完全解析RNN, Seq2Seq, Attention注意力机制

白裳 AINLP 2020-08-27

作者：白裳
知乎专栏：机器学习随笔
本文为授权转载，原文链接点击“阅读原文”直达：
https://zhuanlan.zhihu.com/p/51383402

循环神经网络RNN结构被广泛应用于自然语言处理、机器翻译、语音识别、文字识别等方向。本文主要介绍经典的RNN结构，以及RNN的变种（包括Seq2Seq结构和Attention机制）。希望这篇文章能够帮助初学者更好地入门。

经典的RNN结构

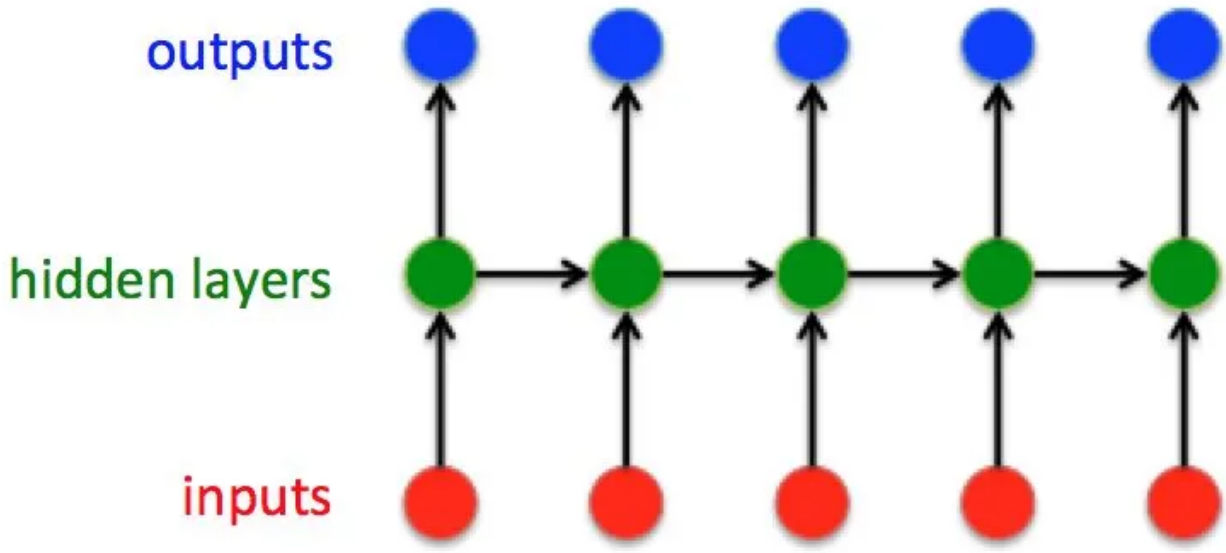


图1

这就是最经典的RNN结构，它的输入是：

$$\text{inputs: } \{x_1, x_2, \ldots, x_t, \ldots, x_T\} \tag{1}$$

输出为：

$$\text{outputs: } \{y_1, y_2, \ldots, y_t, \ldots, y_T\} \tag{2}$$

也就是说，**输入和输出序列必有相同的时间长度！**

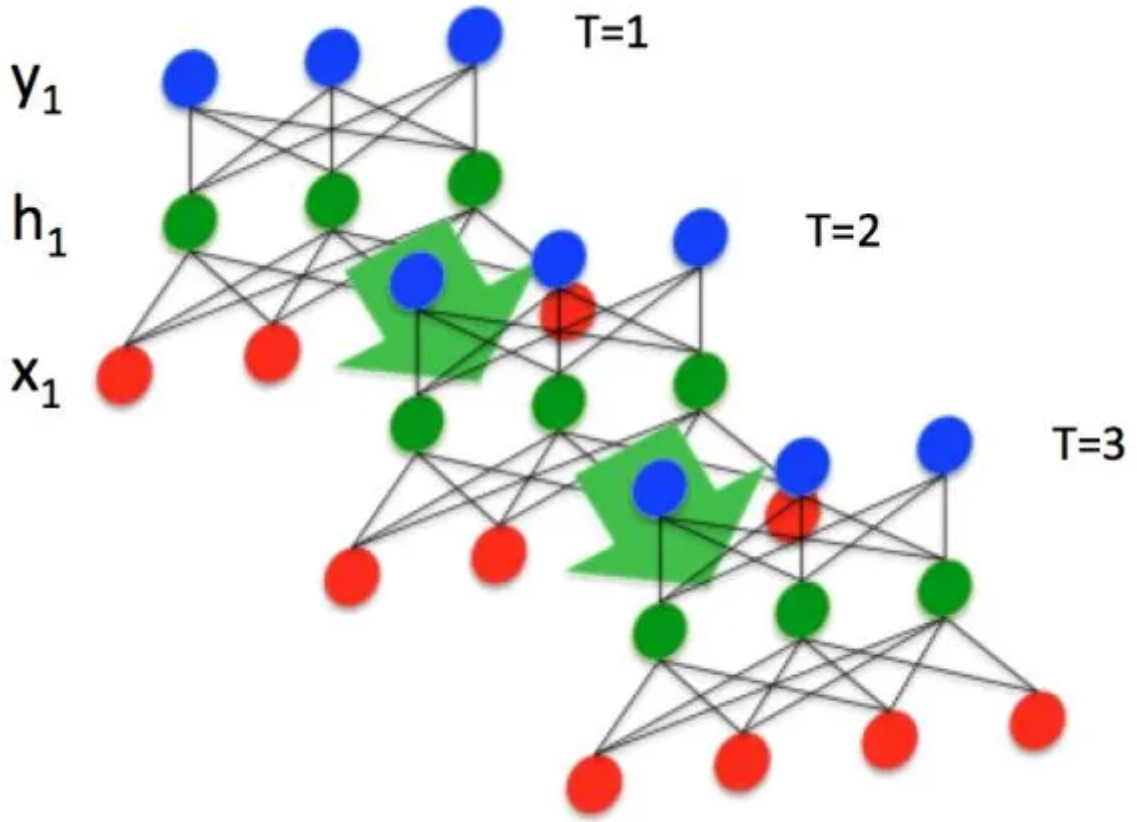


图2

假设输入 x_t（ $t \in \{1, 2, \ldots, T\}$ ）是一个长度为 n_i（ n_{input} ）的列向量：

$$x_t = [x_{t,1}, x_{t,2}, x_{t,3}, \ldots, x_{t,n_i}]^T \tag{3}$$

隐藏层 h_t 是一个长度为 n_h（ n_{hidden} ）的列向量：

$$h_t = [h_{t,1}, h_{t,2}, h_{t,3}, \ldots, h_{t,n_h}]^T \tag{4}$$

输出 y_t 是一个长度为 n_o（ n_{output} ）的列向量：

