#### <https://tykimos.github.io/2018/09/14/ten-minute_introduction_to_sequence-to-sequence_learning_in_Keras/>

#### sequence-to-sequence 학습이란?

sequence-to-sequence(Seq2Seq) 학습은 한 도메인(예: 영어 문장)에서 다른 도메인(예: 불어로 된 문장)으로 시퀀스(sequence)를 변환하는 모델 학습을 의미합니다.

"the cat sat on the mat" -> **[**Seq2Seq model] -> "le chat etait assis sur le tapis"

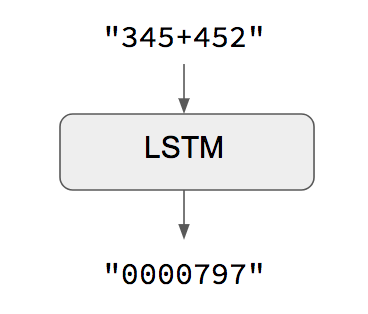
이 모델은 기계 번역 혹은 자유로운 질의응답에 사용됩니다. (자연어 질문을 주어 자연어 응답을 생성) –일반적으로, 텍스트를 생성해야 할 경우라면 언제든지 적용할 수 있습니다.

해당 작업을 다루는 여러 가지 방법이(**RNN** 혹은 **1D convnets**) 있습니다.

*이번 문서에선****RNN****을 사용하고 있습니다.*

#### 자명한(명확한) 사례 : 입력과 출력 시퀀스 길이가 같을 때

입력과 출력 시퀀스 길이가 같을 경우, 케라스 Long Short-Term Memory(LSTM)이나 GRU 계층(혹은 다수의 계층) 같은 모델들을 간단하게 구현할 수 있습니다. [예제 스크립트](https://github.com/fchollet/keras/blob/master/examples/addition_rnn.py)에선 어떻게 RNN으로 문자열로 인코딩된 숫자들에 대한 덧셈 연산을 학습할 수 있는지 보여주고 있습니다.



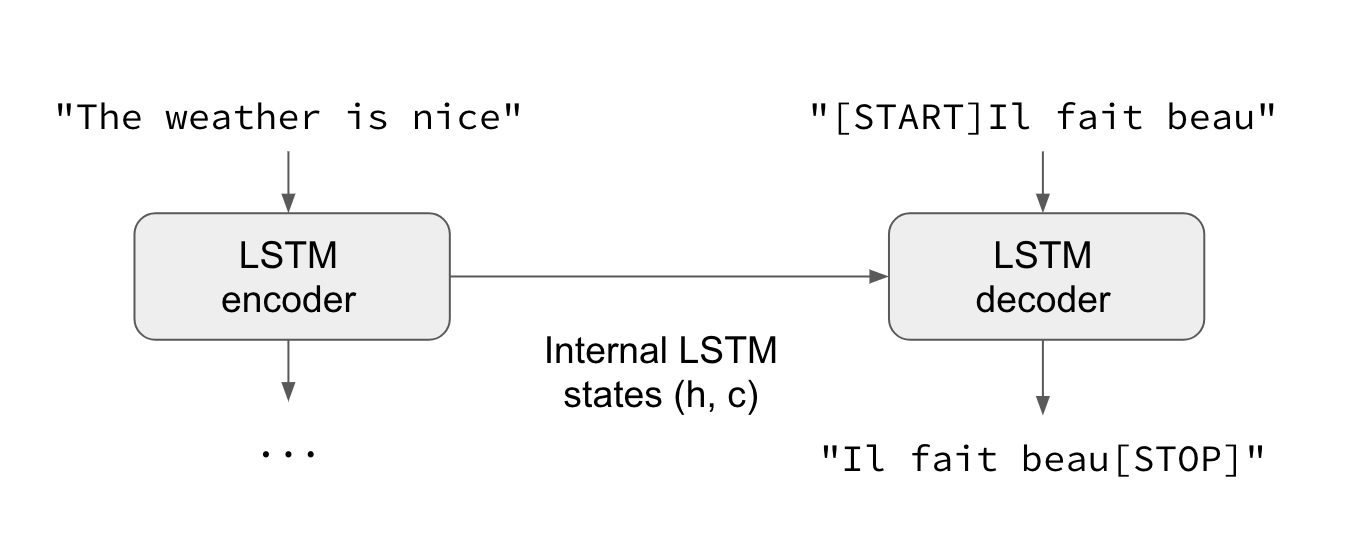
이 방법의 주의점은 주어진 input[...t]으로 target[...t]을 생성 가능하다고 가정하는 것입니다. 일부 경우(예: 숫자된 문자열 추가)에선 정상적으로 작동하지만, 대부분의 경우에는 작동하지 않습니다. 일반적으론, 목표 시퀀스를 생성하기 위해 전체 입력 시퀀스 정보가 필요합니다.

