Machine Learning

10장 머신번역

고려대학교 통계학과
박유성

Contents

머신 번역(Machine translation)

- 머신번역은 sequence-to-sequence 학습의 대표적인 응용분야이다.
- 머신번역은 제 9장에서 논의한 function API를 이용하여 여러 개의 입력과 여러 개의 출력(many-to-many) 아키텍쳐를 적용한 딥러닝모형이다.
- 딥러닝의 난제인 머신번역에 적용된 many-to-many 아키텍쳐는 기대만큼 좋은 성능을 보여주지 못하고 있다.
- 두 번째 머신번역 모형으로, 좀 더 개선된 머신번역을 위해 번역된 언어를 입력데이터로 재사용하는 방법을 논의할 것이다.
- 예를 들어, 영어를 불어로 번역할 때, 영어뿐만 아니라 영어단어에 대응되는 불어단어를 동시에 입력하여 이어지는 불어로의 번역 정밀도를 높이는 sequence-to-sequence 학습방법이다.

머신 번역(Machine translation)

- 목적변수인 불어를 입력으로 이용하는 방법을 teacher forcing이라고 한다. teacher forcing을 이용한 머신번역은 2개의 입력을 가지고 있으므로 이 2개의 입력을 연결하기 위해 encoder모형과 decoder모형으로 구성되어 있다.
- 세 번째 머신번역 모형은 소위 attention을 이용한 머신번역이다.
- attention은 입력언어와 출력언어의 의미상 유사성을 도출한 후, 이 유사성에 비례한 가중치를 입력언어에 부여하여 머신번역의 성능을 향상시키는 기법이다.

01 자료의 사전정리

- 텍스트자료에 제공된 언어의 번역은 순서가 매우 중요하므로 RNN 모형을 적용하기 위한 자료의 사전정리는 매우 중요하다. 먼저 다음과 같이 자료를 내려 받도록 하자.

```
import pandas as pd
import numpy as np
import string
lines= pd.read_table('C:/Users/ysp/Desktop/Deep Learning/english to french.txt', names=['eng', 'fr'])
```

- 위 프로그램에서 pd.read_table은 텍스트자료가 '₩t' 또는 공백(' ')으로 구분되어 있을 때, 텍스트자료를 읽는 함수이다.
- english to french.txt는 하나의 행에 영어가 있고 몇 개의 공간을 띄고 번역된 프랑스어가 있는 텍스트자료이므로 pd.read_table을 사용하고 있다.

01 자료의 사전정리

- 참고로 텍스트자료가 ‘,’로 구분되어 있으면 pd.read_csv를 사용한다.
- 원래의 자료는 총 140,000개의 문장으로 구성되어 있지만 학습시간을 줄이기 위해 아래와 같이 첫 50,000개의 문장으로 데이터를 구성하였고, 예제에서 볼 수 있듯이 영어문장과 불어로 해석된 문장으로 구성되어 있다.
- 영어와 불어는 빈칸으로 구별되어 있고, 대문자와 소문자가 섞여있고 느낌표 등의 punctuation을 포함하고 있다.

```
lines = lines[0:50000]
print(lines.head(3))
print(lines.tail(3))
```

```
   Eng    fr
0 Go. Va !
1 Run! Cours !
2 Run! Courez !
                eng                        fr
49997 They go to work on foot. Ils vont au travail à pied.
49998 They got into the train. Ils montèrent dans le train.
49999 They got into the train. Elles montèrent dans le train.
```

01 자료의 사전정리

- 아래 프로그램에서는 x.lower()함수를 이용하여 영어 및 불어 단어를 모두 소문자로 만들었고, string.punctuation을 이용하여 “#, }, *, <, !”등의 부호를 문장으로부터 제거하고 있다.
- ‘apple, pear, grape’.split(‘,’)=[‘apple’,‘pear’,‘grape’]이고
- ‘ ’.join([‘apple’,‘pear’,‘grape’])=’apple pear grape’
- 아래 프로그램 결과에서 볼 수 있듯이 각 문장의 단어는 빈칸으로 구분된 것을 볼 수 있다.

