ChatGPT와 함께하는 딥러닝 5일 완성

자연어처리(NLP) 및 감성 분석 실습

5일차

서울대학교 평생교육원
Extension College Seoul National University

Topics

- 자연어처리(NLP) 기본 개념 이해
- 토크나이징, 임베딩, 시퀀스 처리 원리 파악
- LSTM 모델 설계 및 감성 분류 실습
- 모델 평가와 실제 문장에 대한 감성 예측 수행
- 전이 학습 모델의 활용 방법 이해

Fine-Tuning Workflow

1. 기본 모델을 개체화하고 사전 훈련된 가중치를 로드
2. trainable = False를 설정하여 기본 모델의 모든 레이어를 동결
3. 기본 모델의 하나 또는 여러 레이어의 출력 위에 새 모델을 생성
4. 새로운 데이터 세트에 대한 새로운 모델을 훈련
5. 모델이 새 데이터에 수렴되면 기본 모델의 전체 또는 일부를 해제하고 매우 낮은 학습률로 전체 모델을 다시 훈련할 수 있음

특성 추출기만 가져오기

- Instantiate a base model and load pre-trained weights into it. 기본 모델을 개체화하고 사전 훈련된 가중치를 로드

```
from tensorflow.keras.applications import *

base_model = VGG16(weights='imagenet’,
                   input_shape=(240, 240, 3),
                   include_top=False) # 특성추출기만 사용
base_model.summary()
```

특성 추출기, 분류기 가져오기

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
base_model = InceptionV3(input_shape=(150, 150, 3),
                         include_top=True, #사전학습 모델의 특성추출기와
                                             분류기 둘다 가져옴
                         weights=None) #적용할 가중치 지정
base_model.summary()
```

Weights

- None: 임의로 초기화 (random initialization)된 가중치를 적용
- imagenet: ImageNet에 대해 사전 훈련된 가중치를 적용 (Default 값)

Base Model 구조 확인

```
from tensorflow.keras.applications import *
```

```
base_model = VGG16(weights='imagenet',
                   input_shape=(240, 240, 3),
                   include_top=False) # 특성추출기만 가져옴
```

```
base_model.input # 입력 텐서
```

```
<KerasTensor: shape=(None, 240, 240, 3) dtype=float32 (created by layer 'input_1')>
```

```
base_model.output # 출력 텐서
```

```
<KerasTensor: shape=(None, 7, 7, 512) dtype=float32 (created by layer 'block5_pool')>
```

```
print(base_model.layers[-1]) # 마지막 레이어
```

```
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D at 0x7802582870a0>
```

```
print(base_model.layers.pop()) #마지막 레이어 제거
```

```
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D at 0x7802582870a0>
```

전이학습 모델 위에 새로운 분류기를 추가

- Create a new model on top of the output of one (or several) layers from the base model. 기본 모델의 하나 또는 여러 레이어의 출력 위에 새 모델을 생성

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Drop
out
from keras.optimizers import Adam

model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=Adam(learnin
g_rate=1e-4),metrics=['accuracy'])
model.summary()
```

Fine-tuning시 학습률을 낮게 설정해 pre-trained weights를 조금씩 업데이트 해줌

모델 컴파일

- Train your new model on your new dataset. 새로운 데이터 세트에 대한 새로운 모델을 훈련

```
# 모델 컴파일
model.compile(optimizer = 'adam',
                    loss = 'sparse_categorical_crossentropy',
                    metrics=['accuracy’])
```

레이어 동결

- 훈련하기 전 base model 의 가중치가 바뀌는 것을 막기 위해 이를 동결

- 베이스 모델 전체를 동결하는 경우

```
base_model.trainable=False

# 훈련 가능한 가중치의 갯수 확인
len(model.trainable_weights) #4
```

trainable = False를 설정하여 기본 모델의 모든 레이어를 동결

- 각 레이어를 순차적으로 동결하는 경우

```
for layer in base_model.layers:
  layer.trainable = False
```

가중치 저장 및 조기종료 설정

```
from keras.callbacks import ModelCheckpoint, EarlyStopping

save_file_name = 'cifar10_transfer_model.h5'
checkpoint = ModelCheckpoint(save_file_name, monitor='val_loss',
                             verbose=1, save_best_only=True,
                             mode='auto')
earlystopping = EarlyStopping(monitor='val_loss', patience=5)
```

모델 훈련

- Train your new model on your new dataset. 새로운 데이터 세트에 대한 새로운 모델을 훈련

```
# 모델 훈련
history = model.fit(x_train, y_train, epochs = 100, validation_data=(x_tes
t, y_test), batch_size= 256, callbacks=[checkpoint, earlystopping])
model.save("cifar10_resnet_model.h5")
```

동결 해제

- Once your model has converged on the new data, you can try to unfreeze all or part of the base model and retrain the whole model end-to-end with a very low learning rate. 모델이 새 데이터에 수렴되면 기본 모델의 전체 또는 일부를 해제하고 매우 낮은 학습률로 전체 모델을 다시 훈련할 수 있음

```
# 모든 층의 동결 해제
base_model.trainable = True

# 훈련할 수 있는 가중치 갯수
len(model.trainable_weights) #30

# 일부 레이어 해제
for layer in base_model.layers[143:]:
  layer.trainable = True
```

전이학습 모델 튜닝기법

- 옵티마이저 변경
- BatchNormalization 추가
- Dropout 추가
- 사전학습 모델 전체 학습
  - base_model.trainable = True
- 데이터 전처리 및 증강
- 분류기에 은닉층 및 노드 추가
- Learning decay를 이용한 유동적인 학습 진행

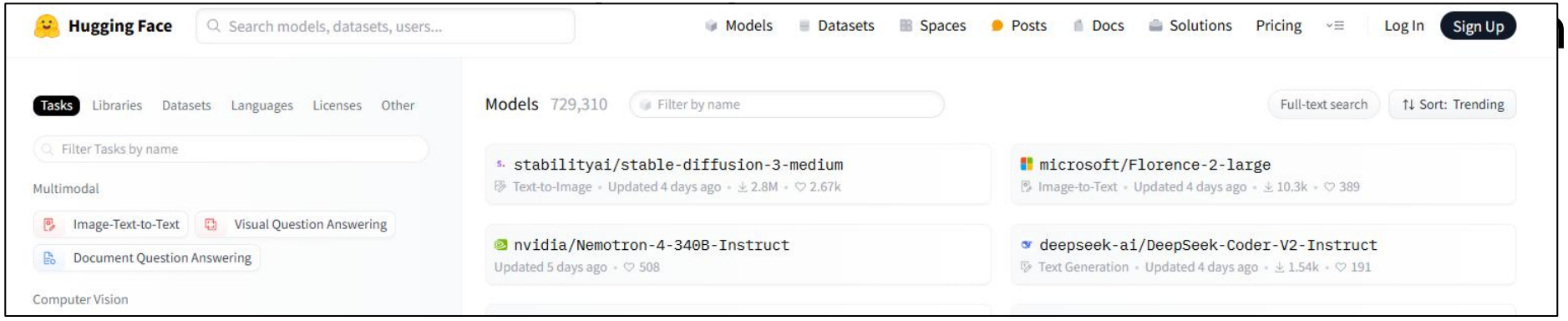
Learning decay:
- 학습률(learning rate)을 점진적으로 감소
- 초반엔 빠르게 학습. 후반엔 작게 조정하여 안정적으로 수렴

HuggingFace Hub

Hugging Face

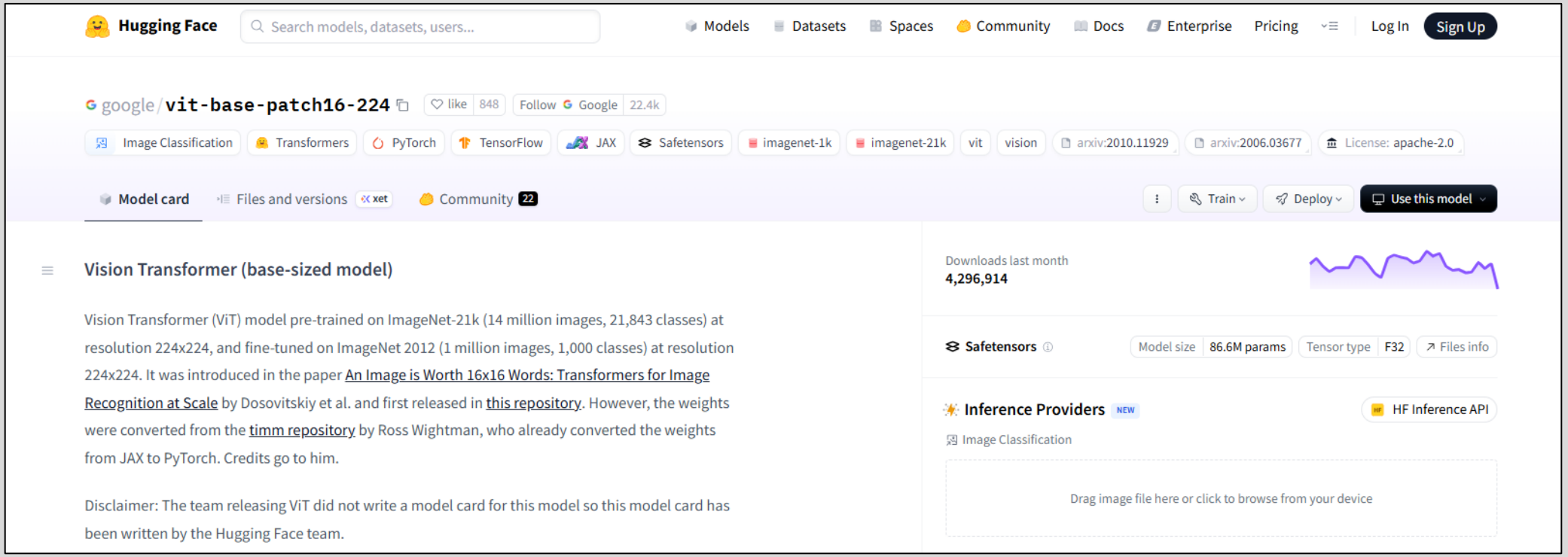
https://huggingface.co/

- A open-source platform that enables anyone to discover, use, and contribute new state-of-the-art models and datasets. 누구나 새로운 최첨단 모델과 데이터 세트를 검색, 사용 및 기여할 수 있는 오픈 소스 플랫폼
- Hosts a wide variety of models, with more than 10,000 publicly available. 10,000개 이상의 공개적으로 사용할 수 있는 다양한 모델을 호스팅
- Utilized for various tasks such as translation, automatic ...ore.



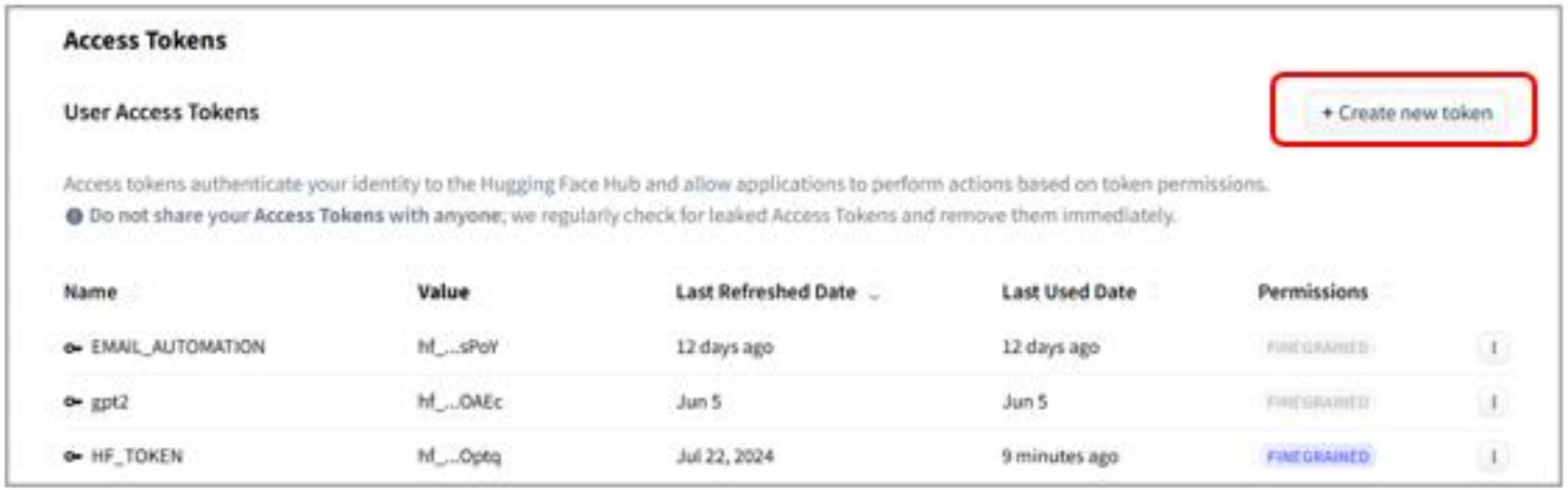
Exercise #1

- 허깅페이스의 사전학습된 비전 모델 (ViT 모델)을 사용하여 이미지를 분류하시오
  - google/vit-base-patch16-224 (Vision Transformer 모델)



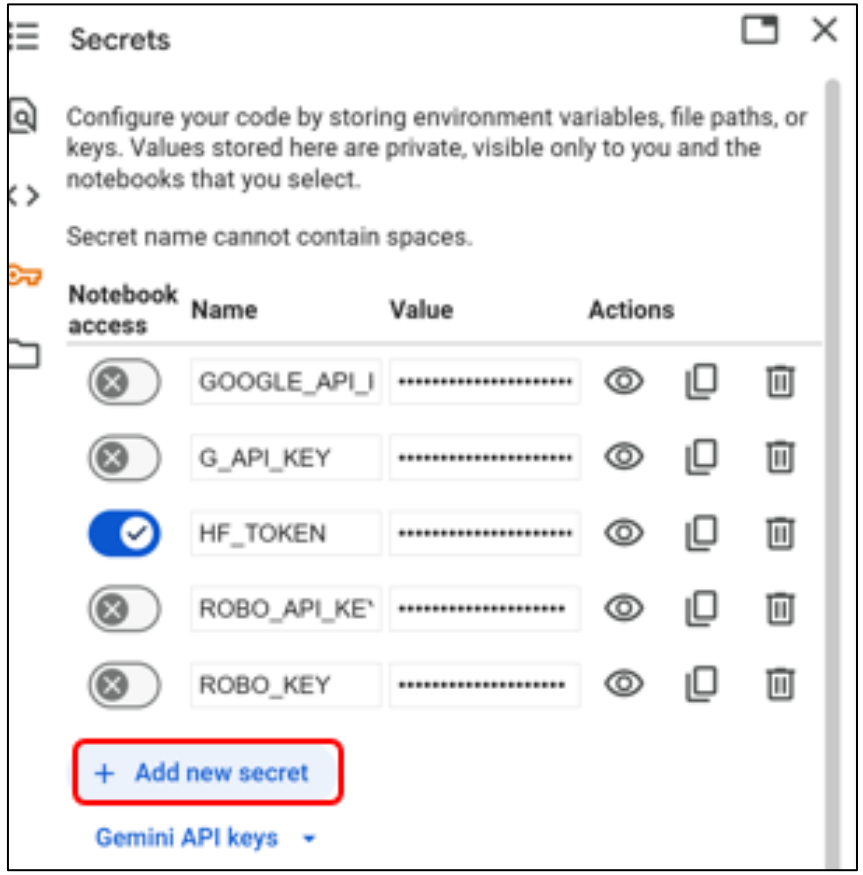
Hugging Face 가입 및 API 키 발급

- 허킹페이스 프로필에 가서 Access Tokens을 선택한 뒤 Create new token을 클릭하고
  새로운 토큰을 만듬. 예) HF_TOKEN



