15. RNN을 이용한 인코더-디코더

인코더-디코더(Encoder-Decoder)

- 인코더-디코더는 신경망 기반 기계번역의 근간으로 활용되고 있음
- 이 구조는 기계번역 이외에도 텍스트 요약, 질의-응답에도 활용되고 있음
- 이 장에서는 Sequence-to-sequence 기법과 이를 이용한 기본적인 번역기 구조를 살펴봄

15장 내용

- 기계번역 기술 현황과 추세
- Keras에서의 text 처리: Tokenizer, pad_sequences, word embedding 등
- Sequence-to-sequence processing
- 영-불 기계번역기
- Keras에서의 Functional API
- Word level 영-불 번역기
- **BLEU** (Bilingual Evaluation Understudy) score: 번역기 성능 평가 기준

기계 번역(Machine Translation) 기술 현황

CNN 기사:

"Squid Game" has become Netflix's most popular show ever, and the importance of streaming is giving new prominence to the role of screenwriter.

Netflix brought Kim Eun-hee on as one of its first creative partners in South Korea, and the streaming service says it has invested nearly $500 million in Korean content in 2021 alone. "She is the tip of the spear," said Keo Lee, Netflix's director of Korean original series. "What I admire most about her is her belief in the strength of a good story."

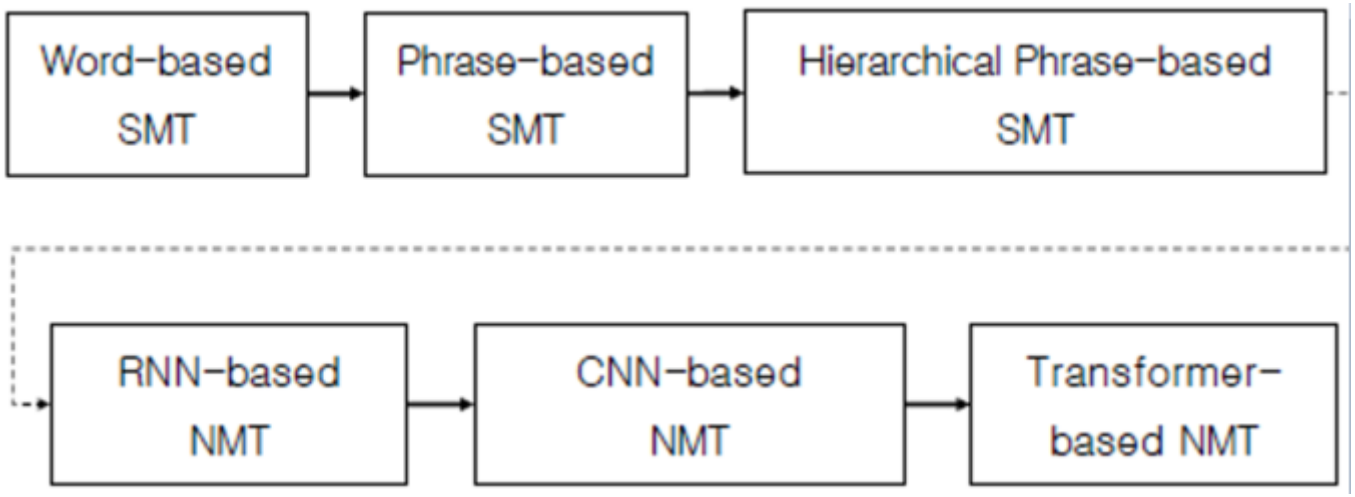
파파고 번역:

"오징어 게임"은 넷플릭스의 가장 인기 있는 쇼가 되었고, 스트리밍의 중요성은 시나리오 작가 역할에 새로운 중요성을 부여하고 있다.

넷플릭스는 김은희를 한국에서 첫 번째 창의적인 파트너 중 한 명으로 데려왔고, 스트리밍 서비스는 2021년에만 거의 5억 달러를 한국 콘텐츠에 투자했다고 말한다. "그녀는 창끝입니다,"라고 넷플릭스의 한국 오리지널 시리즈 감독인 거 리는 말했다. "제가 그녀에 대해 가장 존경하는 것은 좋은 이야기의 힘에 대한 믿음입니다."

기계 번역 기술 추세

- 이전에는 자동 번역 결과가 그다지 좋지 않았음
- 2015년 이후 신경망 기술(Neural MT)이 자동 번역에 사용되면서 수준이 급속히 향상되고 있음
- 이 절에서 소개하고 있는 Sequence-to-sequence 방식은 2014년 논문에서 제시된 신경망 번역 기술임



seq2seq 기술

Keras에서 입력 텍스트 처리 함수

- **Tokenzer:** 입력 텍스트를 숫자로 변환하는 기능을 수행

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

tokenizer = Tokenizer()
tokenizer.fit_on_texts(text_data) # 텍스트의 각 단어에 고유한 숫자를 부여
sequences = tokenizer.texts_to_sequences(text_data) # 단어를 숫자값으로 변환하여 저장
```

- 위에서 구한 sequences는 숫자 리스트로 구성되어 있음
- 전체 단어 리스트는 word_index 함수로 얻을 수 있음

```
word_to_index = tokenizer.word_index
print(word_to_index)
```

Tokenizer 적용 사례

```
from tensorflow.keras.preprocessing.text import Tokenizer
t = Tokenizer()
fit_text = "The earth is an awesome place live"
t.fit_on_texts([fit_text]) # fit_text 만으로 단어별 정수를 배정

test_text = "The earth is an great place live"
sequences = t.texts_to_sequences([test_text])[0]
print("sequences : ",sequences) # great는 단어 집합에 없으므로 출력되지 않음
print("word_index : ",t.word_index) # 단어 집합(vocabulary) 출력
```

여기에 전체 text가 들어가야 함

문자 텍스트 내용을 숫자로 바꿈

```
sequences : [1, 2, 3, 4, 6, 7]
word_index : {'the': 1, 'earth': 2, 'is': 3, 'an': 4, 'awesome': 5, 'place': 6,
'live': 7}
```

Keras에서의 텍스트 시퀀스 처리

- 신경망 입력 텍스트를 일정한 길이로 만들려면 pad_sequences 함수를 이용
- 이 함수는 sequence의 최대 길이를 지정하여 이보다 긴 텍스트는 앞 또는 뒤를 자르고, 짧은 텍스트는 앞에 0을 채움

```
# 최대 길이를 100으로 지정하고 초과하면 각 시퀀스의 앞 쪽을 자른다
X_prime = sequence.pad_sequences(X, maxlen=100, truncating='pre')

# 최대길이를 100으로 지정하고 초과하면 각 시퀀스의 뒷 쪽을 자른다
X_prime = sequence.pad_sequences(X, maxlen=100, truncating='post')
```

pad_sequences 적용 사례

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
pad_sequences([[1, 2, 3], [3, 4, 5, 6], [7, 8]], maxlen=3, padding='pre')
```