01 자료의 사전정리

```
lines['eng']=lines['eng'].apply(lambda x: x.lower())
lines['fr']=lines['fr'].apply(lambda x: x.lower())
exclude = set(string.punctuation)
lines.eng=lines.eng.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
lines.fr=lines.fr.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
print(lines.head(3))
print(lines.tail(3))
```

```
  Eng  fr
0 go va
1 run cours
2 run courez
         eng                                fr
49997 they go to work on foot ils vont au travail à pied
49998 they got into the train ils montèrent dans le train
49999 they got into the train elles montèrent dans le train
```

01 자료의 사전정리

- 아래 프로그램과 같이 번역될 프랑스어 문장은 모두 'start'로 시작하고 'end'로 끝나도록 하고 있다. 10.2절과 10.3절에서 decoder 입력 자료의 예측치를 정의할 때 유용하기 때문이다.
- lines의 크기는 (50000,2)인 것을 확인할 수 있다.

```
lines.fr = lines.fr.apply(lambda x : 'start '+ x + ' end')
print(lines.head(3))
print(lines.tail(3))
print(lines.shape)
```

```
   eng       fr
0 go start va end
1 run start cours  end
2 run start courez  end
         eng                                    fr
49997 they go to work on foot start ils vont au travail à pied end
49998 they got into the train start ils montèrent dans le train end
49999 they got into the train start elles montèrent dans le train end
(50000, 2)
```

01 자료의 사전정리

- 다음 프로그램은 영어와 프랑스어를 토큰화하여 빈도수를 기준으로 80%까지 차지하는 토큰(단어)으로 자료를 재구성하고 있다.
- 아래와 같이 Tokenizer( )함수를 적용한 객체화 변수 eng_tokenizer나 fr_tokenizer에 word_counts 속성을 부여하면 즉,

  eng_tokenizer.word_counts 또는 fr_tokenizer.word_counts

- 를 부여하면 단어와 발생빈도(words and their counts)를 관측순서대로 정리된 dictionary데이터를 만들게 된다. 이를 OrderedDict라고 한다.
- 그러나 이러한 dictionary 데이터는 pandas의 DataFrame으로 읽을 수가 없다. 그러므로 이 데이터를 순수 dictionary로 바꿔야 한다

01 자료의 사전정리

- json.dumps( )함수를 이용하여 ordereddict 형태의 자료를 json 형식으로 바꾼 후, 다시 jason.loads( )함수를 이용하여 jason 형태의 자료로 전환하면 순수 dictionary 자료를 출력하게 된다.
- 아래 결과에 의해, 80%까지 차지하는 총 단어의 수는 영어인 경우, 384 단어이고 프랑스는 총 357 단어인 것으로 나타났다.
- 영어 단어는 final_eng_words에 저장되어 있고 총 357개의 프랑스어 단어는 final_fr_words에 저장되어 있다.
- 80%에 해당하지 않은 단어를 하나의 클래스로 하면, 385개의 영어 단어를 입력하여 358개 프랑스 단어를 구별하는 문제로 요약할 수 있다.
- 즉, 클래스가 358개인 분류문제가 되고 손실함수는 softmax 함수에 의해 계산된 categorical crossentropy가 된다.

01 자료의 사전정리

```
from keras.preprocessing.text import Tokenizer
def create_tokenizer(lines):
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
return tokenizer
import json
eng_tokenizer = create_tokenizer(lines['eng'])
eng_dict=json.loads(json.dumps(eng_tokenizer.word_counts))
df =pd.DataFrame([eng_dict.keys(), eng_dict.values()]).T
df.columns = ['word','count']
df = df.sort_values(by='count',ascending = False)
df['cum_count']=df['count'].cumsum()
df['cum_perc'] = df['cum_count']/df['cum_count'].max()
final_eng_words = df[df['cum_perc']<0.8]['word'].values
fr_tokenizer = create_tokenizer(lines['fr'])
fr_dict = json.loads(json.dumps(fr_tokenizer.word_counts))
df =pd.DataFrame([fr_dict.keys(), fr_dict.values()]).T
df.columns = ['word','count']
df = df.sort_values(by='count',ascending = False)
df['cum_count']=df['count'].cumsum()
df['cum_perc'] = df['cum_count']/df['cum_count'].max()
final_fr_words = df[df['cum_perc']<0.8]['word'].values
print(len(final_eng_words),len(final_fr_words))
```

```
384 357
```

