RNN 循环神经网络

2021年2月9日 11:14

序

上一篇文章讲了神经网络的基本实现，后面的 RNN 与 CNN 等等各种类型的神经网络算法实际上就是对基本方法的改进与拓展，用于解决各种情境下特定的问题。

为什么需要 RNN

考虑这样的场景，我们需要识别这么一个句子的成分构成：我尝试去演奏。首先需要像识别手写数字一样，先将句子分为数个词：我 尝试 去 演奏。注意这个 ‘尝试’ 。这里如果按照经典的 DNN 神经网路算法，则其很有可能被识别为名词而不是动词（也不能说很可能，总之无法针对这种情况做出有效的判断）。而结合上下文我们应当能清楚地知道这是个动词。

由此可以见得，有些情况下的判断需要结合前后的情况才能做出正确的判断（事实上 RNN 只做到了结合前文的判断）。所以就有了 RNN 这一网络结构。

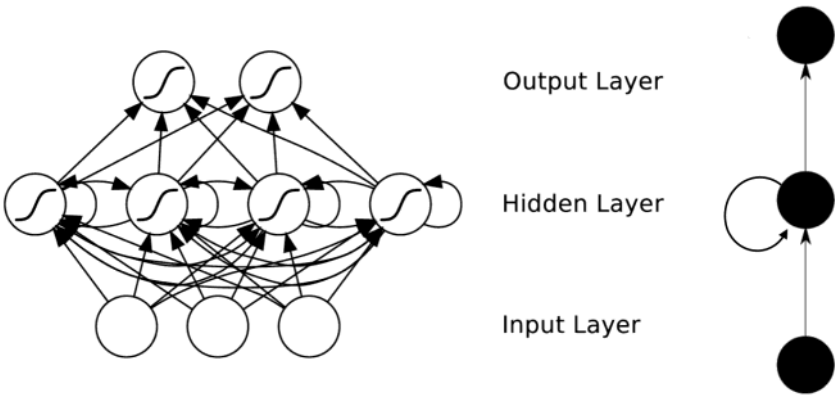
什么是 RNN

RNN 是一种特殊的神经网络结构，它是根据"人的认知是基于过往的经验和记忆"这一观点提出的。它考虑前一时刻的输入，并且随着前面的输入对当前的影响是不断累积的，这种模式就类似于**记忆功能**。

RNN 之所以称为**循环神经网路**，是因为当前的输入不仅包含了原始输入，也包含了上次输入产生的结果附带的输入。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。

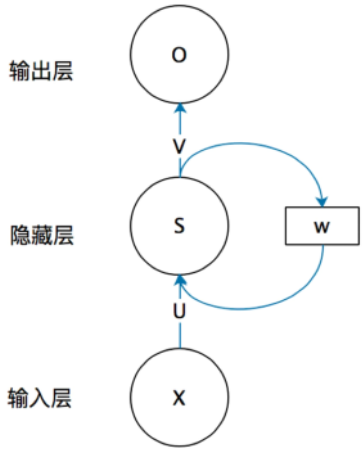
RNN 算法

结构 & 方程



这种图比较好地体现了 RNN 结构与经典神经网络结构之间的关联。可以看到 RNN 多了隐藏层各节点之间的两两关联（全连接网络）。不过个人认为这张图对于后面的算法并不很适合后面算法的理解。

接下来对这张图的各个层级模块进行进一步抽象。

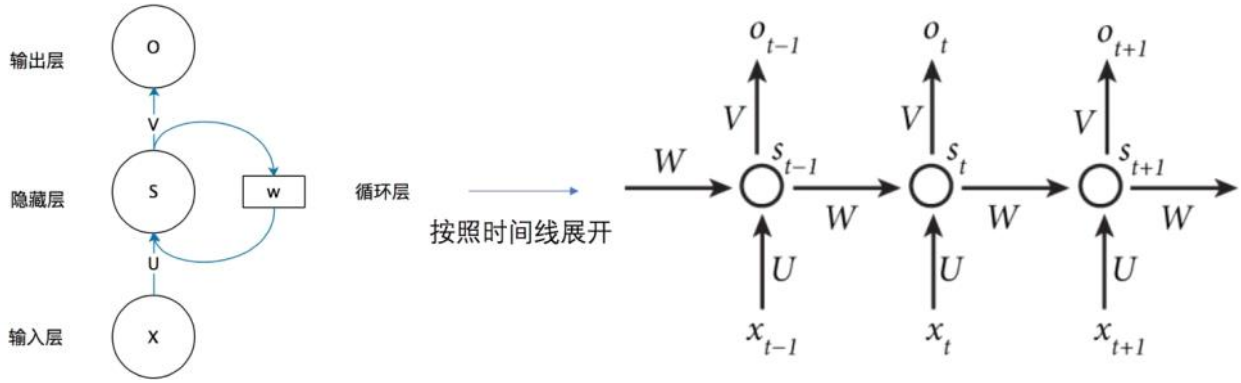


得到了这么一张图，需要对该图中的各个符号的含义做一个解释。事实上这张表格配合图片也解释了整个网络中数据流动的方向。

X	输入矩阵
U	输入层到隐藏层的权重矩阵
S	隐藏层的激活值矩阵（对应到每个神经元来说就是激活值输出）
V	隐藏层到输出层的权重矩阵
O	输出矩阵
W	隐藏层到隐藏层的权重矩阵

