

Day 1

딥러닝을 이용한 자연어 처리 입문

한국과학기술정보연구원

이 홍 석
(hsyi@kisti.re.kr)

❖ Day 1

- ✓ 딥러닝을 이용한 신경망 이해 및 실습
- ✓ 자연어 처리를 위한 텍스트 전처리
- ✓ 자연어 처리를 위한 단어 임베딩
- ✓ Word2Vec 이해 및 실습

- ❖ 자연어 처리에 대한 지식이 없다면 제대로 사용할 수 없음.
- ❖ 점점 시장이 커질 수 밖에 없는 분야. 커리어의 우위.
- ❖ 회사의 도메인과 상관없이 거의 모든 분야에서 활용이 가능.
- ❖ 성능 좋은 NLP 기반 언어 모델들이 오픈 소스로 지속적으로 외부에 공개되고 있는 중
 - ✓ 딥시크(DeepSeek)의 소스코드 공개 등으로 NLP에 대한 이해가 절실하다.



❖ 강의 수준

- ✓ 자연어 처리에 관심이 많은 초심자를 대상으로 한다.
- ✓ 파이썬의 기본문법을 이해하고 있고, 딥러닝과 머신러닝에 대한 기초기식이 있는 분

❖ 강의 목표

- ✓ 텐서플로우의 예제를 통해 직접 코드를 한 줄 한 줄 입력하면 배우는 것을 목표로 한다.
- ✓ 자연어 처리 및 관련 알고리즘에 대한 전반적인 내용은 깊게 다루지는 않는다.

❖ 강의 특징

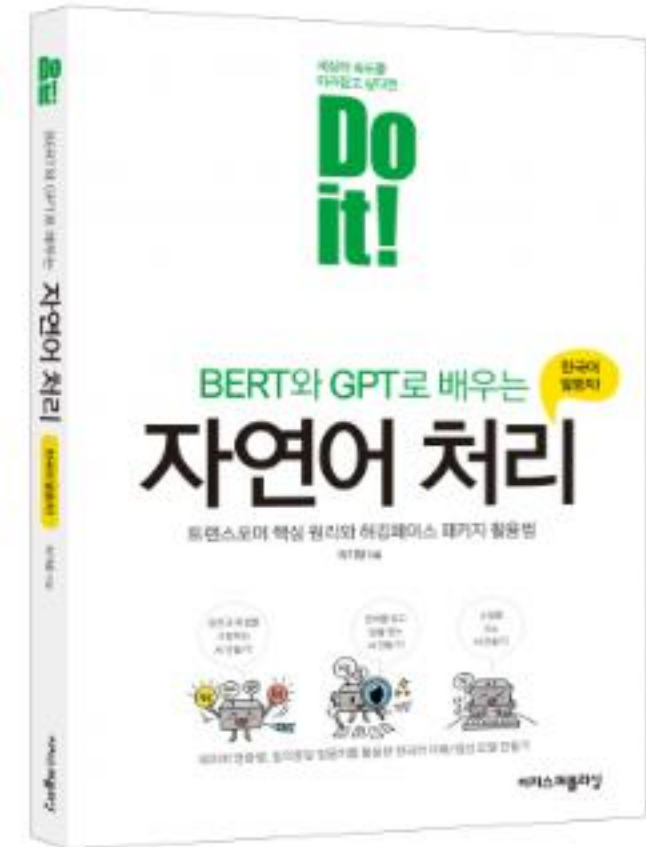
- ✓ 기존 영어 텍스트 데이터 (예, IMDB)외 한글 데이터 처리를 위한 접근법까지 다룬다.
- ✓ 자연어 처리 전반에 대한 개념과 사용할 도구를 배운다.

자연어 처리를 배우는 순서 (차근차근 하세요)

- ❖ Text preprocessing for NLP & Language Model
- ❖ Basic Model & Vectorization
- ❖ Word Embedding (Word2Vec)
- ❖ Text Classification (using RNN & CNN)
- ❖ Sequence to Sequence
- ❖ Attention Mechanism



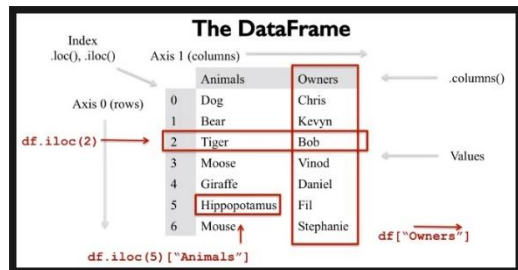
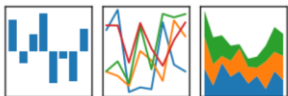
❖ 주요 참고 교재



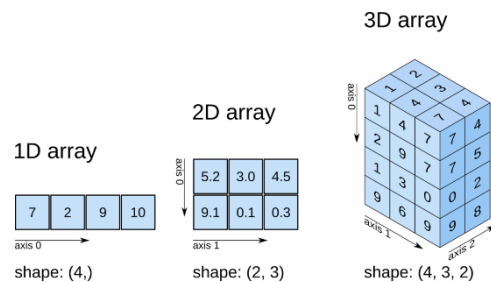
- ❖ Eliza는 튜링 테스트를 해결하기 위해 1960년대 중반에 개발
- ❖ Tay는 Microsoft가 2016년에 출시한 챗봇
- ❖ BERT는 컨텍스트 임베딩 및 향상된 지식 표현을 제공
 - ✓ 딥러닝 NLP 모델은 ELMo, BERT, Big BIRD, ERNIE, Grover, RoBERTa를 포함하여 Muppet 캐릭터의 이름을 따서 명명
- ❖ ChatGPT (Generative Pre-Trained Transformer) ~ OpenAI가 2022년 개발
 - ✓ GPT-4o LLM 최신 모델
- ❖ LaMDA(Language Model for Dialogue Applications)
 - ✓ Google에서 개발한 대화형 챗봇
- ❖ DeepSeek ~ 2025년
 - ✓ Hangzhou DeepSeek Artificial Intelligence Basic Technology Research Co., Ltd에서 개발

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

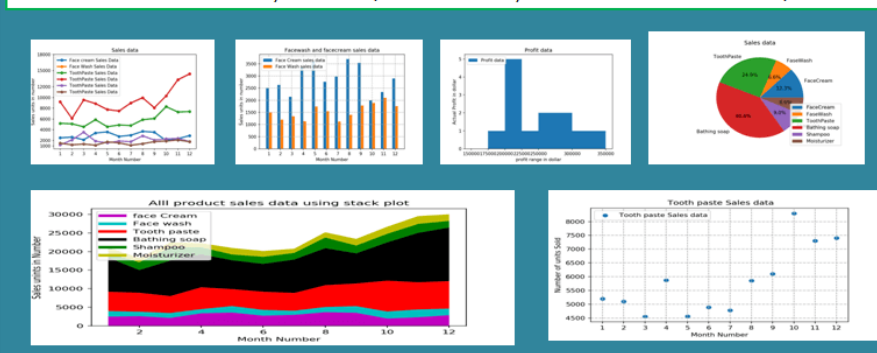


NumPy



Python Matplotlib

Practice Data Visualization In, Practice Questions Online, Solution Provided for Each Question



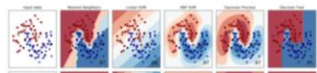
scikit-learn

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...

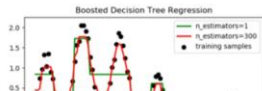


Regression

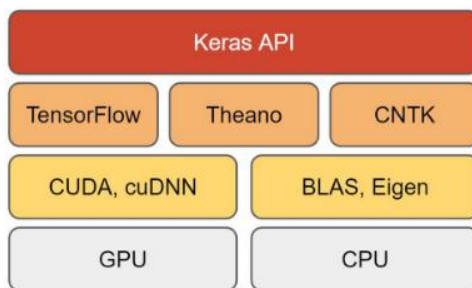
Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

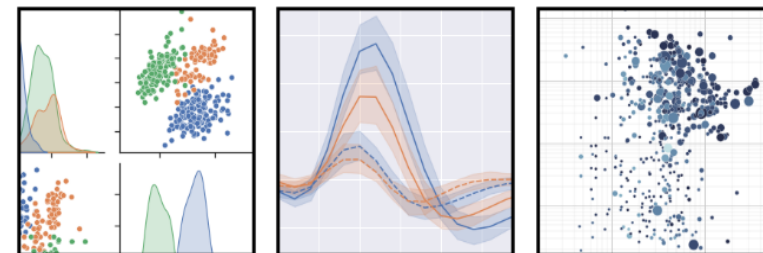
Algorithms: SVR, nearest neighbors, random forest, and more...



TensorFlow PYTORCH



seaborn: statistical data visualization

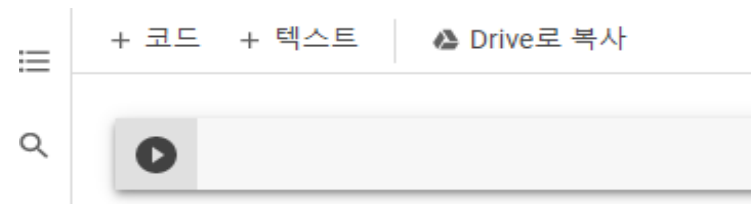


❖ 구글 검색창에 colab 사이트를 찾는다.

- ✓ 주소: <https://colab.research.google.com/>
- ✓ 코랩은 아나콘다 처럼 여러 패키지를 설치가 필요하지 않아서 편리하다.
- ✓ GPU를 사용 가능하며, 교육용으로 10분 정도 계산 시간이면 충분하다. CPU에서 계산을 해도 된다.



+코드/+텍스트 '셀'



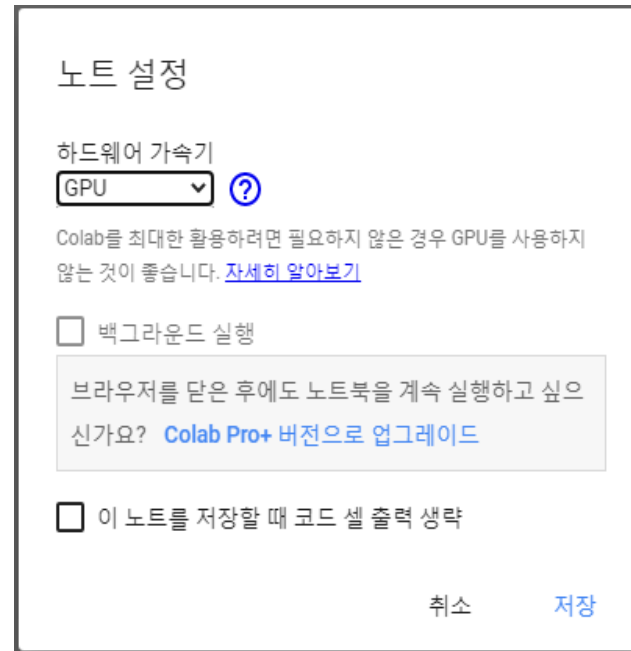
❖ Colab에서 실습할 때의 장점은 GPU를 무료로 사용할 수 있다는 점이다.

✓ 텐서플로우는 기본적으로 64 비트 플랫폼만을 지원한다.

- CPU ~ 64 비트, GPU ~ 32 비트, TPU ~ 8 비트
- 참고로 Intel 가속기 512 비트, 256 비트

✓ Colab에서 GPU 사용은 새 노트에 진입했을 때 상단에서 런타임>런타임 유형 변경을 클릭

- 노트 설정의 하드웨어가속기> GPU를 선택 후 저장



런타임 유형 변경

런타임 유형

Python 3

하드웨어 가속기 ?

- ☒ CPU ☐ T4 GPU ☐ A100 GPU ☐ L4 GPU
- ☐ v2-8 TPU ☐ v5e-1 TPU

프리미엄 GPU를 이용하시겠어요? [추가 컴퓨팅 단위 구매](#)

- ❖ "Pandas"는 "Panel"과 "Data"라는 단어의 축약형
 - ✓ "Python Data Analysis"라는 용어의 축약형
 - ✓ Panel 데이터는 다차원 데이터의 한 형태
 - ✓ Python 기반 도구로 간소화되는 모든 형태의 분석
- ❖ Python Pandas 라이브러리는 2가지 기본 데이터 구조를 제공
 - ✓ Series : 1차원 배열
 - 값 (values)
 - 인덱스(index)
 - ✓ DataFrame : 2차원 배열
 - `df = pd.DataFrame()`

- ❖ Pandas 데이터 프레임(DataFrame)은 2차원 데이터 구조
 - ✓ 레이블이 지정된 축(행 및 열)이 있는 2차원 크기 변경 가능한 행과 열에서 테이블 형식으로 정렬
 - ✓ 데이터, 행 및 열의 세 가지 주요 구성 요소로 구성
 - ✓ “pandas”라고 쓰는 대신 pd를 사용한다.
 - ✓ import pandas as pd

	<i>Name</i>	<i>Team</i>	<i>Number</i>	<i>Position</i>	<i>Age</i>	<i>Height</i>	<i>Weight</i>	<i>College</i>	<i>Salary</i>
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

```
[1] import pandas as pd
```

```
[2] sr = pd.Series([17000, 18000, 1000, 5000],  
                  index=["피자", "치킨", "콜라", "맥주"])
```

```
[3] print('시리즈 출력 :')  
    print('-'*15)  
    print(sr)
```

↪ 시리즈 출력 :

```
-----  
피자    17000  
치킨    18000  
콜라     1000  
맥주     5000  
dtype: int64
```

```
[4] print('시리즈의 값 : {}'.format(sr.values))  
    print('시리즈의 인덱스 : {}'.format(sr.index))
```

↪ 시리즈의 값 : [17000 18000 1000 5000]
시리즈의 인덱스 : Index(['피자', '치킨', '콜라', '맥주'], dtype='object')

❖ Pandas는 CSV, 텍스트, Excel, SQL, HTML, JSON 등 다양한 데이터 파일을 읽을 수 있다.

