RNN以及LSTM的介绍和公式梳理

原创 Dark_Scope 2015-07-25 16:32:32 283957 收藏 182 版权

分类专栏： 机器学习 文章标签： 算法 RNN LSTM

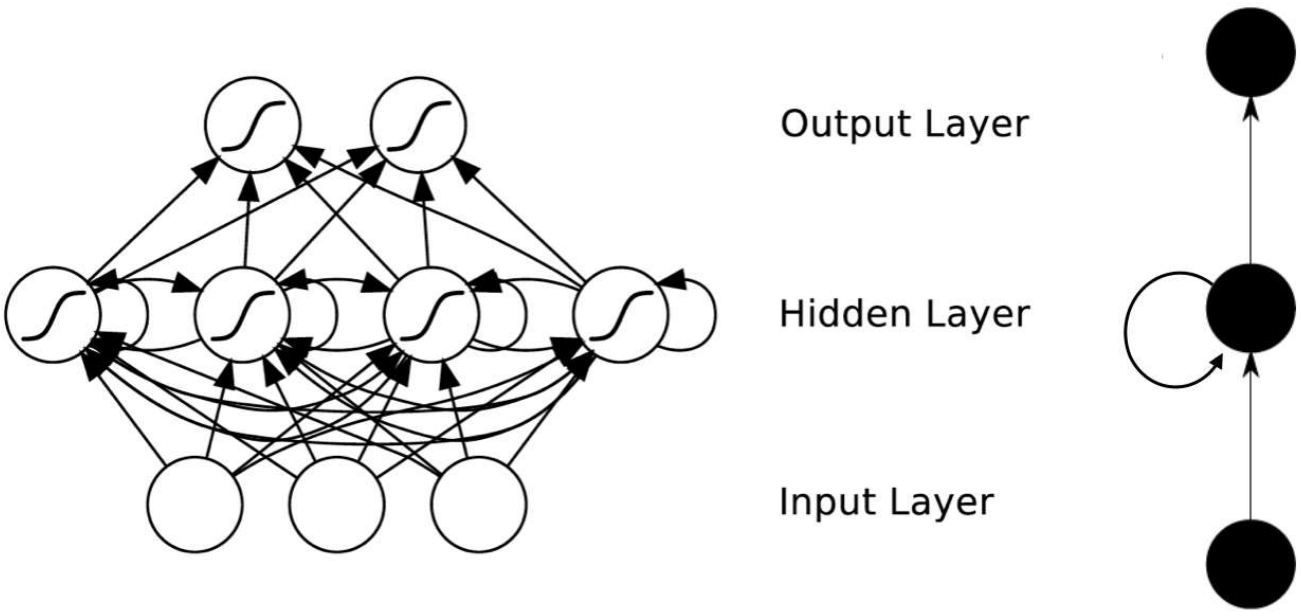
前言

好久没用正儿八经地写博客了，csdn居然也有了markdown的编辑器了，最近花了不少时间看RNN以及LSTM的论文，在组内『夜校』分享过了，再在这里总结一下发出来吧，按照我讲解的思路，理解RNN以及LSTM的算法流程并推导一遍应该是没有问题的。

RNN最近做出了很多非常漂亮的成果，比如Alex Graves的手写文字生成、名声大振的『根据图片生成描述文字』、输出类似训练语料的文字等应用，都让人感到非常神奇。这里就不细说这些应用了，我其实也没看过他们的paper，就知道用到了RNN和LSTM而已O(∩_∩)O