코랩에서 허깅페이스 API 키 설정

- 코랩에서 왼쪽창을 열고 Add new secret에 HF_TOKEN을 등록



허깅페이스 모델 활용 이미지 분류

```
# 라이브러리 설치
!pip install transformers datasets torch torchvision pillow

# 라이브러리 임포트
from transformers import ViTImageProcessor, ViTForImageClassification
from PIL import Image
import torch
import requests
```

허깅페이스 모델 활용 이미지 분류

```
# 이미지 불러오기
image = Image.open('/content/cat.jpg')
```

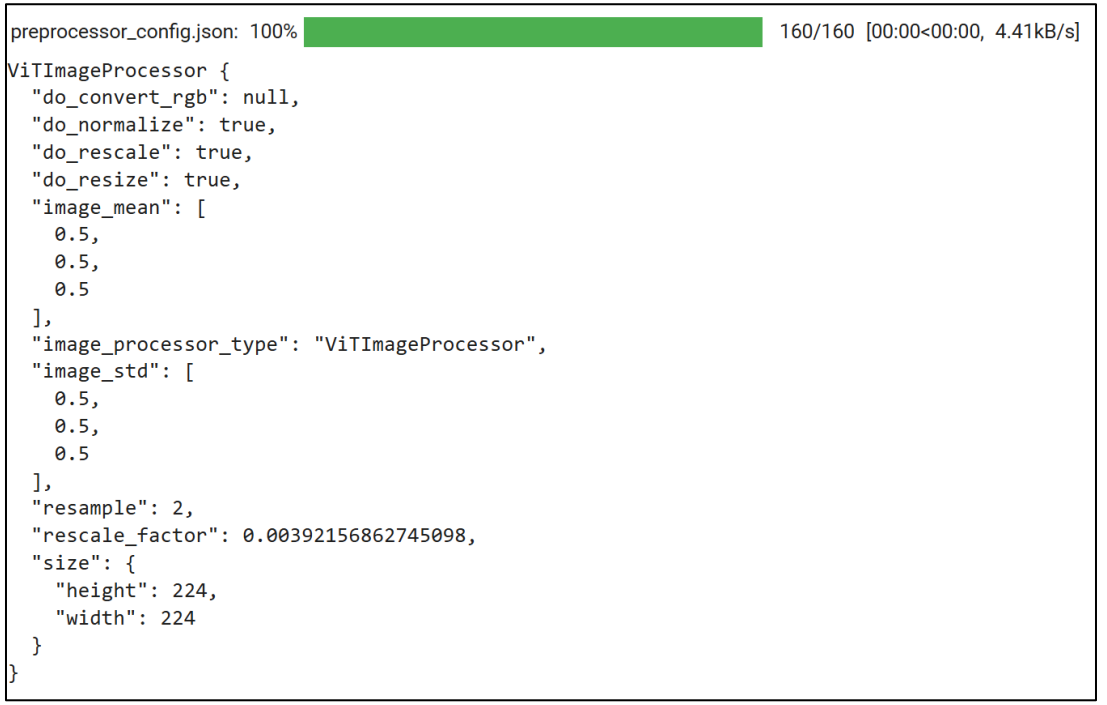
허깅페이스 모델 활용 이미지 분류

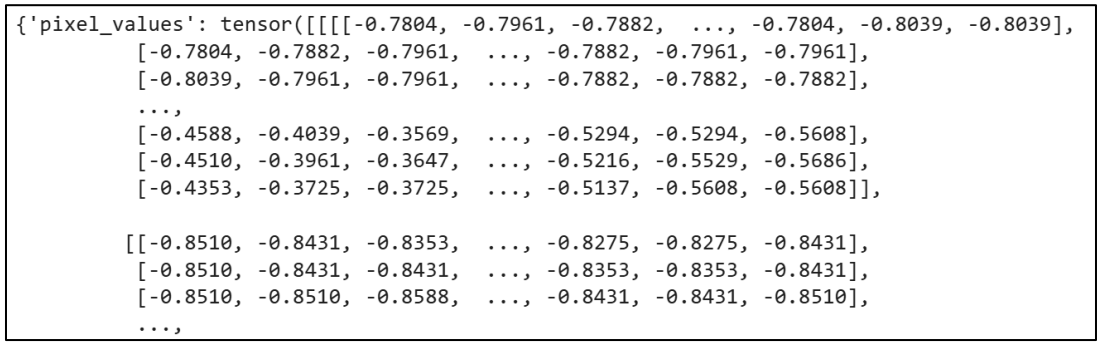
```
# 이미지 전처리
import numpy as np

image = np.array(image)
image.shape
image = image.reshape(1,2304,3456,3)
```

```
array([[[[27, 22, 19],
         [29, 24, 21],
         [28, 23, 19],
         ...,
         [21, 17, 16],
         [21, 17, 16],
         [21, 17, 16]],

        [[26, 21, 18],
         [28, 23, 20],
         [28, 23, 19],
         ...,
         [23, 19, 18],
         [24, 20, 19],
         [24, 20, 19]],

        [[25, 20, 17],
         [27, 22, 19],
         [28, 23, 19],
         ...,
         [25, 21, 20],
         [26, 22, 21],
         [26, 22, 21]],

        ...,
```

허깅페이스 모델 활용 이미지 분류

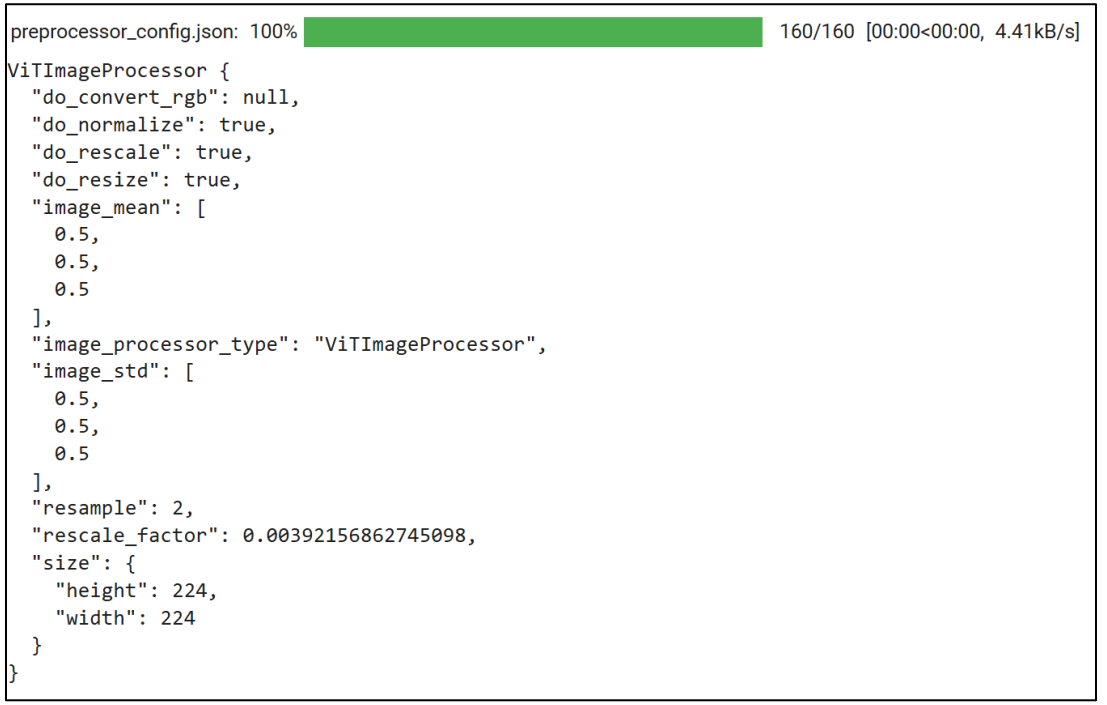
```
# 전처리 도구 로드
processor = ViTImageProcessor.from_pretrained('google/vit-base-patch16-224’)

# 이미지 전처리
inputs = processor(images=image, return_tensors="pt") # 파이토치 텐서 형식으로 반환
```

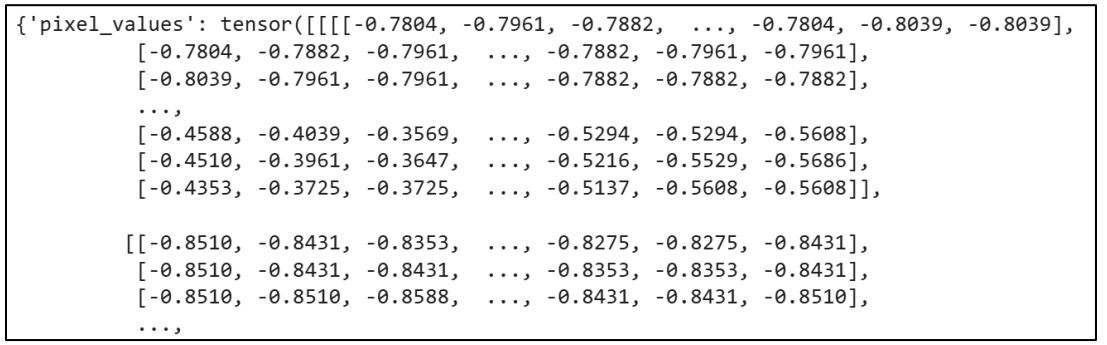




허깅페이스 모델 활용 이미지 분류

```
# 전처리 도구 로드
processor = ViTImageProcessor.from_pretrained('google/vit-base-patch16-224’)

# 이미지 전처리
inputs = processor(images=image, return_tensors="pt") # 파이토치 텐서 형식으로 반환
```





허깅페이스 모델 활용 이미지 분류

```
# 모델 로드
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224’)
```


```
# 모델 추론
with torch.no_grad(): # radient 계산 비활성화
    outputs = model(**inputs) # 입력에 대한 결과값
    logits = outputs.logits # 클래스에 대한 점수
    predicted_class_idx = logits.argmax(-1).item() #가장 높은 점수를 가진 클래스 인덱스 선택
```

허깅페이스 모델 활용 이미지 분류

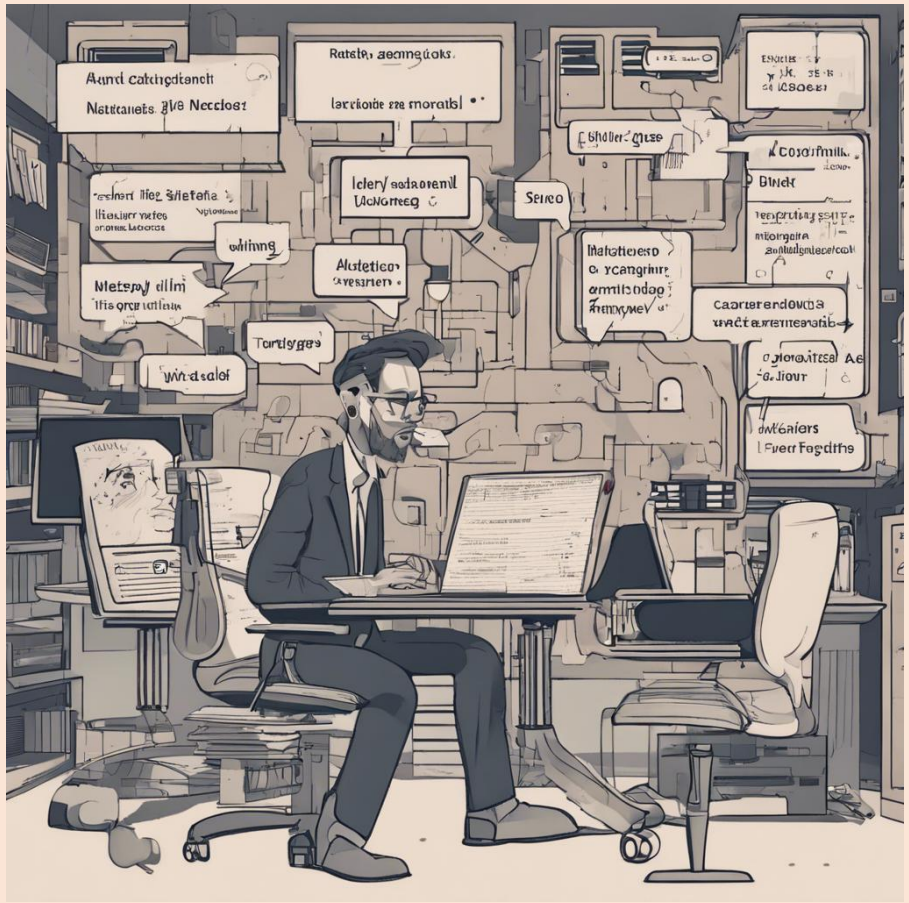
```
# 예측 결과 출력
print(f"예측된 클래스: {model.config.id2label[predicted_class_idx]}") # 사람이 읽을 수 있는 라벨 변환
```

```
예측된 클래스: tabby, tabby cat
```