✓ df = pd.read_csv()

```
[18] import urllib.request
```

```
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt", filename="ratings_train.txt")
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt", filename="ratings_test.txt")
```

↻ ('ratings_test.txt', <http.client.HTTPMessage at 0x7b6e0bef50d0>)

[21] # csv 파일을 사용하는 경우가 많습니다. csv 파일을 데이터프레임으로 로드 할 때는 다음과 같이 합니다.

```
# df = pd.read_csv('csv 파일의 경로')
```

```
df = pd.read_csv('ratings_train.txt', delimiter = '\t') # 탭을 기준으로 자름
```

```
[72] df2.head()
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1

```
df = pd.DataFrame()
```

2) 데이터프레임

```
[5] values = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
      index = ['one', 'two', 'three']
      columns = ['A', 'B', 'C']

      df = pd.DataFrame(values, index=index, columns=columns)

      print('데이터프레임 출력 :')
      print('-'*18)
      print(df)
```

↔ 데이터프레임 출력 :

```
-----
      A  B  C
one   1  2  3
two   4  5  6
three 7  8  9
```

```
[6] print('데이터프레임의 인덱스 : {}'.format(df.index))
      print('데이터프레임의 열이름: {}'.format(df.columns))
      print('데이터프레임의 값 :')
      print('-'*18)
      print(df.values)
```

↔ 데이터프레임의 인덱스 : Index(['one', 'two', 'three'], dtype='object')
 데이터프레임의 열이름: Index(['A', 'B', 'C'], dtype='object')
 데이터프레임의 값 :

```
-----
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```


[73] df2.tail()

	id	document	label
149995	6222902	인간이 문제지.. 소는 원죄인가..	0
149996	8549745	평점이 너무 낮아서...	1
149997	9311800	이게 뭐요? 한국인은 거들먹거리고 필리핀 혼혈은 착하다?	0
149998	2376369	청춘 영화의 최고봉.방황과 우울했던 날들의 자화상	1
149999	9619869	한국 영화 최초로 수간하는 내용이 담긴 영화	0

[74] df2.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  -
 0    id      150000 non-null int64
 1  document 149995 non-null object
 2   label   150000 non-null int64
dtypes: int64(2), object(1)
memory usage: 3.4+ MB
```

[77] print('데이터프레임의 인덱스 : {}'.format(df2.index))

데이터프레임의 인덱스 : RangeIndex(start=0, stop=150000, step=1)

[78] print('데이터프레임의 열이름: {}'.format(df2.columns))

데이터프레임의 열이름: Index(['id', 'document', 'label'], dtype='object')

[82] print('데이터프레임의 값 :')
print('-'*80)
print(df2.values)

데이터프레임의 값 :

```
[[9976970 '아 더빙.. 진짜 짜증나네요 목소리' 0]
[3819312 '흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나' 1]
[10265843 '너무재밌었다그래서보는것을추천한다' 0]
...
[9311800 '이게 뭐요? 한국인은 거들먹거리고 필리핀 혼혈은 착하다?' 0]
[2376369 '청춘 영화의 최고봉.방황과 우울했던 날들의 자화상' 1]
[9619869 '한국 영화 최초로 수간하는 내용이 담긴 영화' 0]]
```

❖ Pandas에서 DataFrame을 CSV로 내보내는 방법

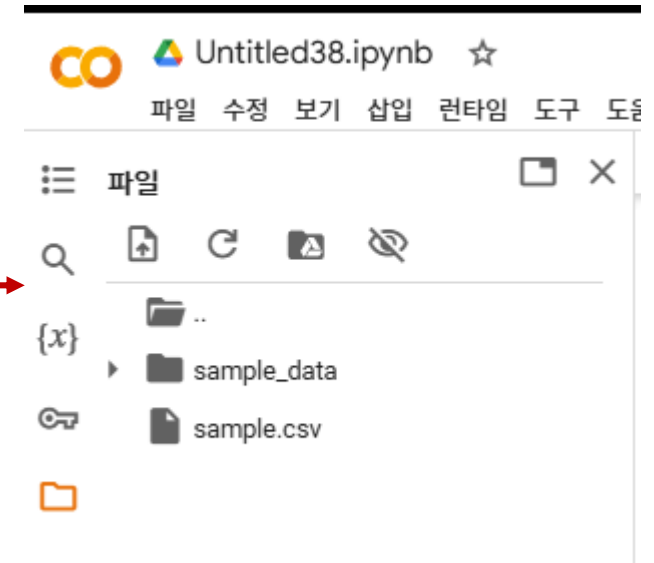
✓ 매서드 `.to_csv()`는 CSV 포맷으로 지정

- 생성된 파일은 왼쪽 폴더 모양을 클릭하면 `sample.csv`를 볼수 있다.

df

	A	B	C
one	1	2	3
two	4	5	6
three	7	8	9

```
[13] # DataFrame을 CSV로 내보내기  
df.to_csv('sample.csv', index=True)
```



✓ 파일 읽어오기

✓ 매서드 `.read_csv()`를 이용한다.

```
# DataFrame을 CSV로 내보내기  
df.to_csv('sample.csv', index=True)
```

```
[15] df2 = pd.read_csv('sample.csv')  
df2
```

df2

Unnamed: 0	A	B	C
0	one	1	2 3
1	two	4	5 6
2	three	7	8 9

❖ 넘파이 (Numpy)는 수치 데이터를 다루는 파이썬 패키지이다.

- ✓ import numpy as np
- ✓ Numpy는 다차원 행렬 자료구조인 ndarray를 사용으로 속도면에서도 순수 파이썬에 비해 압도적으로 빠르다는 장점이 있다.
- ✓ np.array()는 리스트, 튜플, 배열로 부터 ndarray 를 생성한다.

```
# 1차원 배열
vec = np.array([1, 2, 3, 4, 5])
print(vec)
print('vec의 타입 :', type(vec))
print('vec의 차원 :', vec.ndim) # 차원 출력
print('vec의 크기(shape) :', vec.shape) # 크기
```

```
[1 2 3 4 5]
vec의 타입 : <class 'numpy.ndarray'>
vec의 차원 : 1
vec의 크기(shape) : (5,)
```

```
[90] # 2차원 배열
mat = np.array([[10, 20, 30], [ 60, 70, 80]])
print(mat)
print('mat의 타입 :', type(mat))
print('mat의 차원 :', mat.ndim) # 차원 출력
print('mat의 크기(shape) :', mat.shape) # 크기 출력
```

```
[[10 20 30]
 [60 70 80]]
mat의 타입 : <class 'numpy.ndarray'>
mat의 차원 : 2
mat의 크기(shape) : (2, 3)
```

3) np.arange()

```
# 0부터 9까지  
range_vec = np.arange(10)  
print(range_vec)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

5) Numpy 슬라이싱

```
mat = np.array([[1, 2, 3], [4, 5, 6]])  
print(mat)
```

```
[[1 2 3]  
 [4 5 6]]
```

```
[33] # 첫번째 행 출력  
slicing_mat = mat[0, :]  
print(slicing_mat)
```

```
[1 2 3]
```

```
[34] # 두번째 열 출력  
slicing_mat = mat[:, 1]  
print(slicing_mat)
```

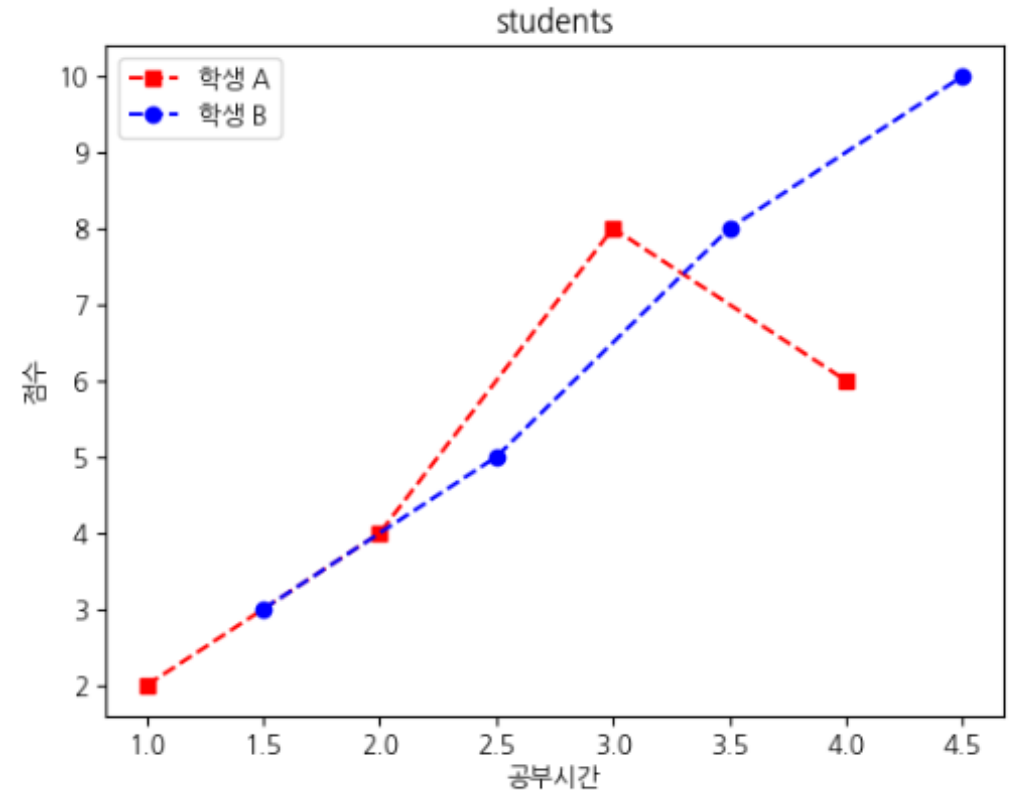
```
[2 5]
```

❖ 맷플롯립 (Matplotlib)은

- ✓ 데이터를 차트 (chart)나 플롯 (plot)으로 시각화하는 패키지이다.
- ✓ Matplotlib을 다 설치하였다면 Matplotlib의 주요 모듈인 pyplot를 관례상 plt라는 명칭으로 임포트한다.
- ✓ 한글 사용을 할 경우 koreanize_matplotlib을 설치한다.

```
[49] import matplotlib.pyplot as plt
      !pip install koreanize-matplotlib
      import koreanize_matplotlib
```

```
plt.title('students')
plt.plot([1,2,3,4],[2,4,8,6], 'rs--')
plt.plot([1.5,2.5,3.5,4.5],[3,5,8,10], 'bo--') # 라인 신규 추
plt.xlabel('공부시간')
plt.ylabel('점수')
plt.legend(['학생 A', '학생 B']) # 범례 삽입
plt.show()
```



