로딩, 사용할 문장 설정, 변수 (차원, 최대 단어 수) 설정

```
1  import numpy as np
2
3  samples = ['AI is too difficult', 'AI is very complicated']
4
5  dimensionality = 10
6  max_length = 5
```

해싱 기법을 이용한 단어의 원-핫 인코딩

- 변수 초기화, 원-핫 인코딩 수행

```
 8  results = np.zeros((len(samples), max_length, dimensionality))
 9  for i, sample in enumerate(samples):
10      for j, word in list(enumerate(sample.split()))[:max_length]:
11          index = abs(hash(word)) % dimensionality
12          print(index)
13          results[i, j, index] = 1.
14
15  print(results)
```

해싱 기법을 이용한 단어의 원-핫 인코딩

```
3
4
6
1
3
4
0
3
[[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

 [[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]]
```

해싱값을 차원값으로 나눈 나머지 값을 인덱스로 설정

각 단어의 인덱스의 위치를 1.로 설정

max_length (최대 단어 수)

dimensionality (해시 크기)

word embedding (단어 임베딩)

- 단어와 벡터를 연관 짓는 방법 중의 하나로 실수형인 밀집 word vector을 사용하는 것
- 원 핫 인코딩으로 만든 벡터는 희소 (sparse)하고 (대부분 0으로 채워짐) 고차원 (단어의 수와 동일)
- 원 핫 인코딩이 수동 인코딩이라면 단어 임베딩은 저차원의 실수형 벡터 (밀집벡터)로 데이터로부터 학습

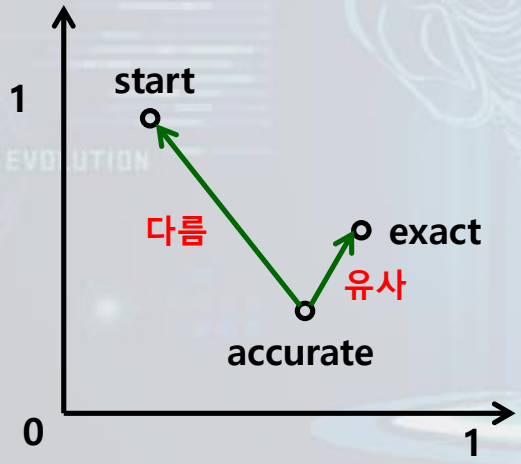
- 단어 임베딩 만드는 방법

  (1) 관심 대상인 문제와 함께 단어 임베딩을 학습 → 랜덤 단어 벡터로 시작해서 가중치 학습

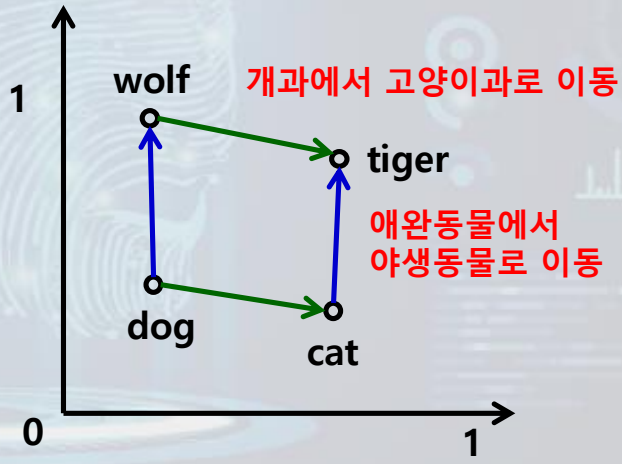
  (2) 사전 훈련된 단어 임베딩 (Pretrained word embedding) : 미리 계산된 단어 임베딩을 사용

Embedding 층을 사용하여 단어 임베딩 학습하기

- 단어와 밀집벡터를 연관 짓는 가장 간단한 방법은 랜덤하게 벡터를 선택하게 하는 것 → 구조적이지 못하고 의미론적이지 못함
- 단어 사이에 있는 의미 관계를 반영해야 함 → 언어를 기하학적 공간에 매핑 (단어 간의 의미적 거리, 방향적 특성 등을 분석)



