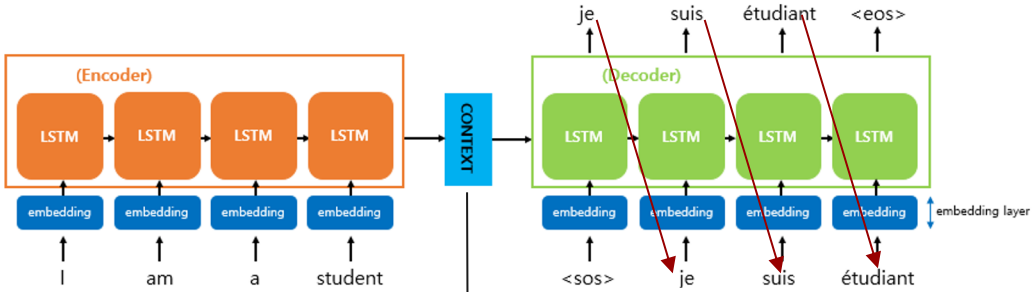
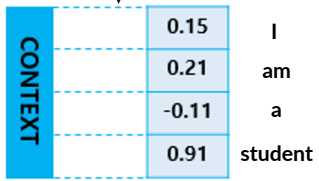
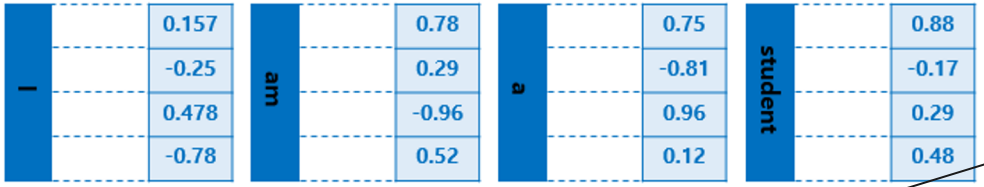
Neural Machine Translation using Seq2Seq LSTM (Keras)

Jaechul Roh (Harry)

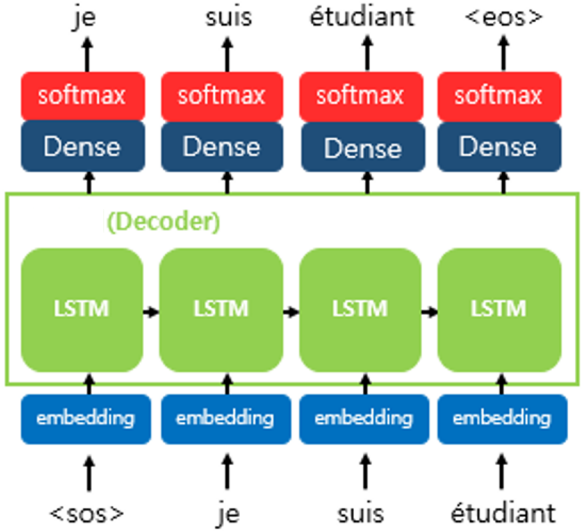
Sequence-to-Sequence (Seq2Seq)