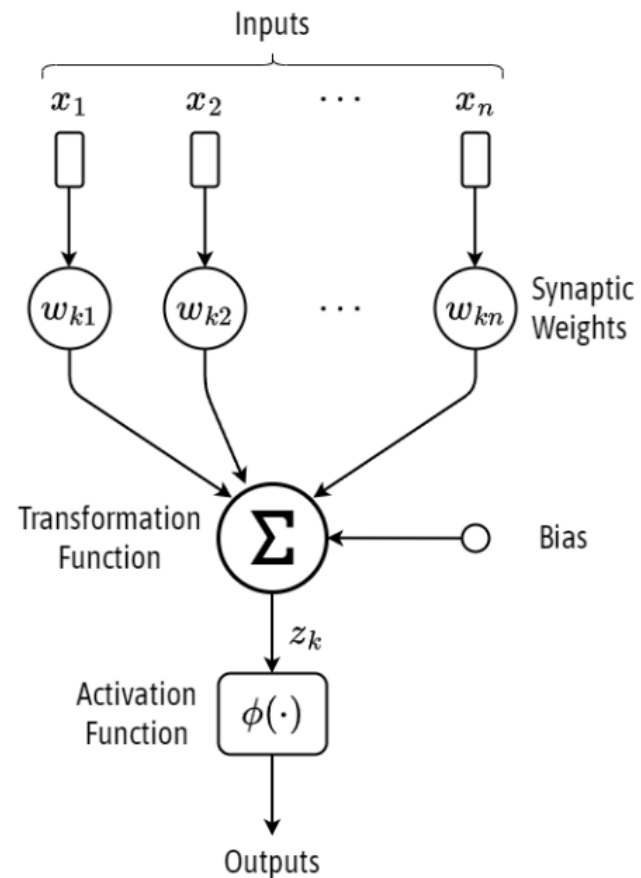
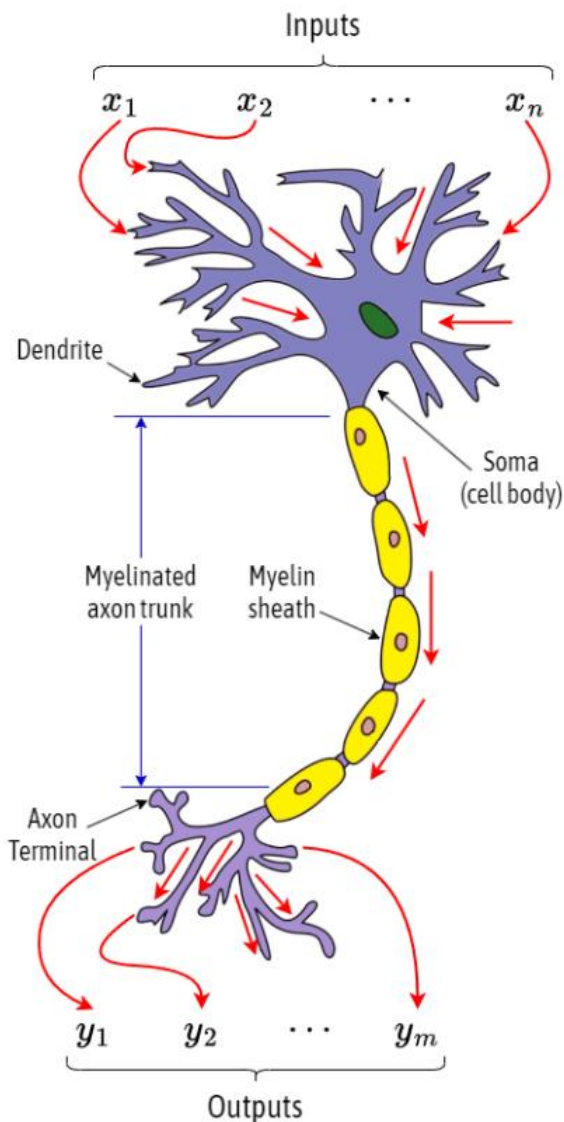
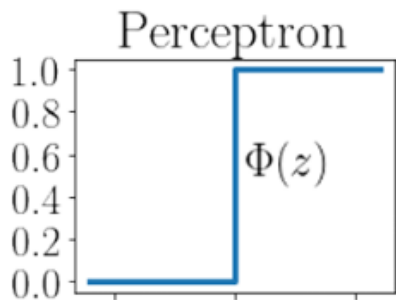
Lecture 02

ANN에서 Deep Learning까지

생물학적 신경(뉴런) 동작 원리

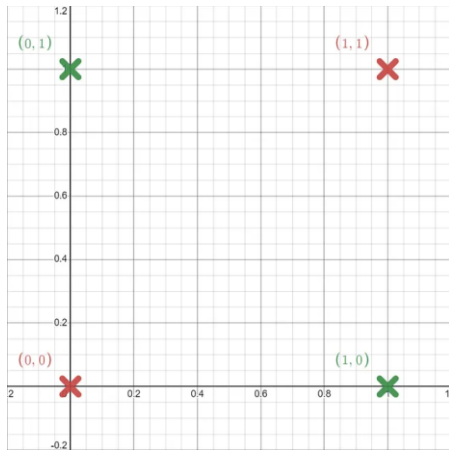
- ✓ 수상돌기(dendrites) : 신호를 수신 (안테나)
- ✓ 세포체(cell body) : 신호를 통합 (프로세서)
- ✓ 축삭(axon) : 신호를 시냅스를 통해 전송 (케이블)
- ✓ 시냅스(Synapse) : 뉴런으로 정보 전달 (커넥터)

1958년 퍼셉트론 신경망으로 AI 봄이 시작됨

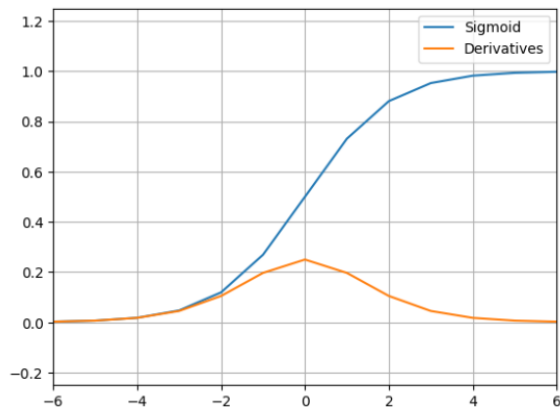


제2의 AI 봄이 시작됨 (1980~) : MLP(다층퍼셉트론)

(1) 단일 퍼셉트론으로 XOR 문제 해결 안됨



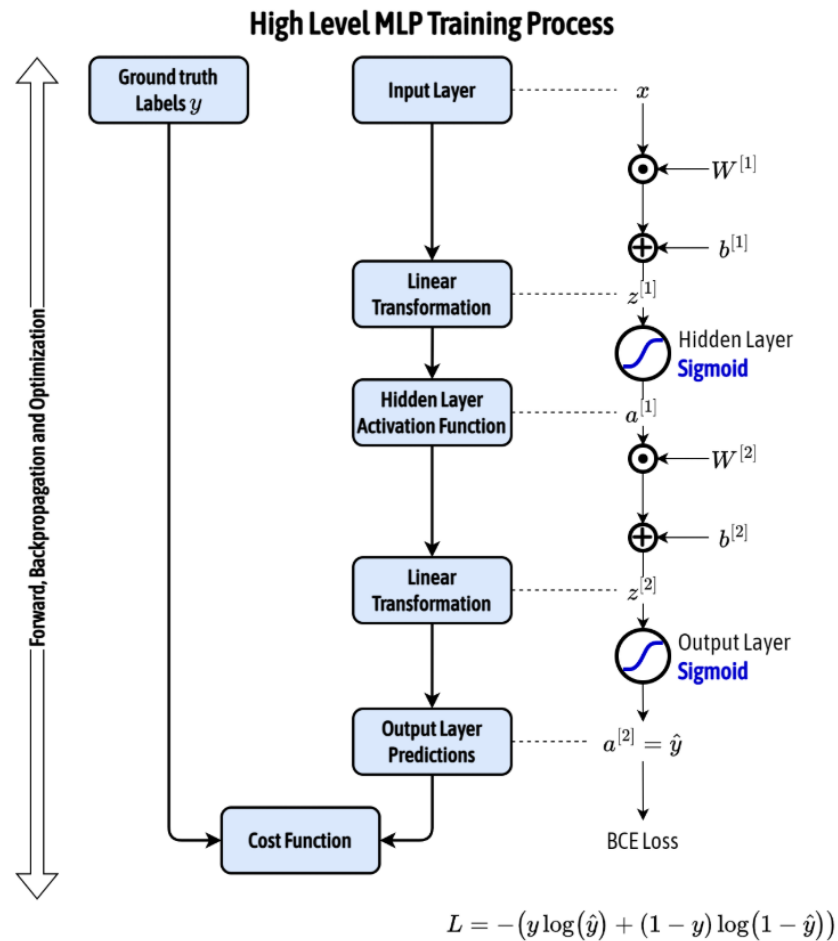
(2) 시그모이드 함수와 미분



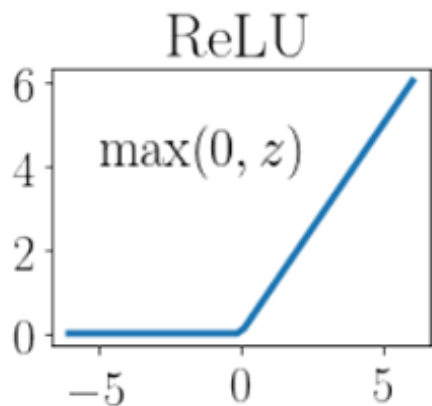
(3) 멀티 퍼셉트론에
역전파알고리즘



(4) MLP로 XOR 문제를 해결하고 다시 AI 봄이 됨

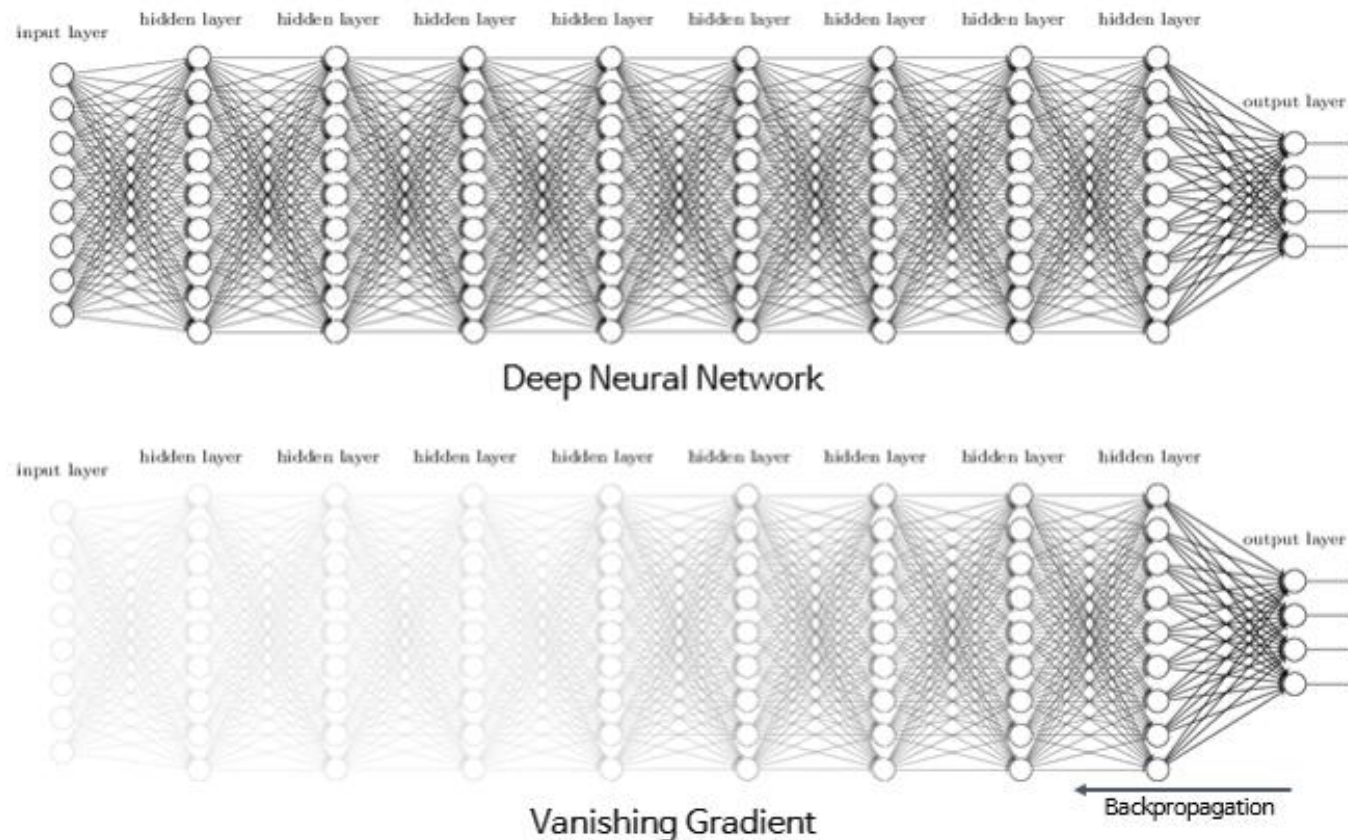


(3) ReLU 활성화함수를 사용으로 VGP를 해결함



(2) VGP 원인은 역전파알고리즘에서
시그모이드의 미분값들의 곱한 결과가 0으로 됨

(4) CNN에서 152개 층을 사용으로 AI의 성공 사례를 만들다.

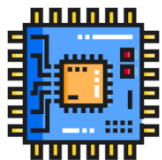


(1) Vanishing Gradient Problem 해결이 안되고 있어서 AI의 2차 겨울됨

AI 기술 발전으로 GPU 컴퓨팅 파워는 해마다 2.3배 성장

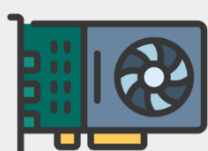
GPU는 딥러닝과 궁합이 잘 맞는다.

