循环神经网络（RNN）的场景与应用

Imagination Tech 2018-05-09

1. 场景与应用

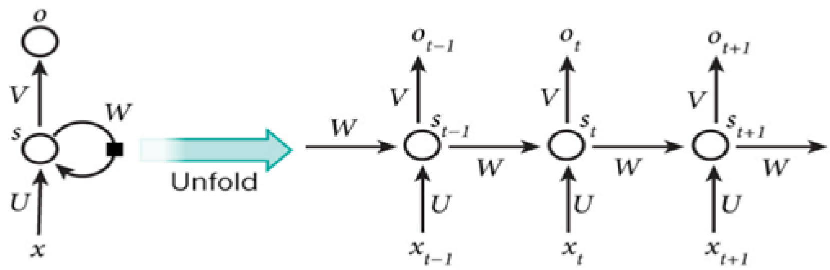
在循环神经网络可以用于文本生成、机器翻译还有看图描述等，在这些场景中很多都出现了RNN的身影。

2. RNN的作用

传统的神经网络DNN或者CNN网络他们的输入和输出都是独立的。对于这些模型输入的数据跟输出的数据大多是关联不太紧密的场景，但是有些场景输入的数据对后面输入的数据是有关系的，或者说后面的数据跟前面的数据是有关联的。例如，对于文本类的数据，当输入某句话的时候，刚开始输入第一个字的时候，再输入这句话的第二个字时候，其实第二个字要输入什么字其实是跟第一个字是有关联的。所以，对于这样一类的场景，通常是要考虑前面的信息的，以至于引入RNN模型。

对于RNN模型为解决这类问题引入了“记忆”这一概念。循环神经网络的循环来源于其每个元素中都执行相同的任务，但是输出依赖于输入和“记忆”两个部分。

3. RNN结构



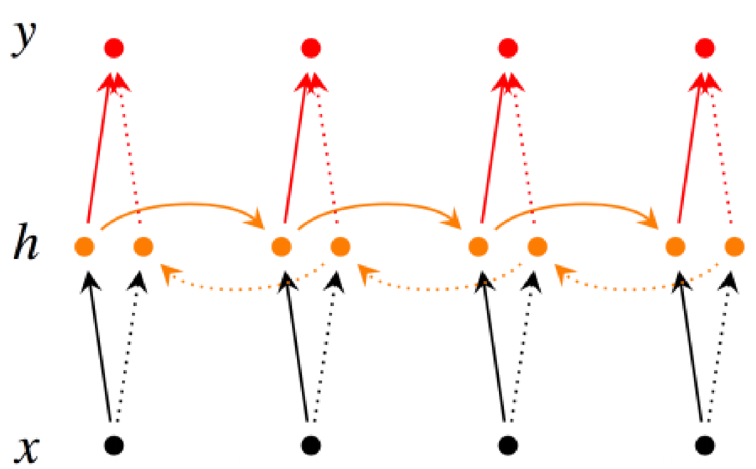
从图中看，对于RNN网络是按照时间序列展开的。对于图中的变量Wt，是在时刻t处的输入，St是时间t处的“记忆”，St=f(UXt+WSt−1 + b)，f可以是tanh等，f取tanh会把数据压缩到一个范围内，有时候也可以使用sigmoid函数。Ot是时间t出的输出，比如是预测下个词的话，可能是softmax输出的属于每个候选词的概率，Ot = softmax(VSt)。对于这里的St已经把Xt合并了，所以Ot的公式只有St。

对于循环神经网络，可以把隐状态St视作为“记忆体”，捕捉之前时间点上的信息。输出Ot有当前时间及之前所有“记忆”共同计算得到的。但由于St是一个有限的矩阵，对于之前的信息并不能完全捕捉到，也会随着时间的变长，对于之前的“记忆”也会“变淡”。对于RNN不同于DNN与CNN，这里的RNN其实整个神经网络都在共享一组参数（U,V,W），这样极大的减小了需要训练的参数。图中的Ot再由写任务下是不存在的，只需要对最后的结果输出就可以。

4. 不同类型的RNN

(1). 双向RNN

通过以上经典的RNN模型，它是只关心当前的输入和之前的“记忆”的，但有些情况下，当前的输入不知依赖于之前的序列元素，还依赖于后面序列的元素。比如，一篇文章，当读第一段时候我们并不知道文章的主体要讲什么内容，但当我们读完第一段的时候需要判断文章主要讲什么内容，这时候就需要读后面的内容才能知道这个文章主要讲的是什么。对于这样的场景需要后面的数据才能更好的预测当前的状态，所以引入了双向RNN，就是为了解决这一类问题的。双向RNN的模型如下：