$$y_t = [y_{t,1}, y_{t,2}, y_{t,3}, \ldots, y_{t,n_o}]^T \tag{5}$$

其中 n_i ， n_h ， n_o 都是由人工设定的。

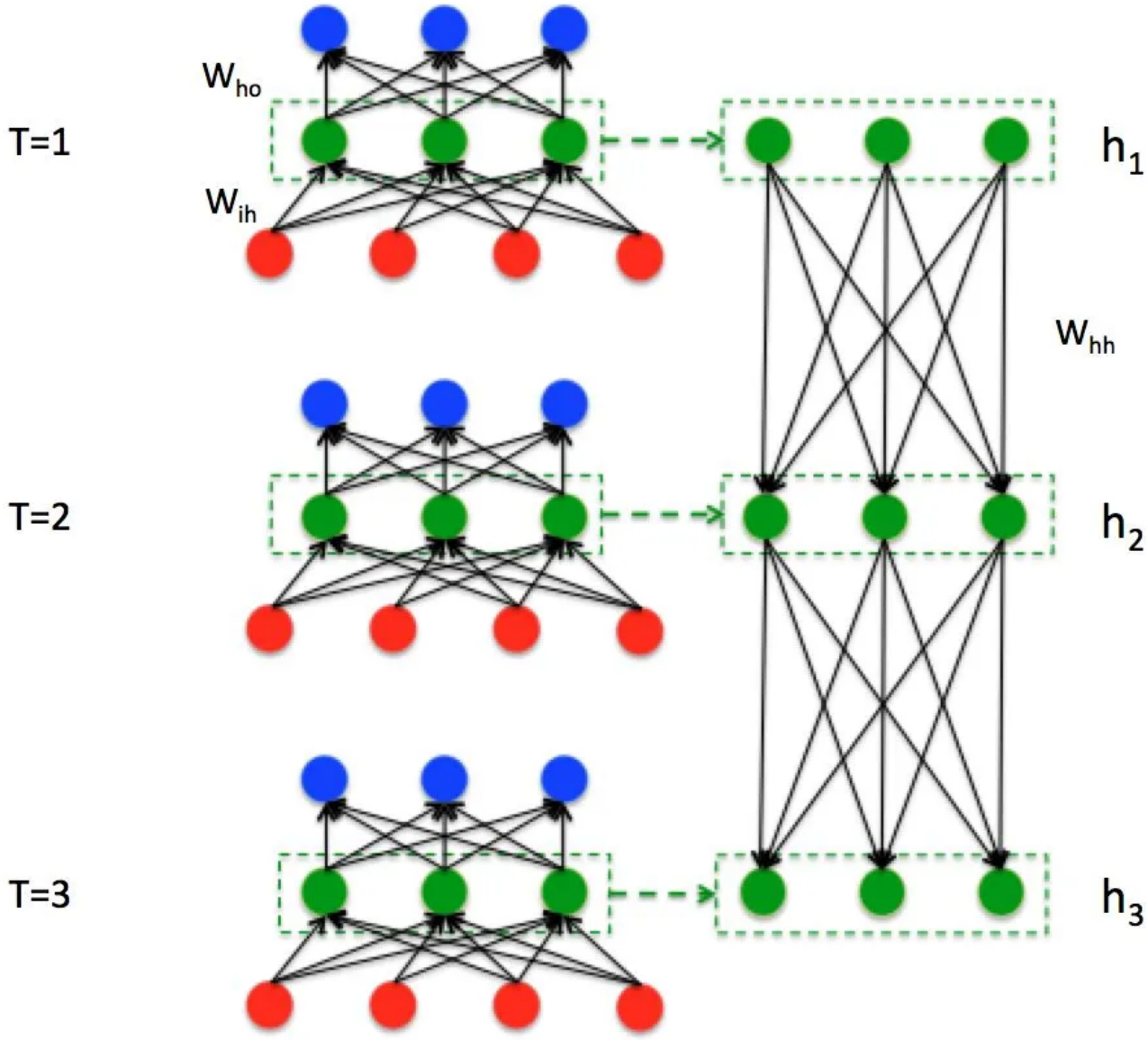


图3

t 时刻输入层--> t 时刻隐藏层：

$$h_t^{ih} = W_{ih} \cdot x_t + b_{ih} \tag{6}$$

$(t-1)$ 时刻隐藏层--> t 时刻隐藏层：

$$h_t^{hh} = W_{hh} \cdot h_{t-1} + b_{hh} \tag{7}$$

t 时刻输入层 **and** $(t-1)$ 时刻隐藏层--> t 时刻隐藏层：

$$\begin{aligned} h_t &= \tanh(h_t^{ih} + h_t^{hh}) \\ &= \tanh((W_{ih} \cdot x_t + b_{ih}) + (W_{hh} \cdot h_{t-1} + b_{hh})) \end{aligned} \tag{8}$$

t 时刻隐藏层--> t 时刻输出层：

$$y_t = W_{ho} \cdot h_t + b_{ho} \tag{9}$$

需要注意的是，**对于任意时刻 $t \in \{1, 2, \ldots, T\}$ ，所有的权值（包括 W_{ih} ， b_{ih} ， W_{hh} ， b_{hh} ， W_{ho} ， b_{ho} ）都相等**，这也就是RNN中的“权值共享”，极大的减少参数量。

其实RNN可以简单的表示为：

$$y_t = \mathtt{RNN}(x_t, h_{t-1}) = \mathtt{RNN}(x_t, x_{t-1}, \ldots, x_2, x_1) \tag{10}$$