#### 일반 사례 : 표준 sequence-to-sequence

일반적으론 입력과 출력 시퀀스 길이가 다르고(예: 기계 번역) 목표 시퀀스를 예측하기 위해 전체 입력 시퀀스 정보가 필요합니다. 이를 위해 고급 설정이 필요하며, 일반적으로 “Seq2Seq models”를 언급할 때 참조합니다. 동작 방법은 하단을 참조하시면 되겠습니다.

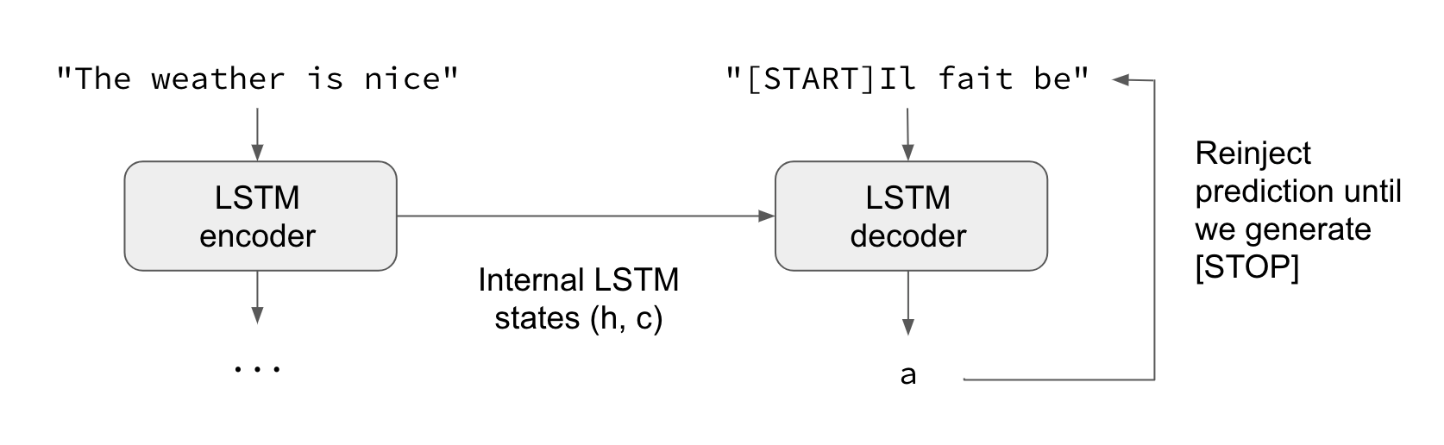
* 하나(혹은 여러 개)의 RNN 계층은 “encoder” 역할을 합니다 : 입력 시퀀스를 처리하고 자체 내부 상태를 반환합니다. 여기서, encoder RNN의 결과는 사용하지 않고 상태만 복구시킵니다. 이 상태가 다음 단계에서 decoder의 “문맥” 혹은 “조건” 역할을 합니다.
* 또 하나(혹은 여러 개)의 RNN 계층은 “decoder” 역할을 합니다 : 목표 시퀀스에서 이전 문자들에 따라 다음 문자들을 예측하도록 훈련됩니다. 상세히 말하면, 목표 시퀀스를 같은 시퀀스로 바꾸지만 후에 “teacher forcing”이라는 학습 과정인, 한 개의 time step만큼 offset\*이 되도록 훈련됩니다. 중요한 건, encoder는 encoder 상태 벡터들을 초기 상태로 사용하고 이는 decoder가 생성할 정보를 얻는 방법이기도 합니다. 사실, decoder는 주어진 target[...t]을 입력 시퀀스에 맞춰서 target[t+1...]을 생성하는 법을 학습합니다.