자연어 처리

Natural Language Processing

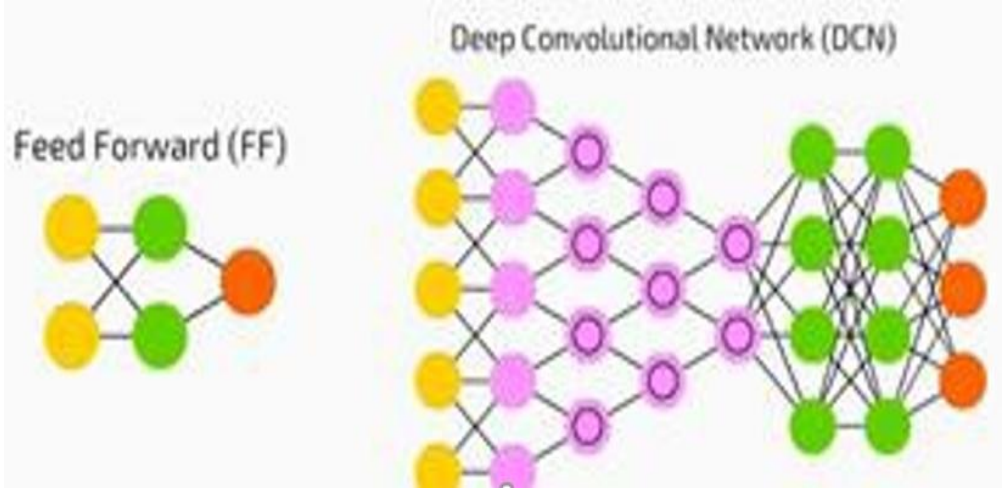
- 순환 신경망(RNN, Recurrent Neural Network)과 변환기(Transformer)와 같은 모델을 사용하여 자연스러운 문장을 생성하고 번역
- 기계 번역, 대화형 챗봇, 텍스트 요약 등에 사용



https://limewire.com/

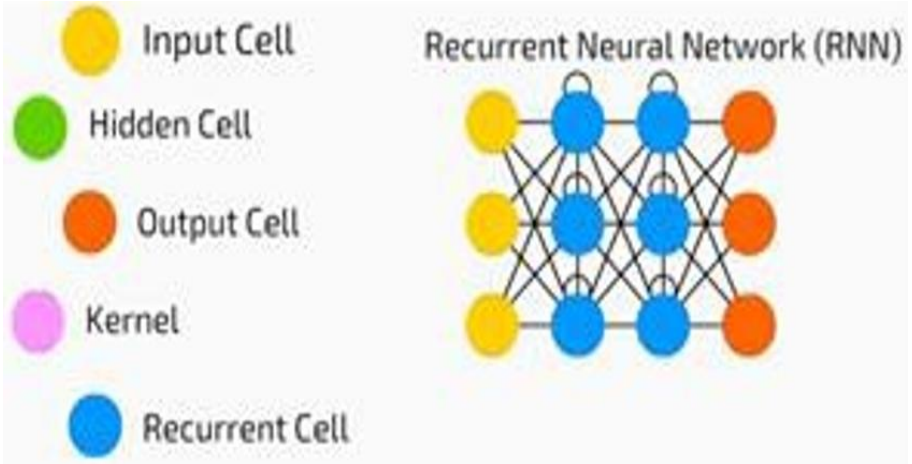
이미지 분류 모델 및 언어 모델

- Convolutional Neural Network
  합성곱 신경망



- Uses convolutional neural networks for image processing.
  이미지 처리를 위해 컨볼루셔널 신경망을 사용

- Recurrent Neural Network
  순환 인공 신경망

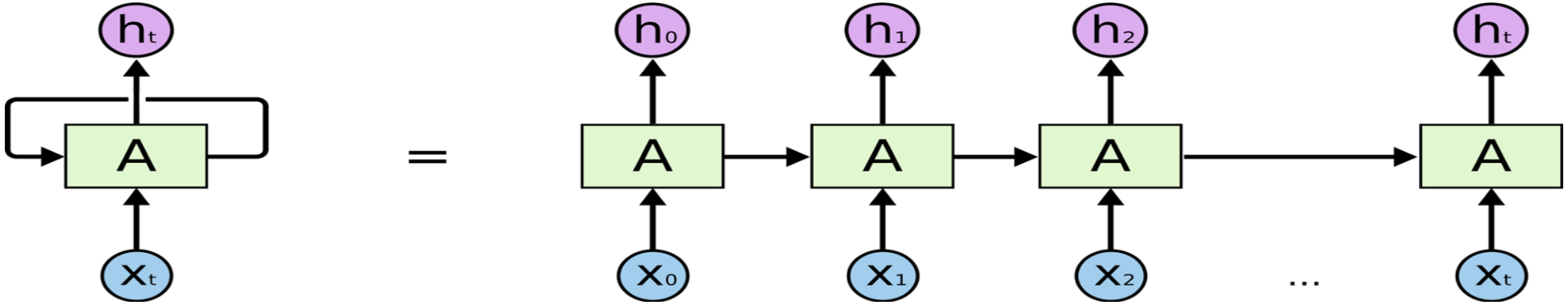


- Uses recurrent neural networks for natural language processing.
  자연어 처리를 위해 순환 신경망을 사용

순환 인공 신경망

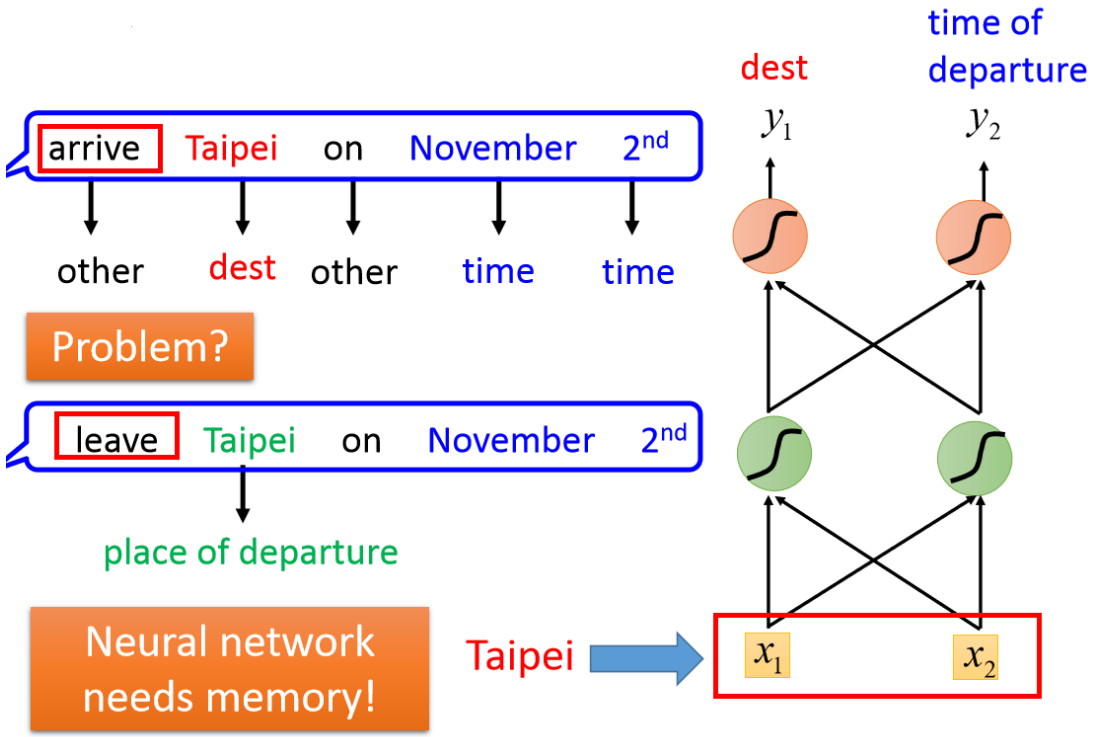
RNN (Recurrent Neural Network)

- A type of Neural Network where the output from previous step are fed as input to the current step. 신경망의 종류로 이전스텝의 결과물이 현재 스텝의 입력값이 됨
- Language modelling or Natural Language Processing (NLP). 자연어처리
- A most powerful models for processing sequential data. 연속된 데이터를 처리하기 가장 좋은 모델



메모리
Memory

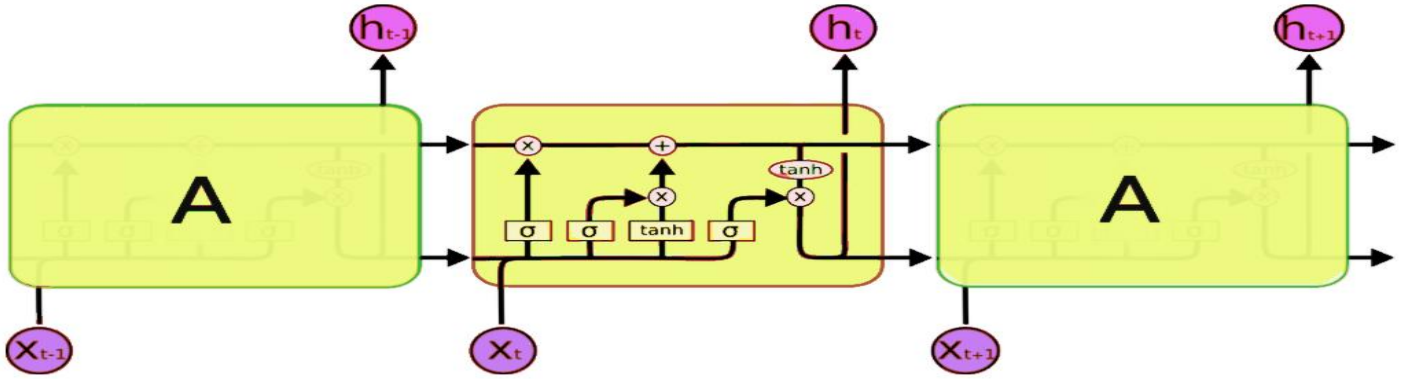
- RNN has a “memory” which captures information about what has been calculated so far. 메모리를 이용하여 지금까지 계산한 결과를 저장해서 예측에 사용



장기단기기억

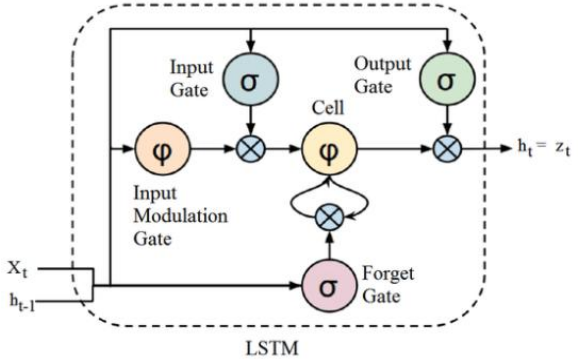
LSTM (Long Short Term Memory)

- A type of RNN that introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network. RNN의 한종류로 은닉층 안에 있는 뉴런 대신 기억셀을 소개
- Solved the problem of long-term dependencies in a sequence. 시퀀스에서 장기의존성 문제를 해결



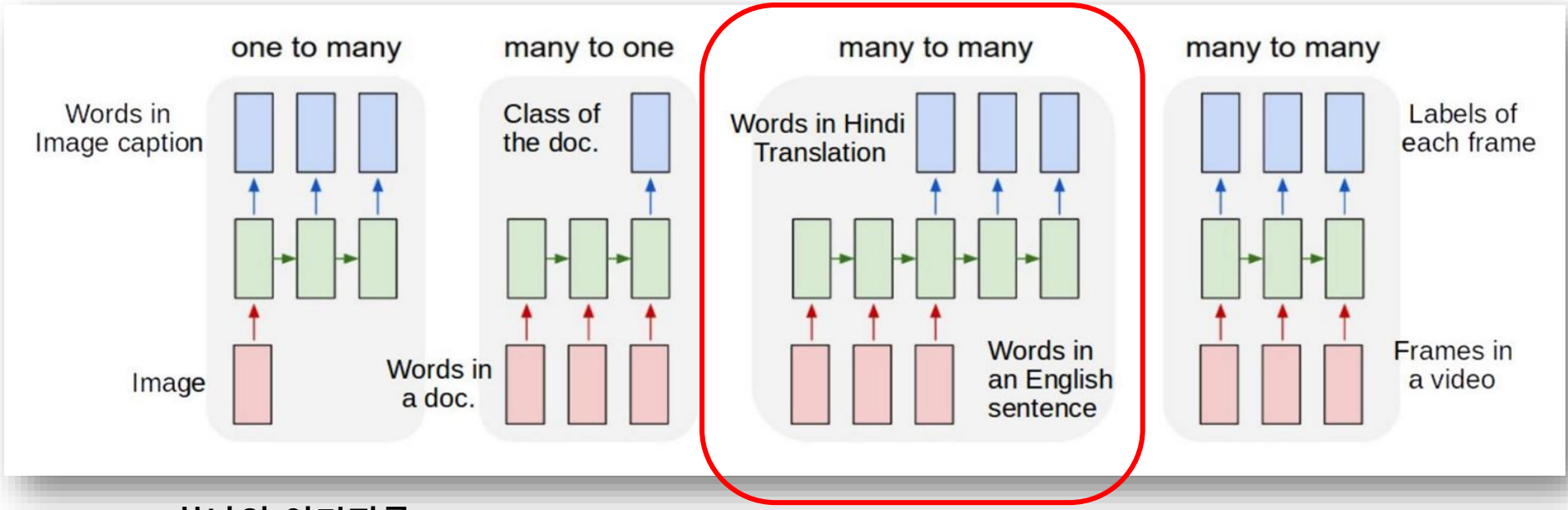
기억게이트
Forget Gate (Remember Vector)

- The output of the forget gate tells the cell state which information to forget by multiplying 0. 0을 곱해서 기억을 지우거나..
- If the output of the forget gate is 1, the information is kept in the cell state. 1을 곱해서 기억을 남기던가..
- LSTM uses a set of gates to control the flow of information. 게이트를 이용하여 정보의 흐름을 조절함



RNN 응용분야

- Used for language modeling and prediction, speech recognition, machine translation, image recognition and characterization. 랭귀지 예측, 언어인식, 기계통역, 이미지 인식에 활용



