

## 딥러닝을 이용한 자연어 처리

인공지능 비서가 갖춰야 할 필수 능력은 사람의 언어를 이해하는 것

-> 자연어 처리(Natural Language Processing, NLP)

자연어 - 우리가 평소에 말하는 음성이나 텍스트

자연어 처리 - 음성이나 텍스트를 컴퓨터가 인식하고 처리하는 것

딥러닝이 등장하면서 자연어 처리 연구가 활발해지기 시작했습니다. 이는 대용량 데이터를 학습할 수 있는 딥러닝의 특성 때문이다.

컴퓨터 알고리즘은 **수치로 된 데이터만 이해할 뿐 텍스트를 이해할 수 없기 때문에** 텍스트를 정제하는 **전처리 과정**이 꼭 필요함!

### 1. 텍스트의 토큰화

(실습코드는 github 참고)

토큰(token) - 텍스트를 단어별, 문장별, 형태소별로 나누었을 때, 작게 나누어진 하나의 단위

토큰화(tokenization) - 텍스트를 잘게 나누는 과정

케라스가 제공하는 **text 모듈의 text\_to\_word\_sequence() 함수**를 사용하면 문장을 단어 단위로 쉽게 나눌 수 있음

**Bag-of-Words 방법** - 같은 단어끼리 따로따로 가방에 담은 뒤 **각 가방에 몇 개의 단어가 들어있는지를 세는** 기법

케라스의 **Tokenizer() 함수**를 사용하면 단어의 빈도 수를 쉽게 계산할 수 있음

(**word\_counts**는 **단어의 빈도 수**를 계산해주는 함수)

(순서를 기억하는 OrderedDict 클래스에 담겨 있는 형태로 출력됨)

**document\_count() 함수** - 몇 개의 문장이 들어있는지 셀 수 있음

**word\_docs() 함수** - 각 단어들이 몇 개의 문장에 나오는가를 세어서 출력할 수 있음(출력되는 순서는 랜덤)

**word\_index() 함수** - 각 단어에 매겨진 인덱스 값을 출력

(각 단어의 빈도수가 **높은 순서**대로 인덱스가 부여됨)

### 2. 단어의 원-핫 인코딩

단어의 출현 빈도만 가지고는 해당 단어가 문장의 어디에서 왔는지, 각 단어의 순서는 어떠한지 등에 관한 정보를 얻을 수 없음

단어가 문장의 다른 요소와 어떤 관계를 가지고 있는지 알아보는 방법이 필요함

-> 원-핫 인코딩 사용!

각 단어를 모두 0으로 바꾸어 주고 원하는 단어만 1로 바꾸어 주는 것이 원-핫 인코딩  
(파이썬 배열의 인덱스는 0부터 시작하므로, 맨 앞에 0이 추가되는 것에 주의)

(0인덱스)	오랫동안	꿈꾸는	이는	그	꿈을	달아간다
	0	0	0	0	0	0

오랫동안 =	[ 0 1 0 0 0 0 0 ]
꿈꾸는 =	[ 0 0 1 0 0 0 0 ]
이는 =	[ 0 0 0 1 0 0 0 ]
그 =	[ 0 0 0 0 1 0 0 ]
꿈을 =	[ 0 0 0 0 0 1 0 ]
달아간다 =	[ 0 0 0 0 0 0 1 ]

`texts_to_sequences()` 함수 - 토큰의 인덱스로만 채워진 새로운 배열 만들기

`to_categorical()` 함수 - 0과 1로만 이루어진 배열로 바꾸어주는 원-핫 인코딩 진행

### 3. 단어 임베딩

원-핫 인코딩을 그대로 사용하면 벡터의 길이가 너무 길어진다는 단점이 있음 -> 공간적 낭비를 해결하기 위해 등장한 것이 바로 단어 임베딩(word embedding)

단어 임베딩으로 얻은 결과가 밀집된 정보를 가지고 있고 공간의 낭비가 적다.

-> 이런 결과가 가능한 이유는 각 단어 간의 유사도를 계산했기 때문

유사도 계산에는 오차 역전파 사용

적절한 크기로 배열을 바꾸어 주기 위해 최적의 유사도를 계산하는 학습 과정을 거친다.

(`Embedding()` 함수 사용)

Embedding() 함수에는 '입력'과 '출력'의 크기를 나타내는 매겨변수 2개가 필요하며 `input_length`를 따로 지정해 단어를 매번 얼마나 입력할지를 추가로 지정할 수 있다.

#### 4. 텍스트를 읽고 긍정, 부정 예측하기

(실습코드는 github 참고)

딥러닝 모델에 입력을 하려면 학습 데이터의 길이가 동일해야 함

**패딩(padding)** - 길이를 똑같이 맞춰 주는 작업

`pad_sequence()` 함수 - 원하는 길이보다 짧은 부분은 숫자 0을 넣어서 채워주고, 긴 데이터는 잘라서 같은 길이로 맞춤

최적화 함수로 `adam()`을 사용하고 오차 함수로는 `binary_crossentropy()`를 사용함. 30번 반복하고나서 정확도를 계산해 출력하게 함.