*offset 의 예: 문자 A의 배열이 ‘abcdef’를 가질 때, ‘c’가 A 시작점에서 2의 offset을 지님*



추론 방식(즉: 알 수 없는 입력 시퀀스를 해석하려고 할 때)에선 약간 다른 처리를 거치게 됩니다.

* 1) 입력 시퀀스를 상태 벡터들로 바꿉니다.
* 2) 크기가 1인 목표 시퀀스로 시작합니다. (시퀀스의 시작 문자에만 해당)
* 3) 상태 벡터들과 크기가 1인 목표 시퀀스를 decoder에 넣어 다음 문자에 대한 예측치를 생성합니다.
* 4) 이런 예측치들을 사용해 다음 문자의 표본을 뽑습니다.(간단하게 argmax를 사용)
* 5) 목표 시퀀스에 샘플링된 문자를 붙입니다.
* 6) 시퀀스 종료 문자를 생성하거나 끝 문자에 도달할 때까지 앞의 과정을 반복합니다.



이같은 과정은 “teacher forcing” 없이 Seq2Seq를 학습시킬 때 쓰일 수도 있습니다. (decoder의 예측치들을 decoder에 다시 기재함으로써)

#### 케라스 예제

실제 코드를 통해 위의 아이디어들을 설명하겠습니다.

예제를 구현하기 위해, 영어 문장과 이에 대해 불어로 번역한 문장 한 쌍으로 구성된 데이터 세트를 사용합니다. ([manythings.org/anki](http://www.manythings.org/anki/)에서 내려받을 수 있습니다.) 다운받을 파일은 fra-eng.zip입니다. 입력 문자를 문자 단위로 처리하고, 문자 단위로 출력문자를 생성하는 문자 수준 Seq2Seq model을 구현할 예정입니다. 또 다른 옵션은 기계 번역에서 좀 더 일반적인 단어 수준 model입니다. 글 끝단에서, Embedding계층을 사용하여 설명에 쓰인 model을 단어 수준 model로 바꿀 수 있는 참고 사항도 보실 수 있습니다.

설명에 쓰인 예제 전체 code는 [Github](https://github.com/fchollet/keras/blob/master/examples/lstm_seq2seq.py)에서 보실 수 있습니다.

진행 과정 요약으론:

* 1) 문장들을 3차원 배열(encoder\_input\_data, decoder\_input\_data, decoder\_target\_data)로 변환합니다.
  + encoder\_input\_data는 (num\_pairs, max\_english\_sentence\_length, num\_english\_characters) 형태의 3차원 배열로 영어 문장의 one-hot 형식 벡터 데이터를 갖고 있습니다.
  + decoder\_input\_data는 (num\_pairs, max\_french\_sentence\_length, num\_french\_characters)형태의 3차원 배열로 불어 문장의 one-hot형식 벡터 데이터를 갖고 있습니다.
  + decoder\_target\_data는 decoder\_input\_data와 같지만 하나의 time step만큼 offset 됩니다. decoder\_target\_data[:, t, :]는 decoder\_input\_data[:, t + 1, :]와 같습니다.
* 2) 기본 LSTM 기반의 Seq2Seq model을 주어진 encoder\_input\_data와 decoder\_input\_data로 decoder\_target\_data를 예측합니다. 해당 model은 teacher forcing을 사용합니다.
* 3) model이 작동하는지 확인하기 위해 일부 문장을 디코딩(decoding)합니다. (encoder\_input\_data의 샘플을 decoder\_target\_data의 표본으로 변환합니다.)

(문장을 디코딩하는)학습 단계와 추론 단계는 꽤나 다르기 때문에, 같은 내부 계층을 사용하지만 서로 다른 모델을 사용합니다.