하나의 이미지를 입력한후 그에 매치되는 단어나 문장으로 출력

단어시퀀스나 문장을 입력하고 정서값을 출력

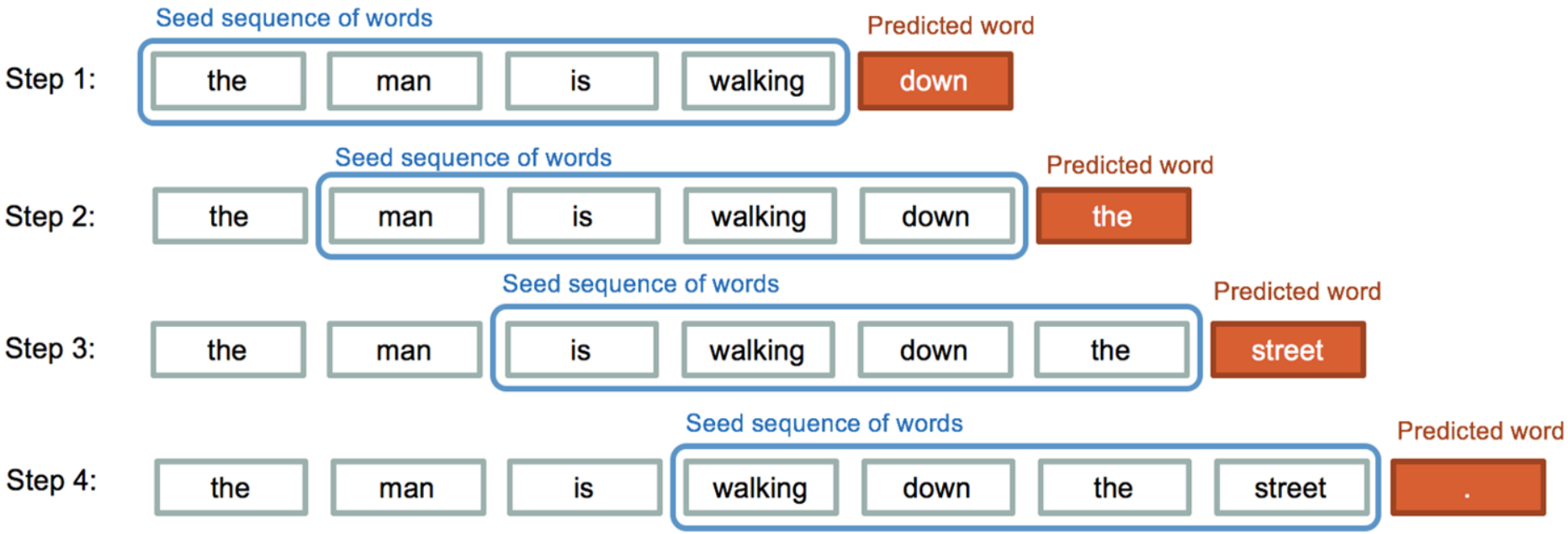
영어문장을 입력해서 다른 문장으로 출력하는 기계번역에 사용

비디오를 프레임단위로 나눠 분류
시계열예측

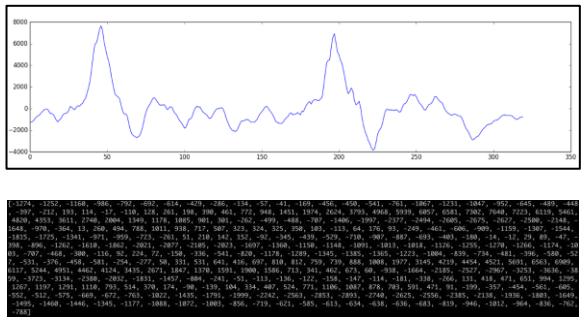
텍스트 생성

Text Generation

- For example, if you want to predict the next word in a sentence, it is better know which words came before that word. 문장에서 다음 단어를 예측할 때 그전에 오는 단어를 알면 더 잘 예측할 수 있음



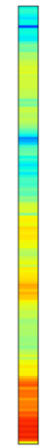
언어인식
Speech Recognition



Input

Stateful Model

Output



20ms slice of audio

Recurrent Neural Network

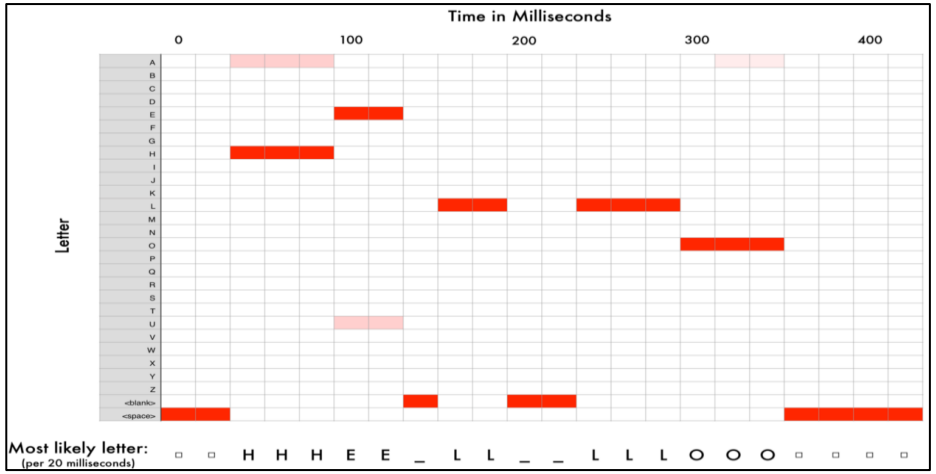
Likelihood saying 'A'

Likelihood saying 'B'

Likelihood saying 'C'

And so on...

The model's current state influences the next calculation.



기계번역
Machine Translation

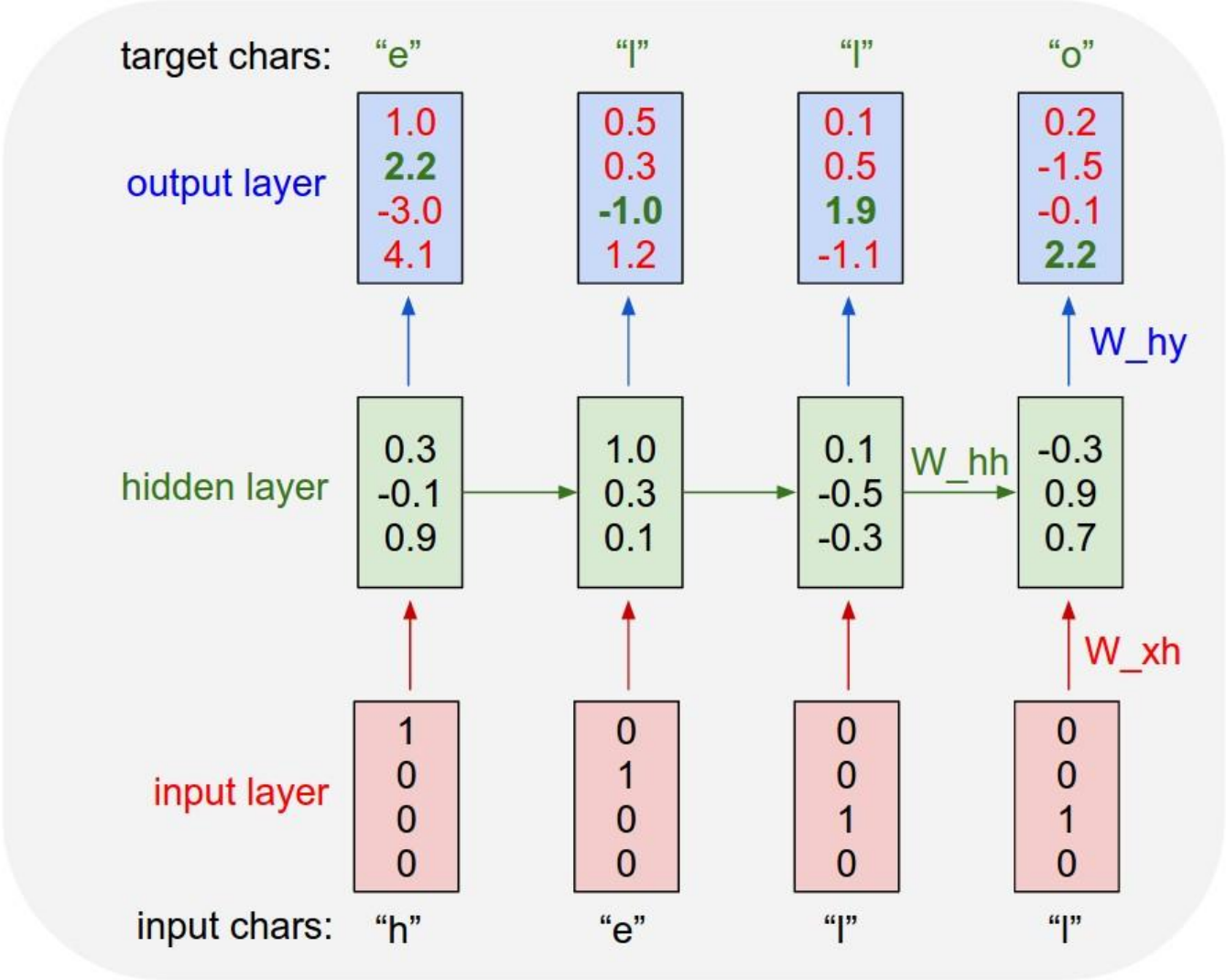
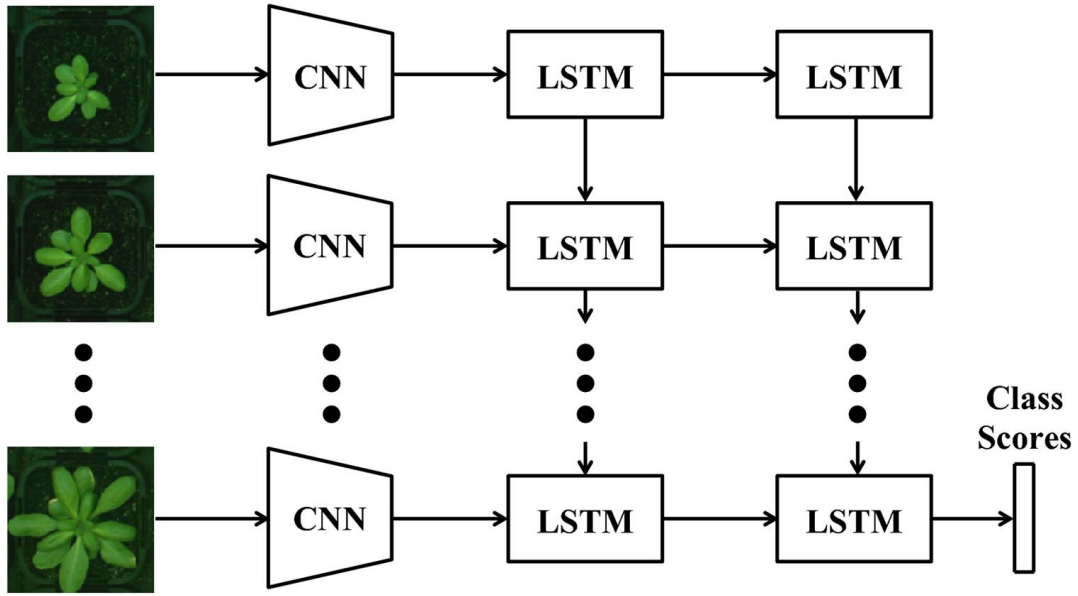


Image Recognition and Characterization 이미지 인식과 글짜



CNN LSTM Model

- One of many to many examples. 다대다 예제 중 하나
- Enter unstructured data by converting it to a number first. 비구조화된 데이터를 먼저 숫자로 바꿔 입력
- Video Frame Classification. 비디오 프레임 분류

머신러닝/딥러닝 기반 텍스트분석

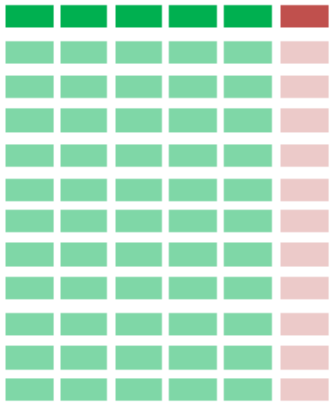
규칙/어휘 기반 감성분석

단어빈도 감성사전

단어	감성점수	빈도수	합계
Happy	1	3	+3
Sad	-1	2	-2
Great	1	2	2
Ugly	-1	1	-1
합계			+2

M/L 기반 감성분석

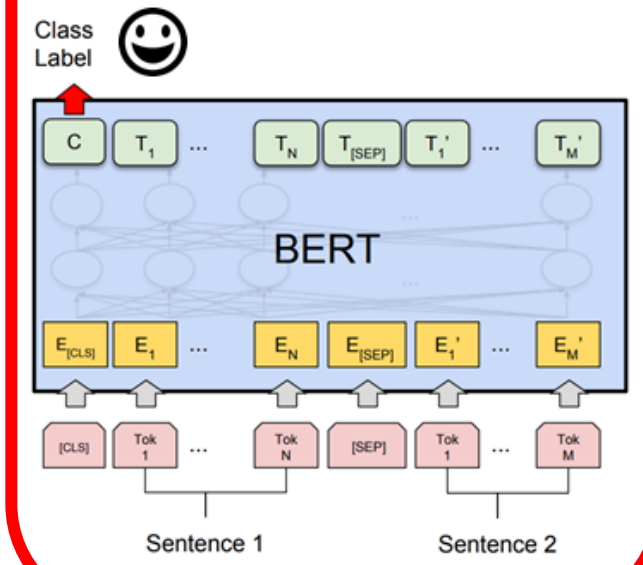
감성 라벨



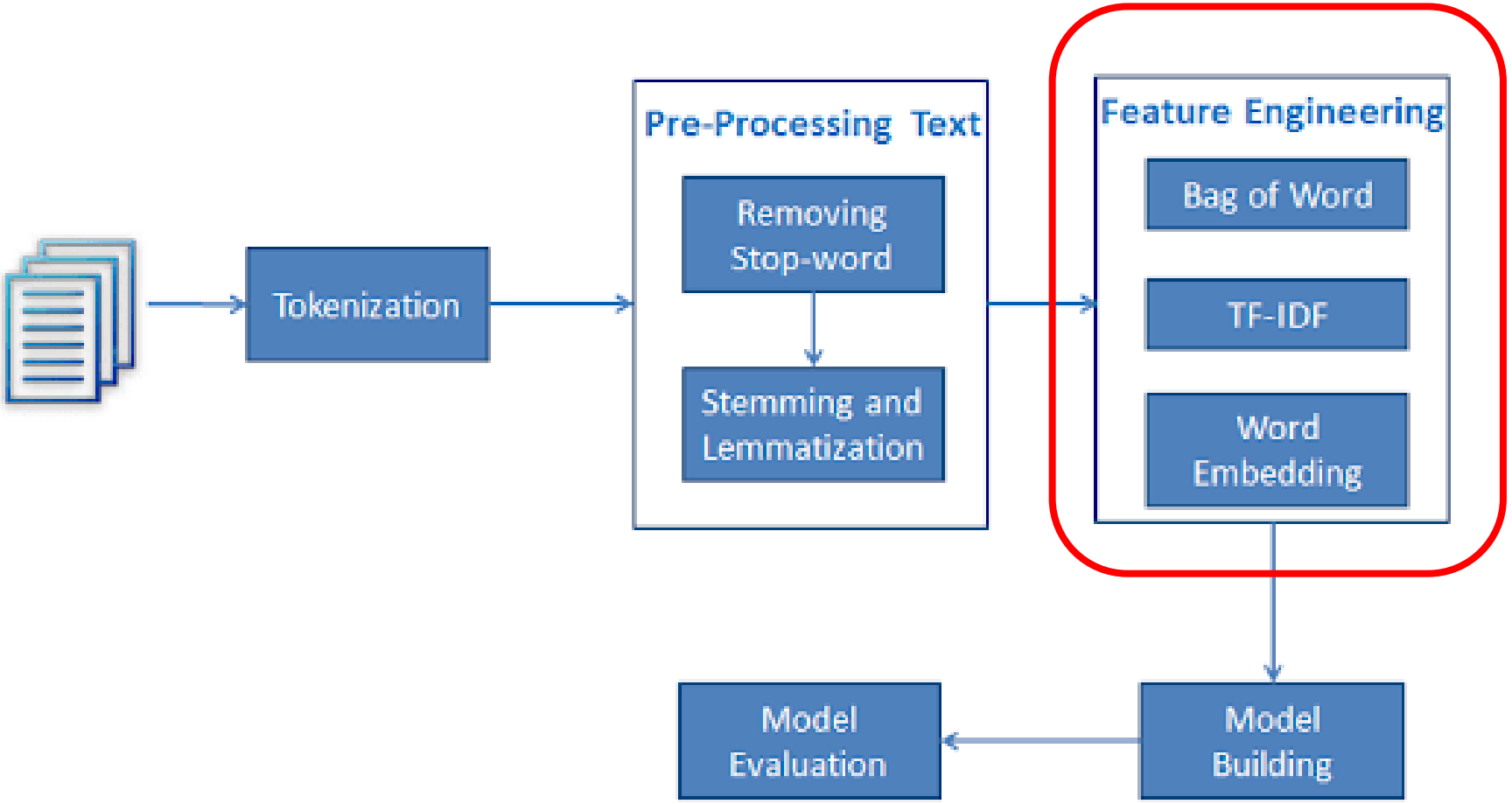
IF-IDF

$$y = f(x)$$

딥러닝 기반 감성분석



Feature Engineering 피쳐 엔지니어링



Word Embedding 워드임베딩

- The collective name for a set of language modeling and feature learning techniques where words or phrases from the vocabulary are mapped to vectors of real numbers.
  단어나 구절을 숫자의 벡터로 바꾸는 방법을 총칭
- The technique is primarily used with Neural Network Models.
  신경망과 함께 사용됨