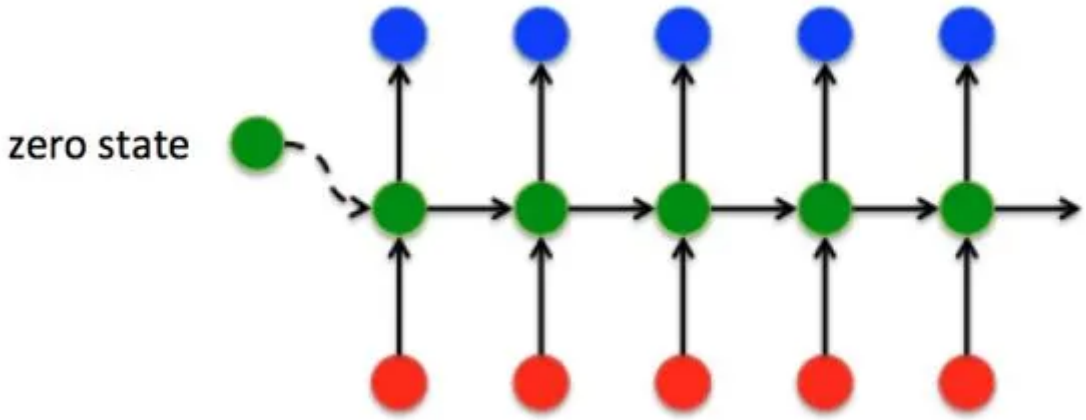


图4

还有一个小细节：在 $t = 1$ 时刻，如果没有特别指定初始状态，一般都会使用全0的 h_0 作为初始状态输入到 h_1 中

$$\text{zero state: } h_0 = [0, 0, \ldots, 0] \tag{11}$$

Sequence to Sequence模型

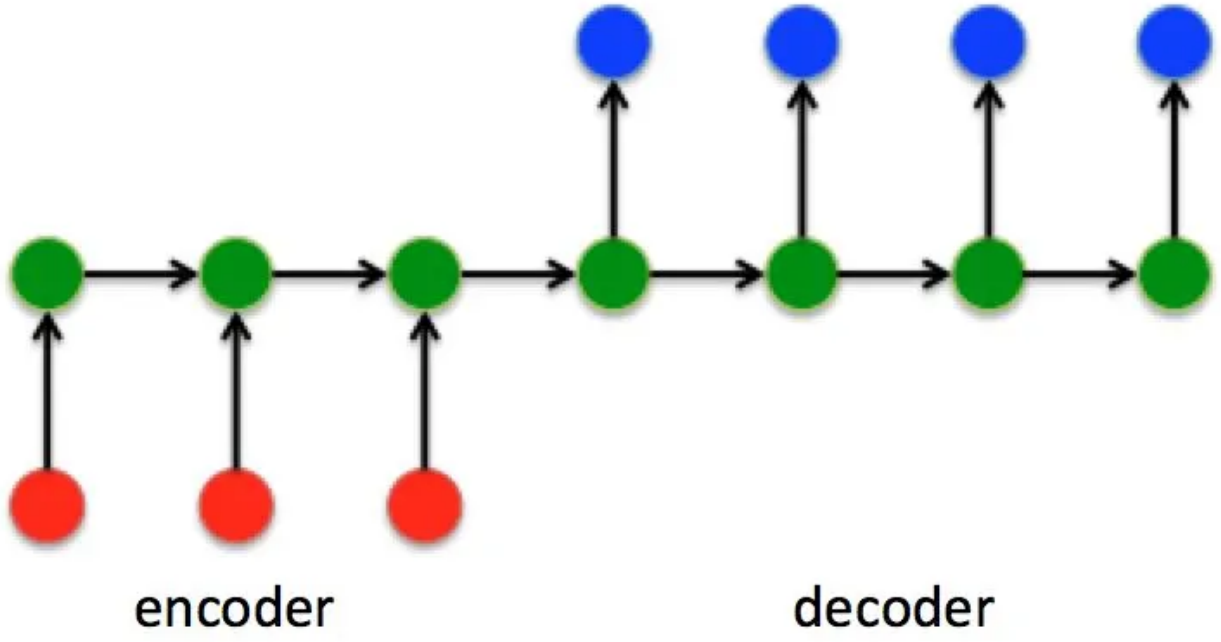


图5

在Seq2Seq结构中，编码器Encoder把所有的输入序列都编码成一个统一的语义向量Context，然后再由解码器Decoder解码。在解码器Decoder解码的过程中，不断地将前一个时刻 $t-1$ 的输出作为后一个时刻 t 的输入，循环解码，直到输出停止符为止。

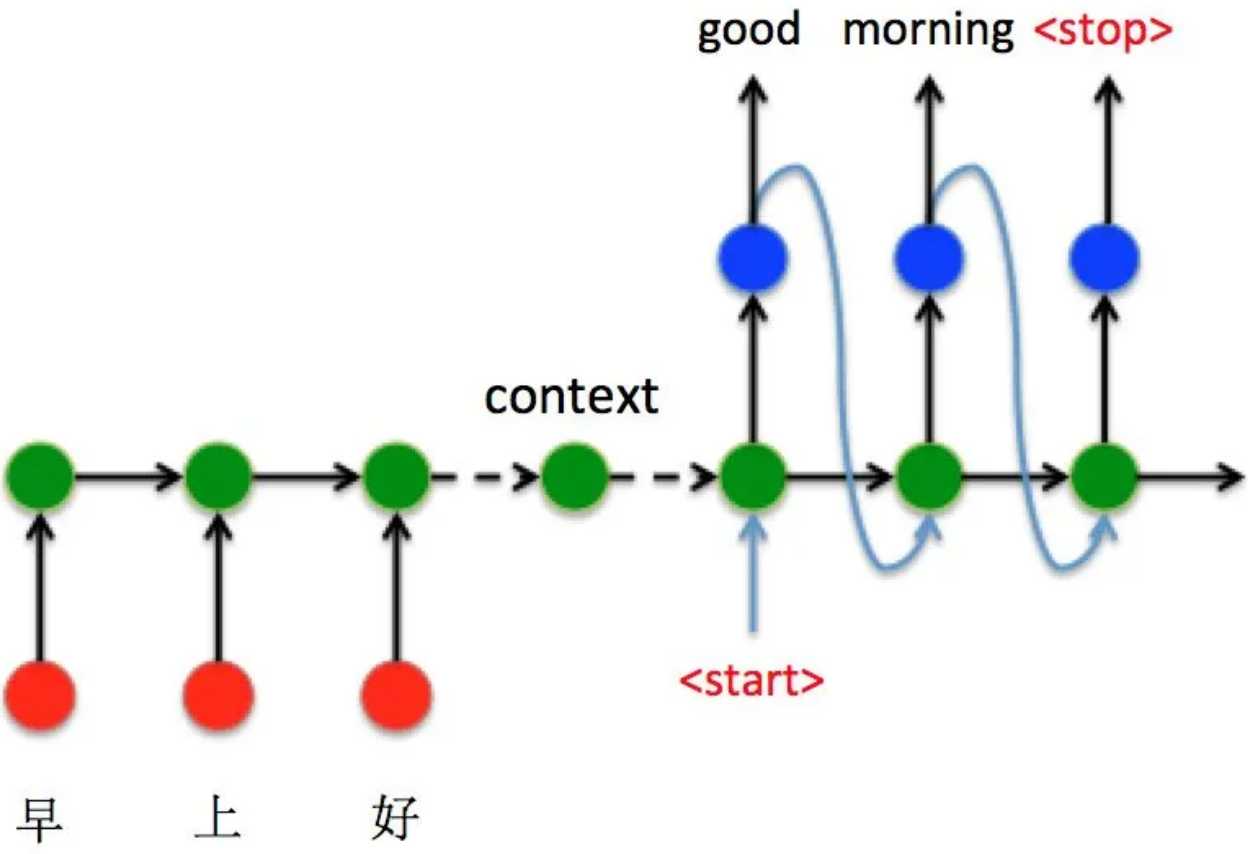


图6

接下来以机器翻译为例，看看如何通过Seq2Seq结构把中文“早上好”翻译成英文“Good morning”：

> 将“早上好”通过Encoder编码，并将最后 $t=3$ 时刻的隐藏层状态 h_3 作为语义向量。以语义向量为Decoder的 h_0 状态，同时在 $t=1$ 时刻输入<start>特殊标识符，开始解码。之后不断的将前一时刻输出作为下一时刻输入进行解码，直接输出<stop>特殊标识符结束。