다음은 원문 저자가 제공하는 model로 keras RNN의 3가지 핵심 특징들을 사용합니다:

* return\_state는 encoder의 출력과 내부 RNN 상태인 리스트를 반환하도록 RNN을 구성하는 인수입니다. 이는 encoder의 상태를 복구하는 데 사용합니다.
* inital\_state는 RNN의 초기 상태를 지정하는 인수입니다. 초기 상태로 incoder를 decoder로 전달하는 데 사용합니다.
* return\_sequences는 출력된 전체 시퀀스를 반환하도록 구성하는 인수(마지막 출력을 제외하곤 기본 동작)로 decoder에 사용합니다.

from keras.models import Model

from keras.layers import Input, LSTM, Dense

*# 입력 시퀀스의 정의와 처리*

encoder\_inputs **=** Input(shape**=**(None, num\_encoder\_tokens))

encoder **=** LSTM(latent\_dim, return\_state**=**True)

encoder\_outputs, state\_h, state\_c **=** encoder(encoder\_inputs)

*# `encoder\_outputs`는 버리고 상태(`state\_h, state\_c`)는 유지*

encoder\_states **=** [state\_h, state\_c]

*# `encoder\_states`를 초기 상태로 사용해 decoder를 설정*

decoder\_inputs **=** Input(shape**=**(None, num\_decoder\_tokens))

*# 전체 출력 시퀀스를 반환하고 내부 상태도 반환하도록 decoder를 설정.*

*# 학습 모델에서 상태를 반환하도록 하진 않지만, inference에서 사용할 예정.*

decoder\_lstm **=** LSTM(latent\_dim, return\_sequences**=**True, return\_state**=**True)

decoder\_outputs, \_, \_ **=** decoder\_lstm(decoder\_inputs,

initial\_state**=**encoder\_states)

decoder\_dense **=** Dense(num\_decoder\_tokens, activation**=**'softmax')

decoder\_outputs **=** decoder\_dense(decoder\_outputs)

*# `encoder\_input\_data`와 `decoder\_input\_data`를 `decoder\_target\_data`로 반환하도록 모델을 정의*

model **=** Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

밑의 2줄로 샘플의 20%를 검증 데이터 세트로 손실을 관찰하면서 모델을 학습시킵니다.

*# 학습 실행*

model**.**compile(optimizer**=**'rmsprop', loss**=**'categorical\_crossentropy')

model**.**fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data,

batch\_size**=**batch\_size,

epochs**=**epochs,

validation\_split**=**0.2)

맥북 CPU에서 1시간 정도 학습한 후에, 추론할 준비가 됩니다. 테스트 문장을 decode하기 위해 반복 수행할 것입니다.

* 1) 입력문장을 encode하고 초기 상태에 decoder의 상태를 가지고 옵니다.
* 2) 초기 상태 decoder의 한 단계와 “시퀀스 시작” 토큰을 목표로 실행합니다. 출력은 다음 목표 문자입니다.
* 3) 예측된 목표 문자를 붙이고 이를 반복합니다.

다음은 추론을 설정한 부분입니다.

encoder\_model **=** Model(encoder\_inputs, encoder\_states)

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]

decoder\_outputs, state\_h, state\_c **=** decoder\_lstm(

decoder\_inputs, initial\_state**=**decoder\_states\_inputs)

decoder\_states **=** [state\_h, state\_c]

decoder\_outputs **=** decoder\_dense(decoder\_outputs)

decoder\_model **=** Model(

[decoder\_inputs] **+** decoder\_states\_inputs,

[decoder\_outputs] **+** decoder\_states)

아래의 코드는 위의 추론 루프를 구현하는 데 사용했습니다.

**def** **decode\_sequence**(input\_seq):

*# 상태 벡터로서 입력값을 encode*

states\_value **=** encoder\_model**.**predict(input\_seq)

*# 길이가 1인 빈 목표 시퀀스를 생성*

target\_seq **=** np**.**zeros((1, 1, num\_decoder\_tokens))

*# 대상 시퀀스 첫 번째 문자를 시작 문자로 기재.*

target\_seq[0, 0, target\_token\_index['\t']] **=** 1.