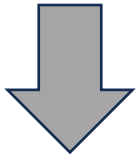
Bag-of-Words (BoW) 백오브워즈

- A way to represent the data in a tabular format with columns representing the total vocabulary of the corpus and each row representing a single observation. The cell (intersection of the row and column) represents the count of the word represented by the column in that particular observation. 단어사전과 비교해서 단어가 몇번 나타났는지 테이블로 정리

	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

인코딩할 텍스트

```
text = ['This is the first document.',
          'This document is the second
document.',
          'And this is the third one.',
          'Is this the first document?’]
```



	and	document	first	is	one	second	the	third	this
0	0	1	1	1	0	0	1	0	1
1	0	2	0	1	0	1	1	0	1
2	1	0	0	1	1	0	1	1	1
3	0	1	1	1	0	0	1	0	1

CountVectorizer

```
from sklearn.feature_extraction.text import
CountVectorizer

cv = CountVectorizer()

x = cv.fit_transform(text)

print(cv.get_feature_names_out()) #단어리스트

print(x.toarray()) #인코딩한 내용 출력

x_df = pd.DataFrame(x.toarray(),
columns=cv.get_feature_names_out())

x_df.head(10)
```

N-Grams 앤그램

- The process of combining the nearby words together for representation purposes where N represents the number of words to be combined together. 옆에 있는 단어들을 함께 묶어주는 과정. N은 합칠 단어의 수

- Example of “Natural Language Processing is essential to Computer Science.”
  - 1-gram or unigram: “Natural, Language, Processing, is, essential, to, Computer, Science”
  - Bigram: “Natural Language, Language Processing, Processing is, is essential, essential to, to Computer, Computer Science”
  - Trigram: “Natural Language Processing, Language Processing is, Processing is essential, is essential to, essential to Computer, to Computer Science”

N-Gram 코드

```
cv2 = CountVectorizer(analyzer='word', ngram_range=(2, 2)) #word n-gram
x2 = cv2.fit_transform(text)
print(cv2.get_feature_names_out())
print(x2.toarray())
x2_df = pd.DataFrame(x2.toarray(), columns=cv2.get_feature_names_out())
x2_df.head(10)
```

	and this	document is	first document	is the	is this	second document	the first	the second	the third	third one	this document	this is	this the
0	0	0	1	1	0	0	1	0	0	0	0	1	0
1	0	1	0	1	0	1	0	1	0	0	1	0	0
2	1	0	0	1	0	0	0	0	1	1	0	1	0
3	0	0	1	0	1	0	1	0	0	0	0	0	1

TF-IDF 티에프 아이디에프

- A way of scoring the vocabulary so as to provide adequate weight to a word in proportion of the impact it has on the meaning of a sentence. 문장에서의 의미에 따라 단어에 가중치를 주는 방법

TF-IDF Formula 공식

- The score is a product of 2 independent scores, term frequency(tf) and inverse document frequency (idf). TF와 IDF를 곱한 값

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

- Log (N/d) where, N is total number of documents and d is the number of documents in which the word appears. 전체문서에서 그 단어를 포함하는 문서의 수

TF and IDF 용어빈도, 역서류빈도

- Term Frequency (TF): frequency of word in the current document. 현재 문장에서 단어의 빈도수
- Inverse Document Frequency(IDF): A measure of how much information the word provides, i.e., if it's common or rare across all documents. 단어가 제공하는 정보의 양을 측정

sentence 1	earth is the third planet from the sun
sentence 2	Jupiter is the largest planet

Word	TF (Sentence 1)	TF (Sentence 2)	IDF	TF*IDF (sentence 1)	TF*IDF (Sentence 2)
earth	1/8	0	log(2/1)=0	0.0375	0
is	1/8	1/5	log(2/2)=0	0	0
the	2/8	1/5	log(2/2)=0	0	0
third	1/8	0	log(2/1)=0.3	0.0375	0
planet	1/8	1/5	log(2/2)=0	0	0
from	0	0	log(2/1)=0.3	0	0
sun	1/8	0	log(2/1)=0.3	0.0375	0
largest	0	1/5	log(2/1)=0.3	0	0.06
Jupiter	0	1/5	log(2/1)=0.3	0	0.06

TF-IDF 코드

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
x3 = tfidf.fit_transform(text)
print(tfidf.get_feature_names_out())
print(x3.toarray())
x3_df = pd.DataFrame(x3.toarray(),
columns=tfidf.get_feature_names_out())
x3_df.head(10)
```

	and	document	first	is	one	second	the	third	this
0	0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085
1	0.000000	0.687624	0.000000	0.281089	0.000000	0.538648	0.281089	0.000000	0.281089
2	0.511849	0.000000	0.000000	0.267104	0.511849	0.000000	0.267104	0.511849	0.267104
3	0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085

RegexpTokenizer 정규식 토크나이저

```
from nltk.tokenize import RegexpTokenizer
sentence  = "Think and wonder, wonder and think."
token = RegexpTokenizer(r"\w+")
words = token.tokenize(sentence)
#similar to words = re.split('\W+', sentence)
print(words)
```

```
['Think', 'and', 'wonder', 'wonder', 'and', 'think']
```

Exercise #2 Sentiment Analysis Example
감성분석 예제

- Read sent_train.tsv 훈련데이터 읽기 (PhraseId를 인덱스로!)

- Summarize the data frame. 데이터프레임 요약
  - head(), shape, dtypes, isna().sum(), describe(), groupby().size()

- Data visualization. 데이터시각화
  - Countplot. 감정에 따른 카운트 막대그래프

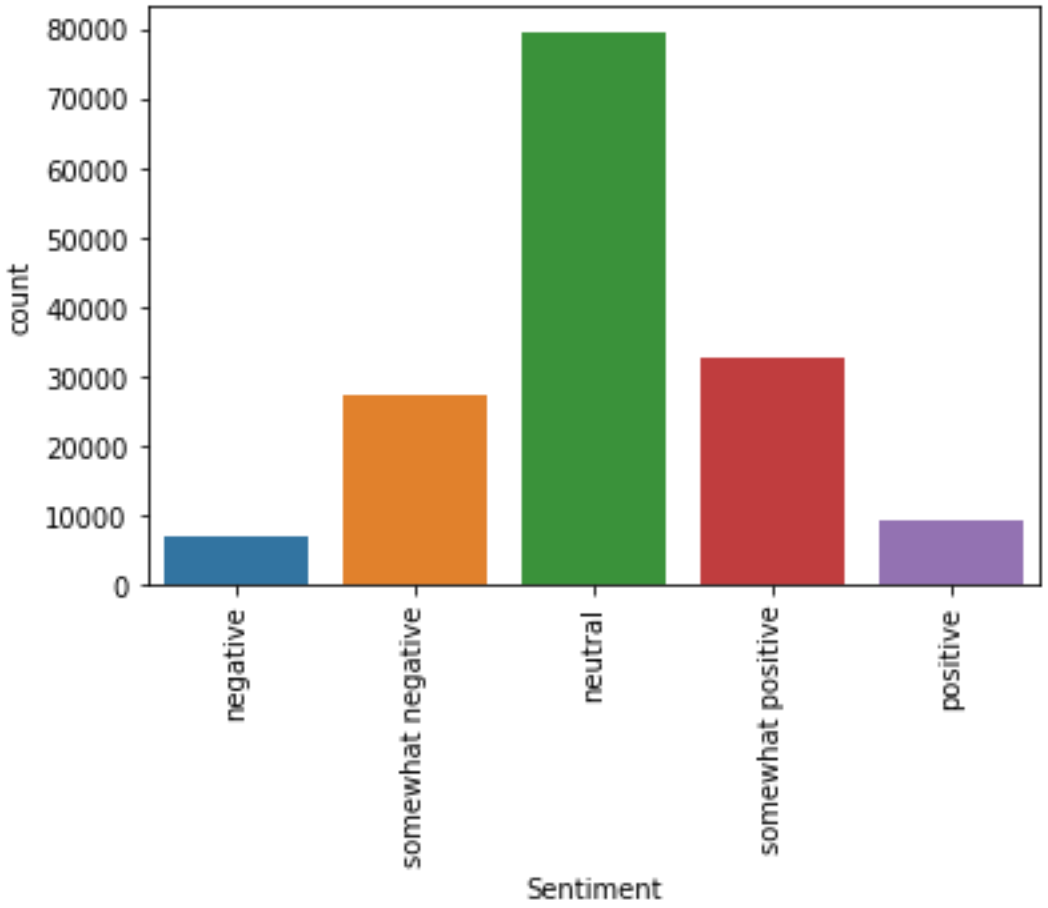
Read train.tsv 훈련데이터 읽기

data - DataFrame

Index	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A serie…	1
1	2	1	A serie…	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2
5	6	1	of esca…	2
6	7	1	of	2
7	8	1	escapad…	2
8	9	1	escapad…	2
9	10	1	demonst…	2
10	11	1	demonst…	2

```
data=pd.read_csv('sent_train.tsv
', sep='\t’)
data.set_index('PhraseId',
inplace=True)
data.head()
data.shape
data.dtypes
data.isna().sum()
data.describe()
```

Count phrase by sentiments and draw bar chart
감정에 따라 카운트, 막대그래프




```
sns.countplot(data.Sentiment)
plt.xticks([0,1,2,3,4],['negative', 'somewhat negative', 'neutral', 'somewhat positive', 'positive'], rotation=90)
plt.show()
```

Exercise #2
Sentiment Analysis Example 감성분석 예제

- Use CountVectorizer or TF-IDF to generate bag of words. 단어를 벡터화
- Remove stop words and punctuations. 불용어 및 문장기호 제거 (시간관계상 data를 1000개로 줄여서 진행)
- Train and test split (.3 test set). 30프로의 테스트셋으로 분리

카운트 벡터라이저

```
token = RegexpTokenizer(r'[a-zA-Z0-9]+') #알파벳, 숫자
cv = CountVectorizer(lowercase=True,#소문자
stop_words='english', #영어불용어
ngram_range = (1,1), #1-gram
tokenizer = token.tokenize) #알파벳, 숫자만 잘라서 인코딩
text_counts= cv.fit_transform(data['Phrase'])
```

	7	absolute	action	actor	actors	actually	adage	age	aggressive	aims	...	windtalkers
0	0	0	0	0	0	0	1	0	0	0	...	0
1	0	0	0	0	0	0	1	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	0	...	0

TFIDF 벡터라이저

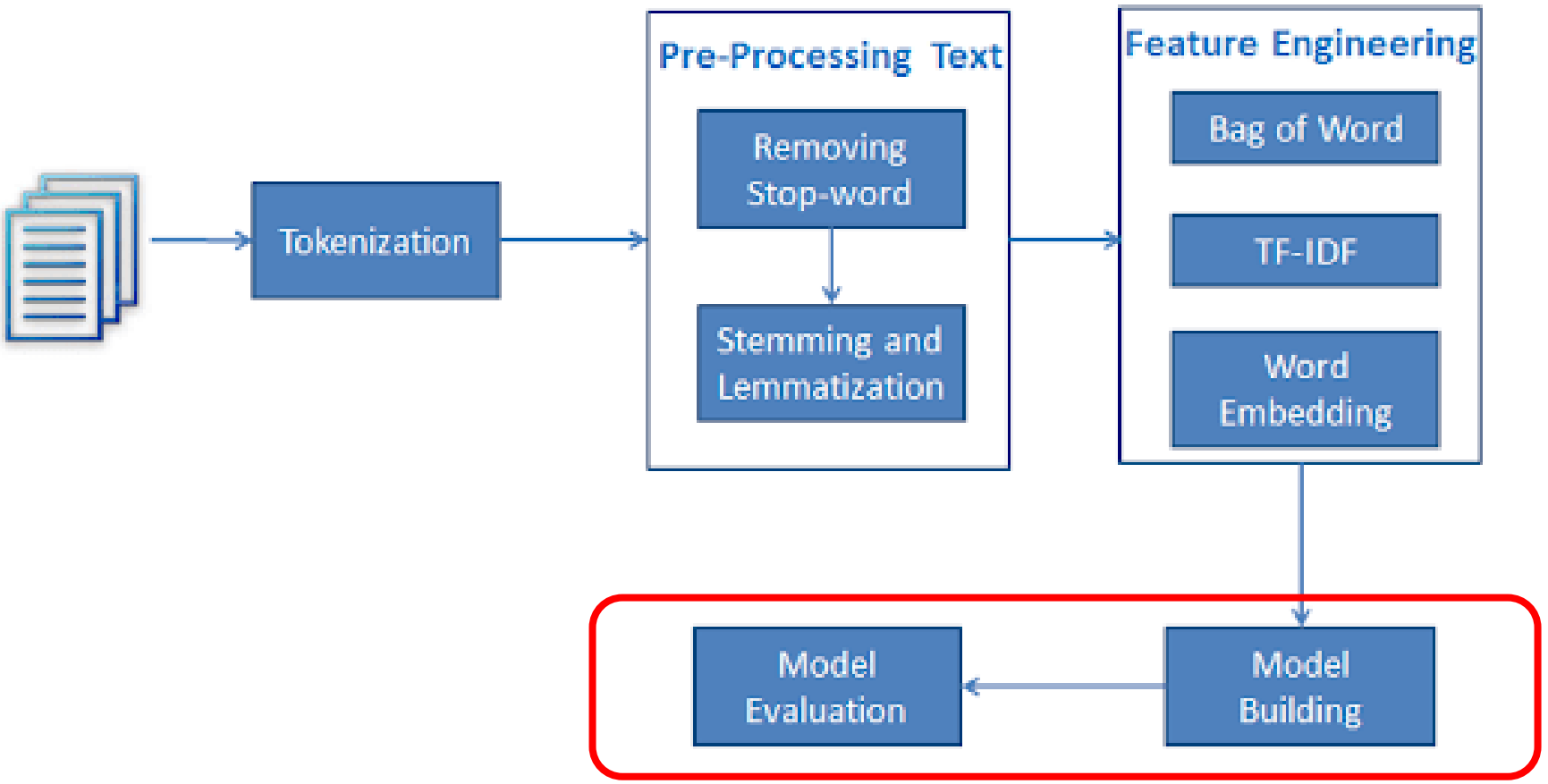
```
token = RegexpTokenizer(r'[a-zA-Z0-9]+') #알파벳, 숫자
tfidf = TfidfVectorizer(lowercase=True,#소문자
stop_words='english', #영어불용어
ngram_range = (1,1), #1-gram
tokenizer = token.tokenize) #알파벳, 숫자만 잘라서 인코딩
text_counts= tfidf.fit_transform(data['Phrase'])
```

	7	absolute	action	actor	actors	actually	adage	age	aggressive	aims	...	windtalkers
0	0.0	0.0	0.0	0.0	0.0	0.0	0.291450	0.0	0.0	0.0	...	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.414287	0.0	0.0	0.0	...	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	...	0.0

Train and Test Split
훈련, 테스트 데이터 분리

