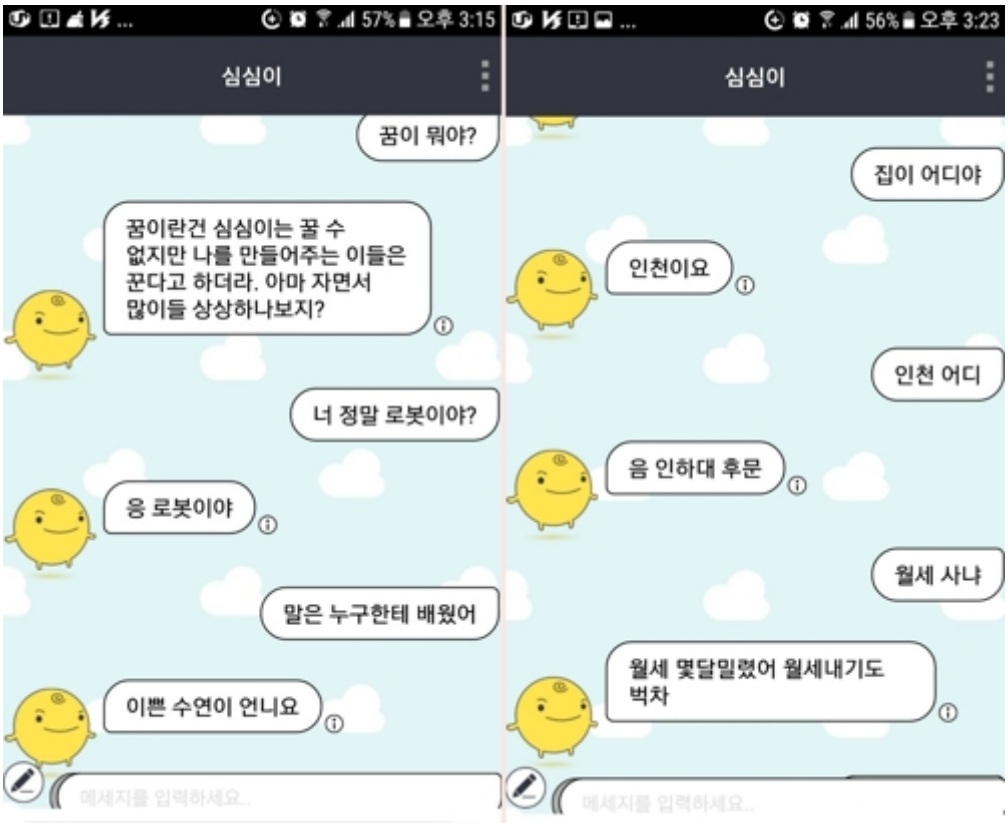
12. 챗봇

챗봇은 인간이 작성한 자연어를 이해하고 적절한 대응을 하며 인간과 대화할 수 있도록 생성된 컴퓨터 프로그램이다. 메신저와 같이 대화형 인터페이스를 가진다. 크게 규칙 기반 방식과 기계 학습 방식 그리고 하이브리드 방식이 있다.



모바일 메신저 이용자가 많아지면서 챗봇이 주목을 받게 되었다. 채팅앱인 WhatsApp 이용자는 2014년에 약 4억 3천만명에서 2017년에는 13억명으로 증가했다. 한국에서도 대부분의 사람이 카카오톡이나 페이스북 메신저를 쓰고 있다. 챗봇은 모바일 메신저와 결합하여 자연어로 사용하기 때문에 고객이 접근하고 사용하기가 쉽다. 또한 자연어 처리의 발전으로 챗봇 수준의 대화는 원활히 처리할 수 있게 되었다.

챗봇 관련 다양한 솔루션이 개발되어 있다. 구글의 다이얼로그플로, 마이크로소프트의 LUIS 등을 사용하면 기계학습과 결합된 챗봇을 손쉽게 만들 수 있다.

12.1. 챗봇의 역사

12.1.1. 튜링 테스트

1950년대에 앨런 튜링(Alan Turing)은 튜링 테스트(Turing Test)를 제안한다. 이 테스트에서 심판은 상대방이 사람 또는 컴퓨터인지 모르는 상태로 종이에 글을 써서 주고 받는 방식으로 대화를 한다. 만약 충분히 대화를 주고 받아도 심판이 사람과 컴퓨터를 구별할 수 없다면, 그 컴

퓨터는 인간 수준의 지능을 가졌다고 볼 수 있다는 것이 튜링의 주장이다. 1990년부터 실제로 이러한 방식의 테스트로 가장 뛰어난 챗봇에 주는 뢰브너 상이 제정되었다.