Word Embedding Vectors

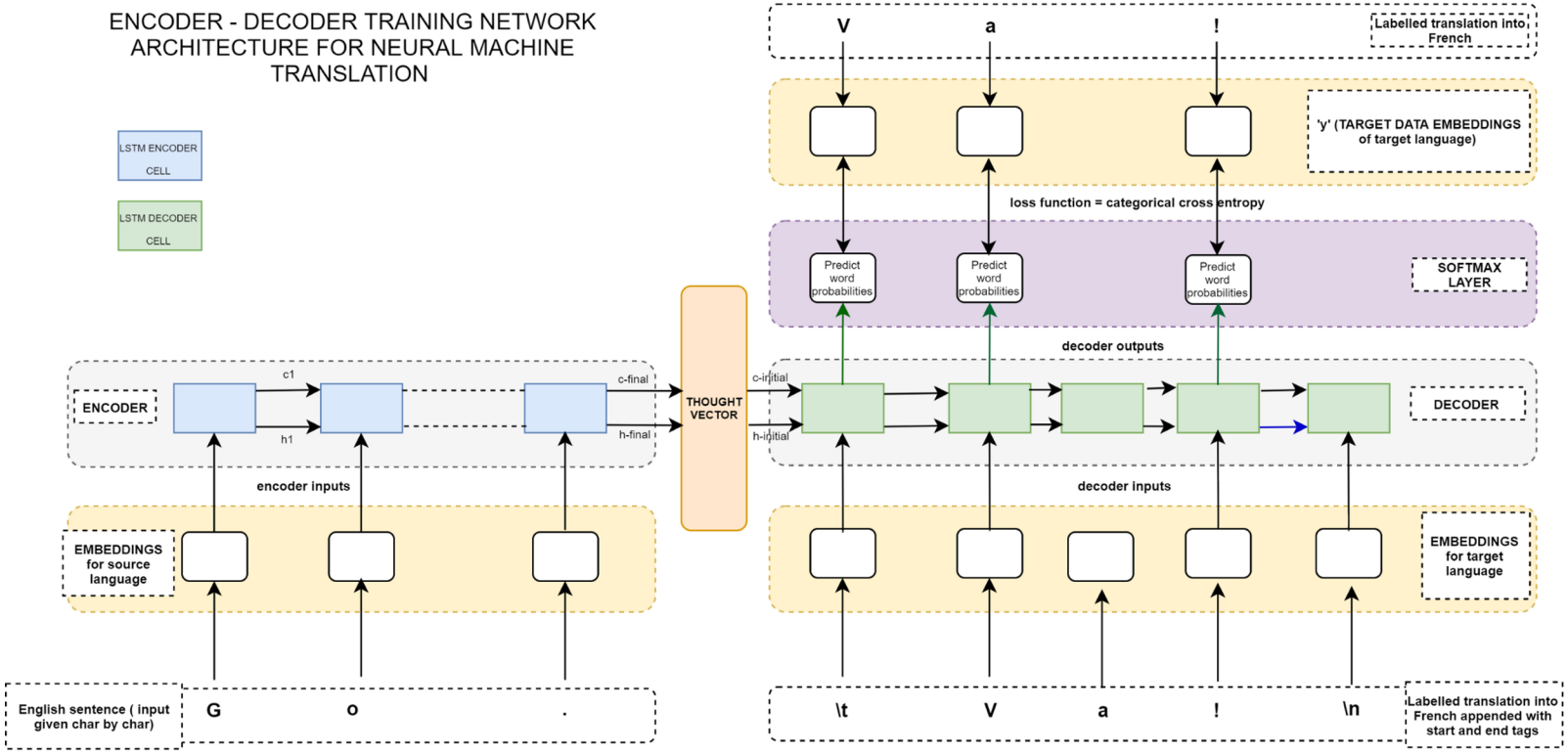


"Dense": Fully connected layer Module (Keras)

Used 4-dimensional example for simplicity
(Normally it uses 256, 512 , … in real life)

"Does NOT use embedding since it is a character-level translation"

Sequence-to-Sequence (Seq2Seq) (2)



Sequence-to-Sequence (Seq2Seq) (3)

Really Good Animation (Jay Alammar github)

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Data Preprocessing

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

Parallel Corpus download

Assign "English" & "French" as column names

```
[ ] import pandas as pd
    from keras.preprocessing.text import Tokenizer
    from keras.preprocessing.sequence import pad_sequences
    from keras.models import Model
    from keras.layers import Dense, Embedding, Input, LSTM
    import numpy as np
```

```
df = pd.read_csv('fra.txt', sep='\t', header=None)
df.drop(df.columns[len(df.columns)-1], axis=1, inplace=True)
df.columns = ['English', 'French']

df = df[0:60000]
df.sample(10)
```