```
from sklearn.model_selection import train_test_split


x_train, x_test, y_train, y_test = train_test_split(df,
data.Sentiment, test_size=0.3, random_state=1)
```

Model Building and Evaluation
모델 구축 및 평가



Exercise #2 Sentiment Analysis Example 감성분석 예제

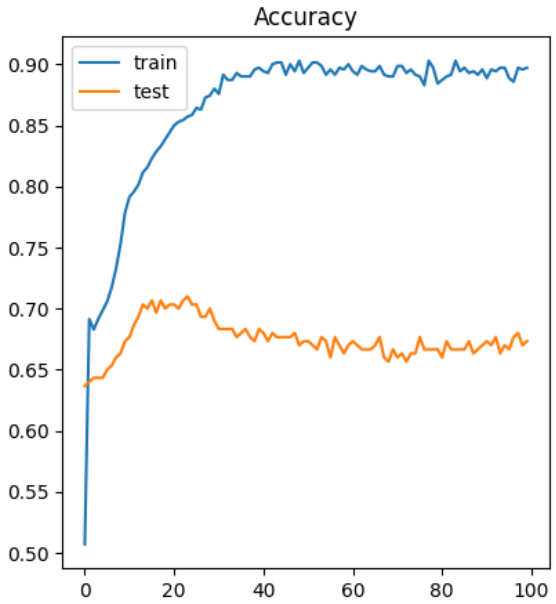
- Build and evaluate a neural network model. 신경망 모델을 만들고 평가
  - 100 Dense with relu, 5 Dense with softmax
  - loss='sparse_categorical_crossentropy', optimizer='adam’
  - History graph
  - accuracy_score, confusion_matrix, classification_report
- Prediction with test data. 테스트 데이터로 예측

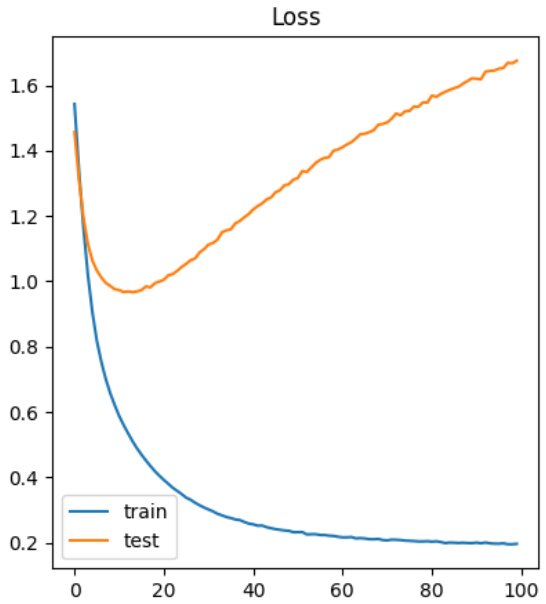
Neural Network Model 신경망모델

```
from keras.models import Sequential

from keras.layers import Dense

y_test.value_counts()

model = Sequential()

model.add(Dense(10, input_dim=x_train.shape[1],
activation='relu'))

model.add(Dense(5, activation='softmax'))
```

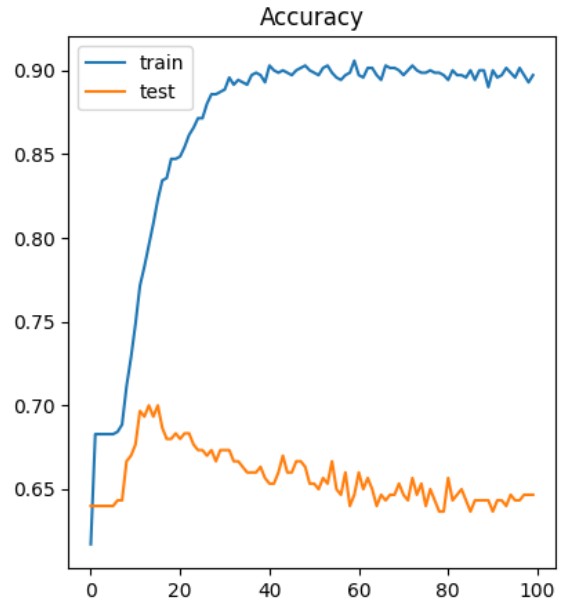
Compile and Fit 컴파일 및 학습

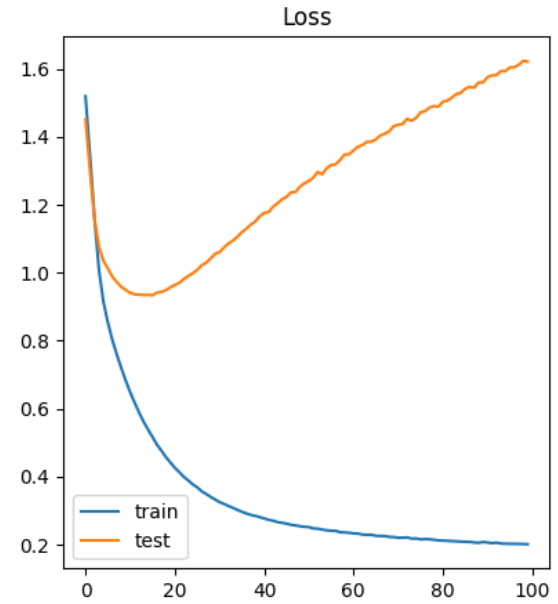
```
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=2,
validation_data=(x_test, y_test))
```





CountVectorizer

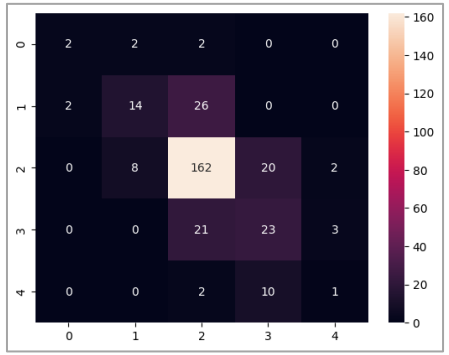




TfidfVectorizer

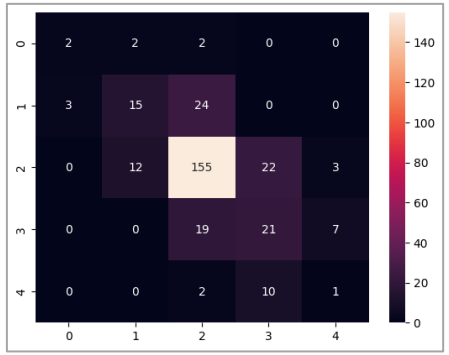
Model Evaluation 모델평가

```
model.evaluate(x_test, y_test) #accuracy
y_pred = model.predict(x_test).round()
accuracy_score(y_test, y_pred) #test performance
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True)
print(classification_report(y_test, y_pred))
```



	precision	recall	f1-score	support
0	0.50	0.33	0.40	6
1	0.58	0.33	0.42	42
2	0.76	0.84	0.80	192
3	0.43	0.49	0.46	47
4	0.17	0.08	0.11	13
accuracy			0.67	300
macro avg	0.49	0.42	0.44	300
weighted avg	0.65	0.67	0.66	300

CountVectorizer



	precision	recall	f1-score	support
0	0.40	0.33	0.36	6
1	0.52	0.36	0.42	42
2	0.77	0.81	0.79	192
3	0.40	0.45	0.42	47
4	0.09	0.08	0.08	13
accuracy			0.65	300
macro avg	0.43	0.40	0.42	300
weighted avg	0.64	0.65	0.64	300

TfidfVectorizer

Prediction 예측

```
test_data = pd.DataFrame({'Phrase':'hello my name is kim'}, index=[0])

test_data = cv.transform(test_data).toarray()

np.argmax(model.predict(test_data))
```

Exercise #2 Amazon Customer Review Example

- Import data. (columns = [‘sentence’, ‘label’])
- X and y split.
- Vectorization (CountVectorizer → data frame)
- Train and test split (sklearn.model_selection, test=.25, random_state))
- Neural Network Model (10 Dense with relu, 1 Dense with sigmoid)
- Compile (adam, binary_crossentropy, accuracy)
- Fit (epochs=100, batch_size=10)
- Evaluate (accuracy and loss)

Keras Embedding Layers
캐라스 임베딩 레이어

- Keras offers an Embedding layer that can be used with text data. 캐라스는 텍스트 데이터에 사용할 수 있는 임베딩레이어를 제공함
- In an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. 임베딩에서는 연속 스페이스에서 단어를 표현하는 밀집벡터로 사용

Keras Embedding Layers
캐라스 임베딩 레이어

```
from keras.layers import Embedding
e = Embedding(input_dim, output_dim, input_length)
```

- input_dim: 총 단어의 개수. 즉, 단어 집합 (vocabulary)의 크기
- output_dim: 임베딩한 후의 벡터의 크기
- input_length: 각 입력 시퀀스의 길이

Word Embedding Example
워드 임베딩 예제

```
from keras.layers import Embedding
#text=[['Hope', 'to', 'see', 'you',
  'soon'],
  ['Nice', 'to', 'see', 'you',
  'again']]
text=[[0, 1, 2, 3, 4],[5, 1, 2, 3,
  6]]
Embedding(7, 2, input_length=5)
```

```
#+-----------+-----------+
#|   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] |
#+-----------+-----------+
```

- 총 단어의 개수는 7개
- 임베딩 벡터의 크기는 2
- 입력시퀀스의 길이는 5

Padding Sequences 시퀀스 패딩

```
from keras.utils import pad_sequences

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

```
array([[1, 2, 3],
       [4, 5, 6],
       [0, 7, 8]])
```

뒤쪽으로 패딩할 때는 post사용

Word Embedding Example
워드임베딩 예제

```
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']

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

- One hot encoding (50 vocab size)
- Padding (4 maxlen, post)
- NN model
  - Embedding (8 output vector size)
  - Flatten, 1 Dense with sigmoid
- Compile
- Fit (epochs = 50)
- Evaluate (accuracy)
- Prediction

One Hot Encoding 원핫인코딩

- 50개의 단어로 전체 단어를 숫자화 시킴

```
from keras.preprocessing.text import one_hot
vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs]
print(encoded_docs)
```

```
[[47, 16], [41, 18], [33, 39], [7, 18], [49], [9], [16, 39], [3, 41], [16, 18], [8, 26, 16, 24]]
```

Padding 패딩

- 4개의 넓이로 숫자를 패딩
- 결과값은 50x4

```
from keras.utils import pad_sequences

max_length = 4

padded_docs = pad_sequences(encoded_docs,
maxlen=max_length, padding='post')

print(padded_docs)
```

```
[[47 16  0  0]
 [41 18  0  0]
 [33 39  0  0]
 [ 7 18  0  0]
 [49  0  0  0]
 [ 9  0  0  0]
 [16 39  0  0]
 [ 3 41  0  0]
 [16 18  0  0]
 [ 8 26 16 24]]
```

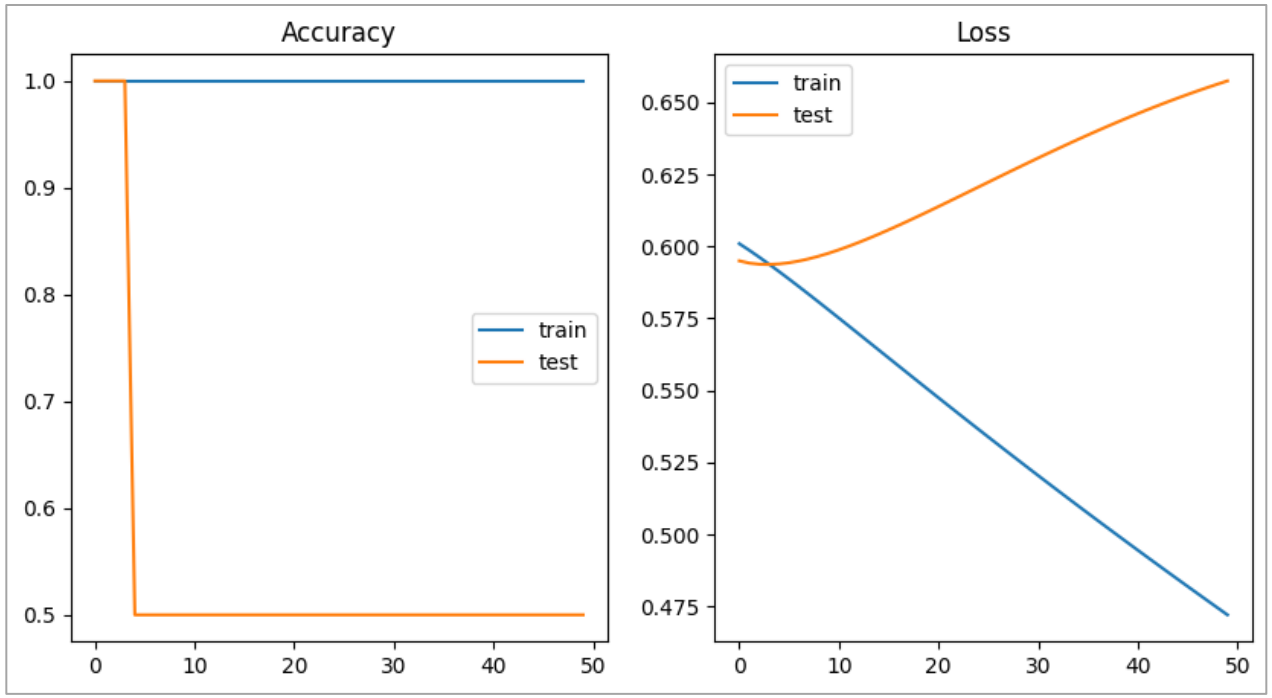
Model Building 모델 구축