CPU



vs

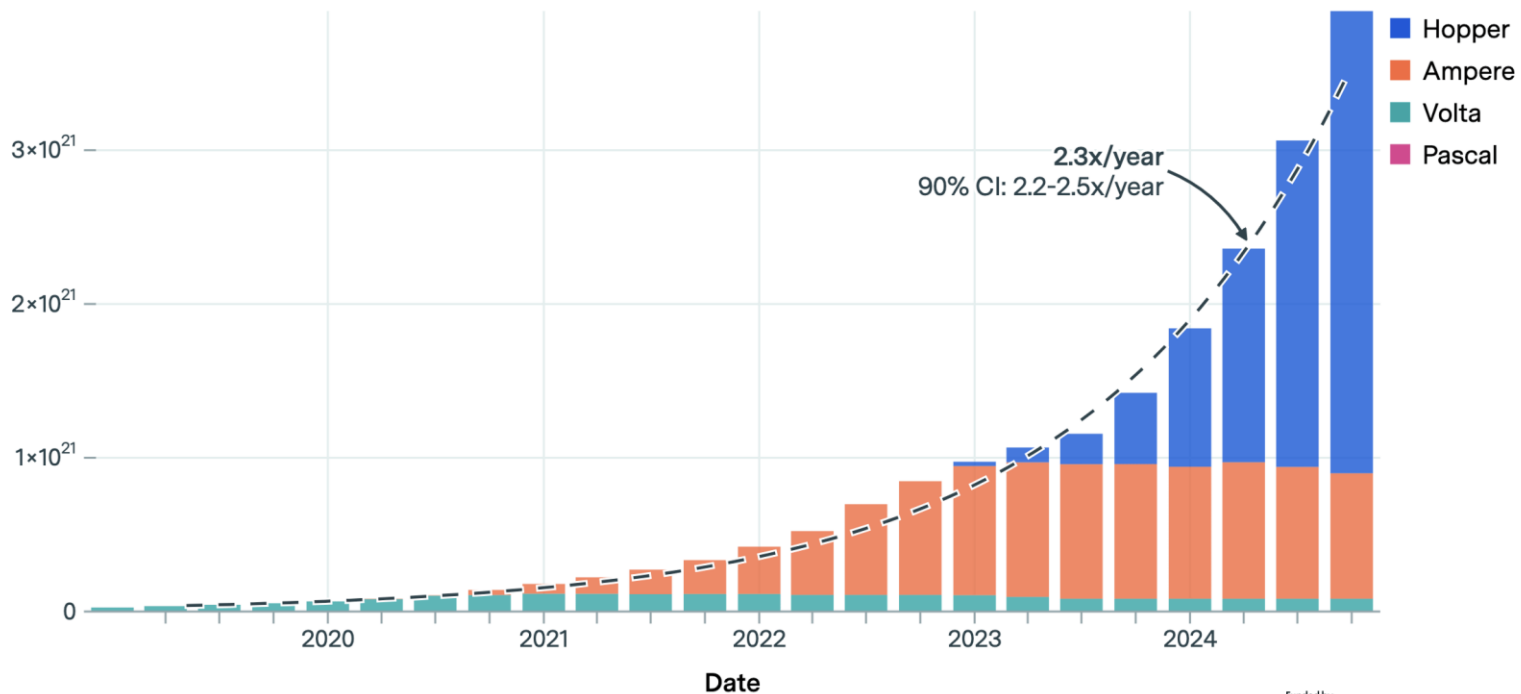
GPU



Total installed NVIDIA computing power by GPU generation

EPOCH AI

Total installed computing power (FLOP/s) ⓘ



CC-BY

Funded by
ARIA epoch.ai

❖ Tokenizer()

- ✓ 토큰화와 정수 인코딩을 위해 사용
- ✓ 훈련 데이터로부터 단어 집합을 생성하고, 해당 단어 집합으로 임의의 문장을 정수 인코딩하는 과정을 보여준다.

```
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
tokenizer = Tokenizer()  
train_text = "The earth is an awesome place live"
```

```
# 단어 집합 생성  
tokenizer.fit_on_texts([train_text])
```

```
# 정수 인코딩  
sub_text = "The earth is an great place live"  
sequences = tokenizer.texts_to_sequences([sub_text])[0]
```

```
print("정 수 인 코 딩 :", sequences)  
print("단 어 집 합 :", tokenizer.word_index)
```

출력 결과를 보면 great 는 단어 집합 (vocabulary) 에 없으므로 출력 되지 않습니다

정 수 인 코 딩 : [1, 2, 3, 4, 6, 7]

단 어 집 합 : {'the': 1, 'earth': 2, 'is': 3, 'an': 4, 'awesome': 5, 'place': 6, 'live': 7}

❖ pad_sequence()

- ✓ 정해진 길이보다 길이가 긴 샘플은 값을 일부 자르고, 정해진 길이보다 길이가 짧은 샘플은 값을 0 으로 채웁니다.
 - 모델의 입력으로 사용하려면 모든 샘플의 길이를 동일하게 맞추어야 할 때가 있습니다.
 - 이를 자연어 처리에서는 패딩 (padding) 작업이라고 하는데, 보통 숫자 0 을 넣어서 길 이가 다른 샘플들의 길이를 맞춰줍니다.
- ✓ padding = 'pre' 를 선택하면 앞에 0 을 채우고
- ✓ padding = 'post' 를 선택하면 뒤에 0 을 채움

```
[6] pad_sequences([[1, 2, 3], [1,2,3, 4, 5, 6,7,8,9], [7, 8]], maxlen=8, padding='pre')
```

```
⇒ array([[0, 0, 0, 0, 0, 1, 2, 3],  
        [2, 3, 4, 5, 6, 7, 8, 9],  
        [0, 0, 0, 0, 0, 0, 7, 8]], dtype=int32)
```

```
[7] pad_sequences([[1, 2, 3], [1,2,3, 4, 5, 6,7,8,9], [7, 8]], maxlen=8, padding='post')
```

```
⇒ array([[1, 2, 3, 0, 0, 0, 0, 0],  
        [2, 3, 4, 5, 6, 7, 8, 9],  
        [7, 8, 0, 0, 0, 0, 0, 0]], dtype=int32)
```

2. 워드 임베딩 (Word Embedding)

- ❖ 워드 임베딩이란 텍스트 내의 단어들을 밀집 벡터 (dense vector)로 만드는 것
 - ✓ 밀집 벡터를 임베딩(embedding vector)라고 한다. 보통 256, 512, 1024 차원을 갖는다.
- ❖ Embedding()
 - ✓ Embedding()은 정수 인코딩이 된 단어들을 입력을 받아서 임베딩을 수행합니다.
 - ✓ (입력될 총 단어수, 임베딩 후 출력되는 크기, 매번 입력되는 단어의 수)

```
from tensorflow.keras.layers import Embedding

# 1. 토큰화
tokenized_text = [['Hope', 'to', 'see', 'you', 'soon'], ['Nice', 'to', 'see', 'you ', 'again']]

# 2. 각 단어에 대한 정수 인코딩
encoded_text = [[0, 1, 2, 3, 4], [5, 1, 2, 3, 6]]

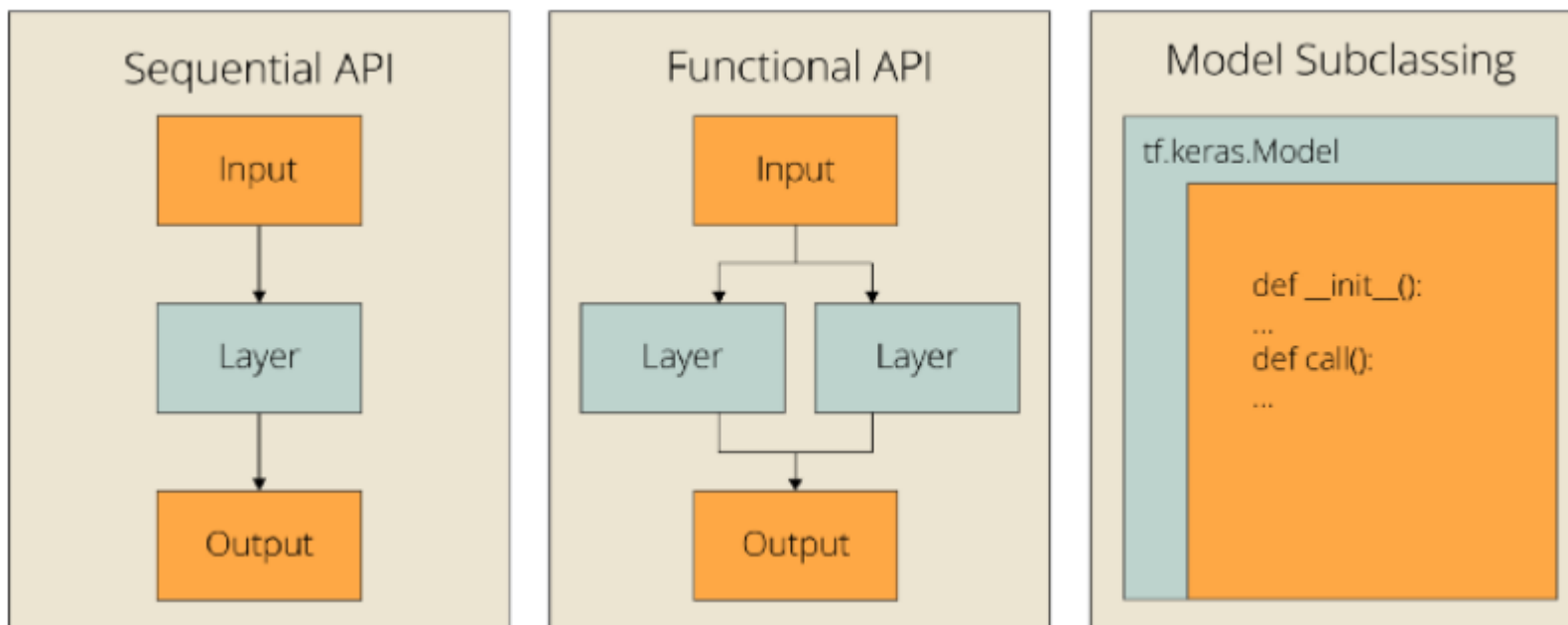
# 3. 위 정수 인코딩 데이터가 아래의 임베딩 층의 입력이 된다.
vocab_size = 7
embedding_dim = 2
Embedding(vocab_size, embedding_dim, input_length=5)
```

Embedding() 출력 예시

index	embedding
0	[1.2, 3.1]
1	[0.1, 4.2]
2	[1.0, 3.1]
3	[0.3, 2.1]
4	[2.2, 1.4]
5	[0.7, 1.7]
6	[4.1, 2.0]

❖ Sequential()

- ✓ 케라스에서는 대부분 모델을 순차적으로 쌓아 올린다.
- ✓ `model.add()`로 여러 개 층을 단계적으로 추가한다.
- ✓ 복잡한 모델은 Functional API를 이용하여 만든다. (예, deep-and-wide() 모델)



❖ Dense() 층의 대표적 인자

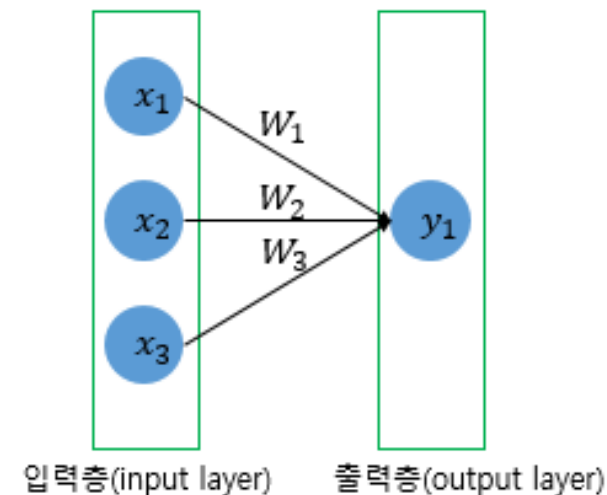
✓ 첫번째는 출력 뉴런 개수, input_dim은 입력 뉴런 개수, 활성화함수

- linear ~ 선형 회귀자; sigmoid ~ 이중 분류자; softmax ~ 다중 분류자; relu ~ 보통 훈련과정에서 사용

❖ summary() : 모델의 정보를 요약해서 보여줍니다

```
from keras.models import Sequential
from keras.layers import Embedding, Dense

vocab_size = 3000
output_dim = 256
input_length = 100
model = Sequential()
model.add(Embedding(vocab_size, output_dim, input_length=input_length))
model.add(Dense(1, input_dim=3, activation='relu'))
```



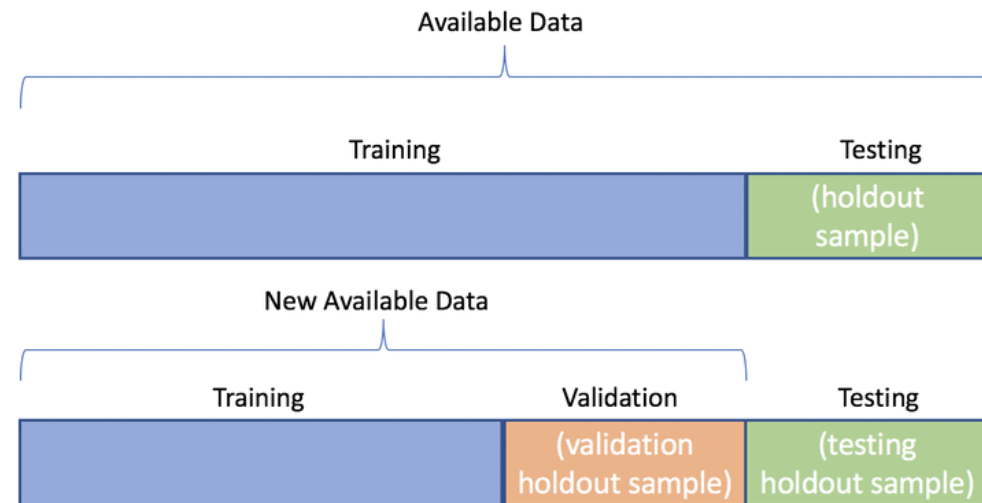
- ❖ `compile()` ~ 손실 함수와 옵티마이저, 메트릭 함수를 선택합니다.
 - ✓ `optimizer` = 훈련 과정을 설정하는 옵티마이저를 설정합니다.
 - ✓ `loss` = 훈련 과정에서 사용할 손실 함수(loss function)를 설정합니다.
 - `mean_squared_error`는 회귀문제
 - `categorical_crossentropy`는 분류 문제이고, 입력의 정수 임베딩
 - `sparse_categorical_crossentropy`는 분류문제이고, 입력 `X_train`의 값이 one-hot 인코딩
 - `binary_crossentropy`도 분류문제를 다룬다.
 - ✓ `metrics` = 훈련을 모니터링하기 위한 지표를 선택합니다.
 - 'acc' ~ 정확도, 'mae'

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

❖ `model.fit()`은 모델을 훈련한다.