01 자료의 사전정리

- 아래 프로그램은 80%에 포함되지 않은 단어를 unknown이라는 의미인 'unk'로 바꾸고 단어 사이를 빈칸으로 만들어서 단어를 구별하고 있다. 아래 예제에서 'extremely'는 'unk'에 속하는 단어임을 알 수 있다.

```
def filter_eng_words(x):
    t = []
    x = x.split()
    for i in range(len(x)):
        if x[i] in final_eng_words:
            t.append(x[i])
        else:
    t.append('unk')
    x3 = ''
    for i in range(len(t)):
        x3 = x3+t[i]+' '
    return x3
filter_eng_words('he is extremely good')
```

```
'he is unk good '
```

01 자료의 사전정리

- 다음 프로그램에서도 80%에 해당하지 않은 프랑스어도 동일하게 'unk'로 전환하고 있다.
- 그런데 filter_eng_words(x)나 filter_fr_words(x)에서 입력되는 x가 하나의 행이면, 예를 들어 lines['eng']가 하나의 행이면 일반적인 함수적용 방법대로 filter_eng_words(lines['eng'])로 적용하면 원하는 결과를 구할 수 있다.
- 그러나 lines['eng']나 lines['fr']은 2D텐서 자료이므로

  lines['eng']=lines['eng'].apply(filter_eng_words) 그리고

  lines['fr']=lines['fr'].apply(filter_eng_words)
- 으로 하면 lines['eng']와 lines['fr']을 한 행씩 각각의 filter_eng_words와 filter_fr_words함수를 적용한다. 물론 apply( )함수에 axis를 줘서 다른 axis에도 적용할 수 있다. default는 axis=0이다.

01 자료의 사전정리

```
def filter_fr_words(x):
    t = []
    x = x.split()
    for i in range(len(x)):
        if x[i] in final_fr_words:
            t.append(x[i])
        else:
    t.append('unk')
    x3 = ''
    for i in range(len(t)):
        x3 = x3+t[i]+' '
    return x3
lines['eng']=lines['eng'].apply(filter_eng_words)
lines['fr']=lines['fr'].apply(filter_fr_words)
```

01 자료의 사전정리

- 다음 프로그램은 데이터에 있는 유일한 영어와 프랑스어 단어의 집합을 만들고 있다.

```
all_eng_words=set()
for eng in lines.eng:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)
all_french_words=set()
for fr in lines.fr:
    for word in fr.split():
        if word not in all_french_words:
            all_french_words.add(word)
input_words = sorted(list(all_eng_words))
target_words = sorted(list(all_french_words))
num_encoder_tokens = len(all_eng_words)
num_decoder_tokens = len(all_french_words)
```

01 자료의 사전정리

- 아래 프로그램에 의해 all_french_words와 final_fr_words의 차이는 추가한 'unk'임을 보여 주고 있으며 all_eng_words는 'unk'를 포함하여 총 385개 단어, all_french_words는 'unk'를 포함하여 358개의 단어를 가지고 있다.

```
print(set(all_french_words) - set(final_fr_words))
print(len(all_eng_words))
print(len(all_french_words))
```

```
{'unk'}
385
358
```

01 자료의 사전정리

- 다음 프로그램은 all_eng_words와 all_french_words를 dictionary 자료형태로 바꾸어 주는 과정이다. {word, index}의 형태로 만들고 있으며 index는 1부터 시작하게 만들고 있다.
- 아래와 같이 'unk'는 336번으로 index가 부여되었고, 'start'는 284, 그리고 'end'는 89번으로 index가 부여되었다. 역으로 index를 부여하고 대응되는 단어를 찾기 위해서는 index-1를 주어야 대응되는 단어를 찾을 수 있다. index를 부여할 때 1부터 부여했기 때문이다.