```
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Embedding
model = Sequential()
model.add(Embedding(vocab_size, 8,
input_length=max_length)) #50개 단어설정, 4변수
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 4, 8)              400

 flatten_2 (Flatten)         (None, 32)                0

 dense_15 (Dense)            (None, 1)                 33

=================================================================
Total params: 433
Trainable params: 433
Non-trainable params: 0
_________________________________________________________________
None
```

Compile and Fit 컴파일 및 학습

```
model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy’])

history = model.fit(padded_docs, labels, epochs=50,
verbose=1, validation_split=.2)
```

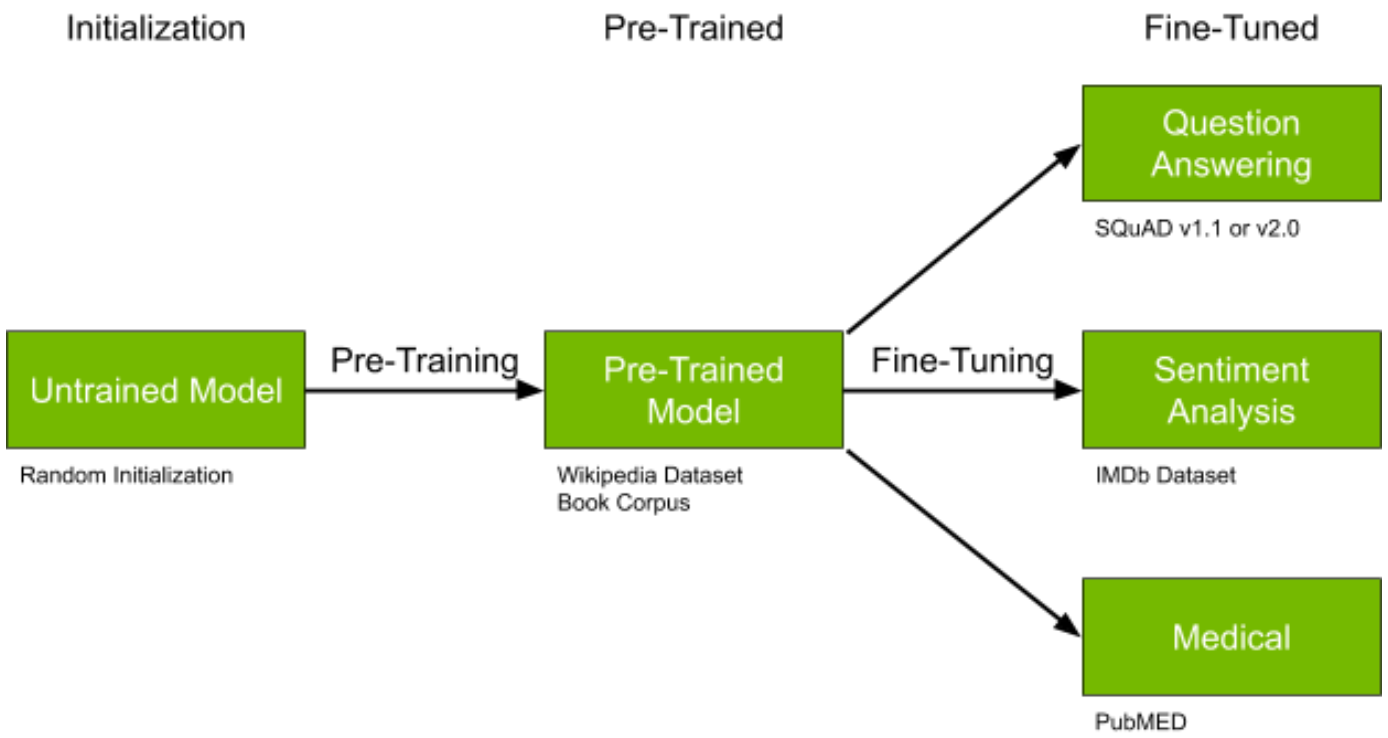


Prediction 예측

```
test_docs = ['no way']

test_docs = [one_hot(d, vocab_size) for d in test_docs]

test_docs = pad_sequences(test_docs, maxlen=max_length,
padding='post')

model.predict(test_docs).round()
```

Fine-Tuning 미세조정

- BERT (Bidirectional Encoder Representations from Transformers)의 핵심 모델은 대규모의 일반 데이터 세트에서 사전 훈련한 다음 질문/답변, 감정 분석 또는 명명된 엔터티 인식과 같은 다양한 작업을 수행하도록 미세 조정할 수 있음

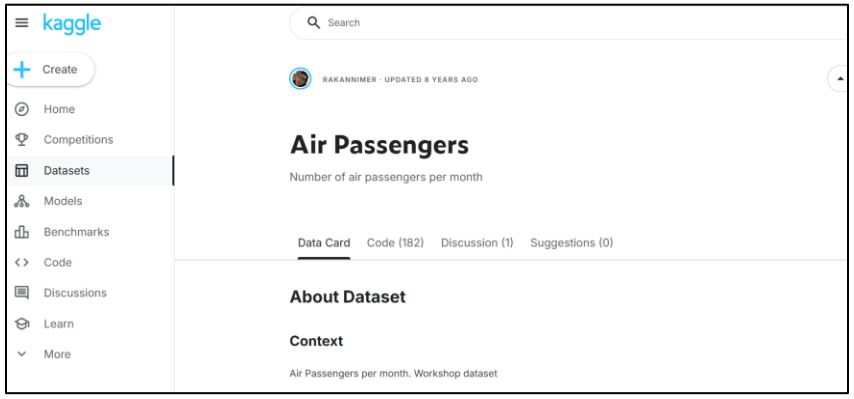


PEFT (Parameter-Efficient Fine Tuning)

- 기본 모델의 대부분의 파라미터를 고정하고, 어댑터 레이어를 추가하여 일부 가중치만 학습하는 방식
- LoRA, QLoRA 등 경량 튜닝 기법으로 빠른 반복 개선 가능
  - LoRA (Low-Rank Adaptation): 사전학습 모델의 가중치는 동결하고, 행렬을 추가하여 일부만 학습. 작은 특정 데이터 세트에 더 적합하도록 수정하는 기계 학습 기술
  - QLoRA (Quantized LoRA): 로라를 기반으로 원래 모델의 가중치를 4비트로 양자화. 양자화 기술을 통합하여 모델 성능을 향상시키면서 메모리 사용량을 더욱 줄임
  - DoRA (Weight-Decomposed LoRA): 로라의 가중치를 크기와 방향 구성요소로 분해하여 더 세밀한 업데이트가 가능하게 함
- Unsloth: QLoRA 기반 프레임워크. LLM 훈련을 최적화 하기 위한 라이브러리. LORA, QLORA를 더 적은 GPU로 학습할 수 있음

여객승객 예제
AirPassengers Example

- AirPassengers dataset includes monthly Airline Passenger Numbers 1949-1960. Build a prediction model for Airline Passenger Number. 여객승객 데이터셋에는 1949년에서 1960년까지의 월별 승객수에 대한 데이터를 포함하고 있습니다. 승객수를 예측하는 모델을 만드시오



Exercise #3

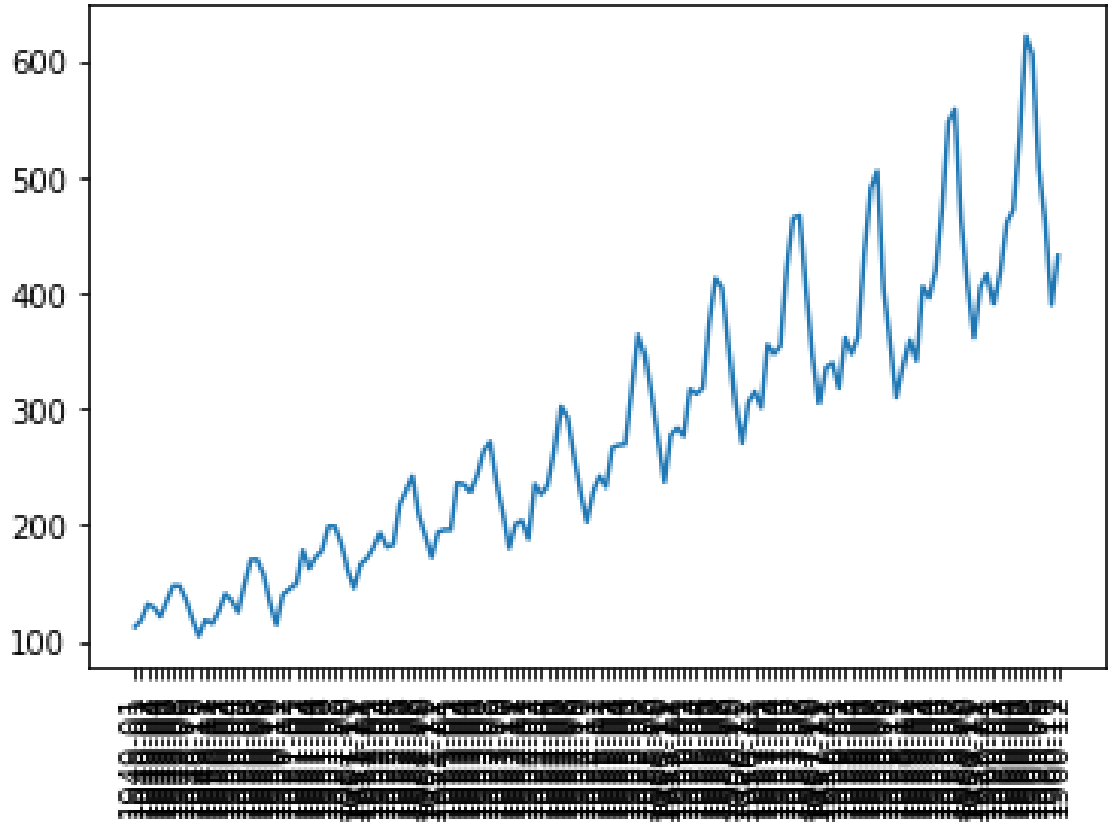
- Use the shift function and build the neural network model for Air Passenger data. 여객 승객데이터를 쉬프트로 이동한 후에 신경망 모델을 만드시오
  - Load air passengers csv file (set_index, plot)
  - Create a new dataframe (shift, dropna)
  - X and y split
  - Neural network model (100 Dense with relu, 1 Dense)

Exercise #3

- Compile (optimizer='adam', loss='mse')
- Fit (epochs=50, validation_split=.2)
- Predict and plot
- Evaluate (loss and val_loss chart)
- Test the data (50, 60, 70)

Time Series Data 시계열데이터

```
air =
pd.read_csv('AirPasseng
ers.csv')
air.set_index('Month',
inplace=True)
#인덱스바꾸기


plt.plot(air)
plt.xticks(rotation=90)
plt.show()
```



shift() 이동

- Used to create copies of columns that are pushed forward (rows of NaN values added to the front) or pulled back (rows of NaN values added to the end). 데이터를 앞이나 뒤로 밀 수 있음

```
df = pd.DataFrame()
df['t'] = [x for x in range(10)]
df['t-1'] = df['t'].shift(1) #아래로 내림
print(df)
```

	t	t-1
0	0	NaN
1	1	0.0
2	2	1.0
3	3	2.0
4	4	3.0
5	5	4.0
6	6	5.0
7	7	6.0
8	8	7.0
9	9	8.0

Data Preparation 데이터 준비

```
df = pd.DataFrame()
df['t'] = air['#Passengers']
df['t+1'] = df['t'].shift(-1) #위로 올림
df['t+2'] = df['t'].shift(-2)
df['t+3'] = df['t'].shift(-3)
print(df)
df.dropna(inplace=True)
```

X and Y Split
독립변수, 종속변수 분리

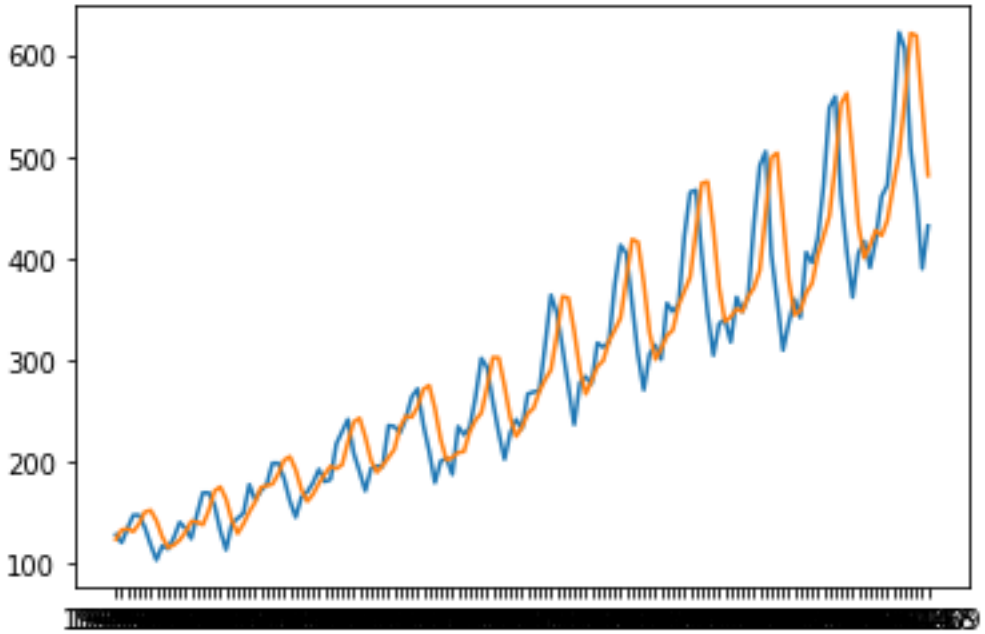
```
y = df['t+3']

x = df.drop('t+3', axis=1)

or

x = df.iloc[:,:-1]

y = df.iloc[:, -1]
```

Neural Network Model
신경망 모델

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(100, activation='relu',
input_dim=3))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

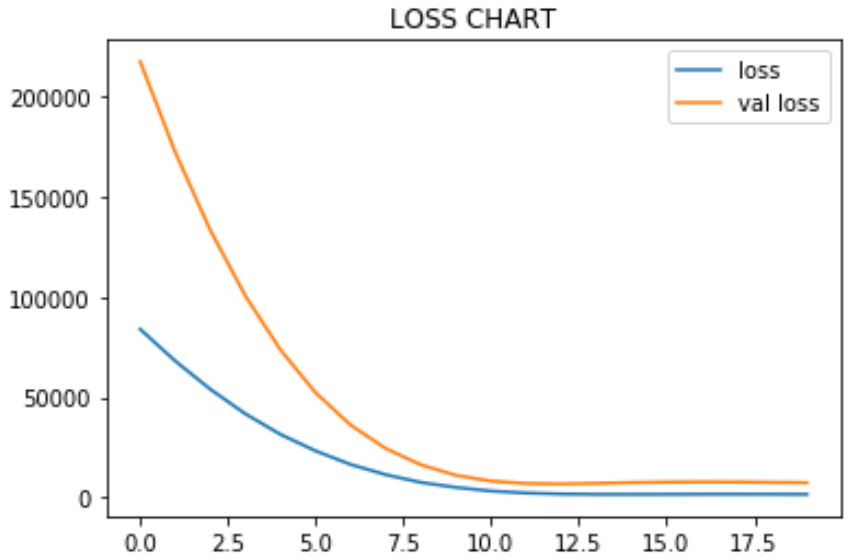
Fit and Prediction 훈련 및 예측