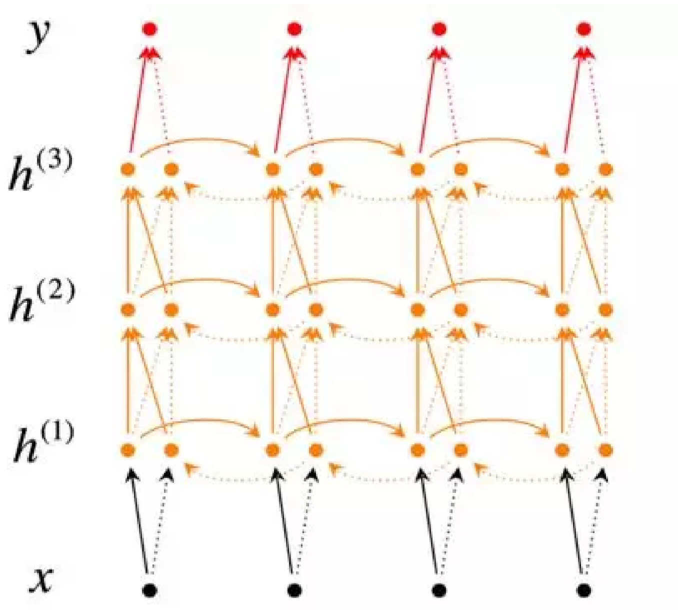
表达式：

$$\overrightarrow{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b})$$
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$
$$y_t = g(U[\overrightarrow{h}_t;\overleftarrow{h}_t] + c)$$

双向RNN是考虑到了前后的“记忆”，能够更好的关联到前后的信息。

(2). 深度双向RNN

对于深度双向RNN和双向RNN的区别是每一步和每一个时间点我们设定了多层的结构。结构如下：



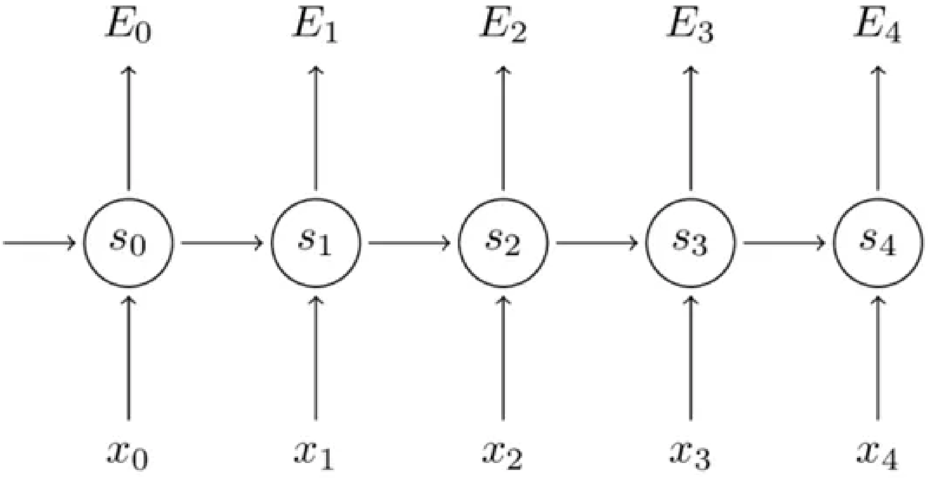
深层双向RNN的表达式：

$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$
$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$
$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

对于深层双向RNN考虑的信息与双向RNN相比变多了，这意味着能够对于某场景，能够关联更多的信息。

5. RNN与BPTT算法

对于循环神经网络的BPTT算法其实是BP算法的一个变体，但是循环神经网络的BPTT是与时间序列有关的。



对于这个问题是个Softmax问题所以这里用交叉熵损失，所以损失函数可以表示为：

$$
\begin{aligned}
E_t(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\
E(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t) \\
&= -\sum_t y_t \log \hat{y}_t
\end{aligned}
$$

对于求所有误差求W得偏导函数为：

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

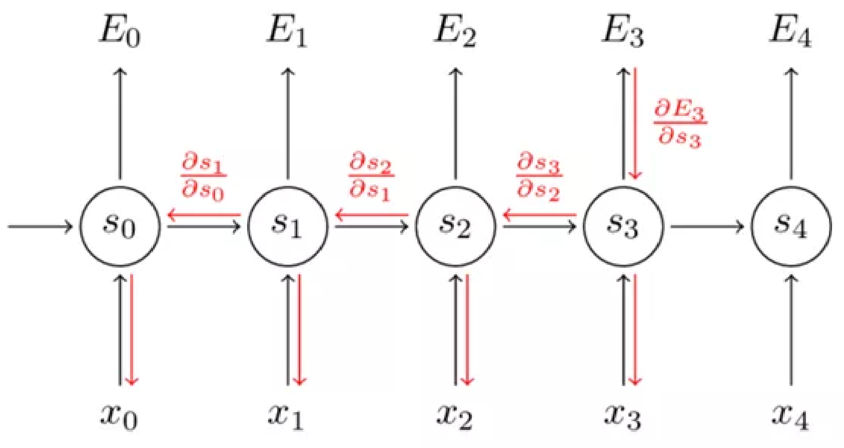
但是，

$$s_3 = \tanh(Ux_t + Ws_2)$$

所以，所以根据求导链式法则可以进一步求得：（又因为他们的权值是共享的）

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

用图表示如下：



通过上面的式子：

$$\frac{\partial E_3}{\partial W}=\sum_{k=0}^{3}\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

通过链式法则，表达式可以进一步表示为：

$$\frac{\partial E_3}{\partial W}=\sum_{k=0}^{3}\frac{\partial E_3}{\partial \hat{y_3}}\frac{\partial \hat{y_3}}{\partial s_3}\left(\prod_{j=k+1}^{3}\frac{\partial s_j}{\partial s_{j-1}}\right)\frac{\partial s_k}{\partial W}$$

通过以上的步骤求得损失函数的偏导后，就可以用SGD算法去做参数的更新。

本文转载自：http://www.cnblogs.com/Zhi-Z/p/8681967.html