	English	French
39183	Why are you so busy?	Pourquoi es-tu si occupé ?
38859	Were you born there?	Êtes-vous née là-bas ?
48590	I got soap in my eyes.	Je me suis pris du savon dans les yeux.
10626	I was cleaning.	J'étais en train de nettoyer.
47342	Don't open it, please.	Ne l'ouvrez pas, s'il vous plaît.
21401	What is going on?	Qu'est-ce qui se passe ?
54091	Do you think I'm right?	Penses-tu que j'ai raison ?
56884	It just has to be done.	Ça doit tout simplement être fait.
40137	Do I seem old to you?	Est-ce que je vous parais vieux ?
39743	Add more water to it.	Ajoutez-y davantage d'eau.

Add '\t' (<sos>) and '\n' (<eos>)
to *start of sentence* and *end of sentence*

```
# <sos> : start of sentence (use '\t')
# <eos> : end of sentence   (use '\n')

df.French = df.French.apply(lambda x: '\t ' + x + ' \n')
df.sample(10)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/generic
A value is trying to be set on a copy of a slice from a Da
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydat
  self[name] = value
```

	English	French
27218	You can come home.	\t Vous pouvez venir à la maison. \n
57116	It's too noisy in here.	\t C'est trop bruyant ici. \n
18022	How tall you are!	\t Comme tu es grande ! \n
13059	Come if you can.	\t Viens si tu peux ! \n
12301	We'll call you.	\t Nous vous appellerons. \n
333	I'm Tom.	\t Je suis Tom. \n
2128	I'm so fat.	\t Je suis tellement grosse. \n
25476	That's my teacher.	\t C'est mon enseignant. \n
18913	I'll fix a drink.	\t Je vais préparer une boisson. \n
31894	Tom gets headaches.	\t Tom souffre du mal de tête. \n

```
# English character set
eng_vocab = set()   # use set instead of list for uniqueness
for sent in df.English:
  for char in sent:
    eng_vocab.add(char)

# French character set
fra_vocab = set()
for sent in df.French:
  for char in sent:
    fra_vocab.add(char)
```

```
ENG_vocab_size = len(eng_vocab) + 1
FRA_vocab_size = len(fra_vocab) + 1

print('No. of English char: ', ENG_vocab_size)
print('No. of French char: ', FRA_vocab_size)
```

```
No. of English char:  80
No. of French char:  105
```

```
# Assign index to each characters
# The characters need to be sorted

ENG_vocab = sorted(list(eng_vocab))
FRA_vocab = sorted(list(fra_vocab))
print(ENG_vocab[30:40])
print(FRA_vocab[30:40])
```

```
['H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q']
['E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']
```