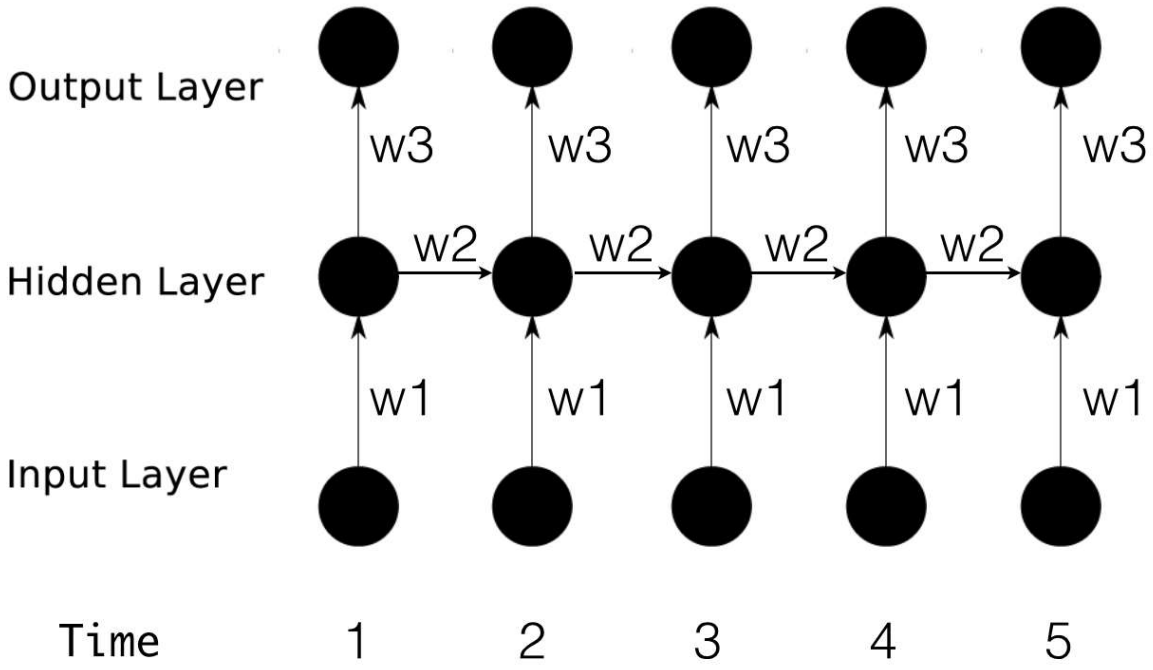
本文就假设你对传统的NN很熟悉了，不会的话参考http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial和我之前的文章http://blog.csdn.net/dark_scope/article/details/9421061学习一下~~

RNN（Recurrent Neural Network）

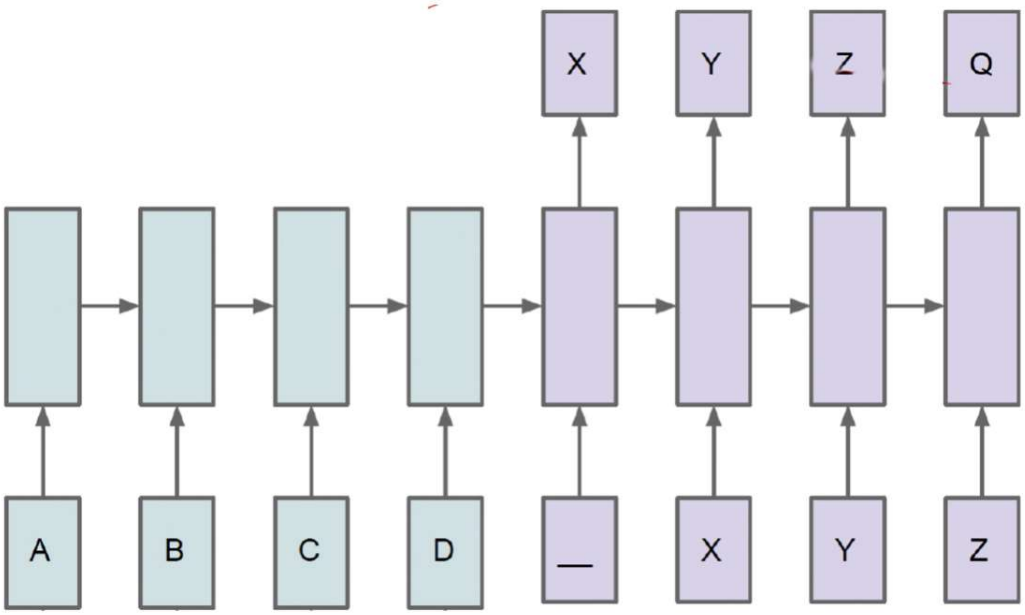


今天我这里讲到的RNN主要是上图这种结构的，即是Hidden Layer会有连向下一时间Hidden Layer的边，还有一种结构是Bidirectional Networks，也就是说会有来自下一时间的Hidden Layer传回来的边，但这不在我们今天的讨论范围内，讲完LSTM，如果你想推导一下Bidirectional Network，应该也是顺理成章的。为了方便推导和描述，我们后面都将左边简化为右边这样一个结构。

