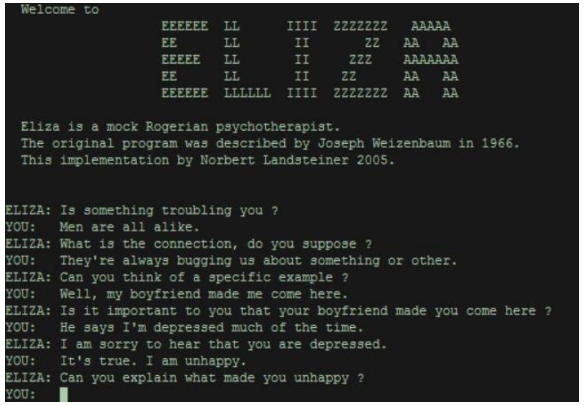
딥러닝 다중 분류

챗봇 엔진 만들기

최 석 재

lingua@naver.com

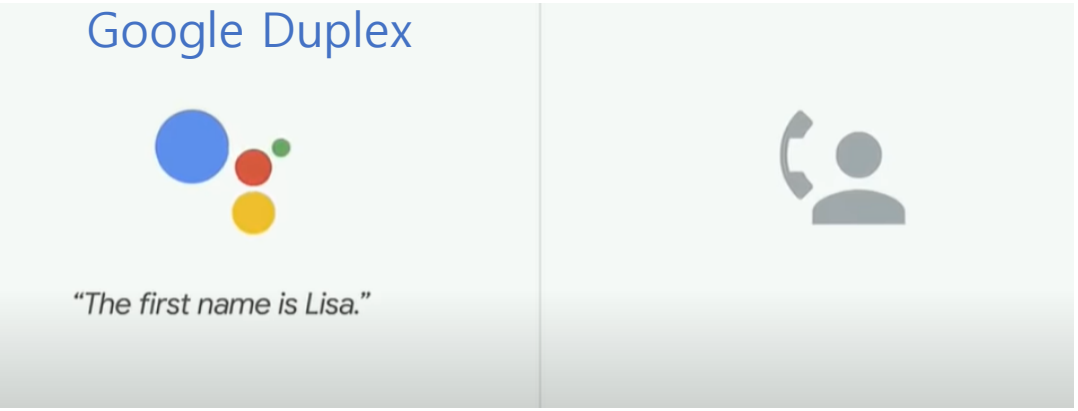
챗봇



- 챗봇은 1966년 MIT에서 ELIZA라는 이름으로 처음 개발
- 빅데이터 및 인공지능을 활용하는 컴퓨터가 고객에게 응대하는 서비스

- 24시간 응대
- 인건비 대비 저렴한 비용으로 운영
- 언택트 시대의 한 방안

- 챗봇에게는 단순하고 반복적인 질의를,
- 사람 상담원에게는 복잡한 질의를 맡김



https://www.youtube.com/watch?v=D5VN56jQMWM

카카오뱅크 챗봇

- 2019년 7월, 카카오뱅크는 특별 이벤트 실시
- 이자 5%, 적금 이자 2배
- 이 이벤트로 고객 1,000만 명 돌파

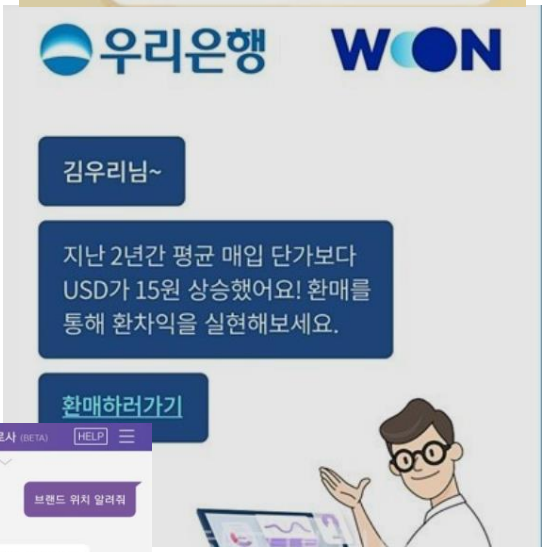
- 파격적인 이벤트로 고객문의 폭발
- 그러나 상담 고객의 50%가 챗봇으로 궁금증 해소

- 현재도 온라인 고객 상담의 35%를 챗봇으로 해결

정보 창구로서의 챗봇

- 우리은행 위비봇
  - 환전 정보, 날씨, 인물 정보 등

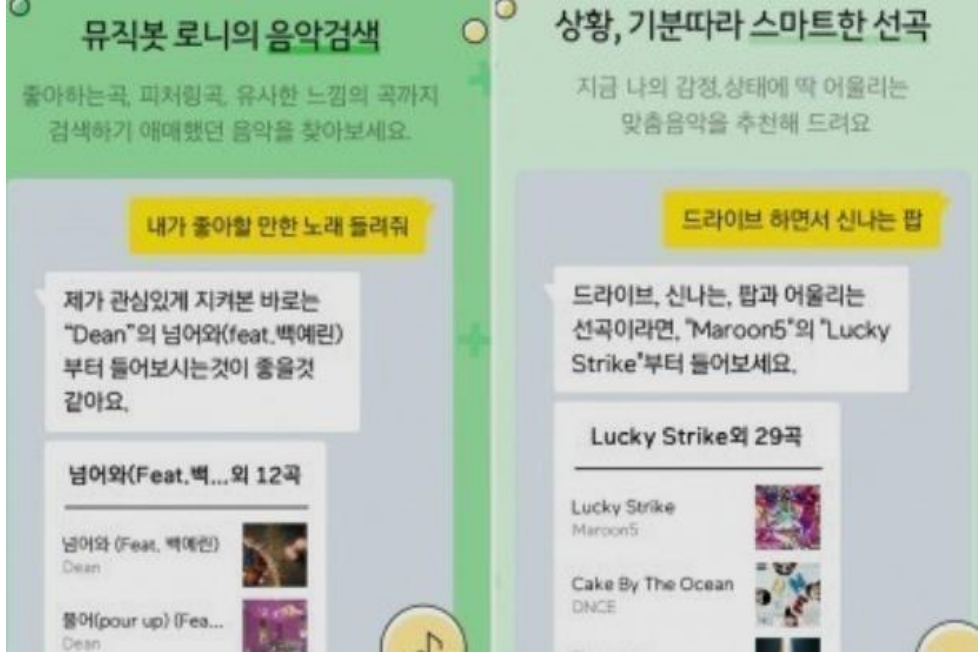
- 롯데쇼핑 로사
  - 맞춤형 상품 추천, 스타일 추천, 영업시간 안내 등