- ✓ 첫번째 인자 = 훈련 데이터에 해당됩니다.
- ✓ 두번째 인자 = 지도 학습에서 레이블 데이터에 해당됩니다.
- ✓ `epochs` = 에포크. 에포크 1은 전체 데이터를 한 차례 훑고 지나갔음을 의미함. 정수값 기재 필요. 총 훈련 횟수를 정의합니다.
- ✓ `batch_size` = 배치 크기. 기본값은 32..
- ✓ `validation_split` = `validation_data`와 동일하게 검증데이터로 훈련데이터의 20%를 사용함
- ✓ `verbose` = 0 : 아무 것도 출력하지 않습니다.
 - 1은 훈련의 진행도를 막대로 보여주고, 2는 미니 배치마다 손실 정보를 출력합니다.

```
model.fit( X_train, y_train,  
          epochs=10, batch_size=32,  
          validation_split=0.2,  
          verbose=0 )
```



❖ evaluate()

- ✓ 모델이 훈련을 통해서 최적의 파라미터를 찾았다. 더 이상 최적화가 필요없어서 1회 계산으로 충분하다.
- ✓ 모델 평가를 위해 test(테스트) 데이터를 통해 학습한 모델에 대한 정확도를 평가한다.
- ✓ 훈련(train) 데이터를 사용하면 모델 평가가 안된다.

❖ predict()

- ✓ 임의의 입력에 대한 모델의 출력값을 확인합니다.
- ✓ 테스트 데이터를 이용하여 입력 값에 대한 예측을 한다.

```
# 위의 fit() 코드의 연장선상인 코드  
model.evaluate(X_test, y_test, batch_size=32)
```

6. 모델의 저장(Save)과 로드(Load)

- ❖ `save()` ~ 인공지능망 모델을 hdf5 파일에 저장합니다.
 - ✓ 복습을 위한 스터디나 실제 어플리케이션 개발 단계에서 구현한 모델을 저장하고 불러오는 일은 중요합니다.
 - ✓ 모델을 저장한다는 것은 학습이 끝난 신경망의 구조를 보존하고 계속해서 사용할 수 있다는 의미입니다.
- ❖ `load_model()` ~ 저장한 모델 파라미터들을 호출한다.

```
▶ model.save("model_name.h5")
```

```
▶ from tensorflow.keras.models import load_model  
model = load_model("model_name.h5")
```

Lecture 03

자연어 처리를 위한 텍스트 전처리

- ❖ 영어는 표준 토큰화 규칙인 Penn Treebank Tokenization를 따른다.
 - ✓ 하이픈으로 구성된 단어는 하나로 유지한다.
 - ✓ doesn't와 같이 아포스트로피로 '접어'가 함께하는 단어는 분리해준다.

Starting a **home-based** restaurant may be an ideal. it **doesn't** have a food chain or restaurant of their own.



['Starting', 'a', '**home-based**', 'restaurant', 'may', 'be', 'an', 'ideal.', 'it', '**does**', '**n't**', 'have', 'a', 'food', 'chain', 'or', 'restaurant', 'of', 'their', 'own', '.']

- ❖ 한국어는 띄어쓰기가 잘 지켜지지 않는다.
 - ✓ 한국어는 어순이 그렇게 중요하지 않다.
- ❖ 주어가 손쉽게 생략된다.
- ❖ 한국어는 교착어이다.
 - ✓ 교착어란 실질적인 의미를 가지는 어간에 조사나, 어미와 같은 문법 형태소들이 결합하여 문법적인 기능이 부여되는 언어를 말한다.
- ❖ 한자어라는 특성상 하나의 음절조차도 다른 의미를 가질 수 있다.

- ❖ 한국어는 대부분의 데이터에서 띄어쓰기가 잘 지켜지지 않는 경향이 있다.
 - ✓ 띄어쓰기가 어렵고, 지키지 않더라도 쉽게 읽을 수 있다.

제가 이렇게 띄어쓰기를 전혀 하지 않고 글을 썼다고 하더라도 글을 쉽게 이해할 수 있습니다.

영어는 띄어쓰기를 하지 않으면 읽기 어려운 언어의 특성이다. 그래서 띄어쓰기가 꽤 엄격하게 지켜지는 편이다.

tobeornottobethatisthequestion

결국 띄어쓰기를 보장해주어야 하는 전처리가 필요할 수도 있다

.

❖ 같은 의미의 문장을 다음과 같이 자유롭게 쓸 수 있다.

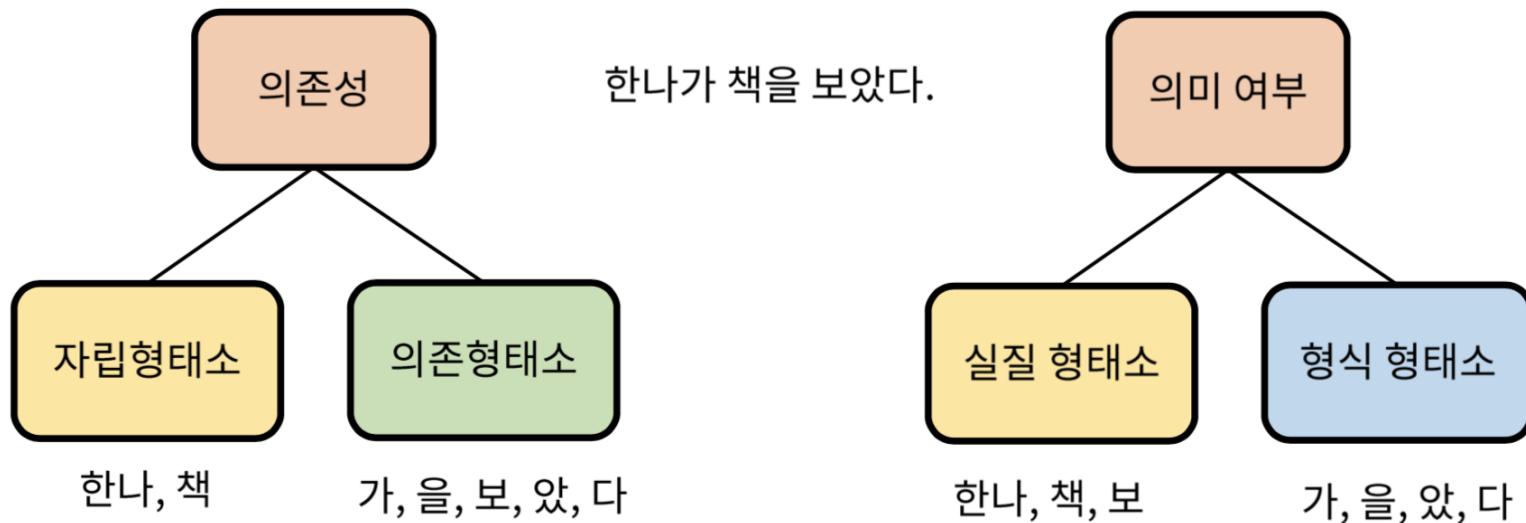
✓ 아래 예제 3.번의 경우 주어까지 생략했다.

1. 나는 운동을 했어. 체육관에서.
2. 나는 체육관에서 운동을 했어.
3. (나는) 체육관에서 운동했어.
4. 나는 운동을 체육관에서 했어.

뒤에서 학습할 다음 단어를 예측하는 Language Model에게는 매우 혼란스러운 상황

한국어는 교착어다.

형태소: 언어학적으로 말을 분석할 때, 의미가 있는 가장 작은 말의 단위



문장을 형태소 단위로 구분하고, 언어적인 구조를 파악하는 것을 형태소 분석이라 한다.

어근, 접두사/접미사, 품사(POS; Part-of Speech) 등을 구분한다.

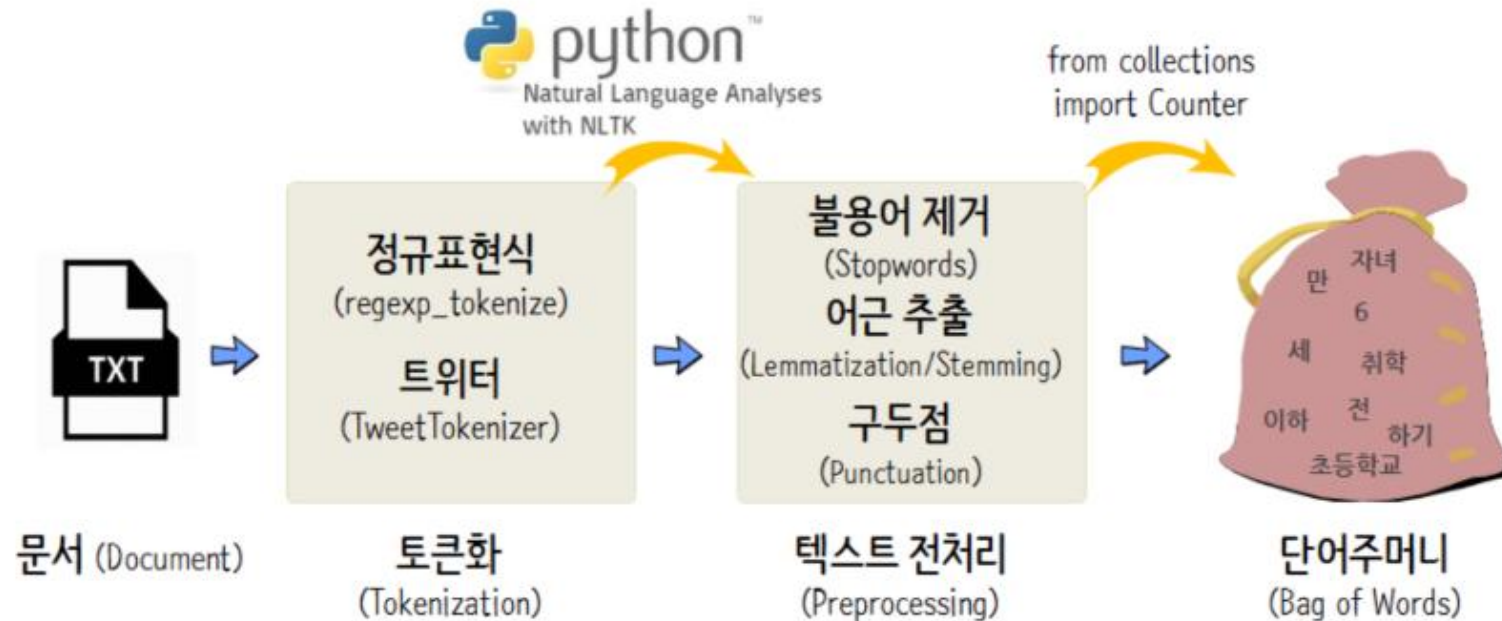
```
!pip install konlpy
from konlpy.tag import Okt
```

```
text = '나는 오늘 행복하다ㅋㅋㅋ. 너도 행복했으면 좋겠다!! 오늘도 좋은하루 보내샘^^'
```

```
okt = Okt()
text_okt = okt.morphs(text)
print(text_okt)
```

```
['나', '는', '오늘', '행복하다', 'ㅋㅋㅋ', '.', '너', '도', '행복했으면', '좋겠다', '!!', '오늘', '도', '좋은', '하루', '보내', '샘', '^^']
```

- ❖ 자연어 처리의 시작인 텍스트 전처리는 텍스트를 사용해서 사전에 처리하는 작업이다.
 - ✓ 텍스트 전처리 과정은 자연어처리에서 절반 이상을 차지한다.



- ❖ 토큰화(tokenization)은 텍스트를 특정 단위인 토큰(token)으로 나누는 작업이다.
 - ✓ 보통 토큰의 기준을 단어(word)이다.
- ❖ `text_to_word_sequence()` 함수를 이용한 단어 토큰화

```
tf.keras.preprocessing.text.text_to_word_sequence(  
    input_text,  
    filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',  
    lower=True,  
    split=' '  
)
```

- 1) 단어를 공백 기준으로 분리한다 (split= ' ')
- 2) '(어퍼스트로피) 문자를 제외한 구두점과 제어문자를 필터링한다.
- 3) 텍스트를 소문자로 변환한다.