Encoder Input / Decoder Input / Decoder Target

```
# make dictionary of {char : index}
ENG_idx_dict = dict([(char, i + 1) for i, char in enumerate(ENG_vocab)])
FRA_idx_dict = dict([(char, i + 1) for i, char in enumerate(FRA_vocab)])

print(ENG_idx_dict)
print(FRA_idx_dict)
```

```
{' ': 1, '!': 2, '"': 3, '$': 4, '%': 5, '&': 6, "'": 7, ',': 8, '-': 9, '.': 10, '/': 11, '0':
{'\t': 1, '\n': 2, ' ': 3, '!': 4, '"': 5, '$': 6, '%': 7, '&': 8, "'": 9, '(': 10, ')': 11, ',
```

```
# Encoder ("English" -> Encoder input)
# Change every sentence in to a sequence of indexes
# Make a list of list --> [ [...], [...], [...] ]

# Encoder input

ENCODER_input = []
for sent in df.English:   # sentence level
  encoded_sent = []
  for english_char in sent:       # character level
    encoded_sent.append(ENG_idx_dict[english_char])

  ENCODER_input.append(encoded_sent)

for i in ENCODER_input[:10]:
  print('Encoded English sentence: ', i)
```

```
Encoded English sentence:  [30, 64, 10]
Encoded English sentence:  [30, 64, 10]
Encoded English sentence:  [30, 64, 10]
Encoded English sentence:  [31, 58, 10]
Encoded English sentence:  [31, 58, 10]
Encoded English sentence:  [41, 70, 63, 2]
Encoded English sentence:  [41, 70, 63, 2]
Encoded English sentence:  [41, 70, 63, 2]
Encoded English sentence:  [41, 70, 63, 2]
Encoded English sentence:  [41, 70, 63, 2]
```

```
# Decoder input

DECODER_input = []
for sent in df.French:
  encoded_sent = []
  for french_char in sent:
    encoded_sent.append(FRA_idx_dict[french_char])

  DECODER_input.append(encoded_sent)

for i in DECODER_input[:5]:
  print('Encoded French sentence: ', i)
```

```
Encoded French sentence:  [1, 3, 48, 53, 3, 4, 3, 2]
Encoded French sentence:  [1, 3, 39, 53, 70, 55, 60, 57, 14, 3, 2]
Encoded French sentence:  [1, 3, 28, 67, 73, 59, 57, 3, 4, 3, 2]
Encoded French sentence:  [1, 3, 45, 53, 64, 73, 72, 3, 4, 3, 2]
Encoded French sentence:  [1, 3, 45, 53, 64, 73, 72, 14, 3, 2]
```

```
# We can actually remove the "<sos>" token
# according to the decoder target structure

# Decoder Target

DECODER_TARGET = []
for sent in df.French:
  encoded_sent = []
  for french_char in sent:
    encoded_sent.append(FRA_idx_dict[french_char])

  encoded_sent = encoded_sent[1:]   # remove <sos> token
  DECODER_TARGET.append(encoded_sent)

for i in DECODER_TARGET[:5]:
  print('Target French sentence: ', i)
```

```
Target French sentence:  [3, 48, 53, 3, 4, 3, 2]
Target French sentence:  [3, 39, 53, 70, 55, 60, 57, 14, 3, 2]
Target French sentence:  [3, 28, 67, 73, 59, 57, 3, 4, 3, 2]
Target French sentence:  [3, 45, 53, 64, 73, 72, 3, 4, 3, 2]
Target French sentence:  [3, 45, 53, 64, 73, 72, 14, 3, 2]
```

Calculating Max Sentence Length / Padding / One-Hot Encoding

```
max_ENG_len = max([len(sent) for sent in df.English])
max_FRA_len = max([len(sent) for sent in df.French])

print("MAX English sentence length: ", max_ENG_len)
print("MAX French sentence length: ", max_FRA_len)
```

```
MAX English sentence length:  23
MAX French sentence length:  76
```

```
# Process padding to match the sentence lengths
ENCODER_input = pad_sequences(ENCODER_input, maxlen=max_ENG_len, padding='post')
DECODER_input = pad_sequences(DECODER_input, maxlen=max_FRA_len, padding='post')
DECODER_target = pad_sequences(DECODER_TARGET, maxlen=max_FRA_len, padding='post')
```

```
from tensorflow.keras.utils import to_categorical

# Process one-hot-encoding
ENCODER_input = to_categorical(ENCODER_input)
DECODER_input = to_categorical(DECODER_input)
DECODER_target = to_categorical(DECODER_target)
```

```
ENCODER_input[0][0:5]
```

```
array([[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],
      dtype=float32)
```