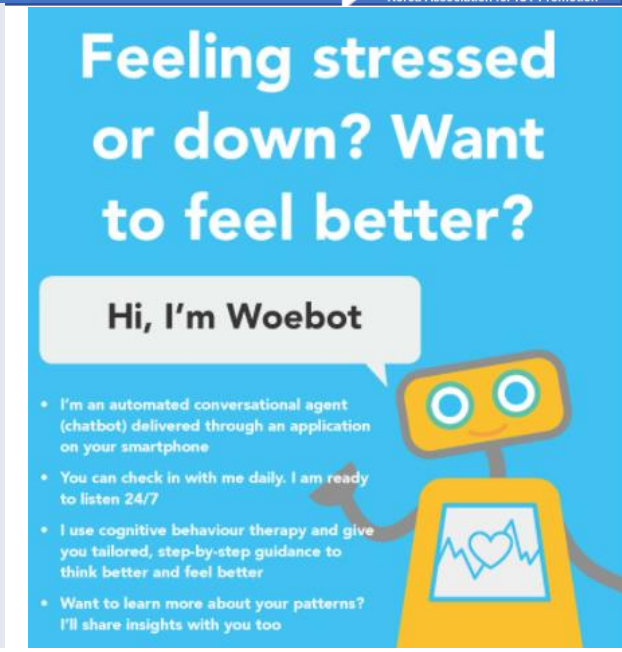
정보 창구로서의 챗봇

- CJ 대한통운 챗봇
  - 택배 조회, 택배 예약, 반품 예약, 배송일정 확인 등

- 멜론 로니
  - 개인별 큐레이션, 음악 관련 질문 응대



정보 창구로서의 챗봇

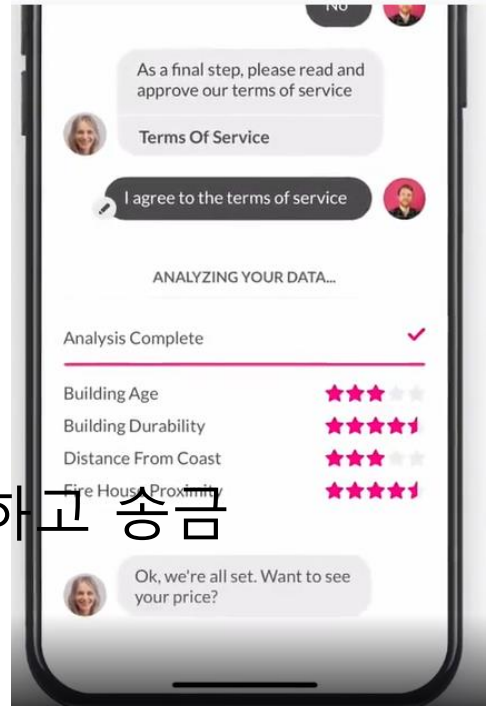


- 농림부 Ask Karen
  - 안전한 육류 준비 방법, 계란 등의 이력 정보 등

- Woebot
  - 우울증 환자의 친구가 되는 챗봇
  - 환자 심리 추적, 심리 패턴 분석, 심리 상태 개선

- Talla
  - 일정관리, 고객관리, 문서검색, 분류 업무 등

- Maya
  - 온라인 주택화재 보험 챗봇
  - 피해사실 증명사진, 서류, 영상을 올리면 3분 안에 보험금을 산정하고 송금



챗봇 발전 단계

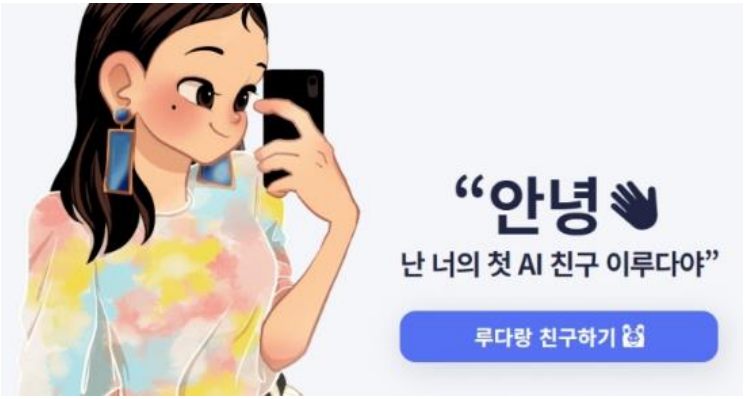
	1단계	2단계	3단계
	챗봇 서비스 (Chatbot)	지능형 비서 (Intelligent Assistant)	감성 비서 (Conscious Assistant)
데이터	텍스트, 음성	텍스트, 음성, 시각자료	텍스트, 음성, 시각자료, 행동인지
주요기술	패턴매칭, 키워드 및 연관어 추출 등	딥러닝, 머신러닝, 자연어처리 등	감성인지기술, 데이터 정형화 기술 등
내용	학습된 내용 질의응답 사용자와 단순한 형태의 소통 검색을 통한 결과 제공	사용자의 패턴 및 상황을 고려한 개인 맞춤형 서비스 간단한 업무처리	감성 교류를 통한 각종 서비스에 대한 선제적 대응

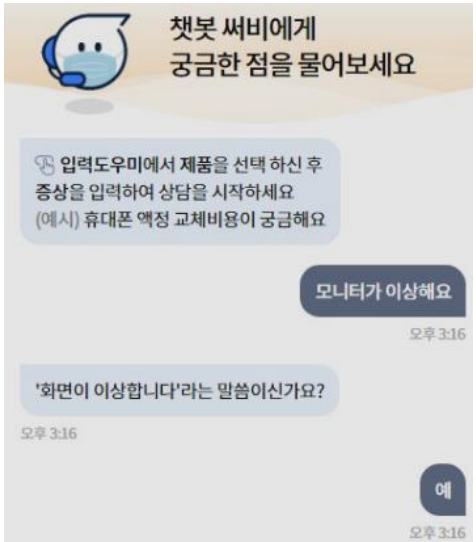
한국정보화진흥원(2018)

챗봇의 종류

- 챗봇은 크게 사람의 말에 자유롭게 대화하는 챗봇과
- 지정된 질문에 답변을 준비해 놓은 QnA 형 챗봇이 있음

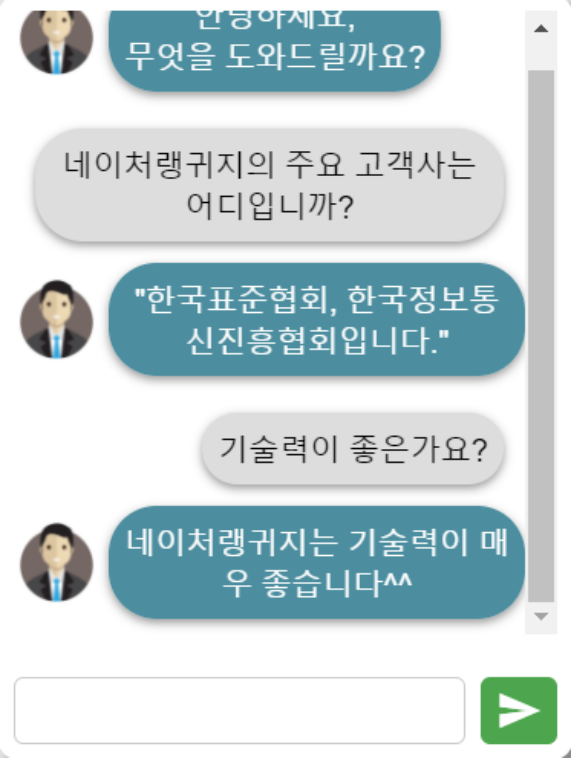
- 자유롭게 대화하는 챗봇은 많은 양의 훈련 데이터를 필요로 하며,
- 오류의 가능성이 있어서 정확한 정보 제공이 필요한 상황에는 부적합
- 비즈니스 챗봇은 대부분 QnA 형을 사용





