https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

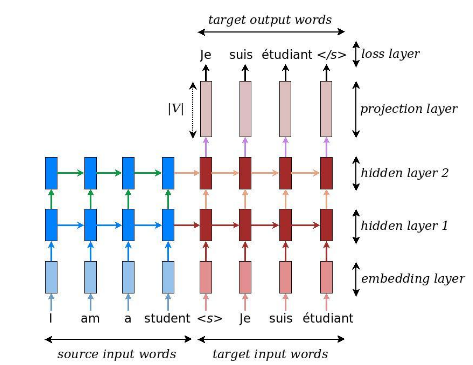
https://github.com/fchollet/keras/blob/master/examples/lstm\_seq2seq.py

首先本代码都是基于函数式模型而不是基于序贯模型的。函数式模型称作Functional，但它的类名是Model，因此我们有时候也用Model来代表函数式模型。一句话，只要你的模型不是类似VGG一样一条路走到黑的模型，或者你的模型需要多于一个的输出，那么你总应该选择函数式模型。函数式模型是最广泛的一类模型，序贯模型（Sequential）只是它的一种特殊情况。

所以开始看代码前前把下面这段看懂了

http://keras-cn.readthedocs.io/en/latest/getting\_started/functional\_API/

版本Keras 2.1.1以上



「编码器」：它处理输入序列并反馈其内部状态。注意我们抛弃了编码器 RNN 的输出，只恢复其状态。该状态在下一步中充当解码器的「语境」。

「解码器」：在给定目标序列先前字母的情况下，它被训练以预测目标序列的下一个字符。

decoder\_input\_data[i, t, target\_token\_index[char]] = 1.

decoder\_target\_data[i, t - 1, target\_token\_index[char]] = 1.

思路：

 在给定 encoder\_input\_data 和 decoder\_input\_data 的情况下，训练一个基本的基于 LSTM 的 Seq2Seq 模型以预测 decoder\_target\_data。我们的模型使用 teacher forcing。

*Teacher Forcing 算法通过将被观察到的序列值作为训练过程中的输入和使用该网络自己的提前一步的预测（one-step-ahead predictions）来进行多步采样（multi-step sampling）。*

训练阶段

# `encoder\_input\_data` & `decoder\_input\_data` into `decoder\_target\_data`

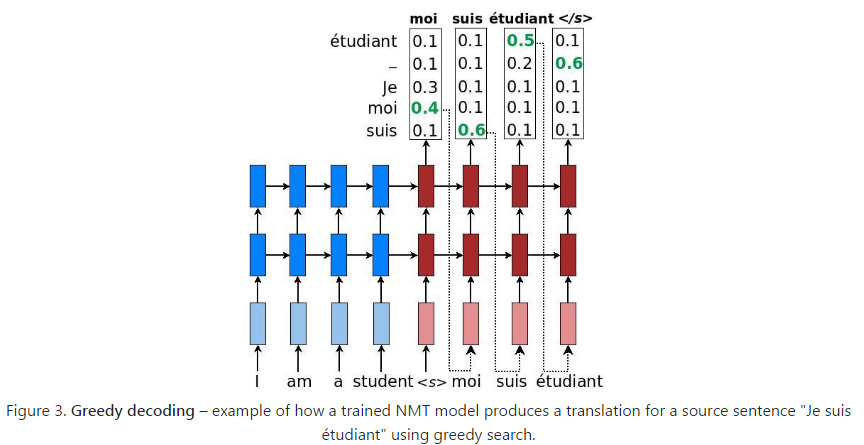
model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

encoder\_inputs:

decoder\_inputs: 

decoder\_outputs: 

In inference mode：



* 1) Encode the input sequence into state vectors.
* 2) Start with a target sequence of size 1 (just the start-of-sequence character).
* 3) Feed the state vectors and 1-char target sequence to the decoder to produce predictions for the next character.
* 4) Sample the next character using these predictions (we simply use argmax).
* 5) Append the sampled character to the target sequence
* 6) Repeat until we generate the end-of-sequence character or we hit the character limit.

输入形状分析：

* encoder\_input\_data is a 3D (num\_pairs, max\_english\_sentence\_length, num\_english\_characters) containing a one-hot vectorization of the English sentences.
* decoder\_input\_data is a 3D (num\_pairs, max\_french\_sentence\_length, num\_french\_characters) containg a one-hot vectorization of the French sentences.
* decoder\_target\_data is the same as decoder\_input\_data but *offset by one timestep*.

代码中：

encoder\_input\_data = np.zeros((len(input\_texts), max\_encoder\_seq\_length, num\_encoder\_tokens),dtype='float32')

decoder\_input\_data = np.zeros( (len(input\_texts), max\_decoder\_seq\_length, num\_decoder\_tokens), dtype='float32')

decoder\_target\_data = np.zeros( (len(input\_texts), max\_decoder\_seq\_length, num\_decoder\_tokens),dtype='float32')

如果是单词级别（进行embedding之后）：

len(input\_texts) 可以理解成有多少句(行)用于训练的话

max\_english\_sentence\_length 每句话最多有多少单词，统一长度

num\_english\_characters 其实是说所有训练语料中一共出现过多少不同的单词，这里搞了个独热码，所以长度设为num\_english\_characters，每一个encoder\_input\_data单词对应其中的一位为1，其余为0

不过这里的例子是字符级别：

max\_english\_sentence\_length 每句话最多有多少字符，统一长度

num\_english\_characters 其实是说所有英文一共多少字母（26个），这里搞了个独热码，所以长度设为num\_english\_characters，

return\_state 构造函数参数配置一个 RNN 层以反馈列表，其中第一个是其输出，下一个是内部的 RNN 状态。这被用于恢复编码器的状态。

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

decoder\_outputs, state\_h, state\_c = decoder\_lstm(…)

return\_sequences 构造函数参数配置一个 RNN 反馈输出的全部序列。这被用在解码器中。

推理：（训练完之后）

* 编码输入语句，检索初始解码器状态。
* 用初始状态运行一步解码器，以「序列开始」为目标。输出即是下一个目标字符。
* 附加预测到的目标字符并重复。

decoder\_model = Model( [decoder\_inputs] + decoder\_states\_inputs,

[decoder\_outputs] + decoder\_states)

更好的写法如下：

decoder\_model = Model(inputs=[decoder\_inputs, decoder\_states\_inputs], outputs=[decoder\_outputs, decoder\_states])

如果只有一个输入或者输出，就不用列表了

model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)可表示为：

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

结果：

Input sentence: Go.

Decoded sentence: Va !

Input sentence: I try.

Decoded sentence: J'essaye.

Input sentence: Oh no!

Decoded sentence: Oh non !

Input sentence: Attack!

Decoded sentence: Attaquez !

…..

好像还不错

不过以上代码是单词级别的

This script demonstrates how to implement a basic character-level sequence-to-sequence model. We apply it to translating short English sentences into short French sentences, character-by-character. Note that it is fairly unusual to do character-level machine translation, as word-level models are more common in this domain.

