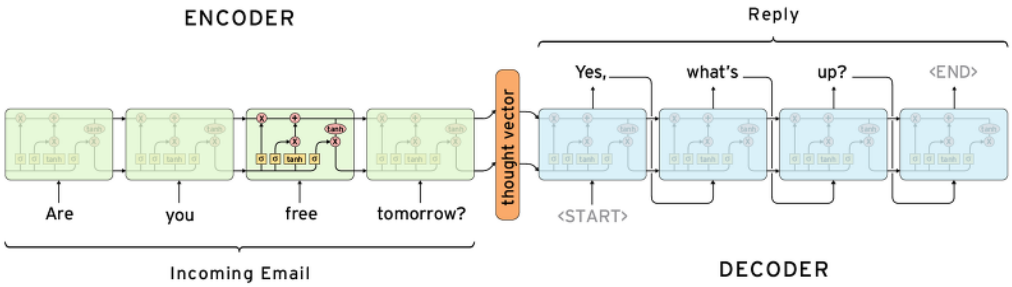
13. 기계 번역

기계번역에서 널리 사용되는 모형은 Sequence-to-sequence(Seq2Seq) 모형이다. 영어를 프랑스어로 번역하는 경우 영어를 출발 언어(source language), 프랑스어를 도착 언어(target language)라고 하는데 출발 언어와 도착 언어의 길이가 맞지 않기 때문에 하나의 LSTM으로는 처리할 수 없다.

Seq2Seq은 두 개의 LSTM을 이어 붙여 이 문제를 해결한다. 먼저 인코더 LSTM은 출발 언어의 문장을 받아들인다. 그리고 특별한 출력 없이 내부 상태를 그대로 디코더 LSTM으로 넘겨준다.

디코더 LSTM은 도착 언어의 언어 모형이다. 이전의 글자나 단어들을 입력으로 주면 다음에 나올 글자나 단어를 예측한다. 일반적인 언어 모형과 다른 점은 초기 상태를 인코더 LSTM으로부터 넘겨받았기 때문에 출발 언어 문장에 따라 도착 언어 문장이 다르게 생성된다는 점이다. 이를 이용해서 기계 번역을 하는 것이다.

!



Seq2Seq는 기계 번역 외에도 입력과 출력이 길이가 다른 시퀀스인 경우에 모두 활용할 수 있다. 예를 들어 이메일이나 채팅에서 자동 답장을 하게 만들 경우, 상대방의 말을 입력으로 하고 답장할 말을 출력으로 하게 하면 된다.

13.1. Seq2Seq 실습

http://www.manythings.org/anki/ 에서 프랑스어-영어 데이터(`fra-eng.zip`)를 다운 받아 압축을 푼다.

만약 google colab을 쓰는 경우에는 아래 코드로 다운 받아 압축을 풀 수 있다.

```
!wget -c http://www.manythings.org/anki/fra-eng.zip && unzip -o fra-eng.zip
```

```
import pandas as pd
```

데이터는 영어와 프랑스어가 탭(\t)으로 구분되어 있고 컬럼명(header)은 없다.

```
df = pd.read_csv('fra.txt', sep='\t', header=None)
```

컬럼명을 붙여준다.

```
df.columns = ['eng', 'fra']
```

데이터의 뒷부분을 확인한다.

```
df.tail()
```

	eng	fra
154878	Top-down economics never works, said Obama. "T...	« L'économie en partant du haut vers le bas, ç...
154879	A carbon footprint is the amount of carbon dio...	Une empreinte carbone est la somme de pollutio...
154880	Death is something that we're often discourage...	La mort est une chose qu'on nous décourage sou...
154881	Since there are usually multiple websites on a...	Puisqu'il y a de multiples sites web sur chaqu...
154882	If someone who doesn't know your background sa...	Si quelqu'un qui ne connaît pas vos antécédent...

13.1.1. 글자 번호 매기기

번역기는 글자 단위로 번역하도록 하자. 사전을 상속받아 글자에 자동으로 번호를 붙여주는 클래스를 만든다.