챗봇의 형태

- 애플리케이션 형태
- 카카오톡, 네이버 톡톡 등 기존 SNS 플랫폼을 사용하는 형태
- 웹 홈페이지에서 사용하는 형태



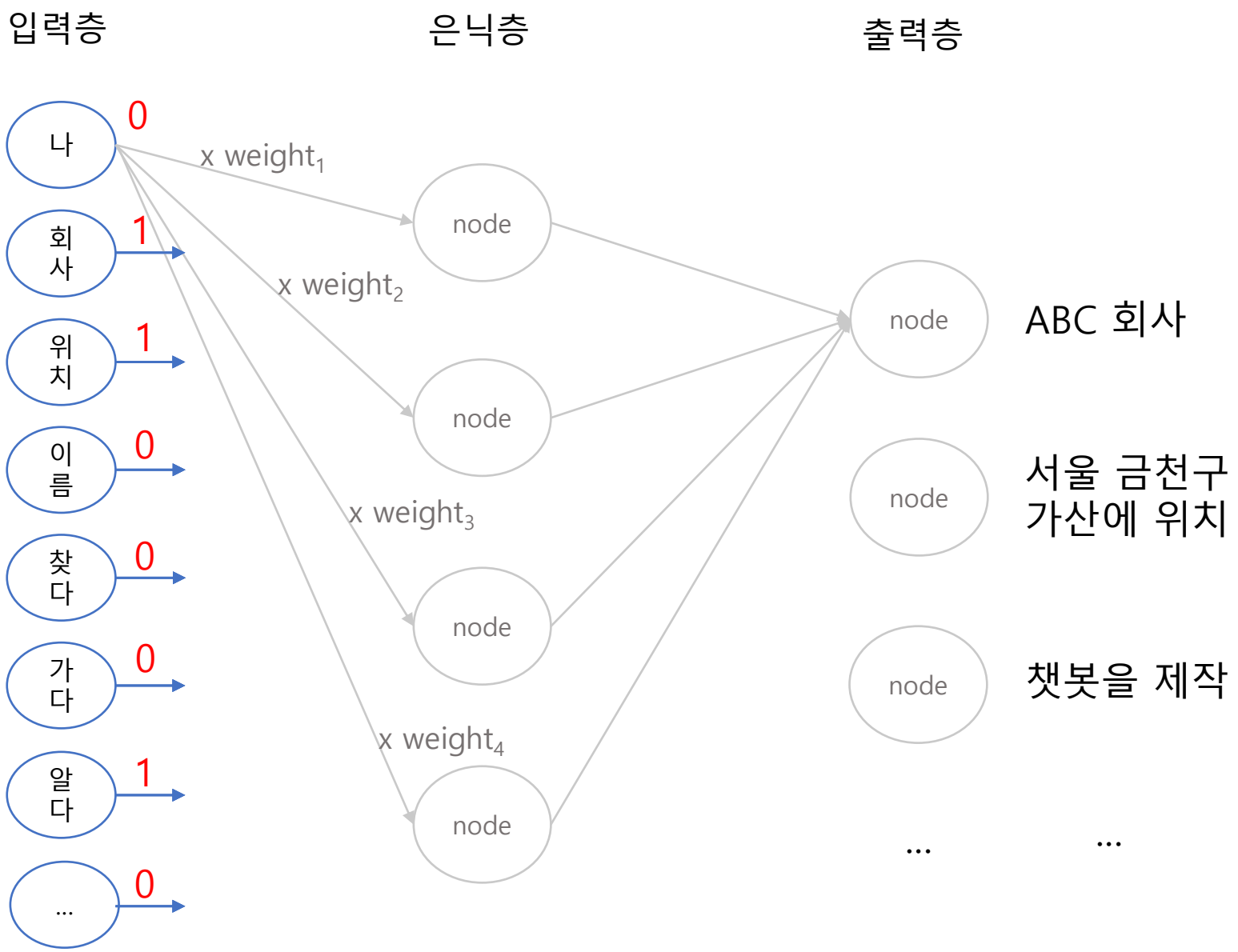
AI 챗봇 실습

다중분류 챗봇

- 딥러닝 챗봇
- 사용자가 입력한 단어를 숫자로 변환한 뒤,

  가중치를 곱하여 은닉층에 보냄

  가장 높은 확률을 갖는 결과를 답변으로 출력



챗봇 DataSet

- 국내 모 렌즈 회사 챗봇의 인공지능 모델에 사용된 데이터의 샘플
- 반품 절차에 관한 문의, 배송에 관한 문의 등 총 10개 종류 문의
- 질의문(Q)과 답변라벨(A)의 두 개 컬럼으로 구성
- 질의문은 동일한 내용의 다른 표현으로 6,322 문장

	A	B
1	저희 안경원 코드 번호 확인이 가능할까요?	매장코드
2	저희 안경원 코드를 잊어버렸습니다	매장코드
3	안경원 코드번호 확인이 어렵습니다 어떻게 해야하나요?	매장코드
4	매장 코드번호 확인이 어렵습니다 어떻게 해야하나요?	매장코드
5	안경원 코드번호 확인하는 방법이 있을까요?	매장코드
6	안경원 코드번호 조회부탁드립니다	매장코드
7	저희 안경원 코드 확인부탁드려요	매장코드
8	안경원 코드 번호를 잊어버렸어요 확인부탁드려요	매장코드
9	안경원 코드 번호 확인을 어디에서 확인 할 수 있나요?	매장코드
10	저희 안경원 코드 어떻게 확인 할 수 있나요?	매장코드
11	저희 매장 코드 번호 확인하고싶어요	매장코드
12	매장 코드 번호 어디에서 볼 수 있나요?	매장코드
13	매장 코드 번호 확인하고싶어요 어디에서 볼 수 있나요?	매장코드
14	매장 코드 번호 조회는 어디에서 할 수 있나요?	매장코드
15	매장 코드 번호 어떻게 알 수 있나요?	매장코드
16	매장 번호 확인하고싶어요 어디에서 볼 수 있나요?	매장코드

구글 드라이브와 연결

```
# from google.colab import auth
# auth.authenticate_user()
```

이 두 줄은 접속이 잘 안되는 경우에만

- from google.colab import drive
- drive.mount('/content/gdrive')

형태소 분석기 관련 설치

- !apt-get update
- !apt-get install g++ openjdk-8-jdk

- !pip install JPype1
- !pip install rhinoMorph

경로 설정

- chat_dir = '/content/gdrive/My Drive/pytest/chatbot/'
- print(chat_dir)

pytest 폴더는 Colab Notebooks 폴더의 바깥 쪽에 둔다

읽기, 쓰기 함수

```
# 데이터 읽기 함수 정의
def read_data(filename, encoding='cp949'):
    with open(filename, 'r', encoding=encoding) as f:
        data = [line.split('₩t') for line in f.read().splitlines()]
    return data

# 데이터 쓰기 함수 정의
def write_data(data, filename, encoding='cp949'):
    with open(filename, 'w', encoding=encoding) as f:
        f.write(data)
```

