要了解深度学习中的注意力模型，就不得不先谈Encoder-Decoder框架，因为目前大多数注意力模型附着在Encoder-Decoder框架下，当然，其实注意力模型可以看作一种通用的思想，本身并不依赖于特定框架，这点需要注意。

Encoder-Decoder框架可以看作是一种深度学习领域的研究模式，应用场景异常广泛。图2是文本处理领域里常用的Encoder-Decoder框架最抽象的一种表示。

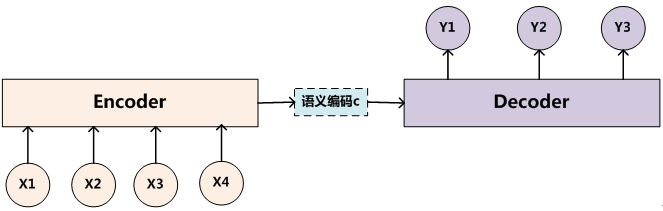


图2 抽象的文本处理领域的Encoder-Decoder框架  
0?wx_fmt=png

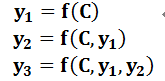
Encoder顾名思义就是对输入句子Source进行编码，将输入句子通过非线性变换转化为中间语义表示C：

0?wx_fmt=png

对于解码器Decoder来说，其任务是根据句子Source的中间语义表示C和之前已经生成的历史信息0?wx_fmt=png来生成i时刻要生成的单词0?wx_fmt=png：

0?wx_fmt=png

图2中展示的Encoder-Decoder框架是没有体现出“注意力模型”的，所以可以把它看作是注意力不集中的分心模型。为什么说它注意力不集中呢？请观察下目标句子Target中每个单词的生成过程如下：



其中f是Decoder的非线性变换函数。从这里可以看出，在生成目标句子的单词时，不论生成哪个单词，它们使用的输入句子Source的语义编码C都是一样的，没有任何区别。

没有引入注意力的模型在输入句子比较短的时候问题不大，但是如果输入句子比较长，此时所有语义完全通过一个中间语义向量来表示，单词自身的信息已经消失，可想而知会丢失很多细节信息，这也是为何要引入注意力模型的重要原因。

理解Attention模型的关键就是这里，即由固定的中间语义表示C换成了根据当前输出单词来调整成加入注意力模型的变化的0?wx_fmt=png。增加了注意力模型的Encoder-Decoder框架理解起来如图3所示。

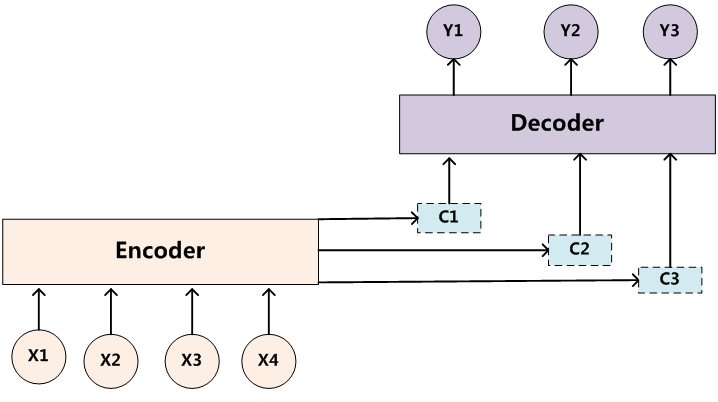
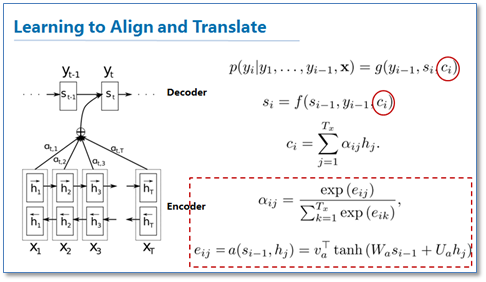


图3 引入注意力模型的Encoder-Decoder框架

**Soft Attention Model：**

**Soft**即上图中的ci是对Encoder中每一个单词都要计算一个注意力概率分布，然后加权得到的。如下图所示：



def attention(self, prev\_state, enc\_outputs):

"""

Attention model for Neural Machine Translation

:param prev\_state: the decoder hidden state at time i-1

:param enc\_outputs: the encoder outputs, a length 'T' list.