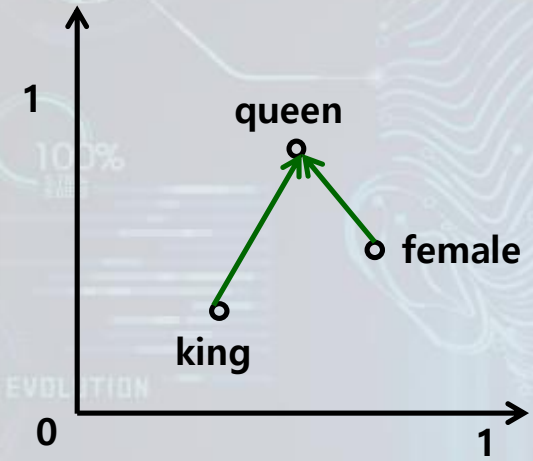
의미적 거리

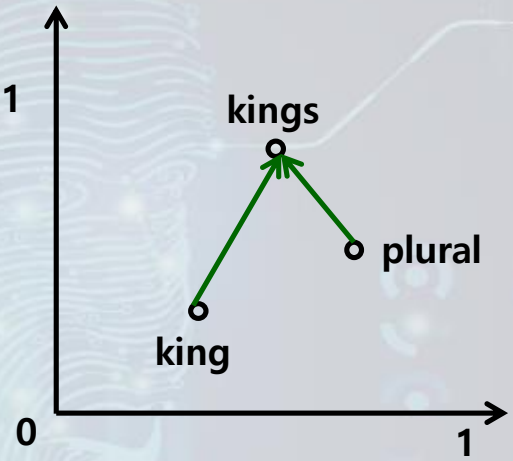


방향적 특성

Embedding 층을 사용하여 단어 임베딩 학습하기

- 실제 단어 임베딩 공간에서 의미 있는 기하학적 변환의 예

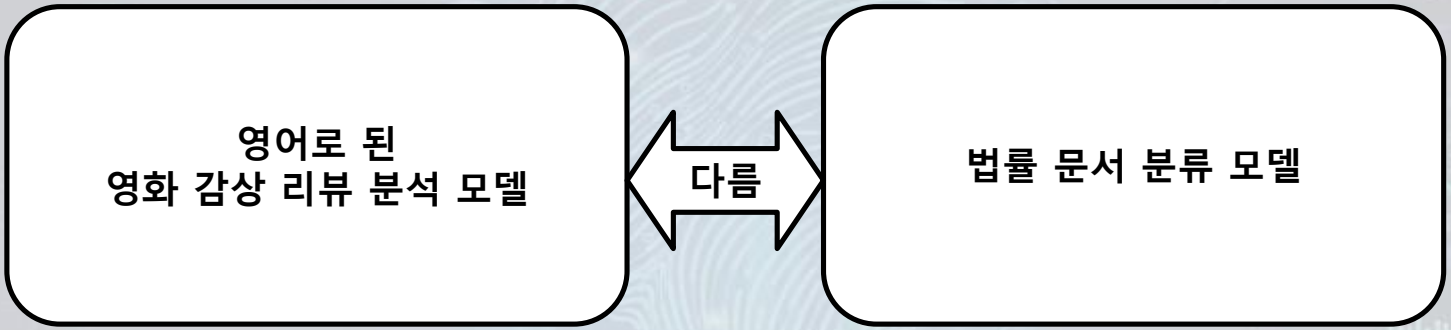




- 향후 사람의 언어를 완벽하게 매핑해서 모든 자연어 처리가 가능한 이상적인 단어 임베딩 공간도 만들어질 것으로 예측

Embedding 층을 사용하여 단어 임베딩 학습하기

- 새로운 작업에는 새로운 임베딩을 학습



Embedding 층을 사용하여 단어 임베딩 학습하기

- Embedding Layer : 정수 인덱스를 밀집 벡터로 맵핑하는 기능

- Embedding Layer
  - input_dim : 단어 사전의 크기 (단어의 종류)
  - output_dim : 단어 인코딩 한 후에 나오는 벡터의 크기 (의미론적 기하공간의 크기)
  - input_length : 단어의 수 (문장의 길이)

```
Embedding(input_dim, output_dim, input_length)
```

- Embedding Layer의 출력 크기 : 샘플 수 * output_dim * input_lenth
- Embedding Layer 다음에 Flatten Layer 온다면 반드시 input_lenth를 지정해야 함 → 입력 크기가 알아야 이를 1차원으로 만들어서 Dense 레이어에 전달할 수 있기 때문
- Embedding 층의 가중치는 랜덤하게 초기화 → 학습을 하면서 역전파를 통해 조정

Embedding 층을 사용하여 단어 임베딩 학습하기

- imdb 데이터 셋 : 스탠포드 대학의 앤드류 마스가 수집한 데이터 셋
- 데이터 셋 주소 : http://ai.stanford.edu/~amaas/data/sentiment/ 또는 https://stanford.io/2w2NUzz
- 인터넷 영화 데이터베이스 (IMDB) : 영화와 관련된 정보와 출연진 정보, 개봉 정보, 영화 후기, 평점에 이르기까지 매우 폭넓은 데이터가 저장된 데이터 셋
- 인터넷 영화 데이터베이스로부터 가져온 양극단의 영화 리뷰 5만개 포함
- 훈련데이터 2만 5,000개, 테스트데이터 2만 5,000개로 나뉘어 있고 50%는 긍정, 50%는 부정 리뷰로 구성