형태소 분석 준비

```
import rhinoMorph
rn = rhinoMorph.startRhino()        # 형태소분석기 기동

data = read_data(chat_dir+'data.txt', encoding='cp949')

print('자료 타입:', type(data))
print('전체 문장수:', len(data))
print('형태소 분석 전 모습:', data[:20])
```

```
filepath:  /usr/local/lib/python3.7/dist-packages
classpath:  /usr/local/lib/python3.7/dist-packages/rhinoMorph/lib/rhino.jar
JVM is already started~
RHINO started!
자료 타입: <class 'list'>
전체 문장수: 6322
형태소 분석 전 모습: [['저희 안경원 코드 번호 확인이 가능할까요?', '매장코드'], ['저희 안경원 코드를 잊어버렸습니다',
```

전체 데이터 형태소 분석

```
import rhinoMorph
rn = rhinoMorph.startRhino()

morphed_data = ''
for data_each in data:
    morphed_data_each = rhinoMorph.onlyMorph_list(rn, data_each[0], pos=['NNG', 'NNP', 'NP', 'VV', 'VA', 'VX', 'XR', 'IC', 'MM', 'MAG'])
    print("morphed_data_each:", morphed_data_each)
    joined_data_each = ' '.join(morphed_data_each)              # 문자열을 하나로 연결
    if joined_data_each:                                        # 내용이 있는 경우만 저장하게 함
        morphed_data += joined_data_each+"₩t"+data_each[1]+"₩n"
print('Morphological Analysis Completed.')

# 형태소 분석된 파일 저장
write_data(morphed_data, chat_dir+'chat_morphed.txt', encoding='cp949')
```

형태소 분석된 데이터 로딩

- data = read_data(chat_dir+'chat_morphed.txt', encoding='cp949')

- print(type(data))
- print(len(data))
- print(len(data[0]))
- print(data[0])

```
<class 'list'>
6322
2
['저희 안경원 코드 번호 확인 가능', '매장코드']
```

- print("연결된 마지막 문장: ", joined_data_each)
- print("마지막 문장의 라벨: ", data_each[1])

```
연결된 마지막 문장:  제품 배송비 없 주문 금액 얼마
마지막 문장의 라벨:  배송비
```

형태소 분석 결과

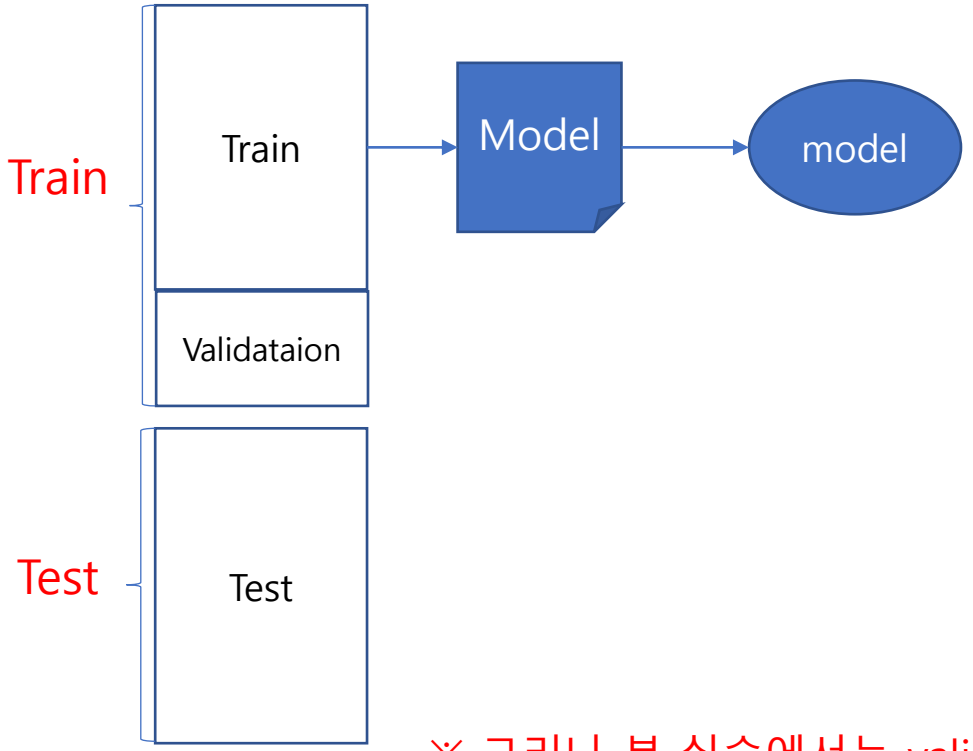
- 형태소 분석 결과를 저장한 data_morphed.txt
- "확인이, 확인하는, 확인부탁, 확인을, ..." → "확인" 으로 통일됨

	A	B
1	저희 안경원 코드 번호 확인이 가능할까요?	매장코드
2	저희 안경원 코드를 잊어버렸습니다	매장코드
3	안경원 코드번호 확인이 어렵습니다 어떻게 해야하나요?	매장코드
4	매장 코드번호 확인이 어렵습니다 어떻게 해야하나요?	매장코드
5	안경원 코드번호 확인하는 방법이 있을까요?	매장코드
6	안경원 코드번호 조회부탁드립니다	매장코드
7	저희 안경원 코드 확인부탁드려요	매장코드
8	안경원 코드 번호를 잊어버렸어요 확인부탁드려요	매장코드
9	안경원 코드 번호 확인을 어디에서 확인 할 수 있나요?	매장코드
10	저희 안경원 코드 어떻게 확인 할 수 있나요?	매장코드
11	저희 매장 코드 번호 확인하고싶어요	매장코드
12	매장 코드 번호 어디에서 볼 수 있나요?	매장코드
13	매장 코드 번호 확인하고싶어요 어디에서 볼 수 있나요?	매장코드
14	매장 코드 번호 조회는 어디에서 할 수 있나요?	매장코드
15	매장 코드 번호 어떻게 알 수 있나요?	매장코드
16	매장 번호 확인하고싶어요 어디에서 볼 수 있나요?	매장코드

	A	B
1	저희 안경원 코드 번호 확인 가능	매장코드
2	저희 안경원 코드 잊어버리	매장코드
3	안경원 코드 번호 확인 어렵 어떻하 하 하	매장코드
4	매장 코드 번호 확인 어렵 어떻하 하 하	매장코드
5	안경원 코드 번호 확인 방법 있	매장코드
6	안경원 코드 번호 조회 부탁 드리	매장코드
7	저희 안경원 코드 확인 부탁 드리	매장코드
8	안경원 코드 번호 잊어버리 확인 부탁 드리	매장코드
9	안경원 코드 번호 확인 어디 확인 하 있	매장코드
10	저희 안경원 코드 어떻하 확인 하 있	매장코드
11	저희 매장 코드 번호 확인 싶	매장코드
12	매장 코드 번호 어디 보 있	매장코드
13	매장 코드 번호 확인 싶 어디 보 있	매장코드
14	매장 코드 번호 조회 어디 하 있	매장코드
15	매장 코드 번호 어떻하 알 있	매장코드
16	매장 번호 확인 싶 어디 보 있	매장코드