12.1.2. 중국어 방

철학자 존 설(John Searle)은 "중국어 방"이라는 사고 실험을 통해 튜링 테스트로 기계의 인공지능 여부를 알 수 없다고 반박한다.

"중국어 방"의 내용은 다음과 같다. 어떤 방에 중국어를 모르는 사람이 들어가 있다. 이 방에는 중국어 질문에 대한 답변 목록이 있다. 밖에서 중국어로 된 질문을 종이에 써서 방의 안에 넣어주면 방 안의 사람은 답변 목록에서 적절한 답변을 찾아 방 밖으로 보내게 된다. 방의 외부에서 관찰할 때는 마치 중국어를 할 줄 아는 사람이 방 안에 있는 것처럼 보이지만 실제로는 그렇지 않다는 것이 설의 지적이다.

"중국어 방"은 매우 간단해 보이지만 '안다'는 것은 무엇인가, '지능'이란 무엇인가 등의 여러 가지 심오한 문제들이 엮여 있다. 그러나 이러한 문제들과 떨어져서 실용적으로만 생각해보면 어떤 한정된 맥락에서는 단순한 답변 목록조차도 진정한 지능을 가진 것처럼 보일 수 있다는 의미도 된다.

12.1.3. 엘리자

1966년에 MIT 교수 조셉 와이젠바움(Joseph Weizenbaum)은 엘리자(ELIZA)라는 초보적인 챗봇을 만들었다. 엘리자는 칼 로저스의 인간중심 심리상담의 원리를 따라 상담가의 역할을 수행한다. 실제로 엘리자는 매우 단순한 규칙에 따라 작동하는 챗봇이다. 이젠바움은 인공지능이라는 것이 실제로는 얼마나 조악한 것인지 보여주려고 엘리자를 만들었지만, 많은 사람들이 엘리자와의 대화에 깊이 빠져드는 것을 보고 깜짝 놀랐다고 전해진다.

```
================================================================================
          EEEEEEEE  L         IIIIIII  ZZZZZZZ      AAA
          E         L            I          Z      A   A
          E         L            I         Z      A     A
          EEEEE     L            I        Z       A     A
          E         L            I       Z        AAAAAAA
          E         L            I      Z         A     A
          EEEEEEEE  LLLLLLLL  IIIIIII  ZZZZZZ     A     A

================================================================================
ELIZA > (PLEASE TYPE IN ALL CAPS) WHAT'S YOUR NAME DEAR ?
AMIT
ELIZA > HI, I'M ELIZA. WHAT DO YOU WANT TO TALK ABOUT ?
AMIT > CAN I TALK ABOUT MY PROBLEM ?
ELIZA > SURE... !
AMIT > DUE TO ON GOING EXAMS IN COLLEGE, I HAVE A LOT OF STRESS.
ELIZA > PLEASE RELAX A LITTLE. SLEEP WELL.
AMIT > THANKS FOR YOUR ADVICE.
ELIZA > NO MENTION.
AMIT > BYE
ELIZA > BYE AND KEEP IN TOUCH...

================================================================================
```

1972년 케네스 콜비는 엘리자를 흉내내어 정신분열증 환자를 흉내낸 패리(Parry)를 개발하였다. 1972년 컴퓨터 통신 국제 학회(ICCC)에서 열린 엘리자와 패리의 대화가 유명하다.

12.1.4. ALICE

1995년에 리처드 월리스(Richard Wallace)가 ALICE (Artificial Linguistic Internet Computer Entity)를 개발하였다. ALICE는 챗봇 프로그램을 직접 작성하지 않아도 대화 규칙을 간단한 템플릿으로 만들 수 있는 기능을 가지고 있었다. ALICE에 기반한 챗봇은 여러 번 뢰브너 상을 수상하기도 했다.

12.2. 인텐트와 엔티티

챗봇을 완전히 기계학습으로 만들려면 수많은 대화 사례를 모아서 학습을 시켜야 한다. 질문-답변의 쌍을 학습시켜서, 사용자가 새로운 질문을 하면 그에 맞는 새로운 답변을 만들어내는 것이다. 이 방법은 학습시키기가 어렵다. 특히 기존의 답변과 비슷한 답변을 하도록 학습하게 하면, 어떤 질문에도 "I don't know" 같은 답변을 하게 되기도 한다. 왜냐하면 어느 질문에도 가능한 흔한 답변이기 때문이다. 또한 현실적으로도 문제가 있다. 기업의 콜센터에서 기존의 질문-답변으로 챗봇을 학습시켰다면 상품이나 정책이 변하더라도 계속해서 기존의 학습된 방식으로만 답변을 하게 된다.