*# 시퀀스들의 batch에 대한 샘플링 반복(간소화를 위해, 배치 크기는 1로 상정)*

stop\_condition **=** False

decoded\_sentence **=** ''

**while** **not** stop\_condition:

output\_tokens, h, c **=** decoder\_model**.**predict(

[target\_seq] **+** states\_value)

*# 토큰으로 샘플링*

sampled\_token\_index **=** np**.**argmax(output\_tokens[0, **-**1, :])

sampled\_char **=** reverse\_target\_char\_index[sampled\_token\_index]

decoded\_sentence **+=** sampled\_char

*# 탈출 조건 : 최대 길이에 도달하거나*

*# 종료 문자를 찾을 경우*

**if** (sampled\_char **==** '\n' **or**

len(decoded\_sentence) **>** max\_decoder\_seq\_length):

stop\_condition **=** True

*# (길이 1인) 목표 시퀀스 최신화*

target\_seq **=** np**.**zeros((1, 1, num\_decoder\_tokens))

target\_seq[0, 0, sampled\_token\_index] **=** 1.

*# 상태 최신화*

states\_value **=** [h, c]

**return** decoded\_sentence

몇 가지 좋은 결과를 얻게 됩니다. (학습 테스트에서 추출한 샘플을 해독하기에 놀랄만한 결과는 아니지만..)

Input sentence: Be nice.

Decoded sentence: Soyez gentil **!**

-

Input sentence: Drop it!

Decoded sentence: Laissez tomber **!**

-

Input sentence: Get out!

Decoded sentence: Sortez !

이로써 keras의 Seq2Seq model에 대한 10분 안에 알려주기 튜토리얼을 마칩니다. 알림 : 설명에 쓰인 예제 전체 code는 [Github](https://github.com/fchollet/keras/blob/master/examples/lstm_seq2seq.py)에서 보실 수 있습니다.

### 참고문서

* [Sequence to Sequence Learning with Neural Networks](https://arxiv.org/abs/1409.3215)
* [Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation](https://arxiv.org/abs/1406.1078)

### 추가 FAQ

#### LSTM 대신 GRU 계층을 사용하려면 어떻게 해야 합니까?

GRU는 오직 상태 1개만 가지지만 LSTM은 상태가 2개가 있기 때문에 실제론 약간 단순합니다. 아래에 GRU 계층을 사용해 학습 모델을 조정하는 방법이 있습니다.

encoder\_inputs **=** Input(shape**=**(None, num\_encoder\_tokens))

encoder **=** GRU(latent\_dim, return\_state**=**True)

encoder\_outputs, state\_h **=** encoder(encoder\_inputs)

decoder\_inputs **=** Input(shape**=**(None, num\_decoder\_tokens))

decoder\_gru **=** GRU(latent\_dim, return\_sequences**=**True)

decoder\_outputs **=** decoder\_gru(decoder\_inputs, initial\_state**=**state\_h)

decoder\_dense **=** Dense(num\_decoder\_tokens, activation**=**'softmax')

decoder\_outputs **=** decoder\_dense(decoder\_outputs)

model **=** Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

#### 정수형 시퀀스가 포함된 단어단계 모델을 사용하려면 어떻게 해야 합니까?

만약 입력이 정수형 시퀀스일 경우(예: 사전에서 색인으로 encode된 단어 시퀀스)라면? Embedding 계층을 통해서 정수형 토큰을 포함시킬 수 있습니다. 구현은 아래와 같습니다:

*# 입력 시퀀스 정의와 처리*

encoder\_inputs **=** Input(shape**=**(None,))

x **=** Embedding(num\_encoder\_tokens, latent\_dim)(encoder\_inputs)

x, state\_h, state\_c **=** LSTM(latent\_dim,

return\_state**=**True)(x)

encoder\_states **=** [state\_h, state\_c]

*# `encoder\_states`를 초기 상태로 사용해 decoder를 설정*

decoder\_inputs **=** Input(shape**=**(None,))

x **=** Embedding(num\_decoder\_tokens, latent\_dim)(decoder\_inputs)

x **=** LSTM(latent\_dim, return\_sequences**=**True)(x, initial\_state**=**encoder\_states)

decoder\_outputs **=** Dense(num\_decoder\_tokens, activation**=**'softmax')(x)