RNN和传统的多层感知机不同的就是跟时间沾上边了，下一时间（理解为step）会受本时间的影响，为了更好地说明这个东西，我们可以将网络按照时间进行展开：



主要的参数就是三部分：在RNN中每一个时间步骤用到的参数都是一样的，要理解清楚的是：一般来说，每一时间的输入和输出是不一样的，比如对于序列数据就是将序列项依次传入，每个序列项再对应不同的输出（比如下一个序列项），举个栗子（预测后面的状态）：

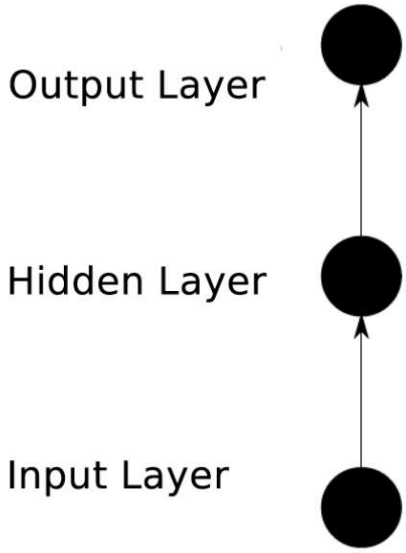


BPTT（Back Propagation Through Time）算法

将RNN展开之后，似乎一切都很明了了，前向传播（Forward Propagation）就是依次按照时间的顺序计算一次就好了，反向传播（Back Propagation）就是从最后一个时间将累积的残差传递回来即可，跟普通的神经网络训练并没有本质上的不同。

前向传播

直接上公式啦：



$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

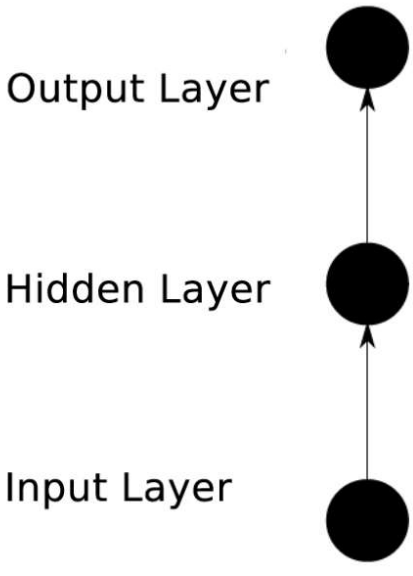
$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

本文用到的公式基本来自Alex的论文，**其中a表示汇集计算的值，b表示经过激活函数计算的值，w是不同节点之间连接的参数（具体睡谁连谁看下标），带下标k的是输出层，带下标h的是隐藏层相关的，除此之外你看到所有带括号的的函数都是激活函数，$\epsilon\epsilon$ 和 $\delta\delta$ 的定义看公式，LL 是最后的Loss function，这里没有给出具体的计算方法，因为这和NN是一样的，**可以看到输出层和普通的NN是完全一样的，接收隐藏层传入的数据并乘以参数求和，只是每一个计算出来的值都有个时间上标t，表示它是t时刻的那个节点。

而隐藏层的计算就是和NN不同的地方，从之前的拓扑图也看到了，隐藏层会接受来自上一时间隐藏层传入的数据，在公式里也体现出来了：第一个求和是和NN一致的，接收来自输入层的数据，第二个是接收来自上一隐藏层的数据。