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

워드 임베딩

- 입력 텍스트를 밀집 벡터로 표현하는 수단으로 Keras에서는 Embedding 레이어를 제공
- Embedding 층은 정수 인코딩이 된 단어들을 임베딩 벡터로 변환
- Embedding 호출시 단어수와 임베딩 벡터 길이를 입력으로 전달
- Embedding 출력은 (단어수, 벡터 길이, 입력 시퀀스 길이)의 3D 텐서를 리턴
- Word2Vec과 같은 실제 임베딩 벡터를 사용할 수도 있지만, 여기서는 자체적으로 생성하는 벡터임

워드 임베딩 사례

```
# 문장 토큰화와 단어 토큰화
text=[['Hope', 'to', 'see', 'you', 'soon'],['Nice', 'to', 'see', 'you', 'again']]
# 각 단어에 대한 정수 인코딩
text=[[0, 1, 2, 3, 4],[5, 1, 2, 3, 6]]
# 위 데이터가 아래의 임베딩 층의 입력이 된다.
Embedding(7, 2, input_length=5)
# 7은 단어의 개수. 즉, 단어 집합(vocabulary)의 크기이다.
# 2는 임베딩한 후의 벡터의 크기이다.
# 5는 각 입력 시퀀스의 길이. 즉, input_length이다.
# 각 정수는 아래의 테이블의 인덱스로 사용되며 Embeddig()은 각 단어에 대해 임베딩 벡터를 리턴
```

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]

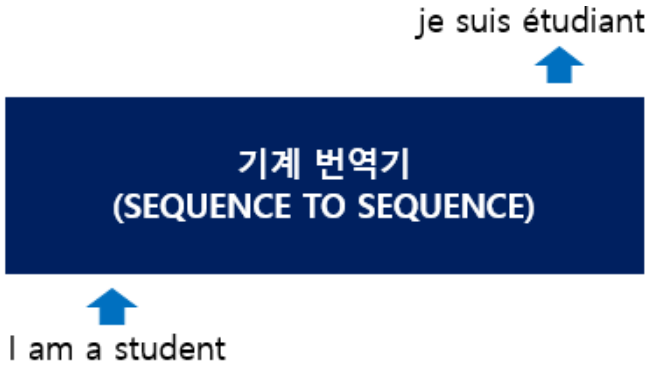
번역기:
Sequence-to-sequence processing

Sequence-to-sequence(seq2seq)

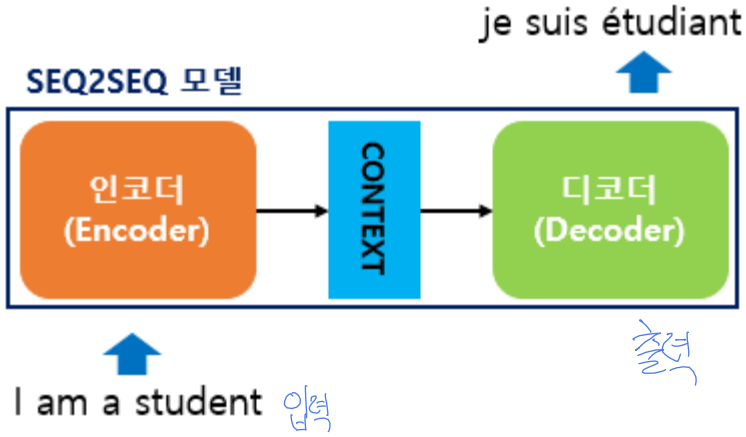
- 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력
- 기계번역(machine translation), chatbot, text summarization 등에 사용됨
- 이 장에서는 Sequence-to-sequence 기법과 이를 keras로 구현하는 방법을 살핌