*# `encoder\_input\_data`와 `decoder\_input\_data`를 `decoder\_target\_data`로 반환하도록 모델을 정의*

model **=** Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

*# 컴파일 & 학습 실행*

model**.**compile(optimizer**=**'rmsprop', loss**=**'categorical\_crossentropy')

*# `decoder\_target\_data`는 `decoder\_input\_data` 같은 정수 시퀀스보단 one-hot 인코딩 형식이 되어야 함.*

model**.**fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data,

batch\_size**=**batch\_size,

epochs**=**epochs,

validation\_split**=**0.2)

#### 학습하는 동안 Teacher forcing을 사용하지 않으려면 어떻게 해야 합니까?

일부 환경에서 완전한 입력-목표 시퀀스 쌍을 버퍼링할 수 없듯이(예를 들어, 만약 매우 긴 시퀀스는 online 학습) 이는 전체 목표 시퀀스로 접근할 수 없기 때문에 Teacher forcing을 사용할 수 없습니다. 이 경우 decoder의 예측값을 입력으로 재입력하여 학습을 실행할 수 있습니다. (그저 추론될 수 있도록)

출력값을 재주입하는 루프를 설계한 모델을 구축하면 다음과 같은 결과를 얻을 수 있습니다.

from keras.layers import Lambda

from keras import backend **as** K

*# 첫 부분은 바꿀 부분 없음.*

encoder\_inputs **=** Input(shape**=**(None, num\_encoder\_tokens))

encoder **=** LSTM(latent\_dim, return\_state**=**True)

encoder\_outputs, state\_h, state\_c **=** encoder(encoder\_inputs)

states **=** [state\_h, state\_c]

*# 한 번에 한 개의 time step이 진행되도록 decoder를 설정*

decoder\_inputs **=** Input(shape**=**(1, num\_decoder\_tokens))

decoder\_lstm **=** LSTM(latent\_dim, return\_sequences**=**True, return\_state**=**True)

decoder\_dense **=** Dense(num\_decoder\_tokens, activation**=**'softmax')

all\_outputs **=** []

inputs **=** decoder\_inputs

**for** \_ **in** range(max\_decoder\_seq\_length):

*# 한 개의 time step에서 decoder 실행*

outputs, state\_h, state\_c **=** decoder\_lstm(inputs,

initial\_state**=**states)

outputs **=** decoder\_dense(outputs)

*# 현재 예측치를 저장(나중에 모든 예측치를 연결할 수 있음)*

all\_outputs**.**append(outputs)

*# 다음 반복을 위해 현 출력 데이터를 입력 데이터로 재지정하고 상태 또한 최신화.*

inputs **=** outputs

states **=** [state\_h, state\_c]

*# 모든 예측치를 연결*

decoder\_outputs **=** Lambda(**lambda** x: K**.**concatenate(x, axis**=**1))(all\_outputs)

*# 앞에서처럼 모델을 정의하고 컴파일.*

model **=** Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

model**.**compile(optimizer**=**'rmsprop', loss**=**'categorical\_crossentropy')

*# 시작문자가 포함된 decoder 입력 정보*

decoder\_input\_data **=** np**.**zeros((num\_samples, 1, num\_decoder\_tokens))

decoder\_input\_data[:, 0, target\_token\_index['\t']] **=** 1.

*# 앞에서처럼 모델 학습*

model**.**fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data,

batch\_size**=**batch\_size,

epochs**=**epochs,

validation\_split**=**0.2)

만약 추가적인 의문점이 있다면, [Twitter](https://twitter.com/fchollet)로 연락해주세요.

*이 글은 2018 컨트리뷰톤에서*[*Contributue to Keras*](https://github.com/KerasKorea/KEKOxTutorial)*프로젝트로 진행했습니다.  
Translator :*[*mike2ox*](https://github.com/mike2ox)*(Moonhyeok Song)  
Translator Email :*[*firefinger07@gmail.com*](mailto:firefinger07@gmail.com)