后向传播



$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right)$$

$$\delta_j^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial a_j^t}$$

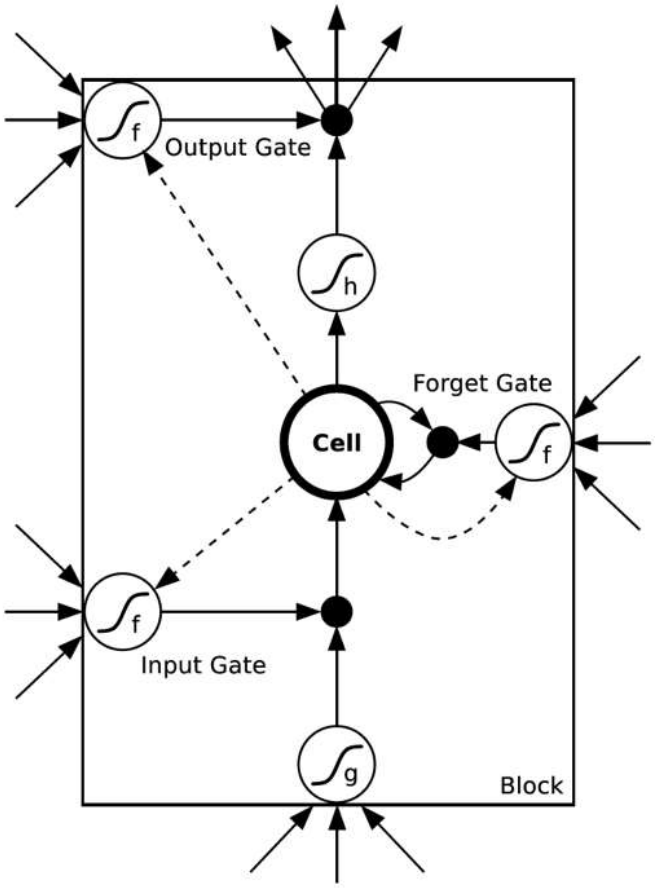
$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

这里主要给出的是计算隐藏层的累积残差的公式，因为输出层和经典的NN是一回事，可以看到第一个公式括号中的两个部分，一个是接收当前时间输出层传回的残差，第二个是接收下一时间隐藏层传回的残差，看着上面的图其实非常好理解。

LSTM（Long-Short Term Memory）

原生的RNN会遇到一个很大的问题，叫做 The vanishing gradient problem for RNNs，也就是后面时间的节点对于前面时间的节点感知力下降，也就是忘事儿，这也是NN在很长一段时间内不得志的原因，网络一深就没法训练了，深度学习那一套东西暂且不表，RNN解决这个问题用到的就叫LSTM，简单来说就是你不是忘事儿吗？我给你拿个小本子把事记上，好记性不如烂笔头嘛，所以LSTM引入一个核心元素就是Cell。

与其说LSTM是一种RNN结构，倒不如说LSTM是RNN的一个魔改组件，把上面看到的网络中的小圆圈换成LSTM的block，就是所谓的LSTM了。那它的block长什么样子呢？



怎么这么复杂……不要怕，下文慢慢帮你缕清楚。理解LSTM最方便的就是结合上面这个图，先简单介绍下里面有几个东西：

1. Cell，就是我们的小本子，有个叫做state的参数东西来记事儿的
2. Input Gate，Output Gate，在参数输入输出的时候起点作用，算一算东西
3. Forget Gate：不是要记东西吗，咋还要Forge呢。这个没找到为啥就要加入这样一个东西，因为原始的LSTM在这个位置就是一个值1，是连接到下一时间的那个参数，估计是以前的事情记太牢了，最近的就不住就不好了，所以要选择性遗忘一些东西。（没找到解释设置这个东西的动机，还望指正）

在阅读下面公式说明的时候时刻记得这个block上面有一个输出节点，下面有一个输入节点，block只是中间的隐层小圆圈~~~

前向传播

一大波公式正在路上。。。。。公式均来自Alex的论文
我们按照一般算法的计算顺序来给出每个部分的公式：

Input Gate

Input Gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{4.2}$$

$$b_\iota^t = f(a_\iota^t) \tag{4.3}$$

带下标L的就是跟Input Gate相关的，回去看上面那个图，看都有谁连向了Input Gate：外面的输入，来自Cell的那个虚线（虚线叫做peephole连接），这在公式立体现在4.2的第一项和第三项，计算就是普通的累积求和。那中间那个是个什么鬼?
带H的是一个泛指，因为LSTM的一个重要特点是其灵活性，cell之间可以互联，hidden units之间可以互联，至于连不连都看你（所以你可能在不同地方看到的LSTM公式结构都不一样）所以这个H就是泛指这些连进来的东西，可以看成是从外面连进了的三条边的一部分。
至于4.3就是简单的激活函数计算而已