当然，上述过程只是Seq2Seq结构的一种经典实现方式。**与经典RNN结构不同的是，Seq2Seq结构不再要求输入和输出序列有相同的时间长度！**

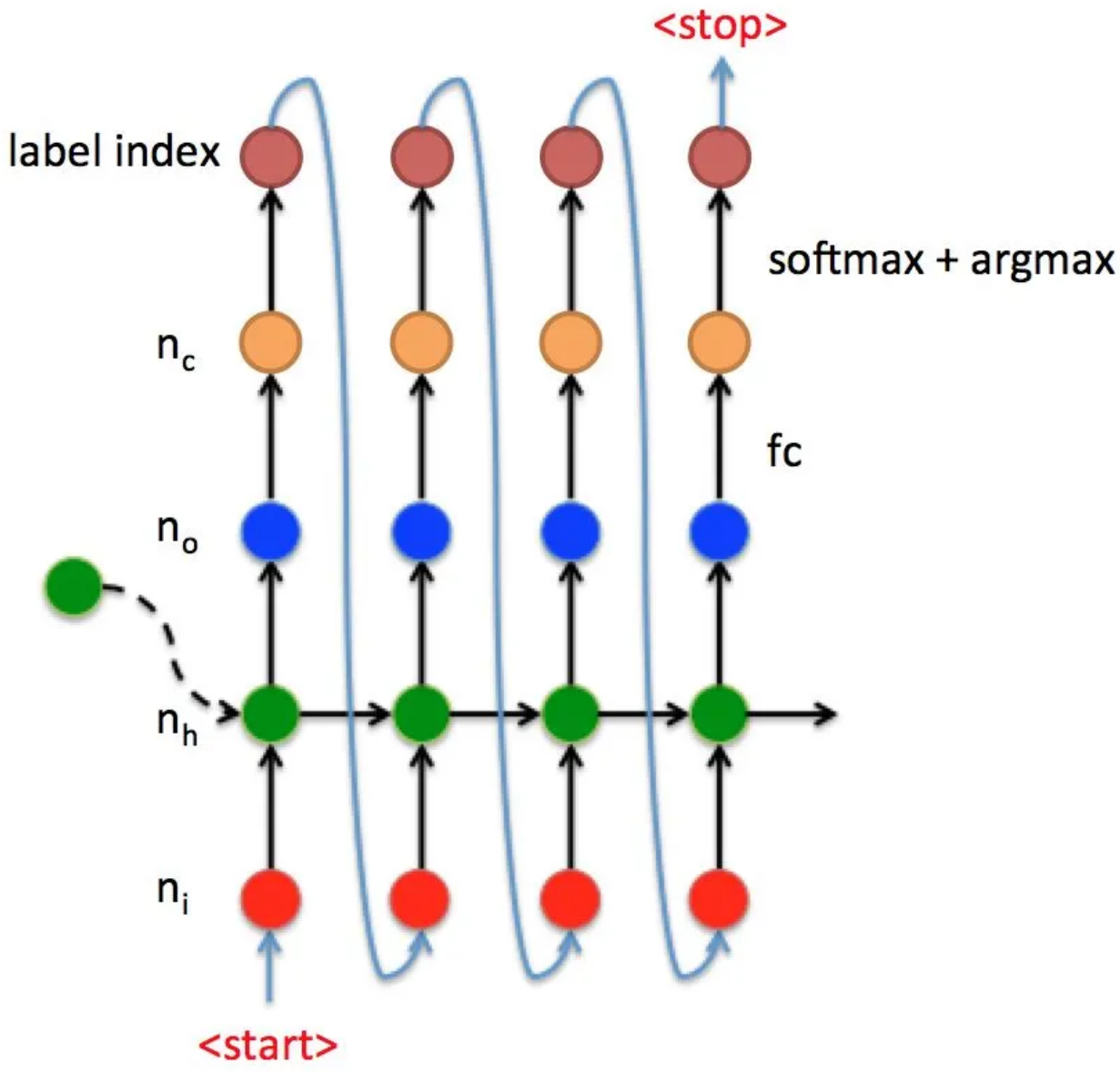


图7

进一步来看上面机器翻译例子Decoder端的 t 时刻数据流，如图7：

> 首先对RNN输入大小为 $[n_i, 1]$ 的向量 x_t （红点）；
> 然后经过RNN输出大小为 $[n_o, 1]$ 的向量 y_t （蓝点）；

接着使用全连接fc将 y_t 变为大小为 $[n_c, 1]$ 的向量 y'_t ，其中 n_c 代表类别数量；
再 y'_t 经过softmax和argmax获取类别index，再经过int2str获取输出字符；
最后将类别index输入到下一状态，直到接收到<stop>标志符停止。

Embedding

还有一点细节，就是如何将前一时刻输出类别index（数值）送入下一时刻输入（向量）进行解码。假设每个标签对应的类别index如下：

```
'<start>' : 0,
'<stop>' : 1,
'good' : 2,
'morning' : 3,
...
```

已知<start>标志符index为0，如果需要将<start>标志符输入到input层，就需要把类别index=0转变为一个 $[n_i, 1]$ 长度的特定对应向量。这时就需要应用**嵌入 (embedding)** 方法。

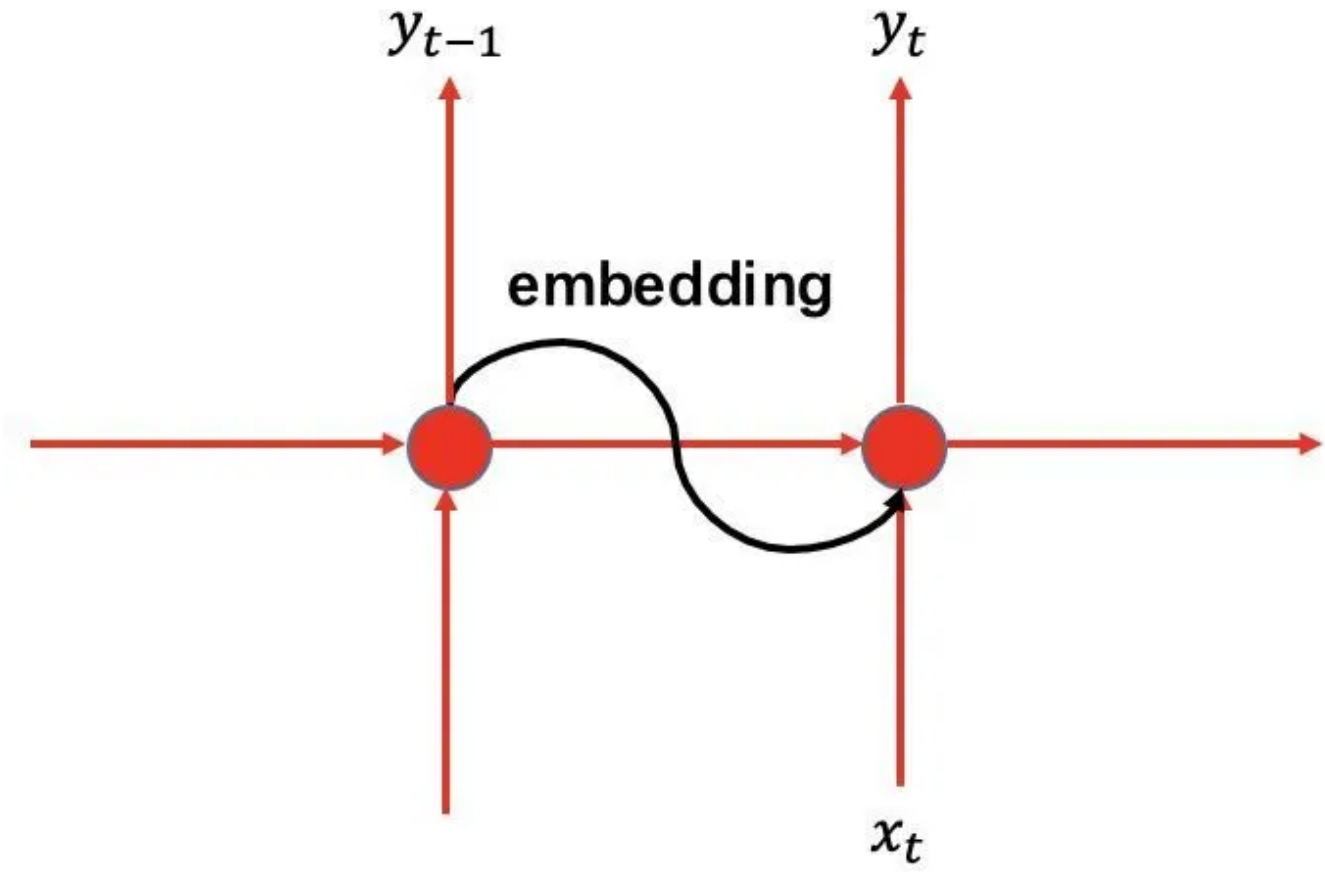


图8 嵌入 (embedding)

假设有 n_c 个词，最简单的方法就是使用 n_c 长度的one-hot编码，词表alphabet如下：

```
'<start>' : 0  <-----> label('<start>')=[1, 0, 0, 0, 0,..., 0]
'<stop>' :  1  <-----> label('<stop>') =[0, 1, 0, 0, 0,..., 0]
'hello':    2  <-----> label('hello')  =[0, 0, 1, 0, 0,..., 0]
'good' :    3  <-----> label('good')   =[0, 0, 0, 1, 0,..., 0]
'morning' : 4  <-----> label('morning')=[0, 0, 0, 0, 1,..., 0]
.......
```

但是使用one-hot编码进行嵌入过于稀疏，所以我们使用一种更加优雅的办法：

首先随机生成一个大小为 $[n_c, n_i]$ embedding随机矩阵：

$$\mathbf{embedding}_{[n_c,n_i]} = \mathrm{random}(n_c, n_i) \tag{12}$$

然后通过start标志的one-hot编码乘以embedding矩阵（即获取embedding矩阵的第 $i_{<\mathrm{start}>} = 0$ 行），作为start标志对应的输入向量送入网络：

$$x_1 = \mathrm{onehot}_{<\mathrm{start}>} * \mathbf{embedding} = \mathbf{embedding}[i_{<\mathrm{start}>}, :] \tag{13}$$