한글 토큰화

```
from tensorflow.keras.preprocessing.text import text_to_word_sequence
text = '나는 오늘 행복하다!! 너도 행복했으면 좋겠다!!'
tokens = text_to_word_sequence(text)
print('토큰:', tokens)
```

토큰: ['나는', '오늘', '행복하다', '너도', '행복했으면', '좋겠다']

!! 와 같은 구두점을 제거 뒤에 ' '(공백문자, whitespace)을 기준으로 단어 토큰화하였다.

영어 토큰화

```
text = 'If you don't try, you can't it.'
tokens = text_to_word_sequence(text)
print('토큰:', tokens)
```

토큰: ['if', 'you', 'don't', 'try', 'you', 'can't', 'it']

알파벳을 소문자로 바꾸면서 마침표나 쉼표 등의 구두점을 제거한다.
하지만 '는 보존하고 있다.

❖ 불용어(stopword)는 자주 등장하지만 분석을 하는 것에 있어서는 큰 도움이 안되는 단어이다.

✓ NLTK에서 100여개의 영어 단어들을 불용어로 미리 정의하였음.

- 불용어 예제: I, my, me, over, 조사, 접미사 등

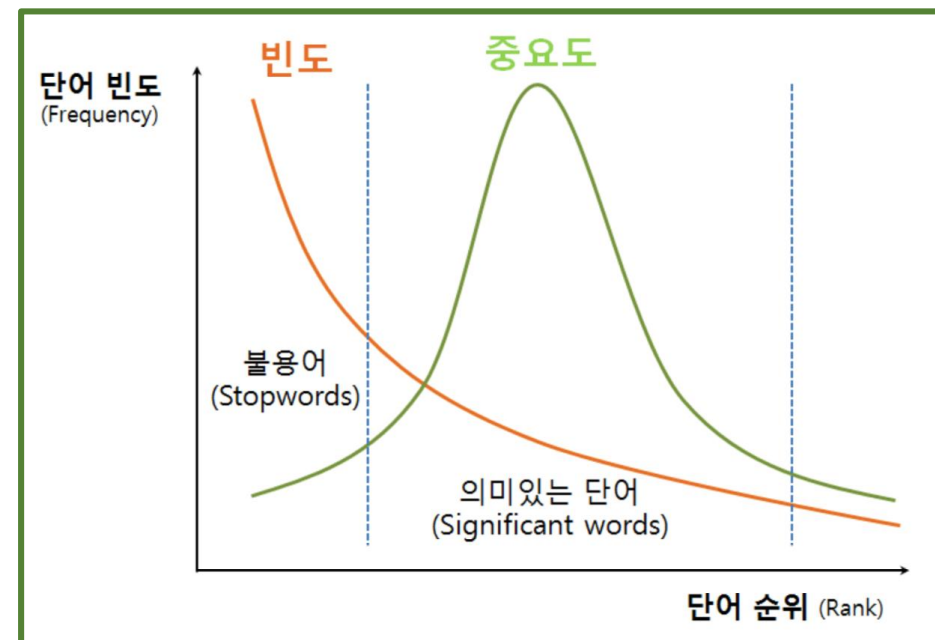
영어 불용어 제거 예제

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from konlpy.tag import Okt

stop_words_list = stopwords.words('english')
print('불용어 개수 :', len(stop_words_list))
print('불용어 10개 출력 :', stop_words_list[:10])
```

불용어 개수 : 179

불용어 10개 출력 : ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]



❖ 정수 인코딩이란

✓ 단어에 정수를 부여하는 방법으로, 단어 빈도수가 높은 순으로 낮은 정수를 부여한다.

❖ Tokenizer() : 문장으로부터 단어를 토큰화와 정수 인코딩한다.

✓ 이 함수는 단어와 숫자의 키(Key)-값(Value) 쌍의 딕셔너리로 반환한다.

```
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
text = [ 'I love my dog', 'I, love my cat', 'You love my dog!']
```

```
tokenizer = Tokenizer(num_words = 100)
```

num_words는 빈도수가 높은 상위 몇 개의 단어만을 사용

```
tokenizer.fit_on_texts(text)
```

```
word_index = tokenizer.word_index
```

문자 데이터를 입력받아서 리스트 형태로 변환

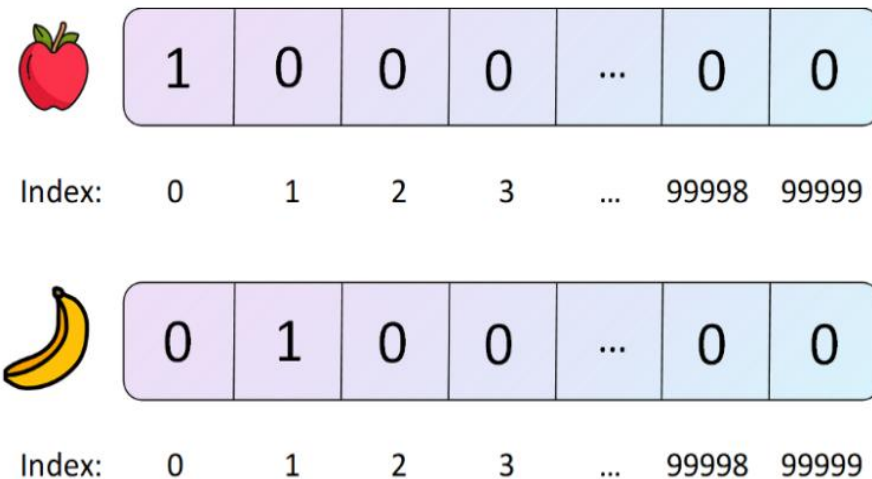
```
print(word_index)
```

```
{'love': 1, 'my': 2, 'i': 3, 'dog': 4, 'cat': 5, 'you': 6}
```




Key-Value(키-값) 쌍

- ❖ '원-핫(one-hot) '이라는 의미는 이진 벡터 중에 하나만 1이고 나머지는 모두 0이라는 것의 의미한다.
 - ✓ 중복은 카운트하지 않은 단어들의 집합이다.

One-Hot embeddings



- ❖ 첫째, 단어 간의 관계나 유사성을 전혀 표현 할 수 없다
- ❖ 둘째, 단어의 집합이 커질 수록 벡터의 차원이 매우 커지며, 이는 저장 공간이 계속 늘어나는 단점이 있다.

					
Man	Woman	King	Queen	Apple	Orange
(5391)	(9853)	(4914)	(7157)	(456)	(6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

❖ 자연어처리에서 padding을 해야하는 이유

- ✓ 모든 문장에 대해서 정수 인코딩을 수행하였을 때 길이는 서로 다를 수 있다.
- ✓ 신경망 입력으로 사용하기 위해서는 가상의 단어를 추가하여 길이를 맞춰준다.

```
text = ['토큰 이해함. ㅎㅎ',  
       '정수 인코딩이 원-핫 인코딩인가? 가물 가물! ㅋㅋ',  
       '이제 슬슬 졸린다. 자연어 처리에서 패딩을 배우고 있는데 이게 뭐지?']
```



정수 인코딩:

```
[[2, 3, 4], [5, 6, 7, 8, 9, 1, 1, 10], [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]]
```

패딩 이후 문장

pre 패딩:

```
[[ 0  0  0  0  0  0  0  0  0  0  2  3  4]  
 [ 0  0  0  0  0  0  0  5  6  7  8  9  1  1 10]  
 [ 0  0  0  0  0 11 12 13 14 15 16 17 18 19 20]]
```

최대 패딩 문장 길이 = 10

Lecture 04

자연어처리를 위한 텍스트 전처리 실습

1. 파이썬 라이브러리 NLTK를 이용한 토큰화

```
[1]: !pip install nltk
```

토큰화

```
[2]: import nltk  
from nltk.tokenize import word_tokenize  
  
nltk.download('puncck_tab')
```

```
[nltk_data] Error loading puncck_tab: Package 'puncck_tab' not found in  
[nltk_data]      index
```

```
[2]: False
```

구두점 제거하기(정제)

```
[3]: tokens = word_tokenize("Hello World!, This is a dog. 한글 테스트!")  
  
# 문자나 숫자인 경우에만 단어를 리스트에 추가한다.  
words = [word for word in tokens if word.isalpha()]  
print(words)  
  
['Hello', 'World', 'This', 'is', 'a', 'dog', '한글', '테스트']
```

2. Keras를 이용한 전처리

토큰화 함수를 이용하자. NLTK 토큰화 모듈과는 약간 다르다.

```
[1]: from tensorflow.keras.preprocessing.text import Tokenizer
```

```
[2]: text = ['I love my dog', 'I, love my cat', 'You love my dog!']
```

```
t = Tokenizer()  
t.fit_on_texts(text)  
print("단어집합 : ", t.word_index)
```

단어집합 : {'love': 1, 'my': 2, 'i': 3, 'dog': 4, 'cat': 5, 'you': 6}

```
[3]: seq = t.texts_to_sequences(text)  
print('text:', text)  
print('seq :', seq)
```

```
text: ['I love my dog', 'I, love my cat', 'You love my dog!']  
seq : [[3, 1, 2, 4], [3, 1, 2, 5], [6, 1, 2, 4]]
```

원-핫 인코딩

원-핫 인코딩

```
[4]: import numpy as np  
from tensorflow.keras.utils import to_categorical
```

```
[5]: text = ["cat", "dog", "cat", "bird"] # 우리가 변환하고 싶은 텍스트  
total_pets = ["cat", "dog", "turtle", "fish", "bird"] # 단어 집합  
print("text=", text)
```

```
text= ['cat', 'dog', 'cat', 'bird']
```

```
[6]: mapping = {} # 변환에 사용되는 딕셔너리를 만든다.  
for x in range(len(total_pets)):  
    mapping[total_pets[x]] = x # "cat"->0, "dog"->1, ...  
print(mapping)
```

```
{'cat': 0, 'dog': 1, 'turtle': 2, 'fish': 3, 'bird': 4}
```

```
[7]: for x in range(len(text)): # 단어들을 순차적인 정수인덱스fh  
    text[x] = mapping[text[x]]  
print("text=", text)
```

```
text= [0, 1, 0, 4]
```

```
[8]: one_hot_encode = to_categorical(text) # 정수인덱스를 원-핫 인코딩  
print("text=", one_hot_encode)
```

```
text= [[1. 0. 0. 0. 0.]  
       [0. 1. 0. 0. 0.]  
       [1. 0. 0. 0. 0.]  
       [0. 0. 0. 0. 1.]]
```


패딩(padding)

```
[1]: from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.preprocessing.text import Tokenizer

[2]: # text = ['I love my dog', 'I, love my cat', 'You love my dog!']
    text = ['토큰 이해함. ㅎㅎ',
            '정수 인코딩이 원-핫 인코딩인가? 가물 가물! ㅋㅋ',
            '이제 슬슬 줄린다. 자연어 처리에서 패딩을 배우고 있는데 이게 뭐지?']
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(text)
    word_inx = tokenizer.word_index