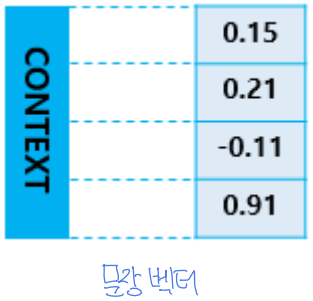
Sequence-to-sequence

- 번역기에서 사용되는 모델

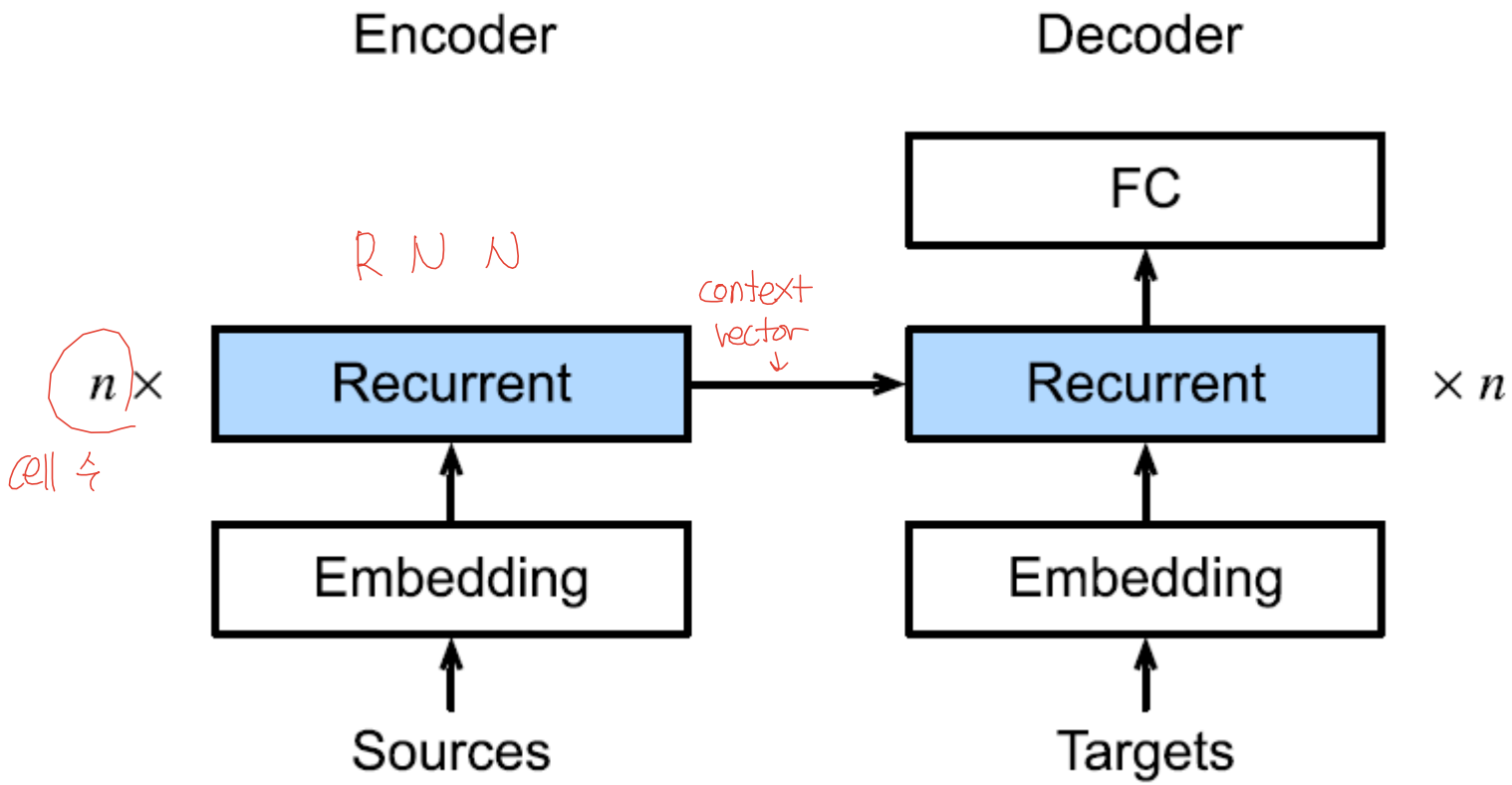


- 인코더-디코더로 구성됨: 인코더는 입력 단어들을 순차적으로 받아 context 벡터를 생성



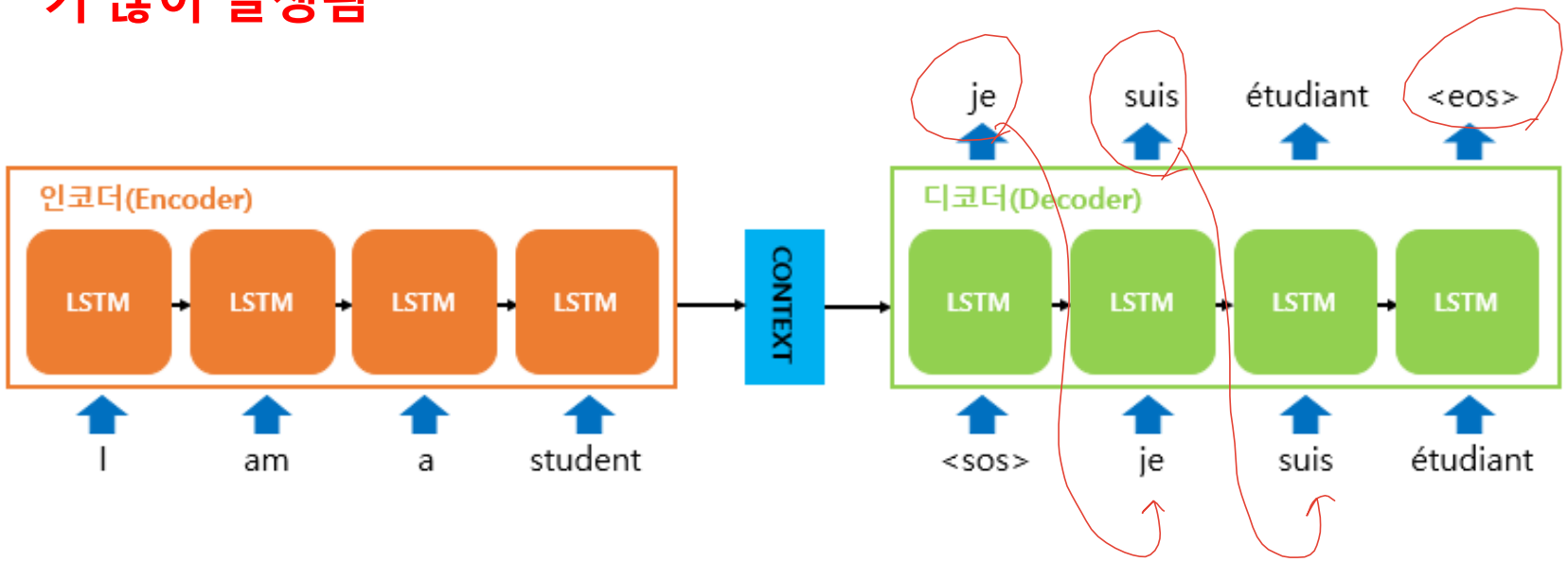


Sequence-to-sequence 구조



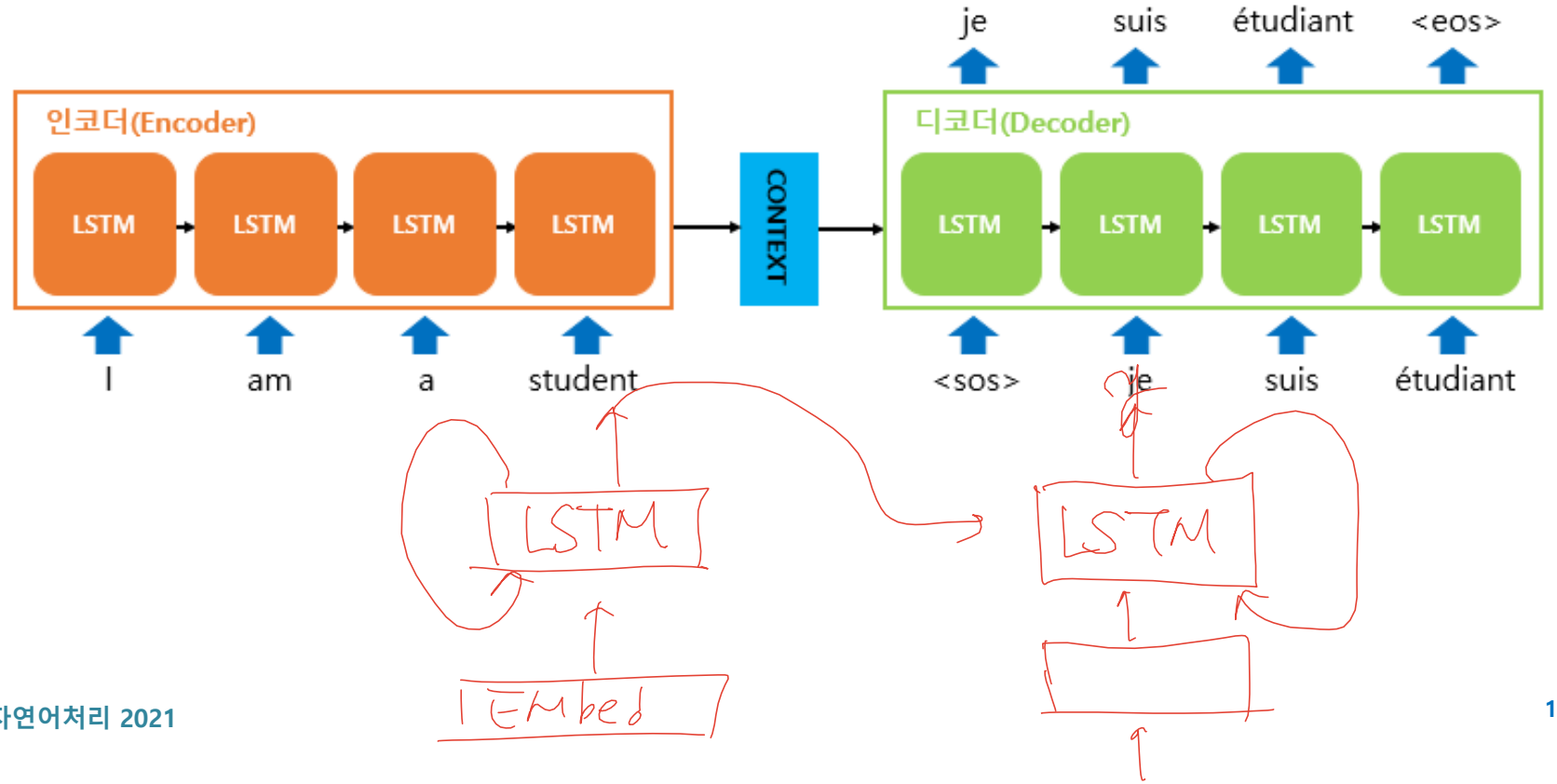
Encoder-decoder architecture

- 인코더와 디코더는 각각 RNN 아키텍처로 구현됨
- 입력 문장은 단어 단위로 쪼개지고 각 단어는 RNN셀의 입력이 됨
- 모든 단어를 입력받은 후 **RNN 셀의 마지막 은닉상태를 Context vector** 라고 함
- **Context vector는 문장 전체의 정보를 가지고 있음: 문장이 길면 오차가 많이 발생됨**



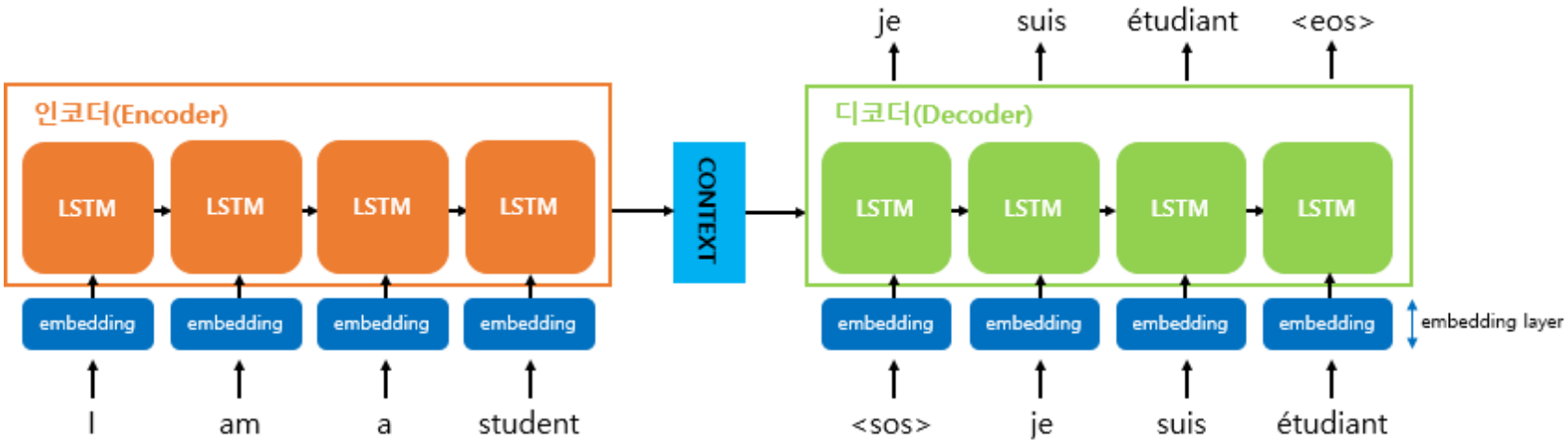
Decoder 동작 방식