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

토크나이저를 초기화한다. `char_level=True` 로 하면 글자 단위로 토큰화한다.

```
eng_tok = Tokenizer(char_level=True)
```

영어에 나온 글자들을 정리한다.

```
eng_tok.fit_on_texts(df['eng'].tolist())
```

영어 데이터를 글자 단위로 토큰화한다.

```
eng_data = eng_tok.texts_to_sequences(df['eng'].tolist())
```

프랑스어 데이터에도 동일한 처리를 해준다.

```
fra_tok = Tokenizer(char_level=True)
fra_tok.fit_on_texts(df['fra'].tolist())
fra_data = fra_tok.texts_to_sequences(df['fra'].tolist())
```

추가로 프랑스어에는 문장의 시작과 끝을 나타내는 기호를 추가해준다.

```
fra_tok.word_index['<START>'] = len(fra_tok.word_index) + 1
fra_tok.word_index['<END>'] = len(fra_tok.word_index) + 1
```

모든 프랑스어 문장의 앞뒤로 시작과 끝 기호를 붙인다.

```
new_fra_data = []
for s in fra_data:
    new_s = [fra_tok.word_index['<START>']] + s + [fra_tok.word_index['<END>']]
    new_fra_data.append(new_s)
fra_data = new_fra_data
```

영어와 프랑스어에 쓰이는 글자 수를 변수에 저장한다.

```
NUM_ENG_CHAR = len(eng_tok.word_index) + 1
NUM_FRA_CHAR = len(fra_tok.word_index) + 1
```

13.1.2. 패딩

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

```
eng_pad = pad_sequences(eng_data, padding='pre')
```

```
fra_pad = pad_sequences(fra_data, padding='post')
```

13.1.3. 모형

Seq2Seq 모형은 입력 언어를 처리하는 인코더와 출력 언어를 처리하는 디코더 2부분으로 되어 있다.

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

인코더

인코더는 영어 문장을 입력으로 받는다.

```
enc_input = Input(shape=(None, NUM_ENG_CHAR))
```

내부 상태를 디코더로 넘겨주어야 하기 때문에 `return_state=True` 로 설정한다.

```
encoder = LSTM(16, return_state=True, return_sequences=False)
```

인코더에 입력을 넣으면 출력과 내부 상태가 나오게 된다.

```
enc_output, state_h, state_c = encoder(enc_input)
```

LSTM에서 내부 상태는 `state_h` 와 `state_c` 두 종류가 있다. 이 중 `state_h` 는 이전 상태의 출력 `enc_output` 과 같다.

내부 상태를 리스트로 묶는다.

```
enc_state = [state_h, state_c]
```

디코더

디코더는 프랑스어를 한 글자씩 받으면 다음 프랑스어 글자를 예측한다.

```
dec_input = Input(shape=(None, NUM_FRA_CHAR))
```

```
decoder = LSTM(16, return_sequences=True, return_state=True)
```

디코더의 내부 상태는 처음에 인코더에서 넘겨 받은 것을 사용한다.

```
dec_output, _, _ = decoder(dec_input, initial_state=enc_state)
```

다음 글자를 예측한다.

```
predict_layer = Dense(NUM_FRA_CHAR, activation='softmax')
```

```
next_char = predict_layer(dec_output)
```

인코더의 입력과 디코더의 입력으로부터 다음 글자의 예측까지를 하나의 모형으로 만든다.

```
model = Model([enc_input, dec_input], next_char)
```

13.1.4. 데이터 생성자

데이터가 매우 크기 때문에 한 번에 one-hot encoding을 하면 메모리가 넘친다. 따라서 한 배치씩 데이터를 인코딩해서 모형에 넣어주는 생성자(generator)를 만들어야 한다. 데이터 생성자는 케라스의 `Sequence` 를 상속하여 만든다.

```
from keras.utils import Sequence
import numpy as np
```

다음은 x와 y 두 가지 데이터를 받아 한 번에 32개씩 one-hot encoding하여 공급하는 클래스이다.

```
class OneHotSequence(Sequence):
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.batch_size = 32

    def __len__(self):
        return len(self.x) // self.batch_size

    def __getitem__(self, idx):
        start = idx * self.batch_size
        end = start + self.batch_size

        # one-hot encoding
        x = np.eye(NUM_ENG_CHAR)[self.x[start:end,]]
        y = np.eye(NUM_FRA_CHAR)[self.y[start:end,]]

        target = np.expand_dims(self.y[start:end,], 2)

        return [x, y], target
```

`np.eye(N)[x]` 형태의 식은 one-hot encoding을 할 때 자주 쓰는 표현이니 기억해두자. `np.eye(N)` 는 항등행렬을 만들어준다. 항등행렬은 모든 값이 0이고 대각원소만 1이다. 아래는 크기가 3인 항등행렬이다.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

이 행렬에서 `[x]` 로 `x` 번째 행들을 뽑아내면 one-hot encoding이 된다. 예를 들어 `x` 가 `[2, 0, 1, 1]` 이면 다음과 같아진다.