在 $t = 1$ 时刻网络输入 x_1 后输出了good字符，那么要在 $t = 2$ 时刻再把good字符的one-hot编码乘以embedding矩阵获取 x_2 ：

$$x_2 = \mathrm{onehot}_{\mathrm{good}} * \mathbf{embedding} \tag{14}$$

同理 $t = 3$ 再把上一时刻输出的morning字符的one-hot编码乘以embedding获取新的 x_3 ：

$$x_3 = \mathrm{onehot}_{\mathrm{morning}} * \mathbf{embedding} = \mathbf{embedding}[i_{\mathrm{morning}}, :] \tag{15}$$

如此不停循环解码。

可以看到，其实Seq2Seq引入嵌入机制解决从label index数值到输入向量的维度恢复问题。在Tensorflow中上述过程通过以下函数实现：

```
tf.nn.embedding_lookup
```

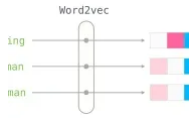
而在pytorch中通过以下接口实现：

```
torch.nn.Embedding
```

需要注意的是：train和test阶段必须使用一样的embedding矩阵！否则输出肯定是乱码。

当然，还可以使用word2vec/glove/elmo/bert等更加“精致”的嵌入方法，也可以在训练过程中迭代更新embedding。这些内容超出本文范围，不再详述。embedding入门请参考：

快速入门词嵌入之word2vec
zhuanlan.zhihu.com



Seq2Seq训练问题

值得一提的是，在seq2seq结构中将 y_t 作为下一时刻输入 $x_{t+1} \Leftarrow y_t$ 进网络，那么某一时刻输出 y_t 错误就会导致后面全错。在训练时由于网络尚未收敛，这种蝴蝶效应格外明显。

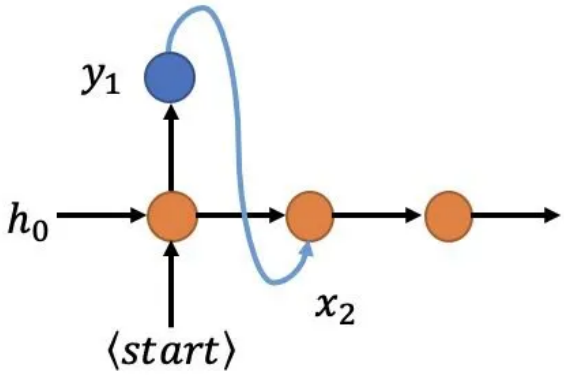


图9

为了解决这个问题，Google提出了大名鼎鼎的Scheduled Sampling（即在训练中 x_t 按照一定概率选择输入 y_{t-1} 或 $t-1$ 时刻对应的真实值，即标签，如图10），既能加快训练速度，也能提高训练精度。

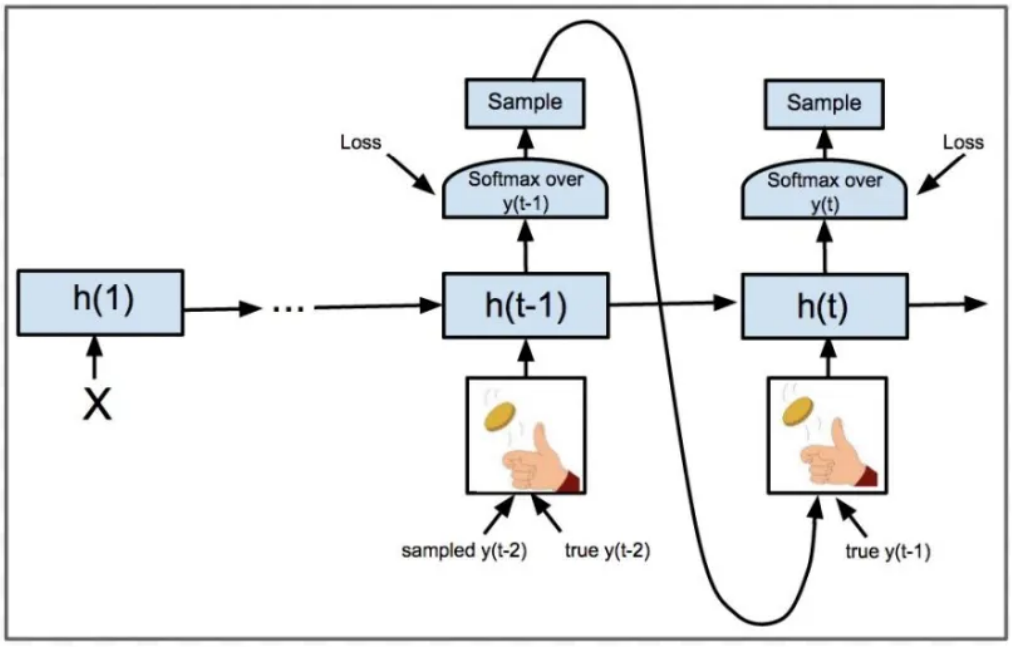


图10

Scheduled Sampling对应文章如下：

Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks
arxiv.org

Attention注意力机制

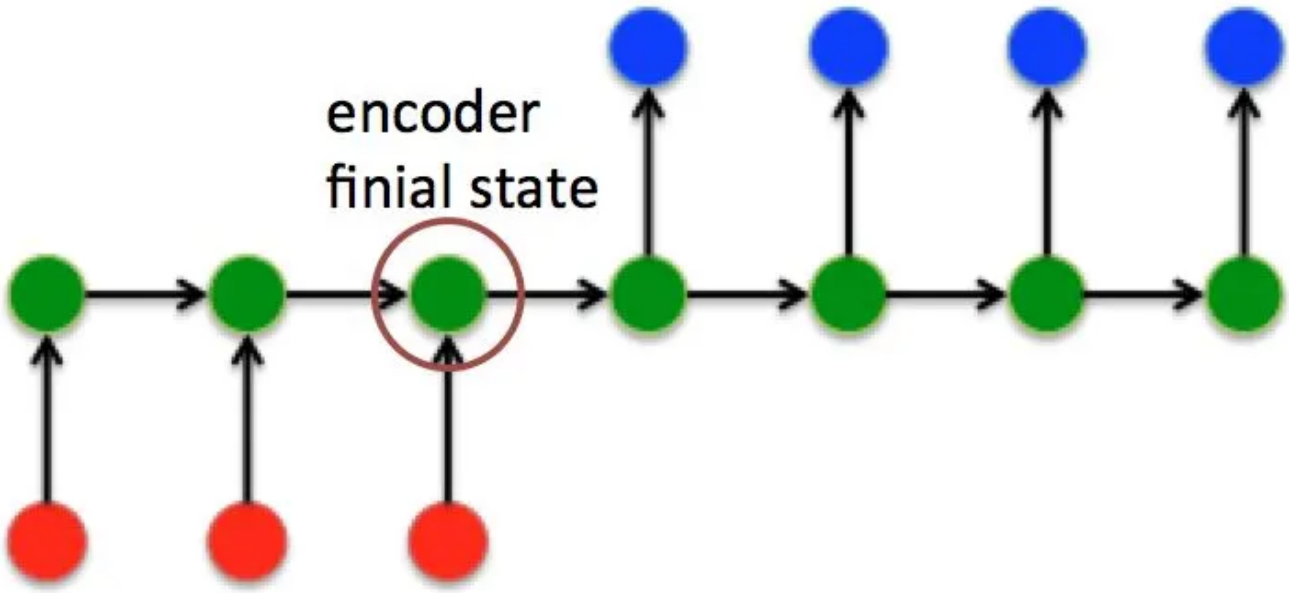
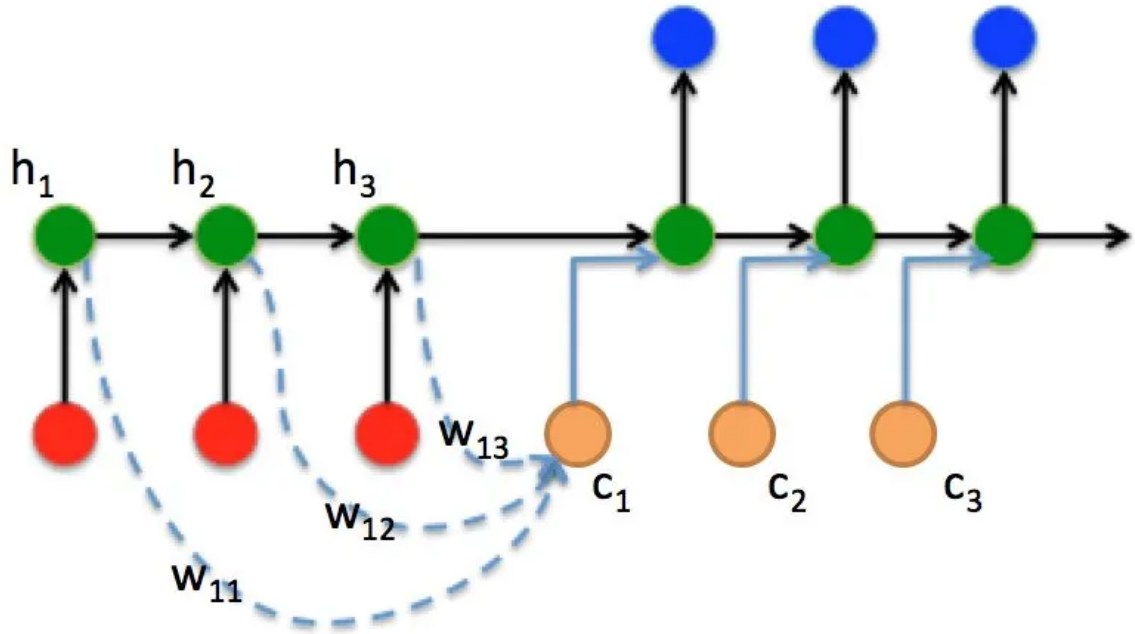