이러한 이유로 현실의 챗봇들은 기계학습과 규칙의 하이브리드 방식을 사용한다. 챗봇에서 기계학습이 사용되는 부분은 주로 사용자의 발언을 분류하고 주요한 키워드들을 뽑아내는 것이다. 예를 들어 "피자를 주문해줘"라고 하면 사용자의 의도는 '주문'으로 분류할 수 있고 주요한 키워드는 '피자'라는 메뉴이다. 이렇게 사용자의 발언 전체의 분류는 **인텐트**(intent)라고 하고, 발언에서 메뉴나 장소, 시간 등에 해당하는부분들은 **엔티티**(entity)라고 한다. 즉, 인텐트 분류와 엔티티 추출이 챗봇에서 기계학습이 활용되는 부분이라고 할 수 있다.

12.3. 시퀀스 레이블링

엔티티 추출에는 시퀀스 레이블링(sequence labeling)을 사용한다. 이는 텍스트와 같은 시퀀스에서 시퀀스 전체가 아닌 각각의 값에 대해 레이블을 붙이는 것이다. 예를 들어 "피자 주문"이라는 문장에서 각각의 글자에 대해 엔티티를 나타내는지(1) 그렇지 않은지(0)로 레이블을 붙이는 것이다.

```
피 1
자 1
주 0
문 0
```

"피자콜라하나씩"이라는 문장에는 '피자'와 '콜라'라는 두 개의 엔티티가 연달아 나온다. 위에서처럼 0/1로만 레이블을 붙이면 구별이 어렵기 때문에 **BIO 방식**을 많이 사용한다(IOB 방식이라고도 한다). 먼저 B는 Beginning의 약자로 엔티티의 시작을 나타낸다. I는 Inside로 엔티티의 내부, 즉 중간이나 끝을 가리킨다. 마지막으로 O는 Outside로 엔티티가 아닌 부분을 나타낸다.

```
피 B
자 I
콜 B
라 I
하 O
나 O
씩 O
```

만약 엔티티가 여러 종류가 있으면 종류별로 B와 I를 붙인다. 예를 들어 메뉴와 장소 2종의 엔티티가 있다면 `메뉴B`, `메뉴I`, `장소B`, `장소I`, `O` 로 5개의 레이블이 생기게 된다.

```
피 메뉴B
자 메뉴I
한 O
판 O
집 장소B
으 O
로 O
```

물론 기계학습을 사용하지 않고 '피자', '집' 등 엔티티에 해당하는 단어를 사전으로 만들어서 분류하는 간단한 방법도 있다. 그러나 엔티티에 해당하는 표현들은 계속해서 새로 생겨나고, 사용자가 항상 정확한 표현을 쓰는 것도 아니기 때문에 사전만으로는 한계가 있다.

기계학습을 사용할 경우 사전에는 "소시지 피자"가 등록되어 있는데 사용자가 "소세지 피자 주문해줘"라고 하면 "소세지 피자"를 추출한 다음 사전에서 가장 비슷한 표현인 "소시지 피자"을 찾는 방식으로 작동한다.

12.5. 엔티티 추출

12.5.1. CoNLL-2003 데이터셋

CoNLL-2003 데이터셋을 다운받는다. 영어 엔티티 추출을 다루는 대회의 데이터셋이다.

```
import requests
```

```
res = requests.get('https://raw.githubusercontent.com/Franck-Dernoncourt/NeuroNER/mast
```

데이터셋은 한 줄에 한 단어씩 표기되어 있고 문장과 문장 사이는 빈 줄로 구분되어 있다. 각 단어의 뒤에는 단어의 품사와 구문, 그리고 엔티티 레이블이 붙어있다.

```
print(res.text[:95])
```

먼저 줄바꿈이 두 번 들어간 부분을 기준으로 끊어 문장 단위로 나눈다.

```
sentences = res.text.split('\n\n')
```

다음으로 문장을 라인으로 나누고 단어와 레이블만 뽑아서 리스트로 바꾼다. 이 과정에 알파벳과 하이픈으로 된 단어의 종류도 함께 수집한다.

```
from collections import Counter
```

```
data = []
word_count = Counter()
label_set = set()

for sentence in sentences[1:]:
    datum = []
    for line in sentence.splitlines():
        word, _, _, label = line.split()
        word = word.lower()

        word_count[word] += 1
        label_set.add(label)

        datum.append((word, label))
    data.append(datum)
```