훈련데이터와 테스트데이터 분리

- 모델은 훈련데이터만 사용하고,
- 모델의 성능은 검증데이터로 중간 검증하고, 테스트데이터로 최종검증



※ 그러나 본 실습에서는 validation 데이터를 test 데이터로도 사용한다

데이터 분리

```
# 훈련데이터와 테스트데이터 분리
data_text = [line[0] for line in data]                    # 데이터 본문
data_label = [line[1] for line in data]                   # 데이터 레이블 부분

from sklearn.model_selection import train_test_split
train_data_text, test_data_text, train_data_label, test_data_label =
train_test_split(data_text, data_label, stratify=data_label)
```

데이터 분리 확인

Counter 클래스를 이용해 train과 test 데이터의 비율을 확인한다

- from collections import Counter
- train_data_label_freq = Counter(train_data_label)
- print('train_data_label_freq:', train_data_label_freq)

- test_data_label_freq = Counter(test_data_label)
- print('test_data_label_freq:', test_data_label_freq)

```
train_data_label_freq: Counter({'반품문의': 1422, '교환문의': 989, '배송문의': 705, '담당자문의': 517, '샘플문의': 460, '매장코드'
test_data_label_freq: Counter({'반품문의': 474, '교환문의': 330, '배송문의': 236, '담당자문의': 172, '샘플문의': 153, '매장코드':
```

데이터의 길이 통계

- import numpy as np
- text_len = [len(line.split(' ')) for line in train_data_text] # 단어 분리 후 개수 저장
- print("최소길이: ", np.min(text_len)) # 2
- print("최대길이: ", np.max(text_len)) # 18
- print("평균길이: ", np.round(np.mean(text_len), 1)) # 6.4
- print("중위수길이: ", np.median(text_len)) # 6.0
- print("구간별 최대 길이: ", np.percentile(text_len, [0, 25, 50, 75, 90, 100]))

```
[ 2.  5.  6.  7.  8. 18.]
```

※ 데이터가 많지 않고, 길이가 크게 길지 않으므로 사용할 단어 개수는 100% 담을 수 있는 20개로 한다

- print("최소길이 문장: ", train_data_text[np.argmin(text_len)])
- print("최대길이 문장: ", train_data_text[np.argmax(text_len)])

```
최소길이 문장:  담당자 통화
최대길이 문장:  박스 외부 표기 되 제품 내부 포장 되 있 제품 일치 않 어떻하 하 하 알리 줄 있
```

데이터 변환

Data Tokenizing

- from keras.preprocessing.text import Tokenizer
- from keras.preprocessing.sequence import pad_sequences
- import numpy as np
- import math

- max_words = 1000 # 데이터셋에서 가장 빈도 높은 n개의 단어만 사용한다
- maxlen = 20 # 각 문장의 길이를 고정시킨다.

- tokenizer = Tokenizer(num_words=max_words) # 상위빈도 1,000 개의 단어만을 추려내는 Tokenizer 객체 생성
- tokenizer.fit_on_texts(train_data_text) # 단어 인덱스를 구축한다
- word_index = tokenizer.word_index # 단어 인덱스만 가져온다

토크나이징 결과 확인

- print('전체에서 %s개의 고유한 토큰을 찾았습니다.' % len(word_index))
- print('word_index type: ', type(word_index))
- print('word_index: ',word_index)

```
전체에서 200개의 고유한 토큰을 찾았습니다.
word_index type:  <class 'dict'>
word_index:  {'하': 1, '확인': 2, '어떻하': 3, '있': 4, '제품': 5, '반품': 6, '어디': 7,
```

Tokenizer 연습

Tokenizing을 할 때는 그 대상들이 별개의 원소로 있어야 한다
먼저 자료가 중첩 리스트로 있을 때의 경우이다

- sample1 = [['사과 감자 옥수수'], ['딸기 감자 옥수수'], ['양파 감자 옥수수'], ['양파 부추 옥수수']]

```
{'사과 감자 옥수수': 1, '딸기 감자 옥수수': 2, '양파 감자 옥수수': 3, '양파 부추 옥수수': 4}
```

단어 분리가 안 되었음

- sample2 = [['사과', '감자', '옥수수'], ['너희', '감자', '옥수수'], ['그들', '감자', '옥수수'], ['양파', '부추', '옥수수']]

```
{'옥수수': 1, '감자': 2, '사과': 3, '너희': 4, '그들': 5, '양파': 6, '부추': 7}
```

단어 분리가 되었음

Tokenizer 연습

그러나 중첩 리스트가 아닌 단일 리스트인 경우에는 별개의 원소로 분리되지 않아도 가능하다

- sample3 = ['사과 감자 옥수수 너희 그들 양파 부추'] ← 현재 데이터 형태

```
{'사과': 1, '감자': 2, '옥수수': 3, '너희': 4, '그들': 5, '양파': 6, '부추': 7}
```

단어 분리가 되었음

- sample4 = ['사과', '감자', '옥수수', '너희', '그들', '양파', '부추']

```
{'사과': 1, '감자': 2, '옥수수': 3, '너희': 4, '그들': 5, '양파': 6, '부추': 7}
```

단어 분리가 되었음

Data Sequencing

텍스트를 숫자로 변환하는 작업을 수행한다

상위 빈도 1,000(max_words)개의 단어만 추출하여 word_index의 숫자 리스트로 변환한다.

- data = tokenizer.texts_to_sequences(train_data_text)
- print("data:", data)

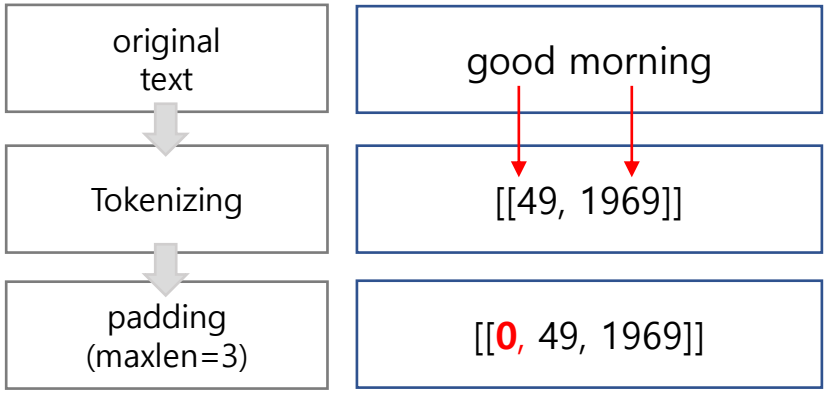
- len_d = [len(d) for d in data]
- print("길이", len_d)
- print("최대 문장 길이: ", max(len_d))
- print("최소 문장 길이: ", min(len_d))

```
data: [[13, 12, 20, 57, 2, 3, 1, 4], [5, 88, 26, 3, 1, 1], [6, 28, 46, 3, 2, 1],
길이 [8, 6, 6, 9, 7, 4, 7, 7, 8, 7, 7, 7, 9, 7, 6, 7, 7, 9, 6, 7, 7, 9, 3, 8, 7,
최대 문장 길이:  18
최소 문장 길이:  2
```