图11

在Seq2Seq结构中，encoder把所有的输入序列都编码成一个统一的语义向量Context，然后再由Decoder解码。由于context包含原始序列中的所有信息，它的长度就成了限制模型性能的瓶颈。如机器翻译问题，当要翻译的句子较长时，一个Context可能存不下那么多信息，就会造成精度的下降。除此之外，如果按照上述方式实现，只用到了编码器的最后一个隐藏层状态，信息利用率低下。

所以如果要改进Seq2Seq结构，最好的切入角度就是：**利用Encoder所有隐藏层状态 h_t 解决Context长度限制问题。**

接下来了解一下attention注意力机制基本思路(Luong Attention)



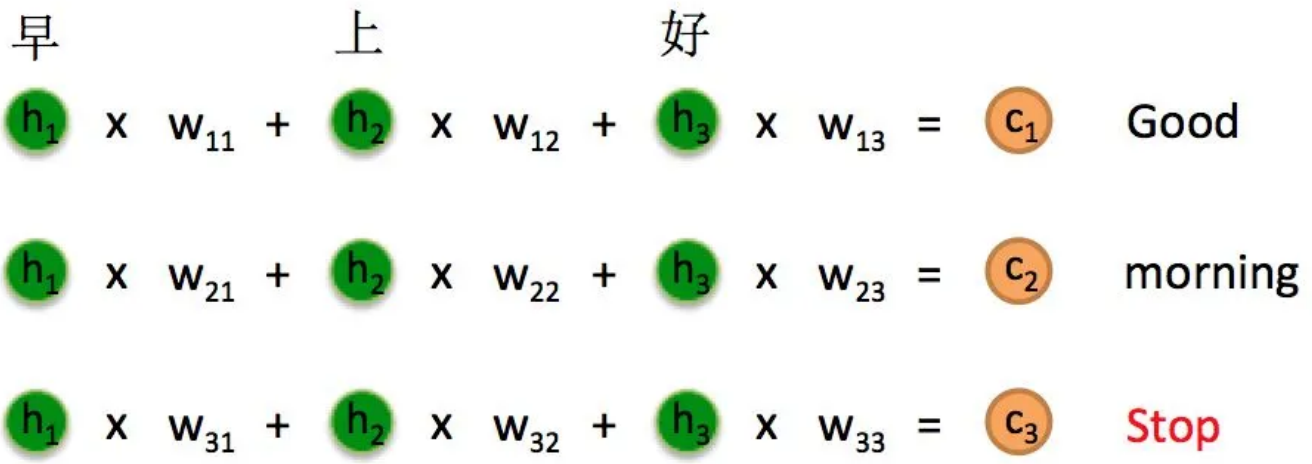


图12

考虑这样一个问题：由于Encoder的隐藏层状态 h_t 代表对不同时刻输入 x_t 的编码结果：

$$\texttt{RNN:}\ x_t \Rightarrow h_t\ \ (t \in \{1, 2, \ldots, T\}) \tag{15}$$

即Encoder状态 h_1 ， h_2 ， h_3 对应编码器对“早”，“上”，“好”三个中文字符的编码结果。那么在Decoder时刻 $t=1$ 通过3个权重 w_{11} ， w_{12} ， w_{13} 计算出一个向量 c_1 ：

$$c_1 = h_1 \cdot w_{11} + h_2 \cdot w_{12} + h_3 \cdot w_{13} \tag{16}$$

然后将这个向量与前一个状态拼接在一起形成一个新的向量输入到隐藏层计算结果：

$$\bar{h}_0 \leftarrow \text{concat}\ (\bar{h}_0, c_1) = \text{concat}(h_3, c_1) \tag{17}$$

Decoder时刻 $t=1$ ：

$$c_2 = h_1 \cdot w_{21} + h_2 \cdot w_{22} + h_3 \cdot w_{23} \tag{18}$$

$$\bar{h}_1 \leftarrow \text{concat}(\bar{h}_1, c_2) \tag{19}$$

Decoder时刻 $t=2$ 和 $t=3$ 同理，就可以解决Context长度限制问题。由于 w_{11} ， w_{12} ， w_{13} 不同，就形成了一种对编码器不同输入 x_t 对应 h_t 的“注意力”机制（权重越大注意力越强）。

那么到底什么是LuongAttention注意力机制?