"""

e\_i = []

c\_i = []

for output in enc\_outputs:

atten\_hidden = tf.tanh(tf.add(tf.matmul(prev\_state, self.attention\_W), tf.matmul(output, self.attention\_U)))

e\_i\_j = tf.matmul(atten\_hidden, self.attention\_V)

e\_i.append(e\_i\_j)

e\_i = tf.concat(e\_i, axis=1)

alpha\_i = tf.nn.softmax(e\_i)

alpha\_i = tf.split(alpha\_i, self.num\_steps, 1)

for alpha\_i\_j, output in zip(alpha\_i, enc\_outputs):

c\_i\_j = tf.multiply(alpha\_i\_j, output)

c\_i.append(c\_i\_j)

c\_i = tf.reshape(tf.concat(c\_i, axis=1), [-1, self.num\_steps, self.hidden\_dim \* 2])

c\_i = tf.reduce\_sum(c\_i, 1)

return c\_i

#对应的decode

def decode(self, cell, init\_state, encoder\_outputs, loop\_function=None):

outputs = []

prev = None

state = init\_state

for i, inp in enumerate(self.decoder\_inputs\_emb):

#decoder\_inputs\_emb是tf.placeholder

#if loop\_function is not None and prev is not None:

# with tf.variable\_scope("loop\_function", reuse=True):

# inp = loop\_function(prev, i)

#if i > 0:

# tf.get\_variable\_scope().reuse\_variables()

c\_i = self.attention(state, encoder\_outputs)

inp = tf.concat([inp, c\_i], axis=1)

output, state = cell(inp, state)#原本没有attention的是decoder\_input和state作为输入

outputs.append(output)

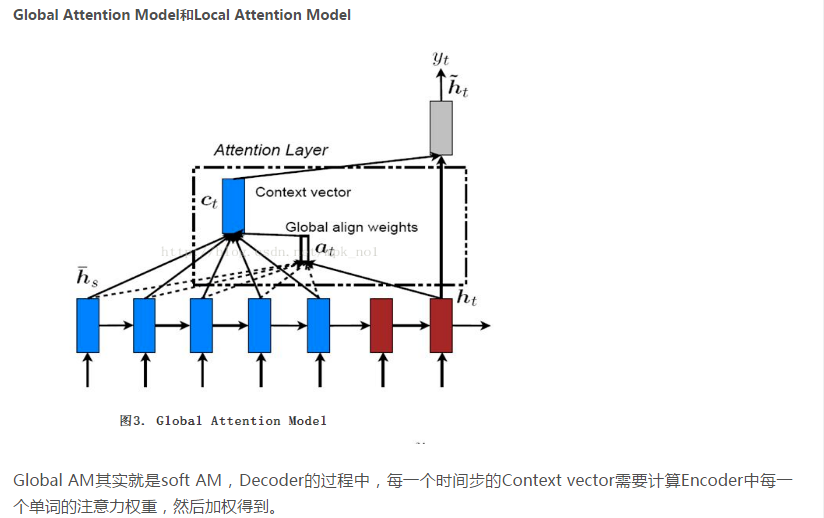
if loop\_function is not None:

prev = output

return outputs

https://www.cnblogs.com/robert-dlut/p/5952032.html

代码摘自 <https://github.com/pemywei/attention-nmt>



其实有Soft AM，对应也有一个Hard AM。Hard AM在图像里证明有用，但是在文本里面用处不大，因为这种单词一一对齐明显要求太高，如果对不齐对后续处理负面影响很大。<https://blog.csdn.net/mpk_no1/article/details/72862348>

https://www.cnblogs.com/robert-dlut/p/5952032.html

<https://segmentfault.com/a/1190000011744246>

https://segmentfault.com/a/1190000011744246