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

이제 인코더 데이터와 디코더 데이터를 이 클래스에 넣어 생성자를 초기화한다.

```
seq = OneHotSequence(eng_pad, fra_pad)
```

생성자를 `len` 함수에 넣으면 실제로는 `__len__` 메소드가 호출되어 배치의 수가 나온다.

```
len(seq)
```

```
4840
```

생성자의 0번째 배치를 `seq[0]` 으로 꺼내보면 실제로는 `__getitem__` 메소드가 호출된다.

13.1.5. 학습

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
```

```
model.fit_generator(seq, epochs=1)
```

```
<keras.callbacks.History at 0x27e8257e160>
```

학습이 매우 오래 걸리므로 꼭 저장을 해둔다.

```
model.save('seq2seq.krs')
```

13.1.6. 번역용 모형

번역용 모형은 학습용 모형과 형태가 조금 다르다.

인코더

인코더는 인코더의 입력에서 인코더가 디코더에 넘겨주는 상태까지를 하나의 모형으로 만든다.

```
encoder_model = Model(enc_input, enc_state)
```

디코더

학습을 할 때는 인코더와 디코더의 입력을 한꺼번에 주기 때문에 인코더에서 디코더로 상태가 바로 넘어간다. 그런데 번역을 할 때는 한 번에 한 글자씩 예측을 하기 때문에, 학습할 때처럼 하면 매번 인코더의 모든 입력을 넣어주어야 한다.

번역을 할 때는 계산을 효율적으로 하기 위해 한 글자를 예측하고 그 때의 상태를 저장해두었다가 다음 글자를 예측할 때는 그 상태를 재사용한다. 이렇게 하면 매번 인코더와 디코더의 모든 입력을 다시 할 필요가 없게 된다.

이를 위해서 학습시킬 때 인코더에서 넘어온 `enc_state` 를 사용한 부분을 입력으로 대체한다.

```
prev_h = Input(shape=(16,))
prev_c = Input(shape=(16,))
```

```
prev_state = [prev_h, prev_c]
```

```
dec_output, current_h, current_c = decoder(dec_input, initial_state=prev_state)
```

```
current_state = [current_h, current_c]
```

```
pred_char = predict_layer(dec_output)
```

```
decoder_model = Model([dec_input] + prev_state, [pred_char] + current_state)
```

13.1.7. 번역

출발 언어 문장 전처리

출발 언어를 토큰화하고 패딩을 한 후

```
_, ENG_LENGTH = eng_pad.shape
```

```
x = eng_tok.texts_to_sequences(["This class is great!"])
```

```
x = pad_sequences(x, ENG_LENGTH, padding='pre')
```

one-hot encoding 하고

```
x = np.eye(NUM_ENG_CHAR)[x]
```

인코더

인코더에 넣어 상태를 가져온다.

```
state = encoder_model.predict(x)
```

디코더

도착 언어 문장을 빈 문자열로 만든다.

```
target_sentence = ''
```

문장 시작을 뜻하는 문자를 one-hot encoding으로 바꾼다.

```
target_seq = np.eye(NUM_FRA_CHAR)[fra_tok.word_index['<START>']]
```

```
target_seq.shape
```

```
(85,)
```

0번째 문서 0번째 글자이므로 형태를 `(1, 1, NUM_FRA_CHAR)` 형태로 바꾼다.

```
target_seq = np.array([[target_seq]])
```

```
target_seq.shape
```

```
(1, 1, 85)
```

변환한 입력을 인코더에서 구한 상태와 함께 디코더에 넣어준다.

```
output_token, h, c = decoder_model.predict([target_seq] + state)
```

첫 번째 문서(`0`)의 마지막 글자(`-1`)에서 가장 확률이 높을 글자의 번호를 구한다.

```
token_index = output_token[0, -1, :].argmax()
```

번호를 글자로 바꾼다.

```
char = fra_tok.index_word[token_index]
```

도착 언어 문장에 추가해준다.

```
target_sentence += char
```

위의 과정을 문장 끝을 나타내는 글자가 나올 때까지 반복한다.

```
while token_index != fra_tok.word_index['<END>']:
    target_seq = np.eye(NUM_FRA_CHAR)[token_index]
    target_seq = np.array([[target_seq]])

    output_token, h, c = decoder_model.predict([target_seq, h, c])

    token_index = output_token[0, -1, :].argmax()

    char = fra_tok.index_word[token_index]
    target_sentence += char

    # 무한 루프 방지를 위해 300글자가 넘으면 강제 중단한다.
    if len(target_sentence) > 300:
        break
```