- 디코더는 context vector를 첫번째 은닉 상태로 사용
- 디코더 초기입력은 심볼 <sos>(start of sentence)가 들어감
- 생성된 단어부터 차례로 다음 단어를 예측하고 이 과정은 심볼 <eos>(end of sentence)가 나올 때까지 반복됨



문자의 표현

- 문자는 보통 Embedding vector로 표현



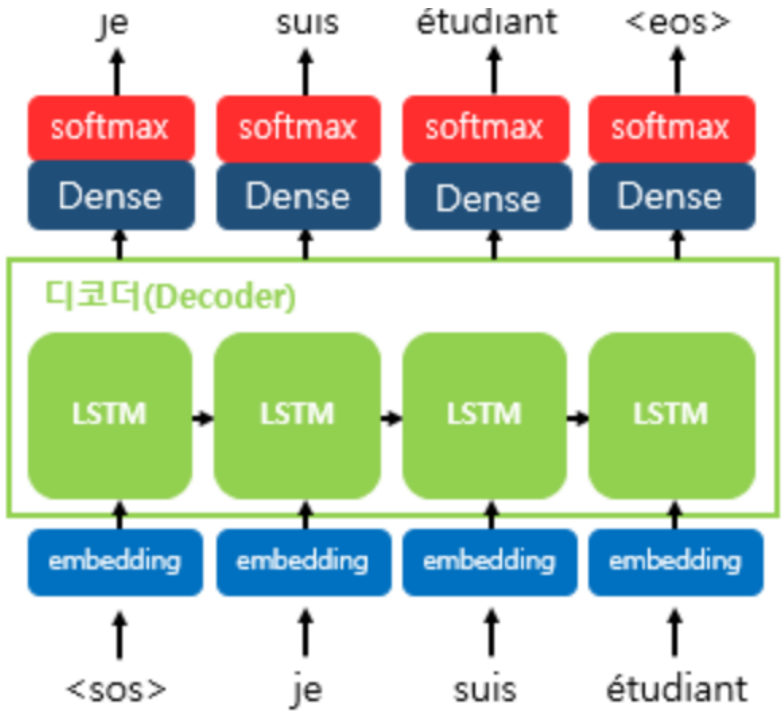
I		am		a		student	
	0.157		0.78		0.75		0.88
	-0.25		0.29		-0.81		-0.17
	0.478		-0.96		0.96		0.29
	-0.78		0.52		0.12		0.48

Decoder 동작 방식

- 디코더는 훈련 과정과 실제 사용 과정에서의 동작 방식이 다름
- **훈련과정:** 인코더가 보낸 컨텍스트 벡터와 실제 정답 문장을 제시했을 때 실제 정답이 나오도록 훈련시킴. 이를 **교사강요(Teacher forcing)**라고 함
- **실제 사용과정:** 컨텍스트 벡터와 <sos>만을 입력으로 받은 후 출력을 생성