01 자료의 사전정리

```
input_token_index = dict([(word, i+1) for i, word in enumerate(input_words)])
target_token_index = dict([(word, i+1) for i, word in enumerate(target_words)])
print(input_token_index['unk'])
print(target_token_index['start'])
print(target_token_index['end'])
print(list(input_token_index.keys())[335])
print(list(target_token_index.keys())[283])
print(list(target_token_index.keys())[88])
```

```
336
284
89
unk
start
end
```

01 자료의 사전정리

- 다음은 RNN 자료에서 시간순서(time steps)를 정의하기 위해 가장 긴 문장을 찾아내고 그 문장 내에 있는 단어 수를 정의하고 있다.
- 영어의 경우, 8개 단어로 구성된 문장이 가장 긴 문장이고 프랑스어의 경우, 17개 단어로 구성된 문장이 가장 긴 문장인 것으로 나타났다. 이러한 8과 17은 모든 문장의 시간순서로 정의하게 된다.

```
length_list=[]
for l in lines.fr:
    length_list.append(len(l.split(' ')))
fr_max_length = np.max(length_list)
length_list=[]
for l in lines.eng:
    length_list.append(len(l.split(' ')))
eng_max_length = np.max(length_list)
print(eng_max_length)
print(fr_max_length)
```

```
8
17
```

01 자료의 사전정리

- 아래 프로그램은 두 개의 입력변수 encoder_input_data, decoder_input_data와 목적변수 decoder_target_data를 정의하고 있다.
- encoder_input_data는 (50000, 8), decoder_input_data는 (50000,17), 그리고 decoder_target_data는 (50000, 17, 359)의 크기를 가지고 있다.
- decoder_target_data의 마지막 축이 num_decoder_tokens+1인 이유는 target_token_index에서 index=0는 그냥 빈 공간이기 때문이다.

```
encoder_input_data = np.zeros((len(lines['eng']), eng_max_length),dtype='float32')
decoder_input_data = np.zeros((len(lines['fr']), fr_max_length),dtype='float32')
decoder_target_data = np.zeros((len(lines['fr']), fr_max_length, num_decoder_tokens+1)
,dtype='float32')
decoder_target_data.shape
```

```
(50000, 17, 359)
```

01 자료의 사전정리

- 다음 프로그램에 의해 영어문장은 encoder_input_data에 저장된다.
- 행은 문장을, 열은 각행을 구성하는 단어의 index로 구성된 2D텐서이다.
- decoder_input_data는 영어문장을 번역한 프랑스어 문장을 저장하고 있으며 각 행은 문장을, 열은 각 문장에 있는 단어의 index로 구성된 2D텐서 자료이다.
- decoder_target_data는 decoder_input_data보다 1시점 앞서게 하여 번역을 예측문제로 전환하고 있다. 즉, 현재의 영어단어와 프랑스어 단어로 다음에 오는 프랑스어 단어를 예측하는 문제로 전환한 것이다.
- 이는 머신번역의 근본적인 한계이며 사람이 하는 의미상의 번역과는 차이가 있다. 이에 따라 decoder_target_data의 구조는 one-hot 벡터 자료가 된다.
- t=1부터 시작하므로 프랑스어 문장의 시작을 나타내는 'start'가 제외되고

01 자료의 사전정리

- t=len(target_text.split( ))-1 에 의해 문장의 마지막을 나타내는 'end'가 나타나는 시간스탭(time steps)으로부터 프랑스어 문장의 최대 시간스탭인 17까지, index=89자리(axis=2의 89자리)에 1로 one-hot 코딩을 하고 있다.
- 그러므로 decoder_target_data는 3D텐서 자료이고 제 1축인 axis=0는 표본, axis=1는 시간스탭, 그리고 axis=2는 359개의 요소로 구성되어
- 제 1축에 대응하는 단어의 index자리(359개의 index 자리 중)에 1을 부여하고 나머지 358개 index 자리에 0을 부여하는 one-hot 벡터자료 구조를 가지고 있다.
- 그러므로 각 문장은 크기가 (17, 359)인 one-hot 벡터자료이며 영어 문장 하나가 입력되면 359개의 클래스를 연속적으로 17번 예측하여 대응하는 프랑스어 문장을 만드는 다중출력(multiple outputs) 구조가 된다.