Forget Gate

Forget Gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$

$$b_\phi^t = f(a_\phi^t)$$

再回去看那个图，连到Forget Gate都有哪些：输入层的输入、泛指的输入、来自cell的虚线，这个和Input Gate就是一回事嘛

Cells

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{4.6}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{4.7}$$

还是老样子，回去看都有啥连到了Cell（这里的cell不是指中间那个Cell，而是最下面那个小圆圈，中间的Cell表示的其实是那个状态值S[c][t]）：输入层的输入，泛指的输入。（这体现在4.6式中）
再看看中间的那个Cell状态值都有谁连过去了：这次好像不大一样，连过去的都是经过一个小黑点汇合的，从公式也能体现出来，分别是：ForgetGate*上一时间的状态 + InputGate*Cell激活后的值

Output Gate

Output Gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{4.8}$$

$$b_\omega^t = f(a_\omega^t) \tag{4.9}$$

老样子，看谁连到了Output Gate：跟其他几个Gate好像完全一样嘛~咦，4.8那个S[c][t]为啥是t，以前都是t-1啊。
这里我也没找到相关的原因，可以理解为在计算OG的时候，S[c][t]已经被计算出来了，所以就不用使用上一时间的状态值了（同样动机不明~~这就是设定好嘛。。。）

最后最后的输出

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \tag{4.10}$$

小黑点，用到了激活后的状态值和Output Gate的结果。
一定按照图的连接来捋一捋，公式还是非常清晰的。

后向传播

又一波公式来袭。。。。。。

$$\epsilon_c^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \qquad \epsilon_s^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta_g^{t+1} \tag{4.11}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t)\sum_{c=1}^{C} h(s_c^t)\epsilon_c^t \tag{4.12}$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t \tag{4.13}$$

Cells

$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t \tag{4.14}$$

Forget Gates

$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t \tag{4.15}$$

Input Gates

$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t \tag{4.16}$$

这次就只贴公式了，因为要每个都讲一下实在是太费功夫了，记住一个要点就是『看上面的图！！』，看看每个要求偏导的东西都有谁会反向传回东西给它，可以看到最复杂的就是4.13了，因为这是对那个状态值求导，它不光连向了三个门（公式后三项，两个本下一时刻，FG是本时刻的），还连向了最后的输出b[c][t]（公式第一项）以及下一时刻的自己（公式第二项），反向传播公式推导用到的唯一数学工具就是链式法则，你要觉得求偏导看不懂，就把它拆成链看就好了。

还有一点，记得最后的Loss Function是每一时间的一个求和，所以当你算当前层输出层传回来的残差都时候就可以忽略其它东西了，举个例子：4.11是对b[c][t]求偏导，而b[c][t]是正向传播LSTM block的输出，输出到谁了？当前层的输出层，下一层的Hidden Layer，这两个东西的最后的Loss function是分开的，彼此之间没有关系，所以公式里是两部分相加。4.11中的

G和之前的H一样，也是泛指，因为它不一定只输出到下一时间的自己，可能还会到下一时间的其他隐层unit，G代表什么纯看你怎么确定的网络结构。

$$\epsilon_t^x = \frac{\partial L}{\partial b_c^t} = \sum_k^K \frac{\partial L}{\partial a_k^t}\frac{\partial a_k^t}{\partial b_c^t} + \sum_g^G \frac{\partial L}{\partial a_g^{t+1}}\frac{\partial a_g^{t+1}}{\partial b_c^t} = (4.11)$$

$$\epsilon tx=\partial L\partial bct=\sum kK\partial L\partial akt\partial akt\partial bct+\sum gG\partial L\partial agt+1\partial agt+1\partial bct=(4.11)$$

后记

推导一遍之后你完全可以自己实现一次了，用到的东西也不复杂，可惜对于RNN和DL这些东西来说，确定网络结构和调参才是对最后效果有着决定性的影响，RNN和LSTM里可以调的东西太多了，每一个未知的激活函数选择，具体网络到底怎么连接，还有学习速率这种老问题。也是个大工程的说
ps.这MD的编辑器还可以啊~~！！

引用

【1】A. Graves. Supervised Sequence Labelling with Recurrent Neural Networks. Textbook, Studies in Computational Intelligence, Springer, 2012.