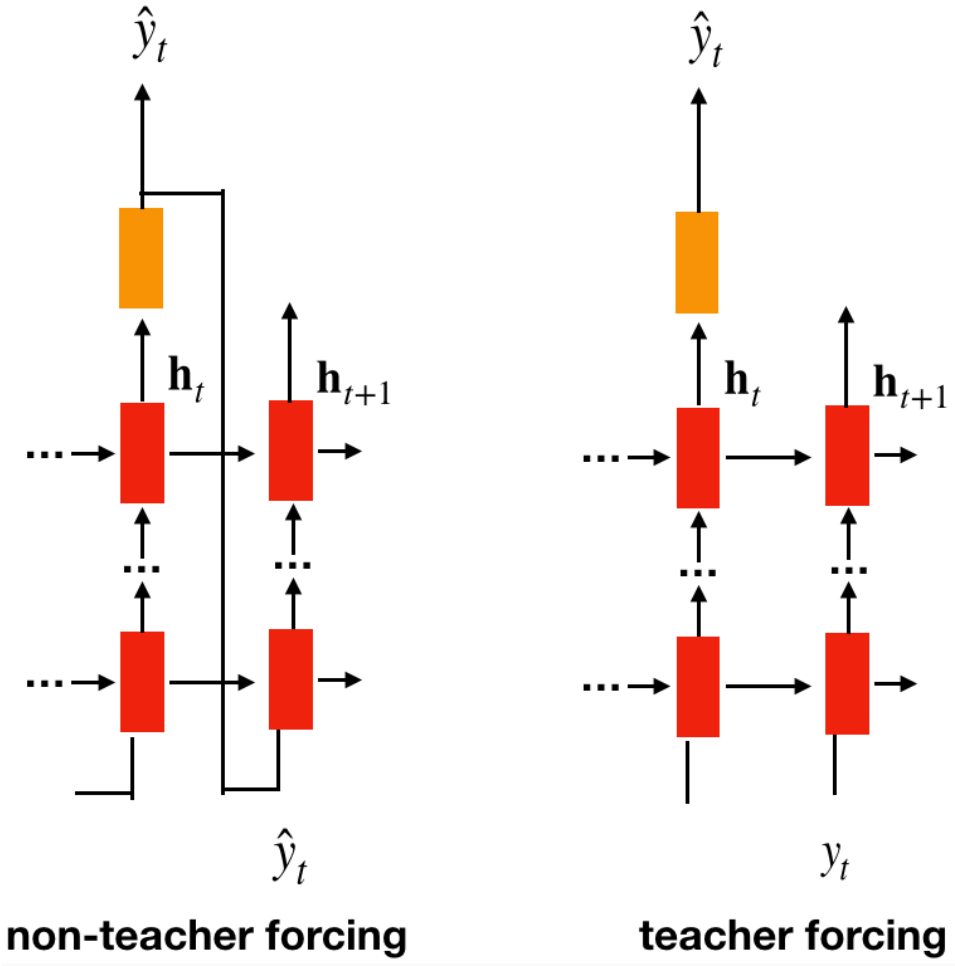
Decoder 구조

- 디코더에서는 각 시점에서 RNN의 출력 벡터가 나오면 softmax 함수를 이용하여 각 단어별 확률값을 반환



교사 강요(Teacher forcing)

- 훈련 시간을 단축하기 위해 훈련시에는 정확한 실제값을 decoder cell의 입력으로 사용하는 것을 **teacher forcing**이라고 함



Word-Level 번역기

- 글자 단위가 아닌 단어 단위로 동작하는 기계 번역기를 구현
- 전처리 과정과 embedding 층을 사용
- keras에서는 Tokenizer와 sequence padding을 사용

사례: 영-불 기계번역기

Corpus

- 병렬 데이터(두 개 이상의 언어 문장들이 병렬로 구성된 데이터)를 이용하여 훈련시키는 영-불 번역기를 고찰
  - 예: '나는 학생이다 ➔ I am a student' 와 같은 입-출력 문장을 제시
- 여기서 사용하는 'fra.txt'에는 영-불 문장이 17만개 정도 저장되어 있는데, 이 중 60,000개의 문장을 이용하여 기계번역기를 구축

영불 코퍼스 파일 읽어 오기

```
http = urllib3.PoolManager()
url ='http://www.manythings.org/anki/fra-eng.zip'
filename = 'fra-eng.zip'
path = os.getcwd()
zipfilename = os.path.join(path, filename)
with http.request('GET', url, preload_content=False) as r, open(zipfilename, 'wb')
as out_file:
      shutil.copyfileobj(r, out_file)

with zipfile.ZipFile(zipfilename, 'r') as zip_ref:
      zip_ref.extractall(path)

lines = pd.read_csv('fra.txt', names=['src', 'tar', 'lic'], sep='₩t') # src와 tar는 입-출력단을 의미
del lines['lic']

lines = lines.loc[:, 'src':'tar']
lines = lines[0:60000] # 6만개만 저장
lines.sample(10)        # 10개의 샘플을 인쇄
```

10개의 샘플 문장

	src	tar
21379	This desk is good.	₩t Ce bureau est bien. ₩n
28472	Did you get my gift?	₩t Avez-vous reçu mon cadeau ? ₩n
54092	Glass is made from sand.	₩t Le verre est fait à partir de sable. ₩n
9188	That's not bad.	₩t Ce n'est pas mal. ₩n
10935	I feel horrible.	₩t Je me sens super mal. ₩n
7423	Do as you want.	₩t Fais comme bon te semble. ₩n
35507	I forgot his address.	₩t J'ai oublié son adresse. ₩n
46827	Can you ride a bicycle?	₩t Sais-tu faire du vélo ? ₩n
56613	It made me a little sad.	₩t Ça m'a rendu un peu triste. ₩n
26400	They are exhausted.	₩t Elles sont exténuées. ₩n