01 자료의 사전정리

- softmax 함수를 이용하여 각 시간스탭에 있는 359개 인텍스에 속할 확률을 예측하게 되므로 총 17개의 categorical crossentropy 손실함수가 계산되며 최종 손실함수는 이러한 17개 손실함수의 합으로 정의된다.

```
for i, (input_text, target_text) in enumerate(zip(lines['eng'], lines['fr'])):
    for t, word in enumerate(input_text.split()):
        encoder_input_data[i, t] = input_token_index[word]
    for t, word in enumerate(target_text.split()):
        decoder_input_data[i, t] = target_token_index[word]
        if t>0:
            decoder_target_data[i, t - 1, target_token_index[word]] = 1.
            if t== len(target_text.split())-1:
                decoder_target_data[i, t:, 89] = 1
print(decoder_input_data.shape,encoder_input_data.shape,decoder_target_data.shape)
```

```
(50000, 17) (50000, 8) (50000, 17, 359)
```

01 자료의 사전정리

- 끝으로 아래와 같이 decoder_input_data에서 0은 모두 89로 대체한다. 적절한 index로 채워지고 남은 0은 모두 'end'를 의미하기 때문이다.

```
for i in range(decoder_input_data.shape[0]):
    for j in range(decoder_input_data.shape[1]):
        if(decoder_input_data[i][j]==0):
            decoder_input_data[i][j] = 89
```

02 sequence-to-sequence 학습

- encoder_input_data만 이용한 가장 간단한 sequence-to-sequence 모형을 고려하여 보자.
- Bidirectional층의 default는 concatenate하므로 Bidirectional층은 (8,128)을 입력으로 받아서 마지막 시간스탭에 대응하는 512개의 노드값을 출력한다.
- 최종 출력이 decoder_target_data의 (batch, 17, 359)이 되어야 하므로 RepeatVector(17)을 주어 Bidirectional층의 출력인 (batch, 512)를 단순하게 17번 복사하여 (batch, 17, 512)를 출력하게 하고 있다.
- LSTM층에 return_sequences=True를 부여했으므로 17개의 시간스탭에 대해 모두 출력하므로 (batch, 17, 256)를 출력한다.

02 sequence-to-sequence 학습

- 이어지는 Dense는 입력된 (17, 256)을 마치 17개의 표본인 것처럼 처리하여 17개의 시간스탭에 동일한 모수를 적용하여 (batch, 17, 359)를 출력하게 된다.
- 17개 시간스탭에 동일한 모수를 적용하므로 필요한 모수는 $256 \times 359 + 359 = 92{,}263$개가 된다.

03 머신번역을 위한 Encoder-Decoder 아키텍쳐

- 앞에서는 프랑스어로 구성된 decoder_input_data를 이용하지 않고 오직 영어로만 구성된 encoder_input_data만을 이용하여 소위, teacher forcing을 이용하지 않았다.
- 전 시간스탭의 단어로 다음 시간스탭의 단어를 예측하는데 decoder_input_data를 이용하므로 decoder_input_data를 teacher forcing이라고 한다.
- 그러나 주의해야 할 것은 decoder_input_data는 모형을 학습시키는 데에 사용할 수 있지만 실제 번역에서는 decoder_input_data가 존재하지 않으므로
- 번역된 단어를 재사용하여 decoder_input_data를 예측하여 teacher forcing을 만들어야 한다.