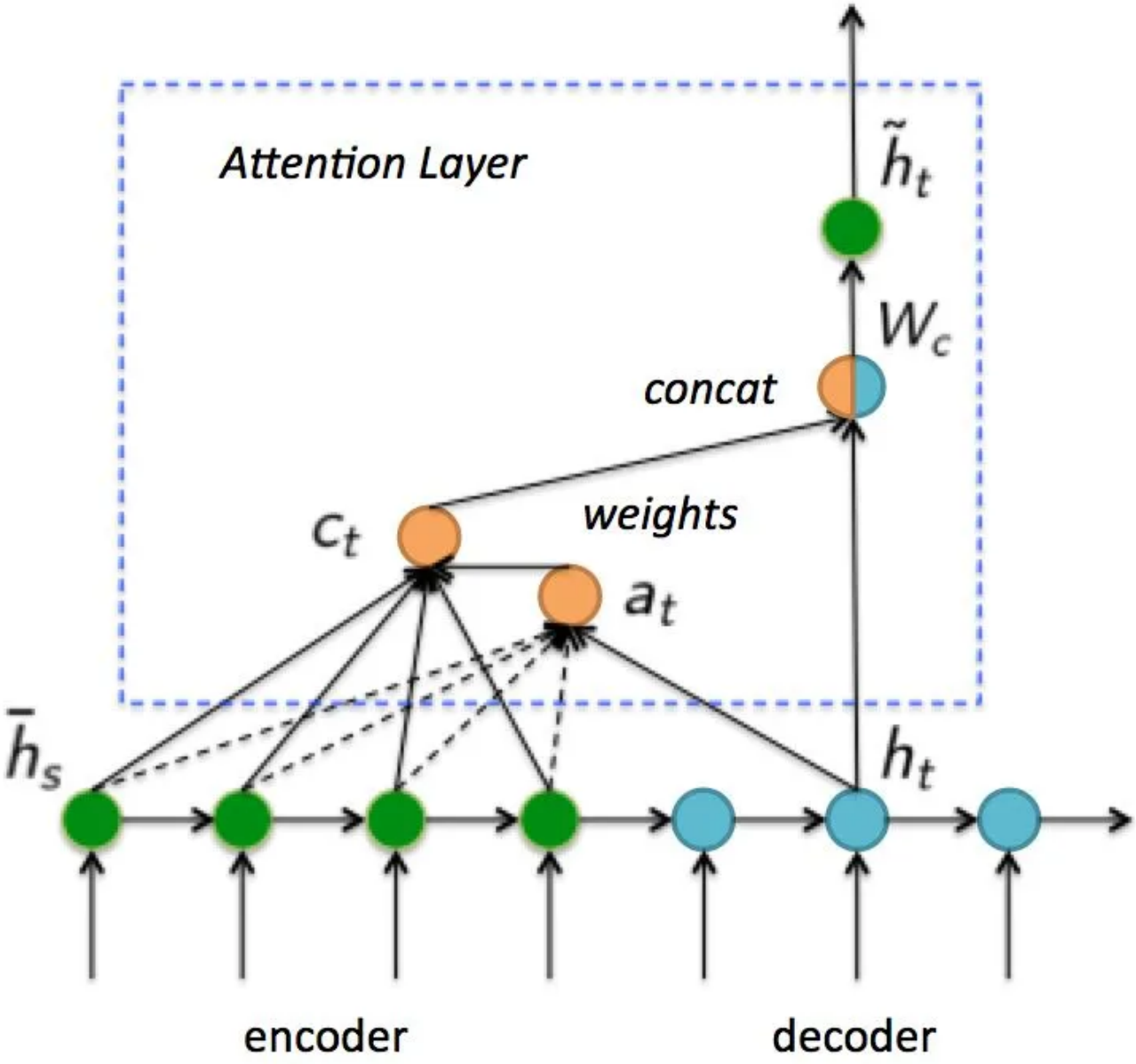


图13

Effective Approaches to Attention-based Neural Machine Translation
arxiv.org

为了说明具体结构，重新定义符号： $\bar{h}_s$ 代表Encoder状态， h_t 代表Decoder状态， $\tilde{h}_t$ 代表Attention Layer输出的最终Decoder状态，如图13。需要说明， $\bar{h}_s$ 和 h_t 是 $[n_h, 1]$ 大小的向量。接下来一起看看注意力机制具体实现方式。

首先，计算Decoder的 t 时刻隐藏层状态 h_t 对Encoder每一个隐藏层状态 $\bar{h}_s$ 权重 $a_t(s)$ 数值：

$$a_t(s) = \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}_{s'}))} \tag{20}$$

这里的 score 可以通过以下三种方式计算：

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s & \text{Dot} \\ h_t^T W_a \bar{h}_s & \text{General} \\ v_a^T \tanh(W_a \cdot \text{concat}(h_t, \bar{h}_s)) & \text{Concat} \end{cases} \tag{21}$$

所谓Dot就是向量内积，而General通过乘以 W_a 权重矩阵进行计算（ W_a 是 $[n_h, n_h]$ 大小的矩阵）。一般经验General方法好于Dot方法，Concat方法略去不讲。

其次，利用权重 $a_t(s)$ 计算所有隐藏层状态 $\bar{h}_s$ 加权之和 c_t ，即生成新的大小为 $[n_h, 1]$ 的Context状态向量：

$$c_t = \sum_s a_t(s) \cdot \bar{h}_s \tag{22}$$

接下来，将通过权重 $a_t(s)$ 生成的 c_t 与原始Decoder隐藏层 t 时刻状态 h_t 拼接在一起：

$$\tilde{h}_t = \tanh(W_c \cdot \text{concat}(c_t, h_t)) = \tanh(W_c \cdot [c_t; h_t]) \tag{23}$$

这里 c_t 和 h_t 大小都是 $[n_h, 1]$ ，拼接后会变大。由于需要恢复为原来形状，所以乘以全连接 W_c 矩阵。当然不恢复也可以，但是会造成Decoder RNN cell变大。

最后，对加入“注意力”的Decoder状态 $\tilde{h}_t$ 乘以 W_{ho} 矩阵即可获得输出：

$$y_t = W_{ho} \tilde{h}_t + b_{ho} \tag{24}$$

也可以根据需要，把新生成的状态 $\tilde{h}_t$ 继续送入RNN继续进行学习。**其中 W_a 和 W_c 参数需要通过学习获得。**

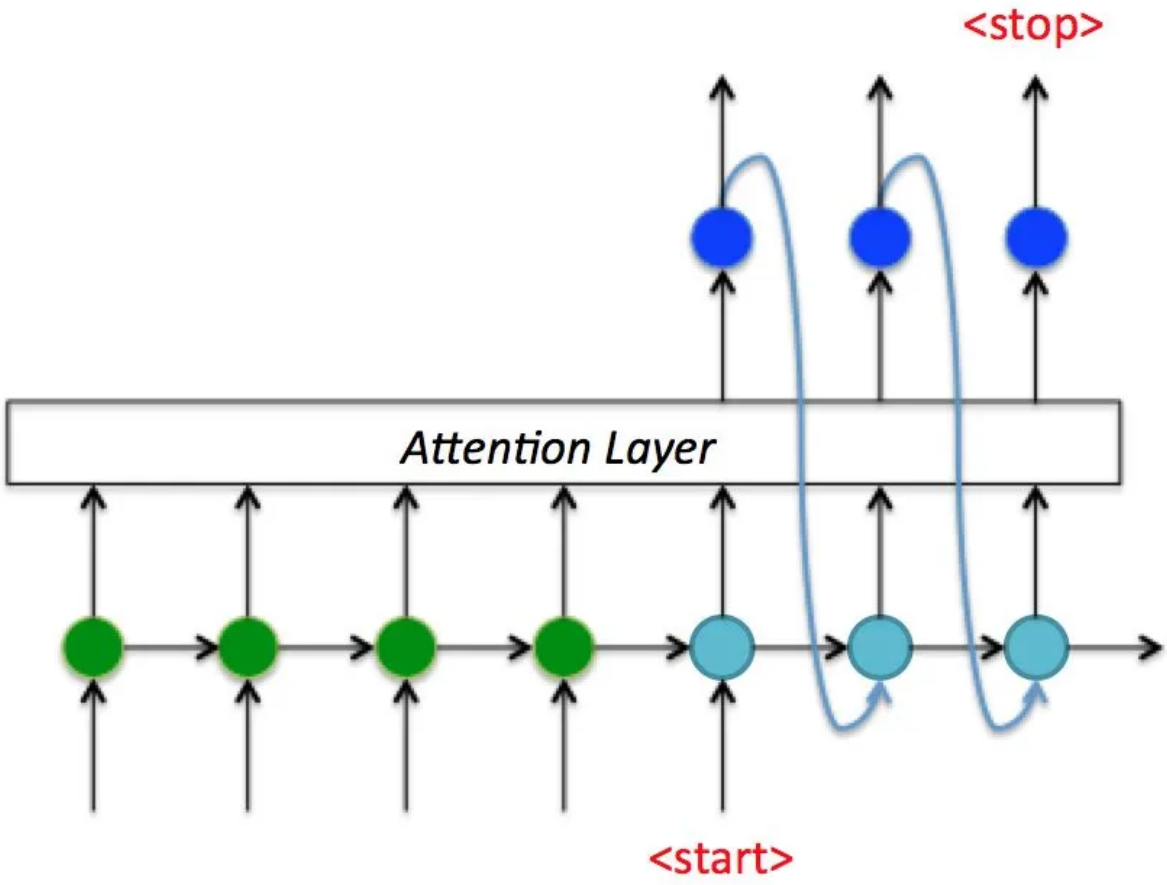


图14

在实际应用中当输入一组 x ，除了可以获得输出 y ，还能提取出 x_t 与 y_t 对应的权重数值 $a_t(s)$ 并画出来，如图15，这样就可以直观的看到时刻 t 注意力机制到底“注意”了什么。