토큰으로 선정된 각 단어에 대하여 index가 배당된다

Data Padding

Padding은 데이터의 길이를 고정시켜 준다
지정된 길이에 모자라는 것은 0으로 채우고, 넘치는 것은 잘라낸다
기본값으로 단어의 선택은 뒤에서부터 한다



결과 확인

길이를 고정시킨다. maxlen의 수만큼으로 2D 텐서를 만든다
20을 넘는 데이터는 잘라내고, 모자라는 데이터는 0으로 채운다

- data = pad_sequences(data, maxlen=maxlen)

- print('data:', data)
- print('data 0:', data[0])
- print('data 0의 길이:', len(data[0]))

```
data: [[ 0  0  0 ...  3  1  4]
 [ 0  0  0 ...  3  1  1]
 [ 0  0  0 ...  3  2  1]
 ...
 [ 0  0  0 ... 41  7  2]
 [ 0  0  0 ... 41  7  2]
 [ 0  0  0 ...  2  3  1]]
data 0: [ 0  0  0  0  0  0  0  0  0  0  0  0 13 12 20 57  2  3  1  4]
data 0의 길이: 20
```

원본과의 비교

- print('train_data_text type:', type(train_data_text))
- print('data type:', type(data))
- print('train_data_text 0:', train_data_text[0])
- print('data 0:', data[0])

```
train_data_text type: <class 'list'>
data type: <class 'numpy.ndarray'>
train_data_text 0: 이번 달 샘플 수량 확인 어떻하 하 있
data 0: [ 0  0  0  0  0  0  0  0  0  0  0  0 13 12 20 57  2  3  1  4]
```

One-Hot Encoding 연습

one-hot encoding은 모든 숫자를 0과 1로만 만든다

숫자로 치환된 라벨이 다중분류에서 올바른 곳에 표시될 수 있게 한다

- sample = [[5, 6, 7], [8, 9, 10]]

- arr = np.zeros((len(sample), 10+1)) # "10"은 11번째에 들어가게 되므로 11개의 공간을 가진 np.array를 만든다 (패딩 0 고려)

- for i, seq in enumerate(sample): # 리스트가 2개이므로 i는 총 2회(0, 1) 반복되며,

  arr[i, seq] = 1. # 각 i에서 리스트의 number가 가리키는 곳의 배열 위치에 1을 기록한다

  [0, [5, 6, 7]]
  [1, [8, 9, 10]]
  [행, 열]

- arr

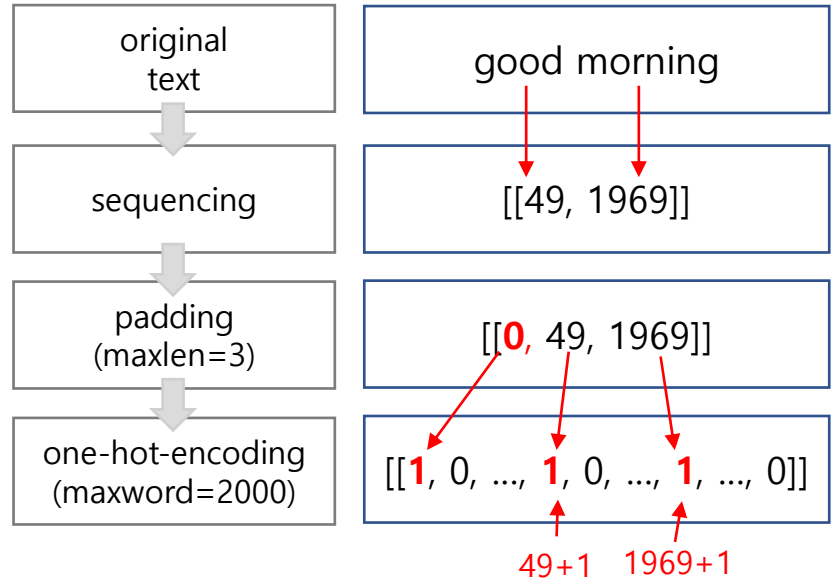
```
In[108]: arr                                    해당 열 위치에 1을 입력한다
Out[108]:
array([[0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0.],   <- i == 0
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1.]])  <- i == 1
        0   1   2   3   4   5   6   7   8   9   10
```

One-Hot Encoding 함수

```
def to_one_hot(sequences, dimension):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

# 원-핫 인코딩 수행
x_train = to_one_hot(data, dimension=max_words)
```

결과 확인



Sequencing 이전 문장

- print('train_data_text type:', type(train_data_text))
- print(len(train_data_text[0]))
- print('texts 0:', train_data_text[0])

Sequencing 이후 문장

- print('data type:', type(x_train))
- print(len(x_train[0]))
- print('x_train [0][0:100]:', x_train[0][0:100])

```
전체에서 200개의 고유한 토큰을 찾았습니다.
word_index type:  <class 'dict'>
word_index:  {'하': 1, '확인': 2, '어떻하': 3, '있': 4, '제품': 5, '반품': 6, '어디': 7,


train_data_text type: <class 'list'>
21
texts 0: 이번 달 샘플 수량 확인 어떻하 하 있


data type: <class 'numpy.ndarray'>
1000
x_train [0][0:100]: [1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 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. 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.]
```

dimension=1000으로 했으므로 각 행은 1,000개를 가지고 있다

문자열 label을 숫자로 치환

```
def labelToInt(labels):
    for count, label in enumerate(labels):
        if label == "배송비":
            labels[count] = 0
        elif label == "담당자문의":
            labels[count] = 1
        elif label == "제품가격":
            labels[count] = 2
        elif label == "배송문의":
                ……
        elif label == "교환문의":
            labels[count] = 8
        elif label == "청구금액":
            labels[count] = 9
    return labels
```

데이터의 label을 숫자로 치환

- print("훈련데이터 label 치환 전:₩n", labels)
- labels = labelToint(labels)
- print("치환 후:", labels)
- print("₩n테스트데이터 label 치환 전:₩n", labels_val)
- labels_val = labelToint(labels_val)
- print("치환 후:", labels_val)

```
훈련데이터 label 치환 전:
 ['샘플문의', '교환문의', '반품문의', '담당자문의', '담당자문의', '
치환 후: [5, 8, 7, 1, 1, 3, 8, 3, 5, 4, 9, 0, 1, 3, 7, 8, 8, 1, 7

테스트데이터 label 치환 전:
 ['반품문의', '교환문의', '배송문의', '샘플문의', '반품문의', '교환
치환 후: [7, 8, 3, 5, 7, 8, 0, 8, 8, 8, 8, 8, 9, 7, 7, 4, 0, 8, 8
```

label의 원-핫 인코딩

label을 원-핫 인코딩 한다

- class_number = 10 # 분류할 클래스의 수

- y_train = to_one_hot(train_data_label, dimension=class_number)
- print(y_train)