불어 문장에서는 '₩t'와 '₩n'은 각각 <sos>와 <eos>를 표시

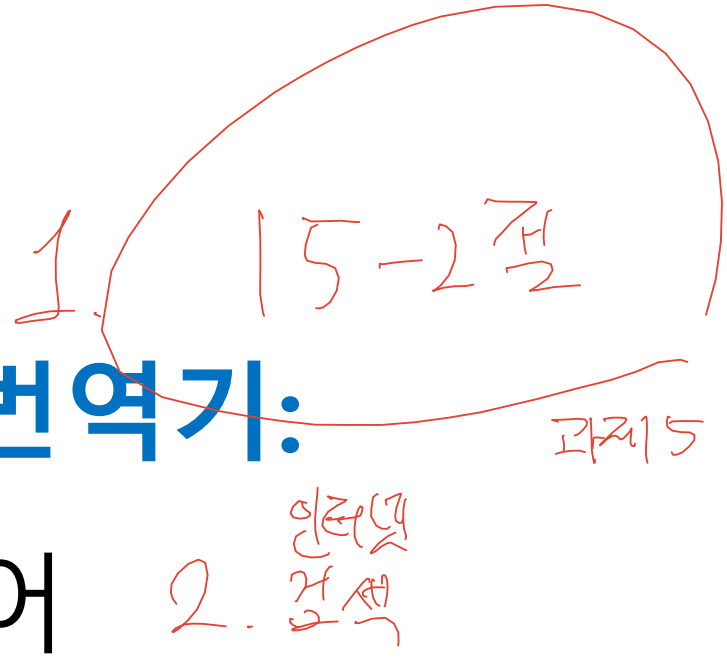
한글-영어 병렬 데이터

- http://opus.nlpl.eu/OpenSubtitles-v2018.php
  - TED 영어 강연 문장들을 한국어로 번역한 파일에 24만개 정도의 문장이 있음(train.tags.en-ko.ko와 train.tags.en-ko.en)
  - 영화 자막을 한글로 번역한 파일에는 140만개 정도의 문장이 있음(OpenSubtitles.en-ko.ko와 OpenSubtitles.en-ko.en)
- 한국 ai-hub에는 총 160만개의 문장이 준비되어 있음
  - 금속노조 현대중공업지부 조합원을 비롯한 3000여명이 새벽 5시께 기상 알림을 듣고 일어나 경찰과 사측 진입에 대비했다.

    "Some 3,000 workers, including members of the Hyundai Heavy Industries branch of the Korean Metal Workers' Union, woke up at 5 a.m. after hearing the morning alarm and prepared for the police and management to enter the company."

Word-level 번역기:

영어-불어



Word-level 번역기 (영어-불어)

- 사용 데이터
  - 앞서 사용했던 영불 병렬 데이터를 이용
  - 33,000개의 샘플을 사용
- 전처리 함수
  - 문자 텍스트 데이터를 Tokenizer 함수들을 이용하여 숫자로 변환
  - 영어와 불어의 각 문장을 일정 길이의 sequence로 변환

load_preprocessed_data 함수

- 입력: 병렬 코퍼스인 fra.txt 파일을 읽음
- 출력: encoder_input (영어), decoder_input, decoder_target (불어)
  - 출력 데이터는 문자로 구성된 텍스트임
- 수행 내용
  - 텍스트에서 단어와 구두점 사이에 공백을 추가
  - 알파벳과 특수 문자(“.”, “?”, “!”, “,”)를 제외하고 모두 공백으로 변환
  - 불어 문장에는 <sos>와 <eos>를 추가
  - 훈련시 Teacher forcing을 사용하므로 실제 시퀀스를 저장하는 decoder_target 문장을 구성

정수 인코딩 과정

- 앞서 설명한 케라스 Tokenizer 함수를 이용하여 문자 텍스트를 정수 시퀀스로 변환
- 수행 절차:
  - `fit_on_texts`: 전체 데이터를 이용하여 각 단어별로 정수를 배정
  - `texts_to_sequences`: 배정된 정수를 이용하여 문자 텍스트를 숫자 시퀀스로 변환
  - `pad_sequences`: 각 문장을 일정한 길이로 패딩

정수 인코딩 코드

```
tokenizer_en = Tokenizer(filters="", lower=False)
tokenizer_en.fit_on_texts(sents_en_in)
encoder_input = tokenizer_en.texts_to_sequences(sents_en_in)
# 문자 텍스트 sents_en_in을 숫자 시퀀스인 encoder_input으로 변환

tokenizer_fra = Tokenizer(filters="", lower=False)
tokenizer_fra.fit_on_texts(sents_fra_in)
tokenizer_fra.fit_on_texts(sents_fra_out)
decoder_input = tokenizer_fra.texts_to_sequences(sents_fra_in)
decoder_target = tokenizer_fra.texts_to_sequences(sents_fra_out)
# 문자 텍스트 sents_fra_in을 숫자 시퀀스인 decoder_input으로 변환

encoder_input = pad_sequences(encoder_input, padding="post")
decoder_input = pad_sequences(decoder_input, padding="post")
decoder_target = pad_sequences(decoder_target, padding="post")
# 숫자 시퀀스 encoder_input, decoder_input을 일정 길이로 패딩. 이 데이터가
신경망 입력으로 사용됨
```

사용된 단어 갯수

- 사용된 단어는 Tokenizer의 word_index 길이를 통해 알 수 있음

```
src_vocab_size = len(tokenizer_en.word_index) + 1
tar_vocab_size = len(tokenizer_fra.word_index) + 1
print("영어 단어 집합의 크기 : {:d}, 프랑스어 단어 집합의 크기 :
    {:d}".format(src_vocab_size, tar_vocab_size))
```

영어 단어 집합의 크기 : 4647, 프랑스어 단어 집합의 크기 : 8022

출력 문장을 생성할 딕셔너리

- 숫자 시퀀스에서 다시 문자 텍스트를 생성하기 위해 단어에서 정수를 얻는 딕셔너리와 정수에서 단어를 얻는 딕셔너리를 저장

```
src_to_index = tokenizer_en.word_index
index_to_src = tokenizer_en.index_word # 훈련 후 결과 비교할 때 사용

tar_to_index = tokenizer_fra.word_index # 훈련 후 예측 과정에서 사용
index_to_tar = tokenizer_fra.index_word # 훈련 후 결과 비교할 때 사용
```