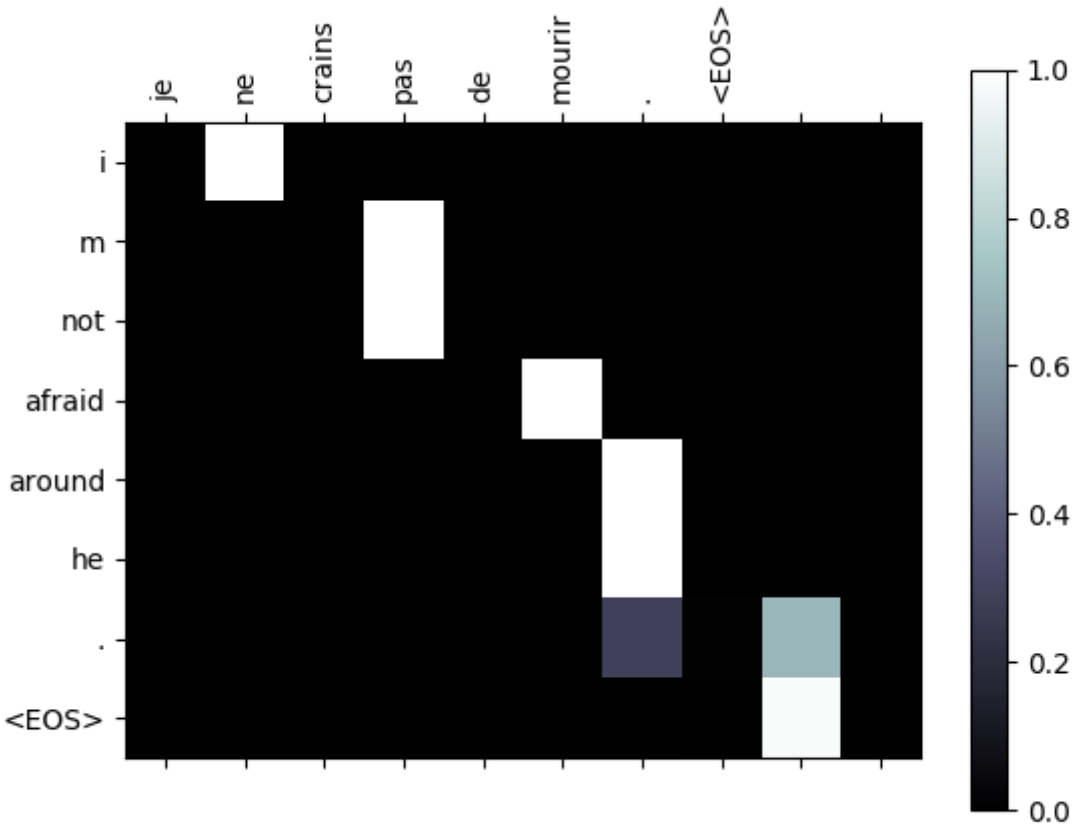


图15 注意力机制中的权重

可以看到，整个Attention注意力机制相当于在Seq2Seq结构上加了一层“包装”，内部通过函数 $\text{score}()$ 计算注意力向量 c_t ，从而给Decoder RNN加入额外信息，以提高性能。无论在机器翻译，语音识别，自然语言处理(NLP)，文字识别(OCR)，Attention机制对Seq2Seq结构都有很大的提升。

如何向RNN加入额外信息

Attention机制其实就是将的Encoder RNN隐藏层状态加权后获得权重向量 c_t ，额外加入到Decoder中，给Decoder RNN网络添加额外信息，从而使得网络有更完整的信息流。

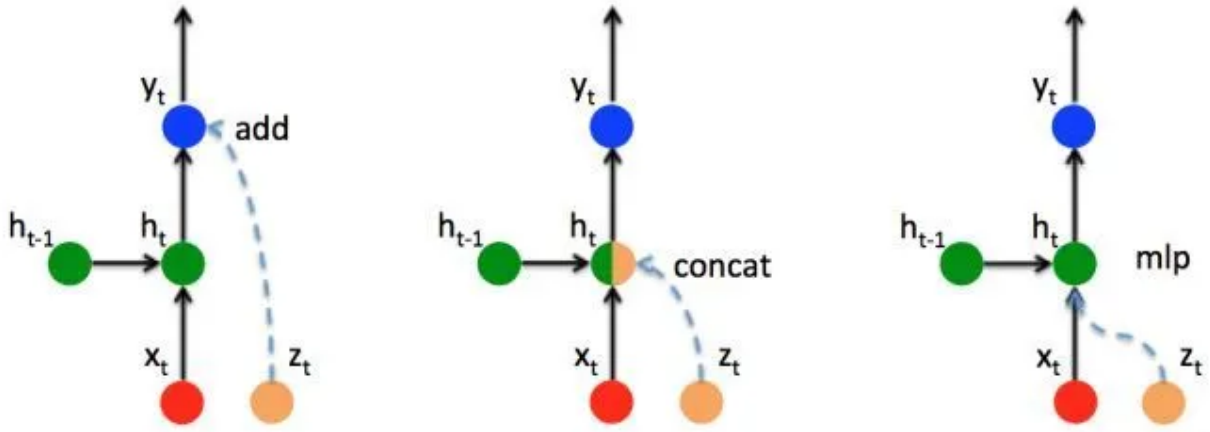


图16 RNN添加额外信息的3中方式

所以，假设有额外信息 z_t （如上文中的注意力向量 c_t ），给RNN网络添加额外信息主要有以下3种方式：

ADD：直接将 z_t 叠加在输出 y_t 上。

$$y_t \leftarrow y_t + z_t \tag{25}$$

CONCAT：将 z_t 拼接在隐藏层 h_t 后全连接恢复维度（不恢复维度也可以，但是会造成参数量加倍）。上篇文章中的LuongAttention机制就使用此种方法。

$$h_t \leftarrow W_c \cdot \text{concat}(h_t, z_t) \tag{26}$$

MLP：新添加一个对 z 的感知单元 $h_t^{zh} = W_{zh} \cdot z_t + b_{zh}$ 。

$$\begin{aligned} h_t &\leftarrow \tanh(h_t^{ih} + h_t^{hh} + h_t^{zt}) \\ &= \tanh((W_{ih} \cdot x_t + b_{ih}) + (W_{hh} \cdot h_{t-1} + b_{hh}) + (W_{zh} \cdot z_t + b_{zh})) \end{aligned} \tag{27}$$

特别说明：上文介绍的LuongAttention仅仅是注意力机制的一种具体实现，不代表Attention仅此一种。事实上Seq2Seq+Attention还有很多很玩法。望读者了解！