[3]: int_enc = tokenizer.texts_to_sequences(text)
    print('정수 인코딩:\n\n', int_enc)
    max_length = 15
    pad_int_enc = pad_sequences(int_enc, maxlen=max_length, padding='pre')
    print("\npre 패딩:\n\n", pad_int_enc)
    pad_int_enc = pad_sequences(int_enc, maxlen=max_length, padding='post')
    print("\npost 패딩:\n\n", pad_int_enc)
```

정수 인코딩:

[[2, 3, 4], [5, 6, 7, 8, 9, 1, 1, 10], [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]]

pre 패딩:

[[0 0 0 0 0 0 0 0 0 0 0 0 2 3 4]
[0 0 0 0 0 0 0 5 6 7 8 9 1 1 10]
[0 0 0 0 0 11 12 13 14 15 16 17 18 19 20]]

post 패딩:

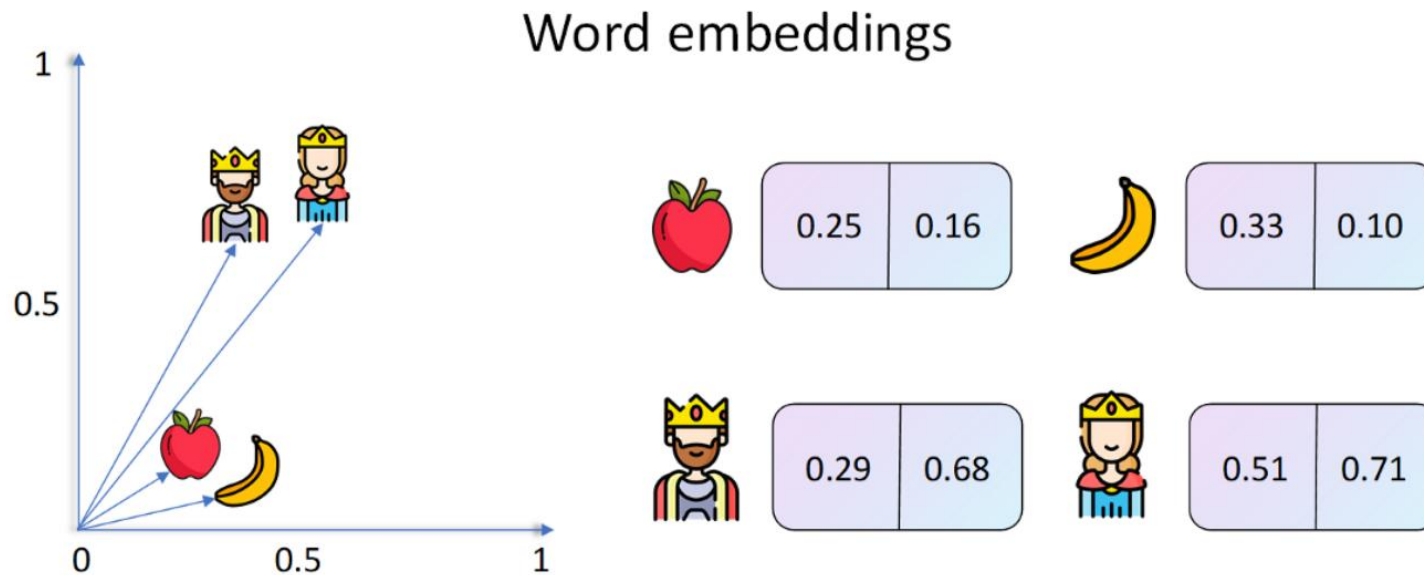
[[2 3 4 0 0 0 0 0 0 0 0 0 0 0 0]
[5 6 7 8 9 1 1 10 0 0 0 0 0 0 0]
[11 12 13 14 15 16 17 18 19 20 0 0 0 0 0]]

Lecture 05

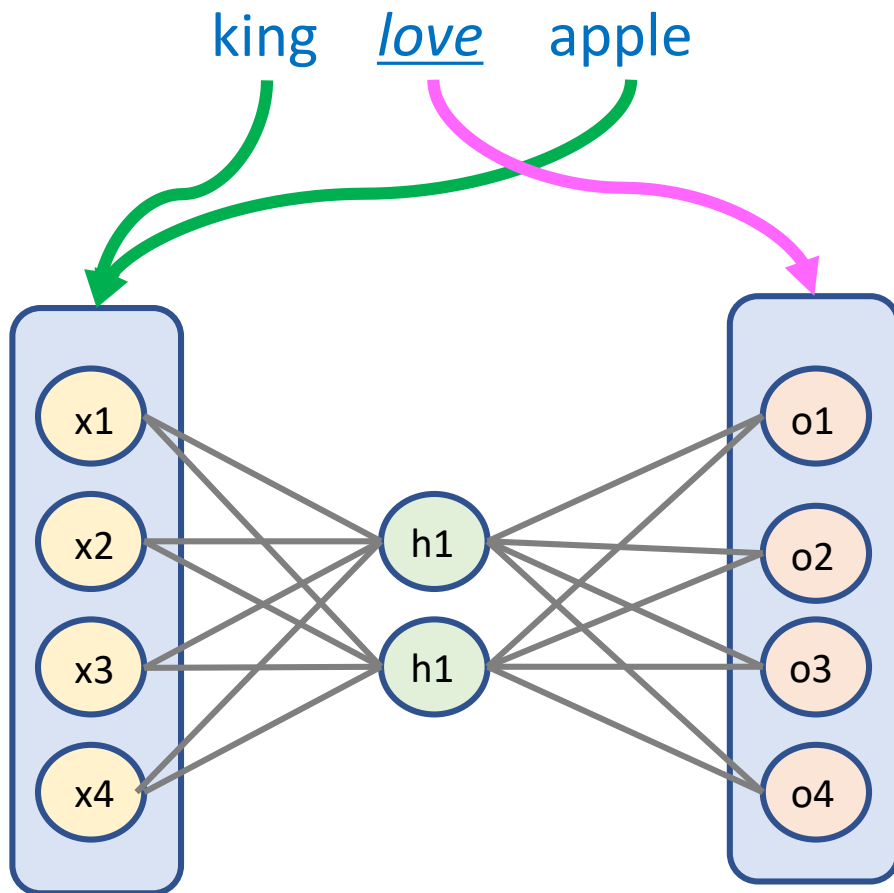
단어 임베딩 소개 및 Word2Vec 실습

- ❖ 벡터화에 신경망을 사용하지 않을 경우
 - ✓ 단어에 대한 벡터 표현 방법 : 원-핫 인코딩
 - ✓ 문서에 대한 벡터 표현 방법 : Document Term Matrix, TF-IDF
 - ✓ Pretrained Language Model의 시대.
- ❖ 벡터화에 신경망을 사용하는 경우 (2008 ~ 2018)
 - ✓ 단어에 대한 벡터 표현 방법 : 워드 임베딩(Word2Vec, GloVe, FastText, Embedding layer)
 - ✓ 문서에 대한 벡터 표현 방법 : Doc2Vec, Sent2Vec
- ❖ 문맥을 고려한 벡터 표현 방법 : ELMo, BERT (2018 - present)
 - ✓ Pretrained Language Model의 시대

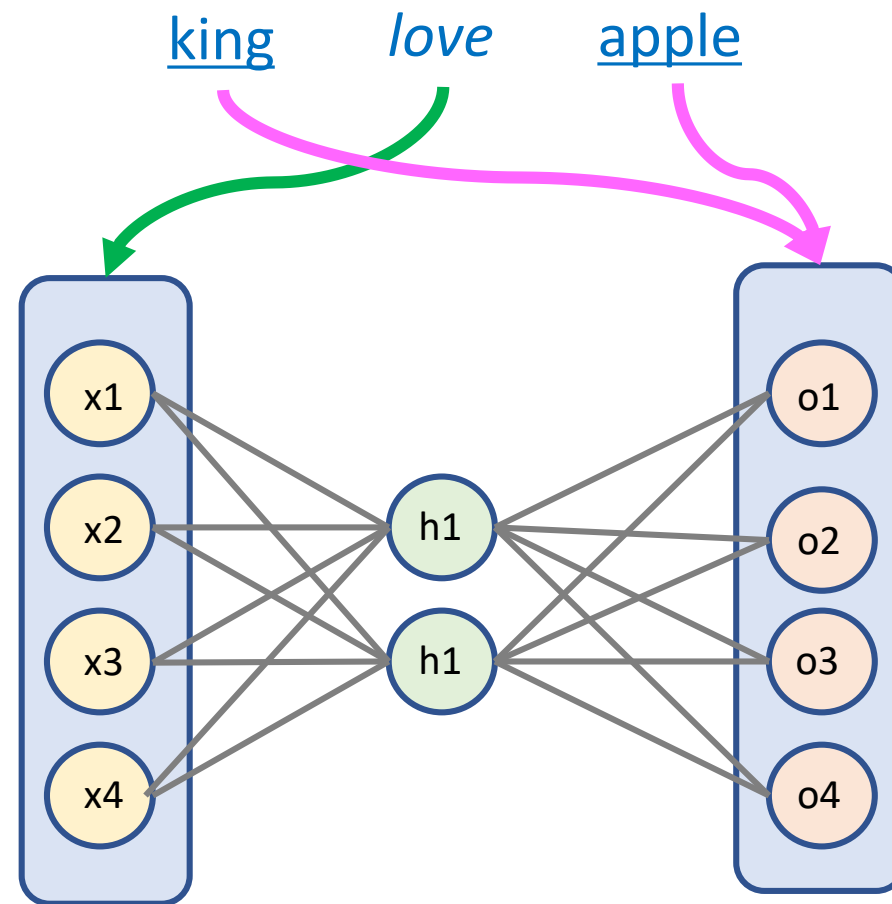
- ❖ 워드임베딩(word embedding)
 - ✓ 두 개의 비슷한 단어가 벡터 공간에서 매우 가까운 위치에 있는 유사한 벡터로 표현됨
 - ✓ 사전 훈련된 워드임베딩 알고리즘으로는 Word2Vec, FastText, Glove등이 있다.
- ❖ Word2Vec 모델은 각 단어의 임베딩 벡터는 해당 단어의 의미와 문맥 context를 반영한다.
 - ✓ Cbow와 Skip-Gram 학습 방법을 이용한다.



CBoW은 전후 문맥을 고려하여
중간에 들어갈 중심 단어를 학습한다.

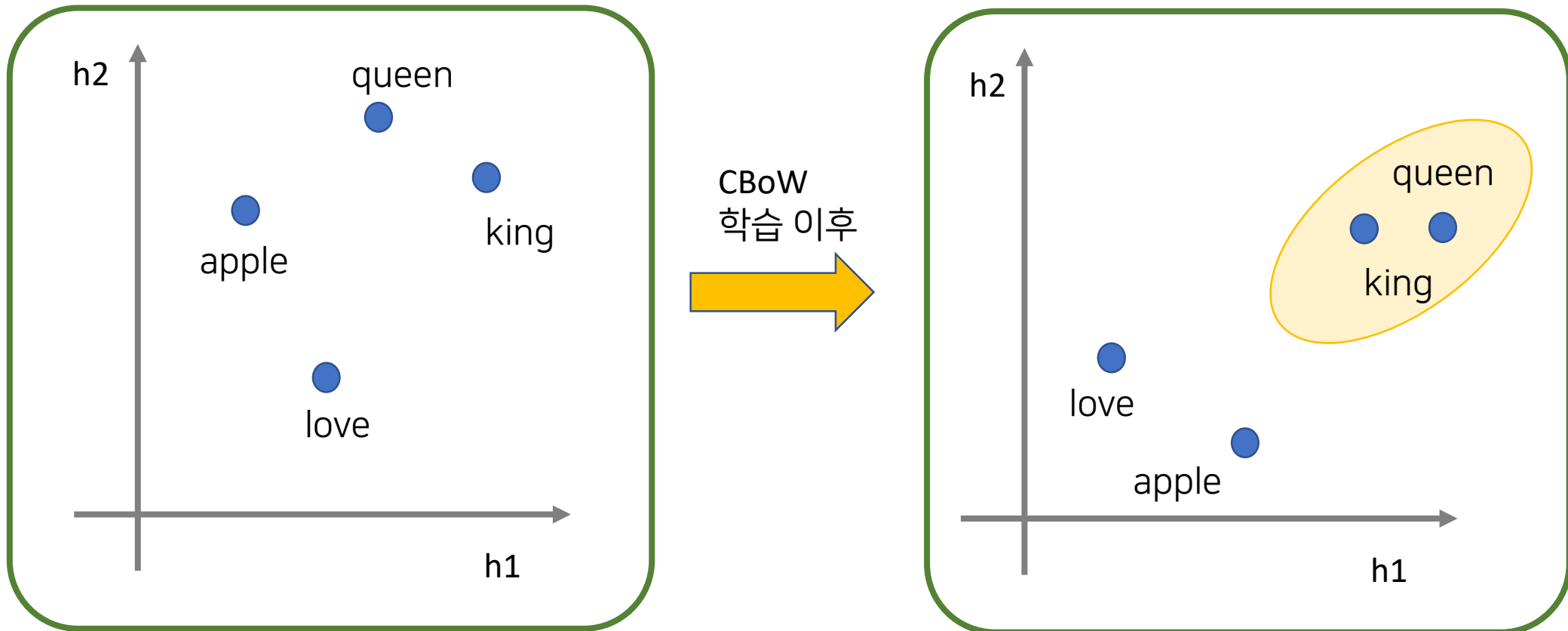


Skip-Gram은 중심단어에서 주변단어를 예측



- ❖ CBoW 학습을 통해서 워드투벡터 모델을 무엇을 학습하나요?
 - ✓ 4차원의 벡터 정보를 2차원(은닉층)으로 배치한다.

Word2Vec은 단어들이 문맥 안에서 그들의 관계성(유사성)을 스스로 학습해 나가는 알고리즘이다.




- ❖ 동형어, 다의어에 대해서는 제대로 훈련되지 않음.
- ❖ 단순히 주변 단어만을 고려하므로 문맥을 고려한 의미를 담고있지는 않음.

Contextual Embedding의 등장.
Ex) ELMo, Context2Vec, BERT 등..





```
[1] import pandas as pd
df=pd.read_csv("https://raw.githubusercontent.com/hongsukyi/Lectures/main/data/naver_movie_train.txt", sep="\t")
```

```
[2] df.head()
```



	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1



```
[3] print(len(df))
```

150000

✓ 편의상 데이터 개수를 줄여주자.

- 150,000개의 10%인 15,000개를 사용하자.