훈련/테스트 데이터 분리

- 33,000 샘플 중 10%인 3,300개를 테스트 데이터로 사용

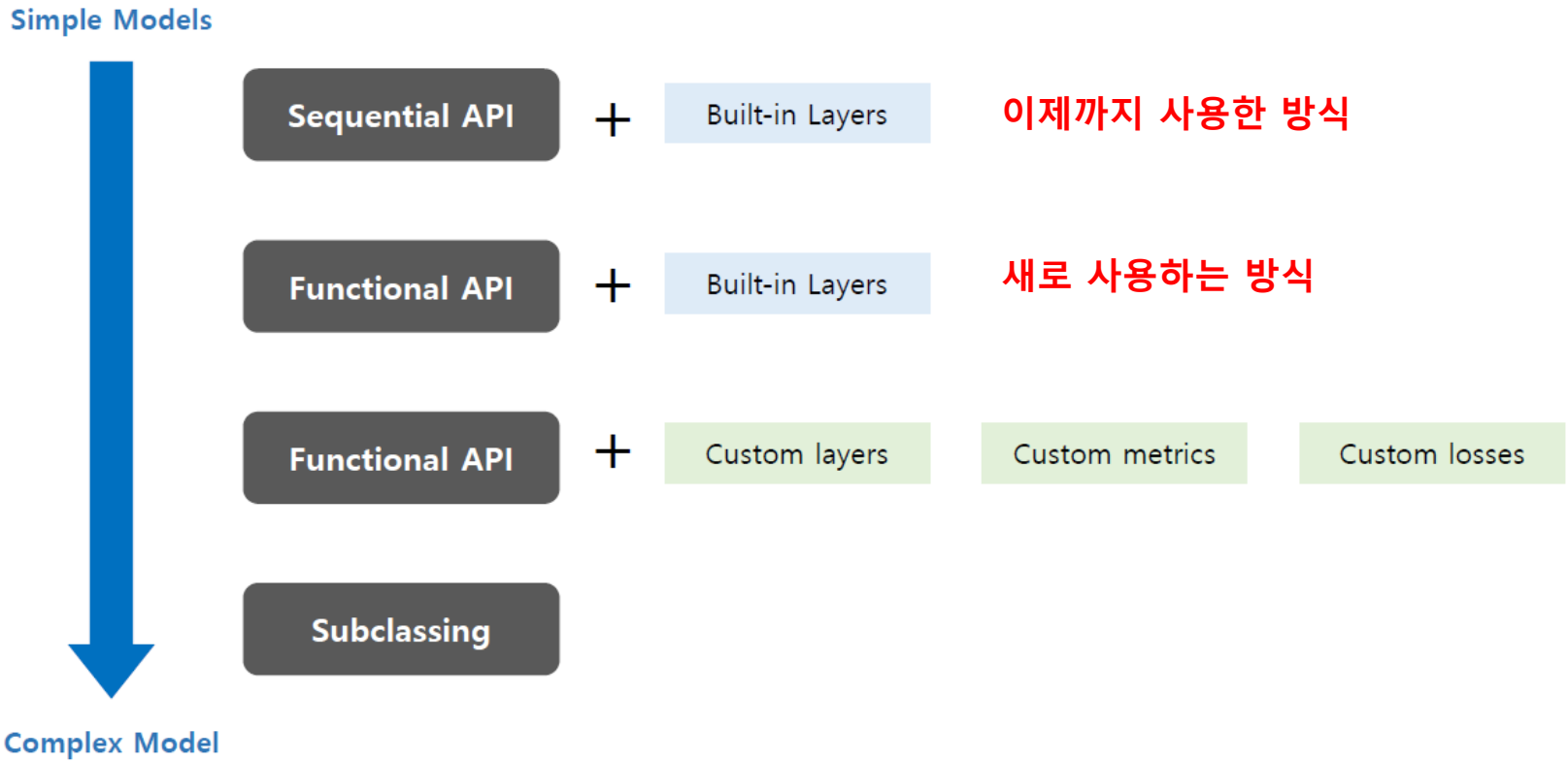
```
n_of_val = int(33000*0.1)

encoder_input_train = encoder_input[:-n_of_val]
decoder_input_train = decoder_input[:-n_of_val]
decoder_target_train = decoder_target[:-n_of_val]

encoder_input_test = encoder_input[-n_of_val:]
decoder_input_test = decoder_input[-n_of_val:]
decoder_target_test = decoder_target[-n_of_val:]
```

Keras 프로그래밍

- 이제까지는 Sequential API 프로그래밍 방식을 사용해 왔음



Functional API

- Functional API에서는 각 layer의 입력과 출력 파라미터를 지정함

```
from keras.models import Sequential, Model
from keras import layers, Input

# 이제까지의 Sequential 방식
seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))

# 같은 코드를 Functional 방식으로 수정하는 경우
input_tensor = Input(shape(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = Model(input_tensor, output_tensor)
```

기계번역기 만들기

- **함수형 API를 이용하여 구현**
- 임베딩 벡터 크기와 LSTM 은닉 상태의 크기를 50개로 설정
- 인코더는 아래와 같음

```
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Masking
from tensorflow.keras.models import Model

latent_dim = 50

# 인코더
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(src_vocab_size, latent_dim)(encoder_inputs) # 임베딩 층
enc_masking = Masking(mask_value=0.0)(enc_emb) # 패딩 0은 연산에서 제외
encoder_lstm = LSTM(latent_dim, return_state=True) # 상태값 리턴을 위해
return_state는 True
encoder_outputs, state_h, state_c = encoder_lstm(enc_masking) # 은닉 상태와 셀
상태를 리턴
encoder_states = [state_h, state_c] # 인코더의 은닉 상태와 셀 상태를 저장
```

기계번역기 만들기(계속)

- 디코더는 아래와 같음

```
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(tar_vocab_size, latent_dim) # 임베딩 층
dec_emb = dec_emb_layer(decoder_inputs) # 패딩 0은 연산에서 제외
dec_masking = Masking(mask_value=0.0)(dec_emb)

# 상태값 리턴을 위해 return_state는 True, 모든 시점에 대해서 단어를 예측하기 위해
return_sequences는 True
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)

# 인코더의 은닉 상태를 초기 은닉 상태(initial_state)로 사용
decoder_outputs, _, _ = decoder_lstm(dec_masking,
                                     initial_state=encoder_states)

# 모든 시점의 결과에 대해서 소프트맥스 함수를 사용한 출력층을 통해 단어 예측
decoder_dense = Dense(tar_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

모델의 구조: model.summary()

영어 단어수: 4663, 불어 단어수: 8038

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, None)]            0
__________________________________________________________________________________________________
input_2 (InputLayer)         [(None, None)]            0
__________________________________________________________________________________________________
embedding (Embedding)        (None, None, 50)          233150    input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 50)          401900    input_2[0][0]
__________________________________________________________________________________________________
masking (Masking)            (None, None, 50)          0         embedding[0][0]
__________________________________________________________________________________________________
masking_1 (Masking)          (None, None, 50)          0         embedding_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                  [(None, 50), (None,       20200     masking[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                [(None, None, 50), (      20200     masking_1[0][0]
                                                                 lstm[0][1]
                                                                 lstm[0][2]
__________________________________________________________________________________________________
dense (Dense)                (None, None, 8038)        409938    lstm_1[0][0]
==================================================================================================
Total params: 1,085,388
Trainable params: 1,085,388
Non-trainable params: 0
```

4663x50

8038x50

50x50x8

50x8038

신경망 훈련과 결과

- 모델의 생성과 훈련

```
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

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

model.fit(x = [encoder_input_train, decoder_input_train], y = decoder_target_train,
          validation_data = ([encoder_input_test, decoder_input_test],
          decoder_target_test), batch_size = 128, epochs = 50)
```