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

검증 및 테스트 데이터 준비

데이터가 많지 않아 별도로 검증 데이터를 만들지 않고 테스트 데이터를 활용한다

문자열을 word_index의 숫자 리스트로 변환
- data_test = tokenizer.texts_to_sequences(test_data_text)

padding으로 문자열의 길이를 고정시킨다
- data_test = pad_sequences(data_test, maxlen=maxlen)

test 데이터를 원-핫 인코딩한다
- x_test = to_one_hot(data_test, dimension=max_words)

test_data_label 원-핫 인코딩
- y_test = to_one_hot(test_data_label, dimension=class_number)
- print(y_test)

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

데이터 확인

Train 데이터와 Test 데이터 확인

- print('훈련 데이터 본문 텐서의 크기:', x_train.shape)
- print('훈련 데이터 레이블 텐서의 크기:', y_train.shape)

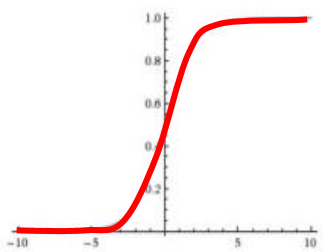
- print('테스트 데이터 텐서의 크기:', x_test.shape)
- print('테스트 레이블 텐서의 크기:', y_test.shape)

```
훈련 데이터 본문 텐서의 크기: (4741, 1000)
훈련 데이터 레이블 텐서의 크기: (4741, 10)
테스트 데이터 텐서의 크기: (1581, 1000)
테스트 레이블 텐서의 크기: (1581, 10)
```

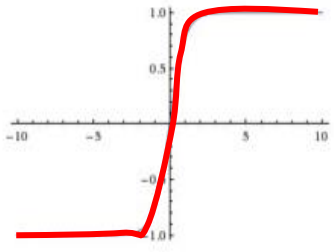
파라미터 지정

- epochs = 20 # 수행할 에포크의 수
- batch_size = 32 # 한 번에 훈련할 배치 사이즈
- model_name = 'train_data_morphed.h5' # 저장될 모델의 이름
- tokenizer_name = 'train_data_morphed.pickle' # 저장될 토크나이저의 이름

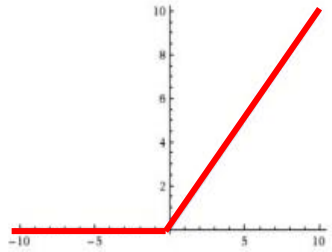
모델 설계



Sigmoid



tanh



ReLU

Define Model

- from keras.models import Sequential
- from keras.layers import Dense, Embedding, Flatten

- model = Sequential() # 모델을 새로 정의

- model.add(Dense(64, activation='relu', input_shape=(max_words,))) # 첫 번째 은닉층
- model.add(Dense(units=32, activation='relu')) # 두 번째 은닉층
- model.add(Dense(units=class_number, activation='softmax')) # 출력층

- model.summary()

conceptually,

sigmoid: $P(class1)$

softmax: $\frac{P(classN)}{P(class1)+P(class2)+P(class3)+\cdots}$

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                64064
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 66,474
Trainable params: 66,474
Non-trainable params: 0
_________________________________________________________________
```

모델 경로 생성

- from keras.callbacks import EarlyStopping
- import os

- checkpoint_path = chat_dir + 'models/'
- checkpoint_dir = os.path.dirname(checkpoint_path)

- if os.path.exists(checkpoint_dir):
- print("{} -- Folder already exists₩n".format(checkpoint_dir))
- else:
- os.makedirs(checkpoint_dir, exist_ok=True)
- print("{} -- Folder created₩n".format(checkpoint_dir))

Compile & Train Model

```
Epoch 1/20
134/149 [=======================>....] - ETA: 0s - loss: 0.9665 - acc: 0.7927
Epoch 1: val_acc improved from -inf to 0.98355, saving model to /content/gdrive/My Drive
149/149 [==============================] - 4s 9ms/step - loss: 0.8885 - acc: 0.8100 - va
Epoch 2/20
145/149 [============================>.] - ETA: 0s - loss: 0.0452 - acc: 0.9940
Epoch 2: val_acc improved from 0.98355 to 0.99557, saving model to /content/gdrive/My Dr
149/149 [==============================] - 1s 4ms/step - loss: 0.0443 - acc: 0.9941 - va
Epoch 3/20
143/149 [===========================>..] - ETA: 0s - loss: 0.0050 - acc: 0.9989
Epoch 3: val_acc improved from 0.99557 to 1.00000, saving model to /content/gdrive/My Dr
149/149 [==============================] - 1s 4ms/step - loss: 0.0050 - acc: 0.9989 - va
Epoch 4/20
149/149 [==============================] - ETA: 0s - loss: 8.6994e-04 - acc: 1.0000
Epoch 4: val_acc did not improve from 1.00000
149/149 [==============================] - 1s 4ms/step - loss: 8.6994e-04 - acc: 1.0000
```

Compile Model

- model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])

과적합 되기 전의 가장 좋은 모델에서 중단하는 기능 사용
32개씩 미니 배치를 만들어 n번의 epoch로 훈련. 배치 개수는 보통 8~512개 중에서 선택
훈련 데이터로 훈련하고, 검증 데이터로 검증한다
반환값의 history는 훈련하는 동안 발생한 모든 정보를 담고 있는 딕셔너리

- from keras.callbacks import ModelCheckpoint
- earlystop_callback = EarlyStopping(monitor='val_acc', patience=1)
- cp_callback = ModelCheckpoint(filepath=checkpoint_path+model_name, monitor='val_acc', verbose=1, save_best_only=True)
- history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_test, y_test), callbacks=[earlystop_callback, cp_callback], verbose=1)
- history_dict = history.history

Save Model

만들어진 모델을 이후에 재사용할 수 있도록 저장한다

- import pickle

model.save(model_name)

같은 단어를 추출하기 위해 훈련데이터에서 사용된 상위빈도 1,000개의 단어로 된 Tokenizer 저장

- with open(tokenizer_name, 'wb') as file:
- pickle.dump(tokenizer, file, protocol=pickle.HIGHEST_PROTOCOL)

※ pickle 프로토콜 최신 버전 사용

모델 성능 확인

Accuracy & Loss
history 딕셔너리 안에 있는 정확도와 손실값을 가져와 본다

- acc = history.history['acc']
- val_acc = history.history['val_acc']
- loss = history.history['loss']
- val_loss = history.history['val_loss']

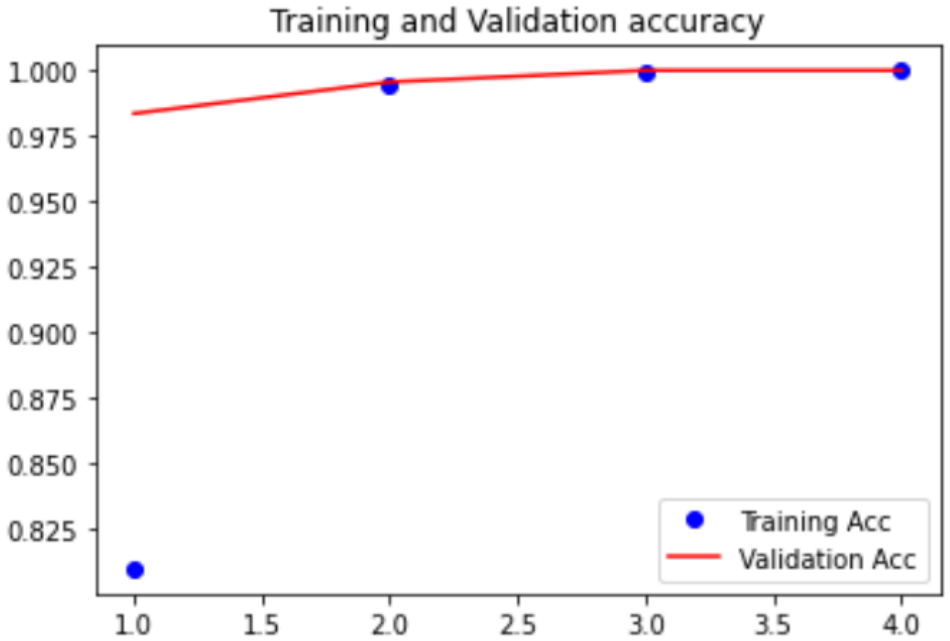
- print('Validation accuracy of each epoch:', np.round(val_acc, 3))
- epochs = range(1, len(val_acc) + 1)