```
df=df[:1000]
print(len(df))
```

1000

네이버 영화 리뷰 데이터 읽어오기


```
[5] # 결측치처리
df.replace("", float("NaN"), inplace=True)
df = df.dropna().reset_index(drop=True)
print('결측치 처리 이후:', len(df))
# 중복 제거
df = df.drop_duplicates(['document']).reset_index(drop=True)
print('중복 제거 이후:', len(df))
# 한글이 아닌 문자 제거
df['document'] = df['document'].str.replace("[^ㄱ-ㅎㅌ-ㅣ가-힣 ]", "")
print('한글 아닌 문자 제거 이후:', len(df))
# 길이가 짧은 데이터 제거
df['document'] = df['document'].apply(lambda x: ' '.join(
    [token for token in x.split() if len(token) > 2]))
print('리뷰 길이가 짧은 것 제거 :', len(df))
```

결측치 처리 이후: 1000
중복 제거 이후: 1000
한글 아닌 문자 제거 이후: 1000
리뷰 길이가 짧은 것 제거 : 1000

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

```
!pip install konlpy
from konlpy.tag import Okt
```

```
[8] df = list(df['document'])
```

```
[9] okt = Okt()
df_token = []
```

```
[10] for sentence in df:
    t_okt = okt.morphs(sentence, stem=True) # 토큰화
    t_sentence = [word for word in t_okt if not word in stopwords]
    df_token.append(t_sentence)
```

```
[11] len(df_token)
```

⇒ 1000

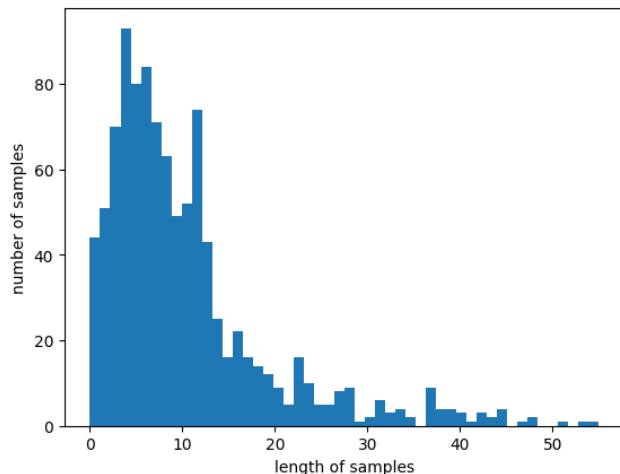
```
print(df_token[:5])
```

⇒ [['더빙', '.', '짜증나다', '목소리'], ['흠', '...', '포스터', '보고'],

```
[13] import matplotlib.pyplot as plt
```

```
[14] print('리뷰의 최대 길이 :', max(len(review) for review in df_token))
print('리뷰의 평균 길이 :', sum(map(len, df_token))/len(df_token))
plt.hist([len(review) for review in df_token], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

리뷰의 최대 길이 : 55
리뷰의 평균 길이 : 10.352



```
(',', 0.9966306090354919)
('...', 0.9959167838096619)
('.', 0.9958727955818176)
('보다', 0.9958419799804688)
('이다', 0.9958160519599915)
('로', 0.9955626130104065)
('..', 0.9955294132232666)
('을', 0.9952145218849182)
('적', 0.995191752910614)
('영화', 0.9948384165763855)
```

```
[22] import gensim      # gensim은 파이썬에서 Word2Vec을 지원
gensim.__version__
from gensim.models import Word2Vec
```

```
▶ embedding_dim = 100
   # sg=0은 CBOW, sg=1은 Skip-gram
   model = Word2Vec(
       sentences = df_token, vector_size = embedding_dim,
       window = 5, min_count = 5, workers = 4, sg = 0 )
```

```
[24] word_vectors = model.wv
      vocabs = list(word_vectors.key_to_index.keys())
```

```
[25] for sim_word in model.wv.most_similar("배우"):
      print(sim_word)
```

```
[29] print(model.wv.similarity('연기', '눈물'))
```

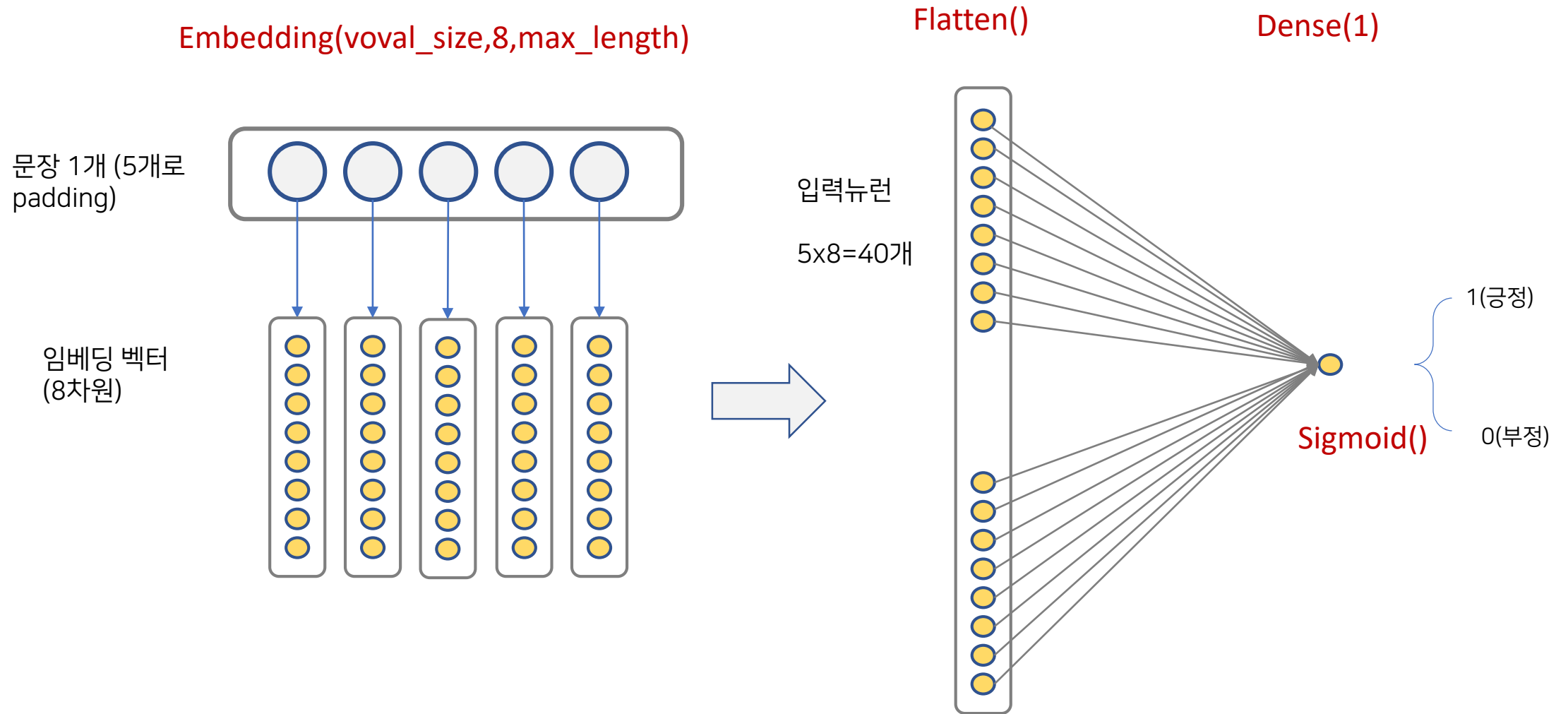
```
⇒ 0.95318097
```

Lecture 06

텍스트 처리를 위한 딥러닝 분류 모델

❖ 케라스을 이용한 분류 문제에서 이해할 사항

- ✓ 라벨의 개수 (분류 문제)
- ✓ 데이터를 훈련, 테스트, 검증으로 나누기
- ✓ 모델을 어떻게 만들 것인지
 - 베이스라인 모델은 입력 1개와 출력 1개 그리고 1개의 은닉층으로 구성
 - 입력에 따라서 임베딩을 할 것인지.
 - 데이터에 따른 딥러닝 모델을 정하자. (MLP/DNN, CNN, RNN, LSTM, GRU, Attention, Transformer)
- ✓ 모델 훈련과정의 파라미터는 디폴트 값사용
- ✓ 모델 훈련 이후 모델 테스트 과정
- ✓ 분석 결과 가시화 등



✓ Keras를 이용한 MLP 모델

텍스트 데이터셋으로 이진 분류하기 (Spammail)

```
[1] import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.layers import Embedding, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
[2] docs = ['additional income',
            'best price',
            'big bucks',
            'cash bonus',
            'earn extra cash',
            'spring savings certificate',
            'valero gas marketing',
            'all domestic employees',
            'nominations for oct',
            'confirmation from spinner']

labels = np.array([1,1,1,1,1,0,0,0,0,0])
```

- one_hot() 함수는 해싱을 이용하여 정수를 생성한다.
- 해시 함수의 충돌 확률을 줄이기 위하여 필요한 것 보다 큰 어휘 크기 50 사용한다.

```
[3] vocab_size = 50  
    encoded_docs = [one_hot(d, vocab_size) for d in docs]  
    print(encoded_docs)
```

⇒ [[32, 27], [20, 23], [41, 31], [22, 23], [20, 27, 22], [6, 38, 22], [17, 39, 27], [16, 39, 34], [43, 21, 28], [39, 12, 22]]

- pad_sequences()함수로 입력 시퀀스의 길이, 즉 단어의 개수를 같게 만든다.
- 여기서는 충분한 단어 개수 (max_length)를 사용한다.

```
[4] max_length = 5  
    padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')  
    print(padded_docs)
```

⇒

[32	27	0	0	0]
[20	23	0	0	0]
[41	31	0	0	0]
[22	23	0	0	0]
[20	27	22	0	0]
[6	38	22	0	0]
[17	39	27	0	0]
[16	39	34	0	0]
[43	21	28	0	0]
[39	12	22	0	0]]

Embedding 임베딩 층에 대하여

- Embedding() 층에서 50개의 어휘이고, 각 문장의 입력 길이는 5이다.
- 각 단어를 임베딩하는 차원(즉, 임베딩 길이)은 10개를 사용하자.
- 한번에 입력에 사용되는 길이는 max_length로 5개 어휘(단어)이다.

✓
0초

```
[5] model = Sequential()  
    model.add(Embedding(vocab_size, 8, input_length=max_length))  
    model.add(Flatten())  
    model.add(Dense(1, activation='sigmoid'))
```

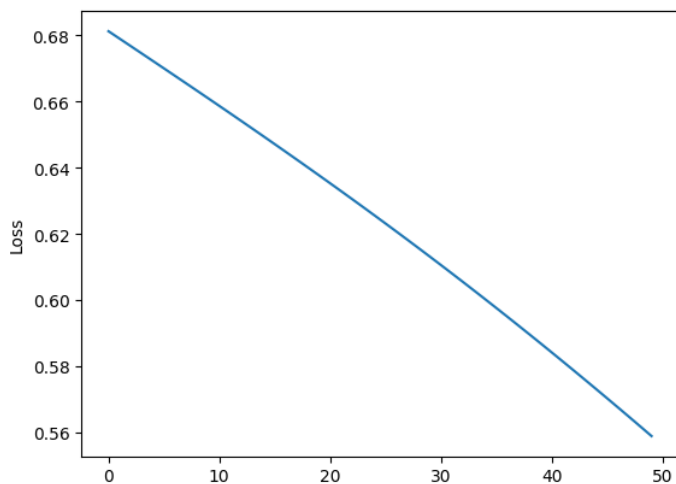
➡ /usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
warnings.warn(

✓
6초

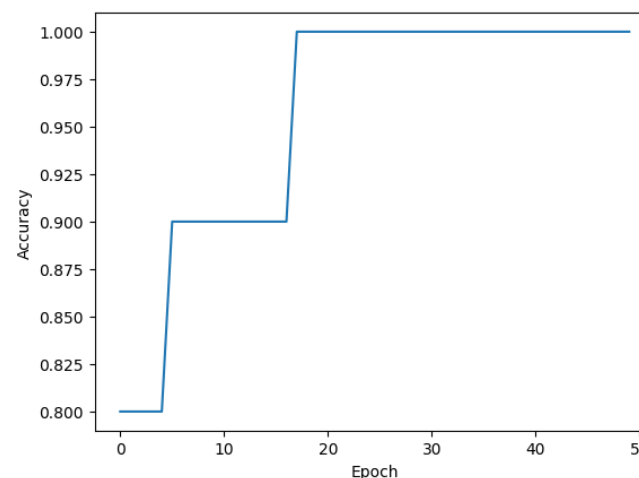
```
[6] model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  
    history= model.fit(padded_docs, labels, epochs=50, verbose=2)
```

➡ Epoch 1/50
1/1 - 2s - 2s/step - accuracy: 0.8000 - loss: 0.6813
Epoch 2/50
1/1 - 0s - 54ms/step - accuracy: 0.8000 - loss: 0.6790
Epoch 3/50
1/1 - 0s - 57ms/step - accuracy: 0.8000 - loss: 0.6768
Epoch 4/50


```
[7] plt.plot(history.history['loss'])
     plt.xlabel('Epoch')
     plt.ylabel('Loss')
     plt.show()
```



```
▶ plt.plot(history.history['accuracy'])
   plt.xlabel('Epoch')
   plt.ylabel('Accuracy')
   plt.show()
```



```
[9] test_doc = ['big income']
     encoded_docs = [one_hot(d, vocab_size) for d in test_doc]
     padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
```

```
print(model.predict(padded_docs))
```

1/1 ————— 0s 193ms/step
[[0.58923656]]

❖ 훈련과정의 입력 데이터를 크게 해보자

- ✓ 문장 임베딩 길이를 변화해보자.
 - `pad_sequences`의 `maxlen=5`에서 다른 숫자로 10, 20 등
- ✓ 각 단어의 임베딩 차원을 증가해보자.
 - `Embedding`의 차원 8에서 10, 20 등

❖ 딥러닝 MLP 모델의 분류 층을 증가시켜보다.

- ✓ `model.add(Dense(16, activation='relu'))`

```
model = Sequential()
model.add(Embedding(vocab_size, 64, input_length=max_length))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

감사합니다.

Korea Institute of Science
and Technology Information

TRUST
KISTIL