Embedding 층을 사용하여 단어 임베딩 학습하기

- 함수 로딩, 사용할 문장 설정, 변수 (차원, 최대 단어 수) 설정

```
1  from keras.datasets import imdb
2  from keras import preprocessing
3  from keras.models import Sequential
4  from keras.layers import Flatten, Dense, Embedding
5
6  # 특성으로 사용할 단어 수
7  max_features = 10000
8  # 사용할 텍스트의 최대 길이 (짧으면 0으로 채움, 길면 자름)
9  maxlen = 20
```

Embedding 층을 사용하여 단어 임베딩 학습하기

- 함수 로딩, 사용할 문장 설정, 데이터 변환

```
11  (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
12  # 훈련 및 테스트 데이터 리스트를 2D 텐서로 변환
13  x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
14  x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

Embedding 층을 사용하여 단어 임베딩 학습하기

- 신경망 설계, 컴파일, 실행

```
17  model = Sequential()
18  model.add(Embedding(max_features, 8, input_length=maxlen))
19  model.add(Flatten())
20  model.add(Dense(1, activation='sigmoid'))
21  model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
22
23  history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
24  print(history)
```

Embedding 층을 사용하여 단어 임베딩 학습하기

```
  32/20000 [..............................] - ETA: 1s - loss: 0.2025 - acc: 0.9062
 1280/20000 [>.............................] - ETA: 0s - loss: 0.2679 - acc: 0.9031
 2656/20000 [==>...........................] - ETA: 0s - loss: 0.2807 - acc: 0.8957
 3904/20000 [====>.........................] - ETA: 0s - loss: 0.2772 - acc: 0.8945
 5184/20000 [======>.......................] - ETA: 0s - loss: 0.2772 - acc: 0.8943
 6464/20000 [========>.....................] - ETA: 0s - loss: 0.2743 - acc: 0.8948
 7840/20000 [=========>....................] - ETA: 0s - loss: 0.2793 - acc: 0.8911
 9184/20000 [==========>...................] - ETA: 0s - loss: 0.2828 - acc: 0.8873
10432/20000 [===========>..................] - ETA: 0s - loss: 0.2847 - acc: 0.8869
11776/20000 [============>.................] - ETA: 0s - loss: 0.2853 - acc: 0.8870
13120/20000 [==============>...............] - ETA: 0s - loss: 0.2844 - acc: 0.8870
14400/20000 [===============>..............] - ETA: 0s - loss: 0.2846 - acc: 0.8864
15808/20000 [================>.............] - ETA: 0s - loss: 0.2840 - acc: 0.8866
16672/20000 [=================>............] - ETA: 0s - loss: 0.2845 - acc: 0.8862
17696/20000 [=================>............] - ETA: 0s - loss: 0.2849 - acc: 0.8859
18752/20000 [==================>...........] - ETA: 0s - loss: 0.2840 - acc: 0.8859
19744/20000 [===================>..........] - ETA: 0s - loss: 0.2839 - acc: 0.8859
20000/20000 [==============================] - 1s 44us/step - loss: 0.2839 - acc: 0.8860
- val_loss: 0.5302 - val_acc: 0.7464
```

사전 훈련된 단어 임베딩 사용하기

- 훈련 데이터가 부족한 경우에 미리 계산된 임베딩 공간에서 임베딩 벡터를 로드할 수 있음 (이미지 분류 문제의 사전 훈련된 컨브넷을 사용하는 이유와 동일)
- 일반적인 특성의 경우에만 적용
- Word2vec 알고리즘 : 2013년 구글의 토마스 미코로프가 개발, 단어를 벡터로 바꿔주는 기능을 하며 구체적인 의미가 있는 속성을 구분 (https://code.google.com/archive/p/word2vec)
  - 단어를 벡터화 할 때 단어의 문맥적 의미를 보존
  - 벡터로 바뀐 단어들은 거리 계산을 통해 유사도를 판단

사전 훈련된 단어 임베딩 사용하기

- **Glove 알고리즘** : 2014년 영문 위키피디아를 사용해 사전에 계산된 임베딩

  - 파일의 이름은 glove.6B.zip이고 압축 파일 크기는 823MB
  - 400,000만개의 단어(또는 단어가 아닌 토큰)에 대한 100차원의 임베딩 벡터를 포함
  - datasets 폴더 아래에 파일 압축을 해제
  - https://nlp.stanford.edu/projects/glove