기계 번역기 동작시키기

- 인코더와 디코더 정의

```
encoder_model = Model(encoder_inputs, encoder_states)

# 이전 시점의 상태들을 저장하는 텐서
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

# 훈련 때 사용했던 임베딩 층을 재사용
dec_emb2= dec_emb_layer(decoder_inputs)

# 다음 단어 예측을 위해 이전 시점의 상태를 현 시점의 초기 상태로 사용
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_states_inputs)
decoder_states2 = [state_h2, state_c2]

# 모든 시점에 대해서 단어 예측
decoder_outputs2 = decoder_dense(decoder_outputs2)

# 디코더 정의
decoder_model = Model(
        [decoder_inputs] + decoder_states_inputs,
        [decoder_outputs2] + decoder_states2)
```

seq2seq 디코더 동작 코드

```
def decode_sequence(input_seq):
    # 입력으로부터 인코더의 상태를 얻음
    states_value = encoder_model.predict(input_seq)
    # <SOS>에 해당하는 원-핫 벡터 생성
    target_seq = np.zeros((1, 1, tar_vocab_size))
    target_seq[0, 0, tar_to_index['\t']] = 1.
    stop_condition = False
    decoded_sentence = ""
    # stop_condition이 True가 될 때까지 루프 반복
    while not stop_condition:
        # 이점 시점의 상태 states_value를 현 시점의 초기 상태로 사용
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
        # 예측 결과를 문자로 변환
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = index_to_tar[sampled_token_index]
        # 현재 시점의 예측 문자를 예측 문장에 추가
        decoded_sentence += sampled_char
        # <eos>에 도달하거나 최대 길이를 넘으면 중단.
        if (sampled_char == '\n' or len(decoded_sentence) > max_tar_len):
            stop_condition = True
        # 현재 시점의 예측 결과를 다음 시점의 입력으로 사용하기 위해 저장
        target_seq = np.zeros((1, 1, tar_vocab_size))
        target_seq[0, 0, sampled_token_index] = 1.
        # 현재 시점의 상태를 다음 시점의 상태로 사용하기 위해 저장
        states_value = [h, c]
    return decoded_sentence
```

결과 확인을 위한 문장 생성

- 정수 시퀀스를 텍스트 시퀀스로 변환

```
def seq2src(input_seq):
    temp=''
    for i in input_seq:
        if(i!=0):
            temp = temp + index_to_src[i]+' '
    return temp

# 번역문의 정수 시퀀스를 텍스트 시퀀스로 변환
def seq2tar(input_seq):
    temp=''
    for i in input_seq:
        if((i!=0 and i!=tar_to_index['<sos>']) and i!=tar_to_index['<eos>']):
            temp = temp + index_to_tar[i] + ' '
    return temp
```

임의의 샘플에 대한 번역 결과

- 정수 시퀀스를 텍스트 시퀀스로 변환

```
for seq_index in [3,50,100,300,1001]:
    input_seq = encoder_input_train[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)

    print("원문 : ",seq2src(encoder_input_train[seq_index]))
    print("번역문 :",seq2tar(decoder_input_train[seq_index]))
    print("예측문 :",decoded_sentence[:-5])
    print("\n")
```

```
원문 : where is your house ?
번역문 : ou est votre maison ?
예측문 : ou est votre maison ?

원문 : we re out of stock .
번역문 : nous sommes a court .
예측문 : nous sommes en train de l exterieur .
```

번역기 성능 평가:

BLEU

BLEU(Bilingual Evaluation Understudy) Score

- 기계 번역기의 성능을 측정하기 위한 평가 방법
- 기계 번역 결과와 사람이 직접 번역한 결과가 얼마나 유사한지 비교
- n-gram의 유사성을 기반으로 측정
- 완벽한 평가 방법은 아니지만 계산속도가 빠름
- 0~1 사이의 값을 가짐(0~100 사이의 %값으로 나타내는 경우도 있음)

BLEU 개념

- 두 개의 문장을 비교할 때 겹치는 n-gram 쌍의 숫자를 비교
- 측정요소는 다음과 같음
  - n-gram을 통한 순서쌍들이 얼마나 겹치는지 측정(Precision)
  - 문장 길이에 대한 과적합 보정(Brevity penalty)
  - 같은 단어가 연속적으로 나올 때 과적합되는 것을 보정(Clipping)

$$BLEU = \min\left(1, \frac{output\ length(\text{예측 문장})}{reference\ length(\text{실제 문장})}\right)\left(\prod_{i=1}^{4} precision_i\right)^{\frac{1}{4}}$$

$precision_{\mathrm{n}}$는 n-gram쌍의 일치도를 의미

문장 비교 사례

- n-gram(1~4)에서 순서쌍들의 일치성
  - **예측 문장:** 빛이 쐬는 노인은 완벽한 어두운곳에서 잠든 사람과 비교할 때 강박증이 심해질 기회가 훨씬 높았다
  - **실제 문장:** 빛이 쐬는 사람은 완벽한 어둠에서 잠든 사람과 비교할 때 우울증이 심해질 가능성이 훨씬 높았다

  - 1-gram precision: $\frac{\text{일치하는 1-gram쌍(예측 문장에서)}}{\text{모든 1-gram쌍(예측 문장에서)}} = \frac{10}{14}$
  - 2-gram precision: $\frac{\text{일치하는 2-gram쌍(예측 문장에서)}}{\text{모든 2-gram쌍(예측 문장에서)}} = \frac{5}{13}$
  - 3-gram precision: $\frac{\text{일치하는 3-gram쌍(예측 문장에서)}}{\text{모든 3-gram쌍(예측 문장에서)}} = \frac{2}{12}$
  - 4-gram precision: $\frac{\text{일치하는 4-gram쌍(예측 문장에서)}}{\text{모든 4-gram쌍(예측 문장에서)}} = \frac{1}{11}$

$$\left(\prod_{i=1}^{4} precision_i\right)^{\frac{1}{4}} = \left(\frac{10}{14} \times \frac{5}{13} \times \frac{2}{12} \times \frac{1}{11}\right)^{\frac{1}{4}}$$

문장 길이에 대한 과적합 보정(Brevity penalty)

- 문장 길이가 다른 경우 점수가 낮아지게 함
  - **예측 문장:** 빛이 쐬는 노인은 완벽한 어두운곳에서 잠듬
  - **실제 문장:** 빛이 쐬는 사람은 완벽한 어둠에서 잠든 사람과 비교할 때 우울증이 심해질 가능성이 훨씬 높았다

$$\min\left(1, \frac{output\ length(\text{예측 문장})}{reference\ length(\text{실제 문장})}\right) = \min\left(1, \frac{6}{14}\right) = \frac{3}{7}$$

nltk를 사용한 BLEU 측정하기

- nltk를 사용하여 BLEU를 계산할 수 있음

```
import nltk.translate.bleu_score as bleu

candidate = 'It is a guide to action which ensures that the military always obeys
the commands of the party’
references = [
    'It is a guide to action that ensures that the military will forever heed Party
commands’,
    'It is the guiding principle which guarantees the military forces always being
under the command of the Party’,
    'It is the practical guide for the army always to heed the directions of the
party’ ]

print(bleu_score(candidate.split(),list(map(lambda ref: ref.split(), references))))
# 이번 챕터에서 구현한 코드로 계산한 BLEU 점수
print(bleu.sentence_bleu(list(map(lambda ref: ref.split(),
references)),candidate.split()))
```

```
# NLTK 패키지 구현되어져 있는 코드로 계산한 BLEU 점수

0.5045666840058485
0.5045666840058485
```

BLEU 점수의 실용성과 한계

- Bleu 점수는 기계 번역기의 성능을 평가하는데 많이 사용되고 있음
- Bleu는 계산하기가 쉽고 언어와 무관하게 사용될 수 있음
- Bleu 점수와 번역의 정확도는 일치하지 않을 수도 있음