```
# fit model
history = model.fit(x, y, epochs=20,
verbose=1, validation_split=.2)


# demonstrate prediction
y_pred = model.predict(x)
Y_pred


# plot prediction
plt.plot(y)
plt.plot(y_pred)
plt.show()
```



Evaluation 평가

```
from sklearn.metrics import r2_score
model.evaluate(x,y)
r2_score(y, y_pred)

tab = pd.DataFrame(history.history)
tab
```




```
plt.plot(tab['loss'], label='loss')
plt.plot(tab['val_loss'], label='val loss')
plt.title('LOSS CHART')
plt.legend()
plt.show()
```

Prediction with Test Data
테스트 데이터로 예측

```
test_data = np.array([[50, 60, 70],])
test_pred = model.predict(test_data,
verbose=0)
print(test_pred)
```

Exercise #4

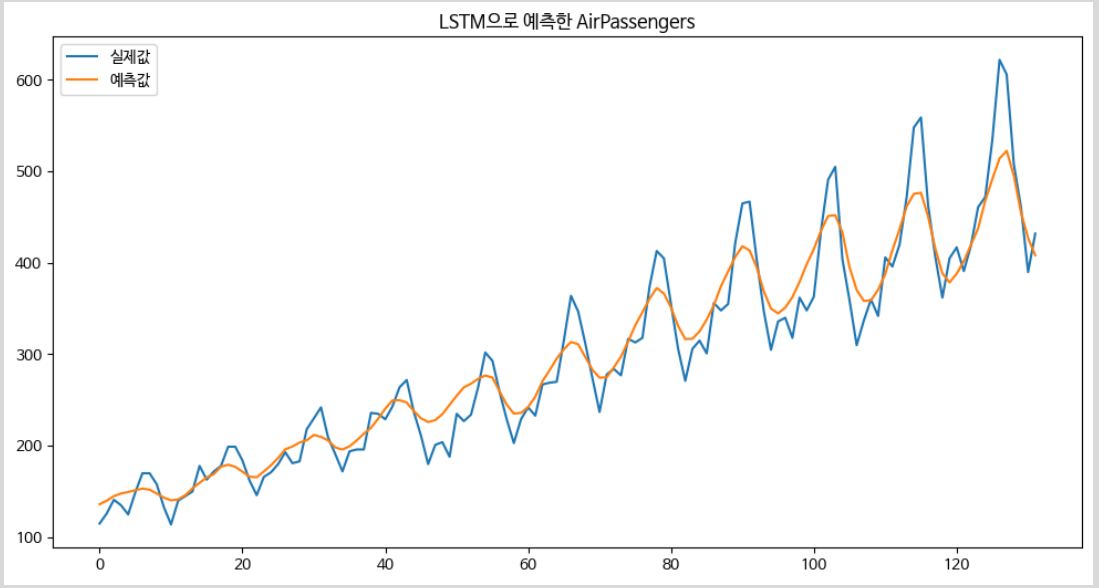
- Use AirPassanger Dataset to build a prediction model using RNN. 여객승객 데이터셋으로 예측모델 만들기
  - Scaling (MinMaxScaler between 0 and 1)
  - Split into x and y (seq_length = 12)
  - Reshape input data to be [samples, time steps, features]

Exercise #4

- Build a model. 모델구축
  - 64 units of LSTM (return_sequences=False)
  - Dropout (0.2)
  - 1 Dense layer
  - adam optimizer and mse loss
- Fit the model. 모델학습
  - 100 epochs, 16 batch size, validation split = 0.2

Exercise #4

- Evaluate the model. 모델 평가
  - Loss graph
- Prediction using test dataset. 예측
  - Inverse transform
- Visualize the result. 결과 시각화



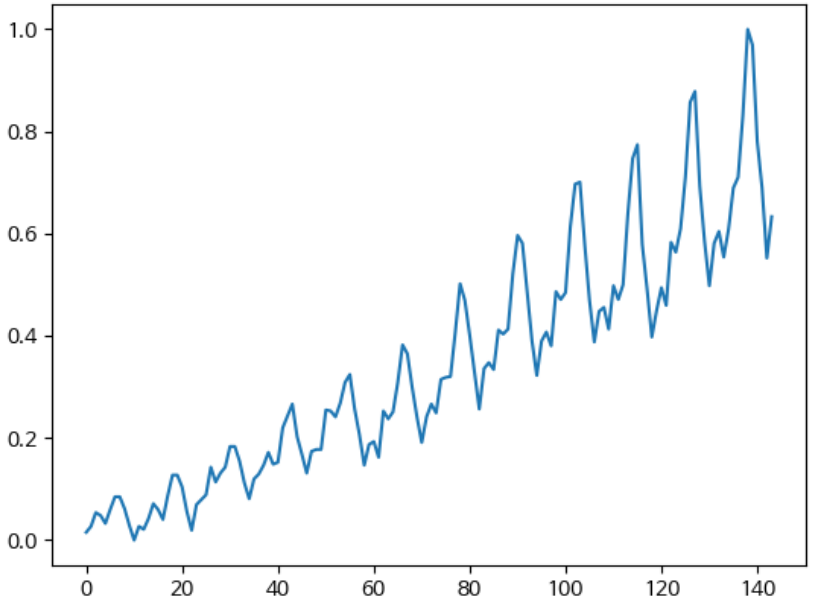
필요한 라이브러리 불러오기

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

Scaling 정규화

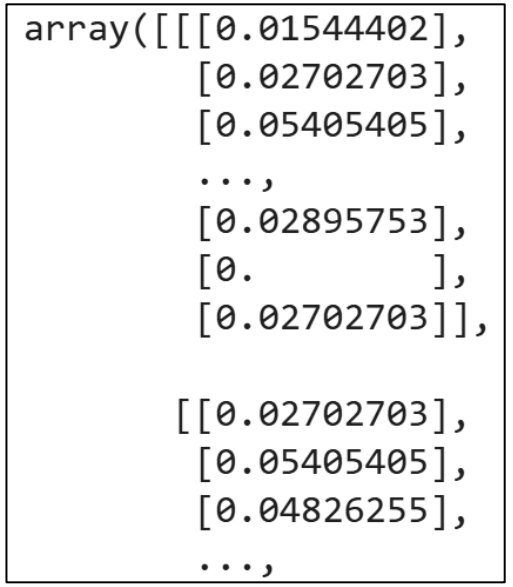
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(air)


# 시각화

plt.plot(air_scaled)

plt.show()
```

- 입력값의 크기가 크거나 범위가 들쭉날쭉하면 학습이 불안정해짐
- Gradient Exploding/Vanishing (기울기 폭발/소실) 문제 발생
- 학습 속도가 빨라짐
- 크기가 큰 특성 하나가 손실에 미치는 영향을 독점

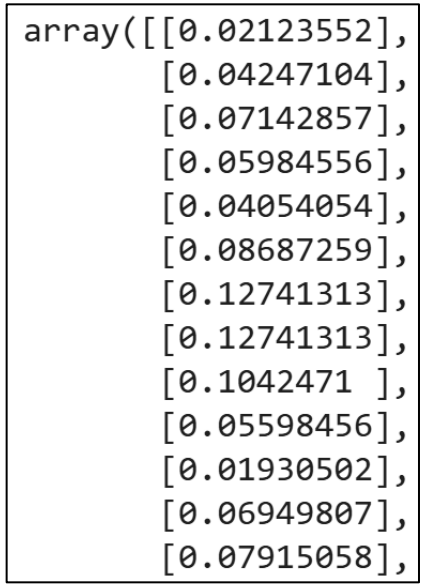


X and Y Split

```
# 시퀀스 데이터 생성 함수
def create_sequences(data, seq_len):
    X, Y = [], []
    for i in range(len(data) - seq_len):
        X.append(data[i:i+seq_len])
        Y.append(data[i+seq_len])
    return np.array(X), np.array(Y)
```



```
array([[[0.01544402],
        [0.02702703],
        [0.05405405],
        ...,
        [0.02895753],
        [0.        ],
        [0.02702703]],

       [[0.02702703],
        [0.05405405],
        [0.04826255],
        ...,
```



```
array([[0.02123552],
       [0.04247104],
       [0.07142857],
       [0.05984556],
       [0.04054054],
       [0.08687259],
       [0.12741313],
       [0.12741313],
       [0.1042471 ],
       [0.05598456],
       [0.01930502],
       [0.06949807],
       [0.07915058],
```

```
# 시퀀스 데이터 생성
seq_length = 12  # 1년 단위 예측
X, Y = create_sequences(scaled_data, seq_length)
```

Reshape Input Values 입력 모양 바꾸기

- Reshape x to be [samples, time steps, features]. 데이터셋의 모양을 바꿔줌. [데이터의 개수, 타임스텝, 1]

```
# LSTM 입력 형태로 reshape
X = X.reshape(X.shape[0], X.shape[1], 1)
X.Shape # (132, 12, 1)
```

LSTM 입력값

- Must be 3D. 3차원이어야 함
- The 3 input dimensions are: samples, time steps, and features. 3차원은 샘플의 수, 타임스텝의 수, 특성의 수
- The input_shape argument on the first hidden layer takes the number of time steps and features. 첫번째 히든레이어의 입력모양은 타임스텝의 수와 특성의 수를 받음

LSTM 모델

```
# LSTM 모델 구성
model = Sequential()
model.add(LSTM(64, input_shape=(seq_length, 1), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
model.summary()
```

return_sequences

- 출력 형식을 조절
- True: 모든 시점의 출력을 시퀀스로 반환
- False: 마지막 시점의 출력만 반환

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 64)	16,896
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Total params: 16,961 (66.25 KB)
Trainable params: 16,961 (66.25 KB)
Non-trainable params: 0 (0.00 B)

모델 학습

```
# 컴파일
model.compile(optimizer='adam', loss='mse’)

# 모델 학습
history = model.fit(X, Y, epochs=100, batch_size=16,
verbose=1, validation_split=.2)
```

Evaluation 평가

model.evaluate(X, Y)

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0022
0.0029964423738420001
```

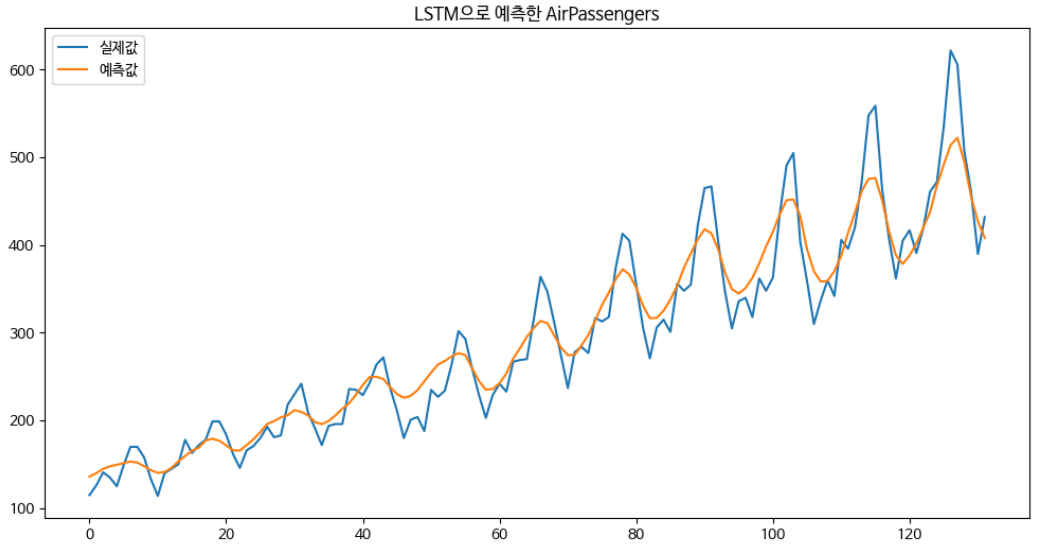
```
from sklearn.metrics import mean_squared_error
train_rmse = np.sqrt(mean_squared_error(y_train_inverse,
  train_pred_inverse))
print('Train Score: %.2f RMSE' % (train_rmse))
test_rmse = np.sqrt(mean_squared_error(y_test_inverse,
  test_pred_inverse))
print('Test Score: %.2f RMSE' % (test_rmse))
```

Prediction and Invert Predictions
예측과 예측 결과 역정규화

```
# 예측
pred = model.predict(X)

# 예측 결과 역정규화
predicted = scaler.inverse_transform(pred)
actual = scaler.inverse_transform(Y)
```

Forecasting Plot 예측 그래프

```
# 시각화
plt.figure(figsize=(12,6))
plt.plot(actual, label='실제값')
plt.plot(predicted, label='예측값')
plt.title('LSTM으로 예측한 AirPassengers')
plt.legend()
plt.show()
```



Evaluation 평가

```
model.evaluate(X, Y)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0022
0.0029964423738420001
```

```
# 히스토리 그래프
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```