TRAINING Seq2Seq Model

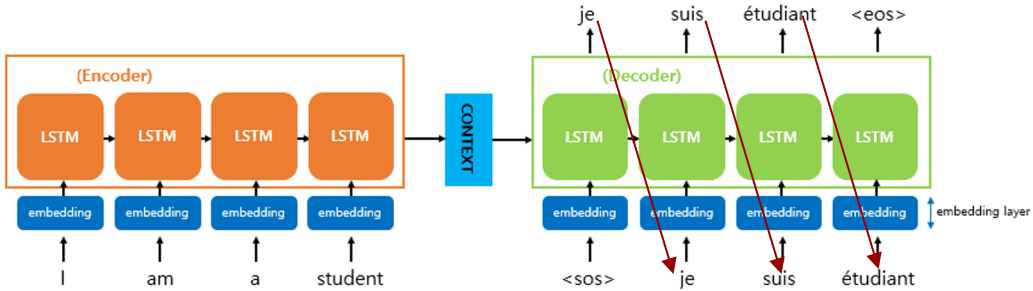
Seq2Seq LSTM TRAINING Model

ENCODER LSTM

Encoder LSTM

```
# LSTM(units, return_state, return_sequences)
# "units": dimensionality of output space
# "return_state": whether to return the last state in addition to the output
# "return_sequences": whether to return the LAST OUTPUT
#                     (in the output sequence or full sequence)

# state_h: hidden state (SHORT TERM memory)
# state_c: cell state (LONG TERM memory)


ENCODER_inputs = Input(shape=(None, ENG_vocab_size))
ENCODER_lstm = LSTM(units=256, return_state=True)

ENCODER_outputs, state_h, state_c = ENCODER_lstm(ENCODER_inputs)

ENCODER_states = [state_h, state_c]

ENCODER_outputs
```

```
<KerasTensor: shape=(None, 256) dtype=float32 (created by layer 'lstm')>
```



Context vector = number of hidden units in the ENCODER RNN

= dimensionality of the output space (256)

return_state=True : need to pass the state values of the ENCODER to the DECODER

ENCODER_outputs, state_h, state_c = ENCODER_lstm(ENCODER_inputs)

ENCODER_states = [state_h, state_c)

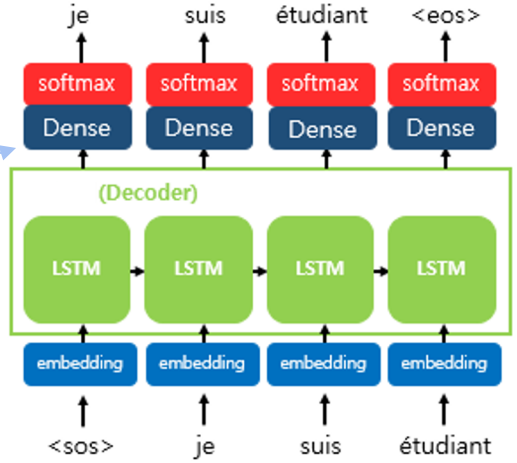
Seq2Seq LSTM TRAINING Model (2)

Decoder LSTM

```
[ ] DECODER_inputs = Input(shape=(None, FRA_vocab_size))
    DECODER_lstm = LSTM(units=256, return_sequences=True, return_state=True)

    DECODER_outputs, _, _ = DECODER_lstm(DECODER_inputs, initial_state=ENCODER_states)

    DECODER_softmax_layer = Dense(FRA_vocab_size, activation='softmax')
    DECODER_outputs = DECODER_softmax_layer(DECODER_outputs)
```

```
INPUTS = [ENCODER_inputs, DECODER_inputs]
OUTPUTS = DECODER_outputs

model = Model(INPUTS, OUTPUTS)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, None, 80)]   0           []

 input_2 (InputLayer)           [(None, None, 105)]  0           []

 lstm (LSTM)                    [(None, 256),        345088      ['input_1[0][0]']
                                 (None, 256),
                                 (None, 256)]

 lstm_1 (LSTM)                  [(None, None, 256),  370688      ['input_2[0][0]',
                                 (None, 256),                     'lstm[0][1]',
                                 (None, 256)]                     'lstm[0][2]']

 dense (Dense)                  (None, None, 105)    26985       ['lstm_1[0][0]']

==================================================================================================
Total params: 742,761
Trainable params: 742,761
Non-trainable params: 0
__________________________________________________________________________________________________
```



DECODER LSTM

1. At each time step, the DECODER_LSTM will give an output vector.
2. The corresponding vector will return the probability of each "token" in the output sequence through the SOFTMAX function.
3. Then, the DECODER decides the specific token at each time step based on the probability