3번 문장의 첫 10단어만 확인해보자. 엔티티의 종류로는 단체(ORG) 인물(PER)과 장소(LOC) 그리고 기타(MISC)가 있다.

```
data[3][:10]
```

```
[('the', 'O'),
 ('european', 'B-ORG'),
 ('commission', 'I-ORG'),
 ('said', 'O'),
 ('on', 'O'),
 ('thursday', 'O'),
 ('it', 'O'),
 ('disagreed', 'O'),
 ('with', 'O'),
 ('german', 'B-MISC')]
```

2만 종 이상의 단어가 있다.

```
len(word_count)
```

```
21010
```

12.5.2. 전처리

단어 목록을 정렬한다.

```
import operator
```

```
word_list = sorted(word_count.items(), key=operator.itemgetter(1), reverse=True)
```

10번 넘게 나온 단어들을 많이 나온 순서대로 단어에 번호를 붙인다. 단, 패딩을 위해 0번과 OOV(Out of Vocabulary)를 위해 1번을 비워둔다.

```
word_index = {w: i + 2 for i, (w, n) in enumerate(word_list) if n > 10}
```

잘 나오지 않는 단어를 제외하고 2천여개 단어에만 번호가 붙었다.

```
len(word_index)
```

```
2180
```

레이블은 모두 9종이 있다.

```
label_set
```

```
{'B-LOC', 'B-MISC', 'B-ORG', 'B-PER', 'I-LOC', 'I-MISC', 'I-ORG', 'I-PER', 'O'}
```

레이블에 모두 번호를 붙인다.

```
label_index = {label: i for i, label in enumerate(label_set)}
```

```
index_label = {i: label for label, i in label_index.items()}
```

단어와 레이블을 모두 번호로 붙인다.

```
xs = []
ys = []

for datum in data:
    x = []
    y = []

    for word, label in datum:
        x.append(word_index.get(word, 1))
        y.append(label_index[label])

    xs.append(x)
    ys.append(y)
```

```
xs[0]
```

```
[990, 1, 206, 630, 7, 1, 217, 1, 3]
```

```
ys[0]
```

```
[7, 0, 4, 0, 0, 0, 4, 0, 0]
```

패딩을 한다.

```
from keras.preprocessing.sequence import pad_sequences
```

```
pad_xs = pad_sequences(xs, padding='post')
```

레이블은 패딩을 할 때 0대신 'O' 로 패딩한다.

```
pad_ys = pad_sequences(ys, padding='post', value=label_index['O'])
```

데이터를 분할한다.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(pad_xs, pad_ys, test_size=.2, rand
```

y_train 에 한 차원을 덧붙여준다.

```
import numpy
```

```
y_train = numpy.expand_dims(y_train, 2)
```

문장의 최대 길이와 단어 수를 구한다.

```
_, MAXLEN = pad_xs.shape
NUM_WORDS = len(word_index) + 2
```

```
NUM_LABELS = len(label_index)
```

12.5.3. LSTM

가장 간단하게 생각해볼 수 있는 모형은 LSTM이다.

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

```
lstm = Sequential()
lstm.add(Embedding(input_dim=NUM_WORDS, output_dim=8, input_length=MAXLEN, mask_zero=T
lstm.add(LSTM(16, return_sequences=True))
lstm.add(TimeDistributed(Dense(NUM_LABELS, activation='softmax')))
```

```
lstm.summary()
```

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

```
lstm.fit(X_train, y_train, epochs=2)
```

```
<keras.callbacks.History at 0x231a7f30be0>
```

12.5.4. 양방향 LSTM

```
from keras.layers import Bidirectional
```

양방향 LSTM은 `Bidirectional` 을 사용하는 것 외에 큰 차이는 없다.

```
bilstm = Sequential()
bilstm.add(Embedding(input_dim=NUM_WORDS, output_dim=8, input_length=MAXLEN, mask_zero
bilstm.add(Bidirectional(LSTM(16, return_sequences=True)))
bilstm.add(TimeDistributed(Dense(NUM_LABELS, activation='softmax')))
```

LSTM이 순방향과 역방향 2개가 들어가기 때문에 출력이 16에서 32로 늘어난다.

```
bilstm.summary()
```

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

```
bilstm.fit(X_train, y_train, epochs=2)
```

```
<keras.callbacks.History at 0x2319d345940>
```