03 머신번역을 위한 Encoder-Decoder 아키텍쳐

- Encoder-Decoder 아키텍쳐는 아래 프로그램과 같이 encoder 모형과 decoder 모형으로 구성되어 있다.
- encoder 모형에서 사용하는 LSTM 층에 return_state=True 만 부여했으므로 마지막 시간스탭(time steps)의 은닉노드 state_h와 상태노드 state_c를 출력하게 된다.
- 여기에서 encoder_outputs는 state_h와 같으며 state_h와 state_c의 크기는 (batch, 256)인 2D 텐서이다. 이 state_h와 state_c는 encoder_states로 decoder 모형의 LSTM의 상태 및 은닉노드의 초기값으로 전달된다.
- decoder 모형의 LSTM 층에 return_sequences=True를 부여했으므로 decoder_outputs은 모든 시간스탭의 은닉층 값을 가지게 되므로 (batch, 17, 256) 3D 텐서자료가 된다.

03 머신번역을 위한 Encoder-Decoder 아키텍쳐

- LSTM층이 initial_state를 지정하기 위해서는 반드시 LSTM을 먼저 객체화 한 후, 사용하여야 한다. 즉, decoder모형과 같이 LSTM을 decoder_lstm으로 객체화한 후 decoder_lstm(dex, initial_state=encoder_states)으로 하여야 한다.

- Encoder-Decoder 아키텍쳐는 입력이 2개이고 출력이 다중이며 encoder 모형과 decoder 모형이 연결되므로 function API를 사용하여야 한다.

03 머신번역을 위한 Encoder-Decoder 아키텍쳐

- 이미 언급했듯이, 실제 번역에서는 목적변수인 decoder_target_data보다 한 시점 빠른 프랑스 단어로 구성된 decoder_input_data가 존재할 수 없다.
- 그러므로 학습된 teacher forcing을 이용한 머신번역 모형을 이용하기 위해서는 decoder_input_data를 추정해야 한다.
- decoder_input_data의 모든 문장은 index=284를 가진 'start'로 출발하므로 encoder_input_data를 이용하여 'start' 다음 프랑스어 단어를 예측하고 이를 decoder_input_data에 저장한다.
- 다음 단계로 encoder_input_data와 예측된 단어를 가진 decoder_input_data를 이용하여 두 번째 시간스탭에 나타날 프랑스어 단어를 예측하고 이를 decoder_input_data에 저장한다.
- 이를 최종 시간스탭까지 반복하여 입력된 영어문장의 번역을 완성한다.

04 Attention을 이용한 머신번역

- decoder_input_data를 이용한 teacher forcing으로 머신번역의 성능을 향상시킨 앞 절의 모형에 attention을 추가한 머신번역 모형을 논의하고자 한다.
- attention은 앞 절에서 논의한 encoder_input_data 중 decoder_input_data를 잘 설명해주는 단어에 좀 더 가중치를 부여해 주는 기법을 말한다.
- teacher forcing을 이용한 encoder-decoder 아키텍쳐는 encoder 모형의 은닉층을 decoder LSTM 층에 연결하여 encoder의 정보를 이용하였다.
- 이 절에서 사용할 Attention을 이용한 encoder-decoder 아키텍쳐도 앞 절의 아키텍쳐와 동일한 구조를 가지고 있다.

04 Attention을 이용한 머신번역

- 추가적으로 encoder 모형과 decoder 모형의 출력 값을 이용하여 encoder와 decoder의 유사성(similarity)으로 해석되는 attention을 encoder에 적용하여 가중 encoder를 모형에 반영한다.
- 다음 프로그램은 teacher forcing의 encoder 모형과 decoder 모형과 유사하나, encoder 모형의 LSTM층에 return_sequences=True를 추가하여 encoder_outputs의 크기가 (batch, 8, 256)으로 3D 텐서로 증가하였고
- 과대적합을 방지하기 위해 Dropout 층을 추가하였다. decoder 모형에서도 Dropout 층을 추가하였으며 LSTM에서는 return_state=True를 제거하였으며 마지막 Dense 층을 제거하였다.

Q & A