Seq2Seq LSTM TRAINING Model (3)

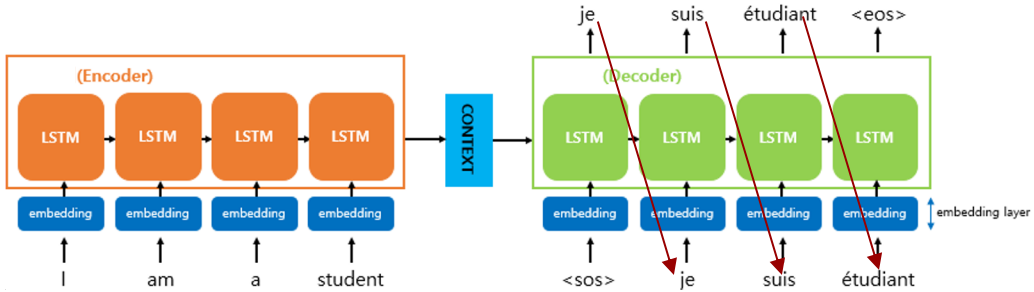
```
[ ]  x = [ENCODER_input, DECODER_input]

     model.fit(x=x,
               y=DECODER_target,
               batch_size=64,
               epochs=5,
               validation_split=0.2)
```

```
Epoch 1/5
750/750 [==============================] - 40s 47ms/step - loss: 0.7535 - val_loss: 0.6638
Epoch 2/5
750/750 [==============================] - 33s 43ms/step - loss: 0.4570 - val_loss: 0.5355
Epoch 3/5
750/750 [==============================] - 33s 44ms/step - loss: 0.3816 - val_loss: 0.4718
Epoch 4/5
750/750 [==============================] - 33s 44ms/step - loss: 0.3393 - val_loss: 0.4341
Epoch 5/5
750/750 [==============================] - 33s 43ms/step - loss: 0.3107 - val_loss: 0.4160
<keras.callbacks.History at 0x7fccee617f50>
```

```
print(ENCODER_input.shape)
print(DECODER_input.shape)
print(DECODER_target.shape)
```

```
(60000, 23, 80)
(60000, 76, 105)
(60000, 76, 105)
```

"Train the model based on the ENCODED English Text and French Text"



TESTING Seq2Seq Model

Seq2Seq LSTM TESTING Model

TESTING the trained Seq2Seq Model

- The structure of the Seq2Seq model is different when it is TRAINED and when it is TESTED.
- When the English sentence goes into the ENCODER, it returns the "hidden state" and the "cell state"
- Send the "\t", which corresponds to 'SOS' token is sent to the DECODER
- The translation is processed until the sentence meets "\n", which corresponds to 'EOS' token

```
encoder_model = Model(inputs=ENCODER_inputs, outputs=ENCODER_states)
encoder_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_1 (InputLayer)         [(None, None, 80)]        0

 lstm (LSTM)                  [(None, 256),             345088
                               (None, 256),
                               (None, 256)]

=================================================================
Total params: 345,088
Trainable params: 345,088
Non-trainable params: 0
_________________________________________________________________
```

ENCODER MODEL

ENCODER_outputs, state_h, state_c = ENCODER_lstm(ENCODER_inputs)

ENCODER_states = [state_h, state_c]

Key difference between TRAINING model & TESTING model

TRAINING:

- Stacked the Encoder & Decoder in ONE model
- DECODER_LSTM uses the *initial state* as ENCODER states

TESTING:

- Uses 2 different models: "*encoder_model*", "*DECODER_model*"
- encoder_model returns the "ENCODER_states"
  - ENCODER_state = [state_h, state_c]

Seq2Seq LSTM TESTING Model (2)