```
Validation accuracy of each epoch: [0.984 0.996 1.    1.    ]
```

Plotting Accuracy

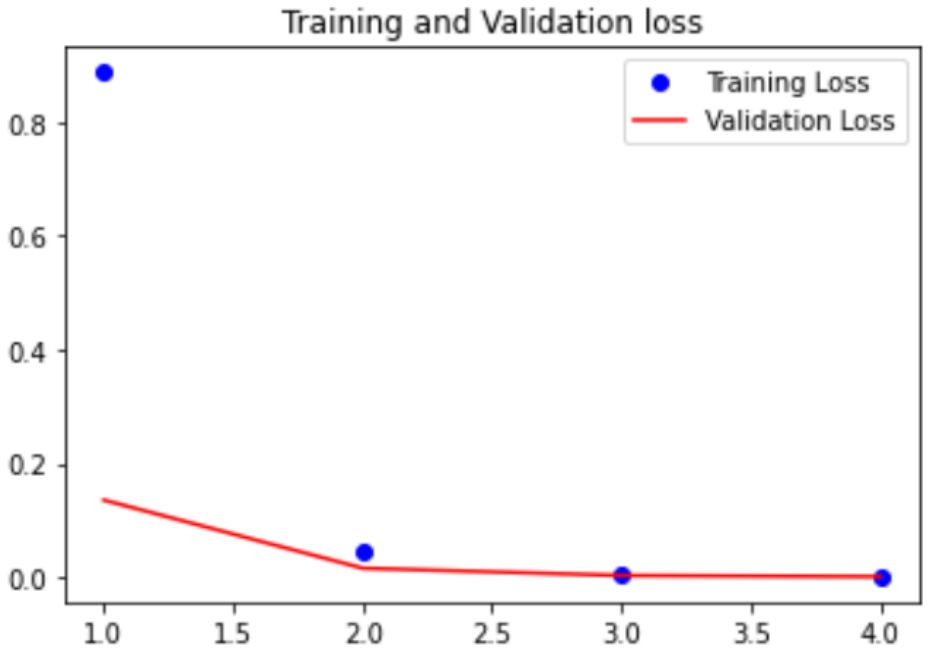
- import matplotlib.pyplot as plt

- plt.plot(epochs, acc, 'bo', label='Training Acc')
- plt.plot(epochs, val_acc, 'r', label='Validation Acc')
- plt.title('Training and Validation accuracy')
- plt.legend(loc=4)
- plt.show



Plotting Loss

- plt.plot(epochs, loss, 'bo', label='Training Loss')
- plt.plot(epochs, val_loss, 'r', label='Validation Loss')
- plt.title('Training and Validation loss')
- plt.legend(loc=1)
- plt.show()



Load Model

저장된 모델을 불러와 사용한다

- from keras.models import load_model

- loaded_model = load_model(model_name)

- with open(tokenizer_name, 'rb') as handle:
- loaded_tokenizer = pickle.load(handle)

Test Data Evaluation

모델에 분류할 데이터와 그 정답을 같이 넣어준다

- test_eval = loaded_model.evaluate(x_test, y_test)

모델이 분류한 결과와 입력된 정답을 비교한 결과

- print('prediction model loss & acc:', test_eval)

```
50/50 [==============================] - 0s 3ms/step - loss: 0.0027 - acc: 1.0000
prediction model loss & acc: [0.0026912169996649027, 1.0]
```

숫자형 라벨을 답변으로 치환

```
def intToLabel(label_int):
    labels = ''
    if label_int == 0:
        labels = '20,000원 이상 주문하시면 배송비가 없습니다'
    elif label_int == 1:
        labels = '담당자는 홈페이지에 사번을 넣으시면 자세한 정보를 알 수 있습니다'
    elif label_int == 2:
        labels = '공급가와 소비자가는 홈페이지 > ... > 가격조회 에서 확인 가능합니다'
                                        ......
    elif label_int == 9:
        labels = '청구금액은 매달 12일 이후에 홈페이지 > ... > 청구금액에서 확인 가능합니다'
    return labels
```

1문장 예측

- user_input = input("내용을 입력하세요: ")
- morphed_input, poses = rhinoMorph.wholeResult_list(rn, user_input, pos=['NNG', 'NNP', 'NP', 'VV', 'VA', 'XR', 'VCN', 'MAG', 'MAJ', 'IC']) # 형태소와 품사를 모두 가져온다
- text = [morphed_input]

- data = loaded_tokenizer.texts_to_sequences(text)
- data = pad_sequences(data, maxlen=maxlen)
- x_test = to_one_hot(data, dimension=max_words)

- prediction = loaded_model.predict(x_test)
- label_int = np.argmax(prediction)
- label = intToLabel(label_int)
- print(label)

```
내용을 입력하세요: 배송기간은 어떻게 돼?
배송에는 보통 2일이 소요되며, 빠른 배송을 선택하시면 1일 안에 책임배달합니다
```

테스트 예문

- 제품을 교환하고 싶어요
- 매장 코드 번호 확인은 어떻게 해요?
- 이번 달 청구금액 확인은 어디서 하나요?
- 제품에 표기된 제품과 내부포장 제품이 다른데 어떡하죠?
- 담당자 번호 확인은 어떻게 해요?
- 이번 달 반품 마감일 확인 어디서 가능합니까?
- 배송 제품은 언제 도착하나요?
- 샘플 신청은 어떻게 하나요?
- 제품의 공급가와 소비자가 알고싶어요
- 얼마이상 주문해야 배송비 안 붙나요?