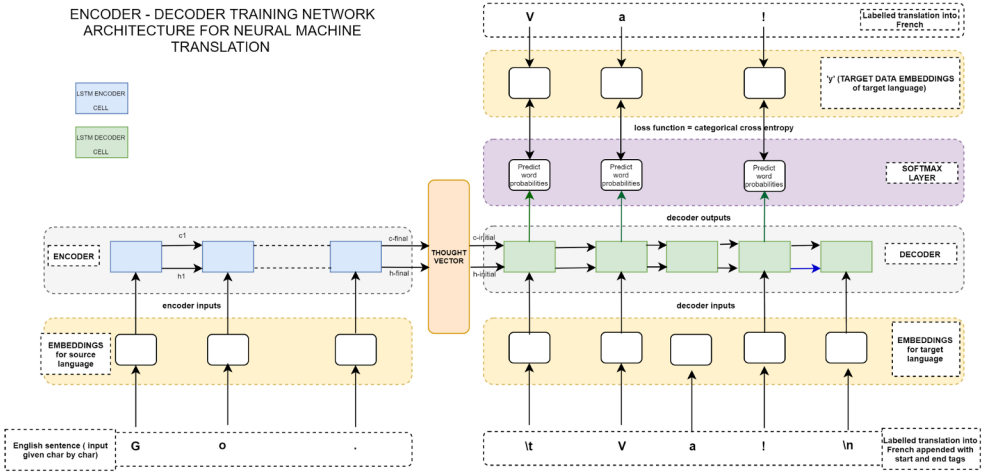
```
# Tensor that will store the previous states
DECODER_state_input_h = Input(shape=(256,))
DECODER_state_input_c = Input(shape=(256,))
DECODER_states_inputs = [DECODER_state_input_h, DECODER_state_input_c]

DECODER_outputs, state_h, state_c = DECODER_lstm(DECODER_inputs,
                                                 initial_state=DECODER_states_inputs)

# Store the hidden state and the cell state
# which we did not use in the training process
DECODER_states_outputs = [state_h, state_c]
DECODER_outputs = DECODER_softmax_layer(DECODER_outputs)

# Input: We use both the Decoder inputs list & Decoder state at INPUT
# Output: Decoder output list & Decoder state at OUTPUT
INPUTS = [DECODER_inputs] + DECODER_states_inputs
OUTPUTS = [DECODER_outputs] + DECODER_states_outputs

DECODER_model = Model(inputs=INPUTS,
                      outputs=OUTPUTS)

DECODER_model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, None, 105)]  0           []

 input_3 (InputLayer)           [(None, 256)]        0           []

 input_4 (InputLayer)           [(None, 256)]        0           []

 lstm_1 (LSTM)                  [(None, None, 256),  370688      ['input_2[0][0]',
                                 (None, 256),                     'input_3[0][0]',
                                 (None, 256)]                     'input_4[0][0]']

 dense (Dense)                  (None, None, 105)    26985       ['lstm_1[1][0]']

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

DECODER_MODEL

TESTING:

- DECODER_model input: [DECODER_inputs] + DECODER_states_input
- DECODER_model returns:
  - [DECODER_outputs] + DECODER_states_ouput



Seq2Seq LSTM TESTING Model (2)

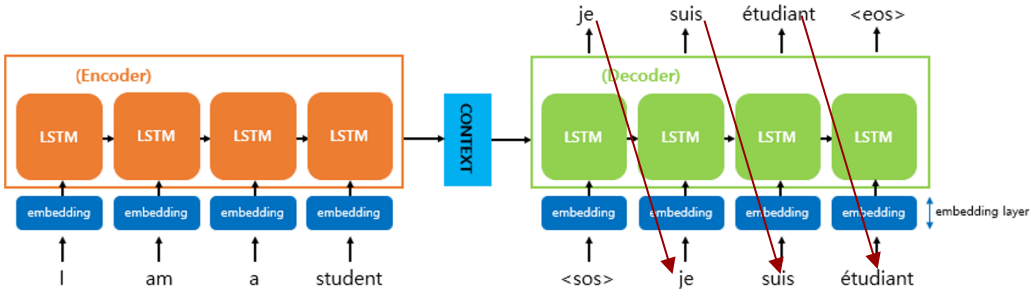
```python
def decoding_func(INPUT_seq):
  # Receive state from the input of the ENCODER
  states_value = encoder_model.predict(INPUT_seq)

  # Create One-Hot Vector for <SOS> ('\t')
  target_sequence = np.zeros((1, 1, FRA_vocab_size))
  SOS_idx = FRA_idx_dict['\t']
  target_sequence[0, 0, SOS_idx] = 1

  stop_condition = False
  decoded_sentence = ""

  # Repeat the loop until it meets the stop_condition as True
  while not stop_condition:
    # Use the state value of the previous timestep
    # as the initial state of the current timestep
    output_tokens, h, c = DECODER_model.predict([target_sequence] + states_value)

    # Convert predicted sequence as CHARACTERS
    sampled_token_index = np.argmax(output_tokens[0, -1, :])
    sampled_char = idx_FRA_dict[sampled_token_index]

    # Add the predicted character to the decoded sentence
    decoded_sentence += sampled_char

    # Stop the loop if
    #   1. Meets <EOS> ("\n")
    #   2. Went over the length of current sentence
    if (sampled_char == '\n' or len(decoded_sentence) > max_FRA_len):
      stop_condition = True

    # Use current predicted character as INPUT of the next timestep
    target_sequence = np.zeros((1, 1, FRA_vocab_size))
    target_sequence[0, 0, sampled_token_index] = 1

    # Store the current state_value to use for the next timestep
    states_value = [h, c]

  return decoded_sentence
```

```python
# Previous dictionary: {token : idx}
# Modified dictionary: {idx : token}

idx_ENG_dict = dict((y, x) for x,y in ENG_idx_dict.items())
idx_FRA_dict = dict((y, x) for x,y in FRA_idx_dict.items())

print(idx_ENG_dict)
print(idx_FRA_dict)
```

```
{1: ' ', 2: '!', 3: '"', 4: '$', 5: '%', 6: '&', 7: "'", 8: ',
{1: '\t', 2: '\n', 3: ' ', 4: '!', 5: '"', 6: '$', 7: '%', 8: '
```

Key feature of the decoding_func function:

"pass both target_sequence AND states_value to the DECODER model"



Result & Evaluation

```
for index in [3, 50, 100, 300, 1001]:
  input_sequence = ENCODER_input[index : index + 1]
  decoded_sent = decoding_func(input_sequence)

  print(20*"-")

  input_sentence = df.English[index]
  answer = df.French[index][2:len(df.French[index]) - 1]
  predicted_sentence = decoded_sent[1:len(decoded_sent) - 1]

  print("Input sentence: ", input_sentence)
  print("Answer sentence: ", answer)
  print("Predicted sentence: ", predicted_sentence)
```

```
--------------------
Input sentence:  Hi.
Answer sentence:  Salut !
Predicted sentence:  Apporte-le !
--------------------
Input sentence:  I see.
Answer sentence:  Aha.
Predicted sentence:  Je me suis sinte.
--------------------
Input sentence:  Hug me.
Answer sentence:  Serrez-moi dans vos bras !
Predicted sentence:  La chambre est en train de mentir.
--------------------
Input sentence:  Help me.
Answer sentence:  Aidez-moi.
Predicted sentence:  Aidez-vous !
--------------------
Input sentence:  I am sure.
Answer sentence:  Je suis sûr.
Predicted sentence:  Je suis désolé.
```

First obtain the encoded input of a specific sentence

Testing on different sentences:

index [3, 50, 100, 300, 1001]

Model tend to show even different length of sentences as output

Model may improve if we *increase the number of epochs*

Reference

- “1) 시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq).” 위키독스, wikidocs.net/24996. Accessed 12 Jan. 2022.
- Chollet, By Francois. “A Ten-Minute Introduction to Sequence-to-Sequence Learning in Keras.” *Keras*, blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html. Accessed 12 Jan. 2022.
- Alammar, Jay. “Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention).” *Jalammar*, jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention. Accessed 12 Jan. 2022.