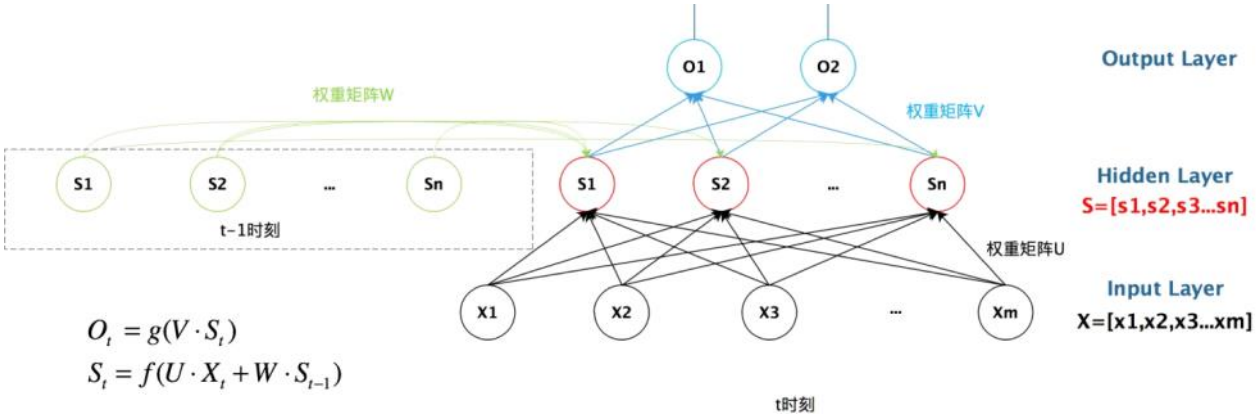
// 一开始分析到这里的时候我是比较疑惑的，因为按照之前的带权输入 z = w·a + b 的概念，为什么这里没有出现偏差这一参数 b。事实上，因为 a 输入量只会因为 w 权重变形，而与偏差 b 无关。

如果将上图展开，可以得到这么一张摊开后的图。



可以清楚的看到数据流动正如之前那张表格描述。隐藏层神经元的激活值在他们之间流动互相传递信息，保存‘记忆’。

可以进一步得到更加具体的结构图，并将他们之间存在的关系用数学公式的形式表达出来。



注意左下角的这两条方程。

$$O_t = g(V \cdot S_t)$$

$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$

这就是 RNN 神经网络向前传递的完整计算方程。St（当前激活输出） 的值不仅取决于 Xt 输入，同时还取决于 St-1（上次激活输出） 的值。

RNN 的反向传播算法 BPTT

目标

得到 RNN 代价函数关于 U、V、W的梯度。

RNN 与 DNN 的不同之处在于 RNN 不仅需要考虑结构上 o -> s ->x 的误差传递，还要考虑时间上 st -> st-1 -> ... 的误差传递。

但是从向前传递的公式中可以看出，V 的梯度是与时间传递无关的，而 U、W 的误差是会随时间传递的。那么就先从最简单的 V 入手。

- 激活函数 Φ 为 Sigmoid 函数
- 变换函数 ϕ 为 Softmax 函数
- 损失函数 L 为 Cross Entropy （即代价函数）

// 这里采用什么函数不重要，因为具体函数型只会影响到后面具体求导时的步骤，这里只从方程上讨论求各参数梯度的实现。

经过一系列的链式法则，求导化简（事实上这其中的过程目前不必深究）。得到

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{n} \left(\frac{\partial L_t}{\partial o_t} * \varphi'(o_t^*) \right) \times s_t^T$$

看到这个公式是有十足的亲切感的。这不正是 BP 反向传播算法中 BP1 与 BP4 对输出层误差的处理吗。是完全一致的，这也可以看出 BPTT 与 BP 根本上是同源的。

接下来对 U、W 进行分析。

- 首先对二者计算时间上的"局部梯度"：(我也没细看，总之就是链式法则的应用了)

$$\frac{\partial L_t}{\partial s_t^*} = \frac{\partial s_t}{\partial s_t^*} * \left(\frac{\partial s_t^T V^T}{\partial s_t} \times \frac{\partial L_t}{\partial V s_t} \right) = \phi'(s_t^*) * \left[V^T \times \left(\frac{\partial L_t}{\partial o_t} * \varphi'(o_t^*) \right) \right]$$

$$\frac{\partial L_t}{\partial s_{k-1}^*} = \frac{\partial s_k^*}{\partial s_{k-1}^*} \times \frac{\partial L_t}{\partial s_k^*} = \phi'(s_{k-1}^*) * \left(W^T \times \frac{\partial L_t}{\partial s_k^*} \right), \quad (k = 1, \ldots, t)$$

- 再根据时间上计算出的梯度，计算出全局梯度：

$$\frac{\partial L_t}{\partial U} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_k^*} \times \frac{\partial s_k^*}{\partial U} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_k^*} \times x_k^T$$

$$\frac{\partial L_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_k^*} \times \frac{\partial s_k^*}{\partial W} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial s_k^*} \times s_{k-1}^T$$

再一次地，这次的结果就是带上时间通道误差

$$\sum_{k=1}^{t} \frac{\partial L_t}{\partial s_k^*}$$

的 BP 反向传播表达式。

再看一眼 RNN 向前传递的方程，与 BP 算法方程。果然能感受到数学的大道至简之美。

$$O_t = g(V \cdot S_t)$$

$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \qquad \text{(BP2)}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \text{(BP3)}$$

$$\frac{\partial C}{\partial w_{j\,k}^l} = a_k^{l-1} \delta_j^l \qquad \text{(BP4)}$$

问题

这次的反向传播没有涉及偏差 b 的误差纠正，是否是与 U、V、W 这些权值矩阵融合考虑

了?

梯度爆炸与梯度消失的问题

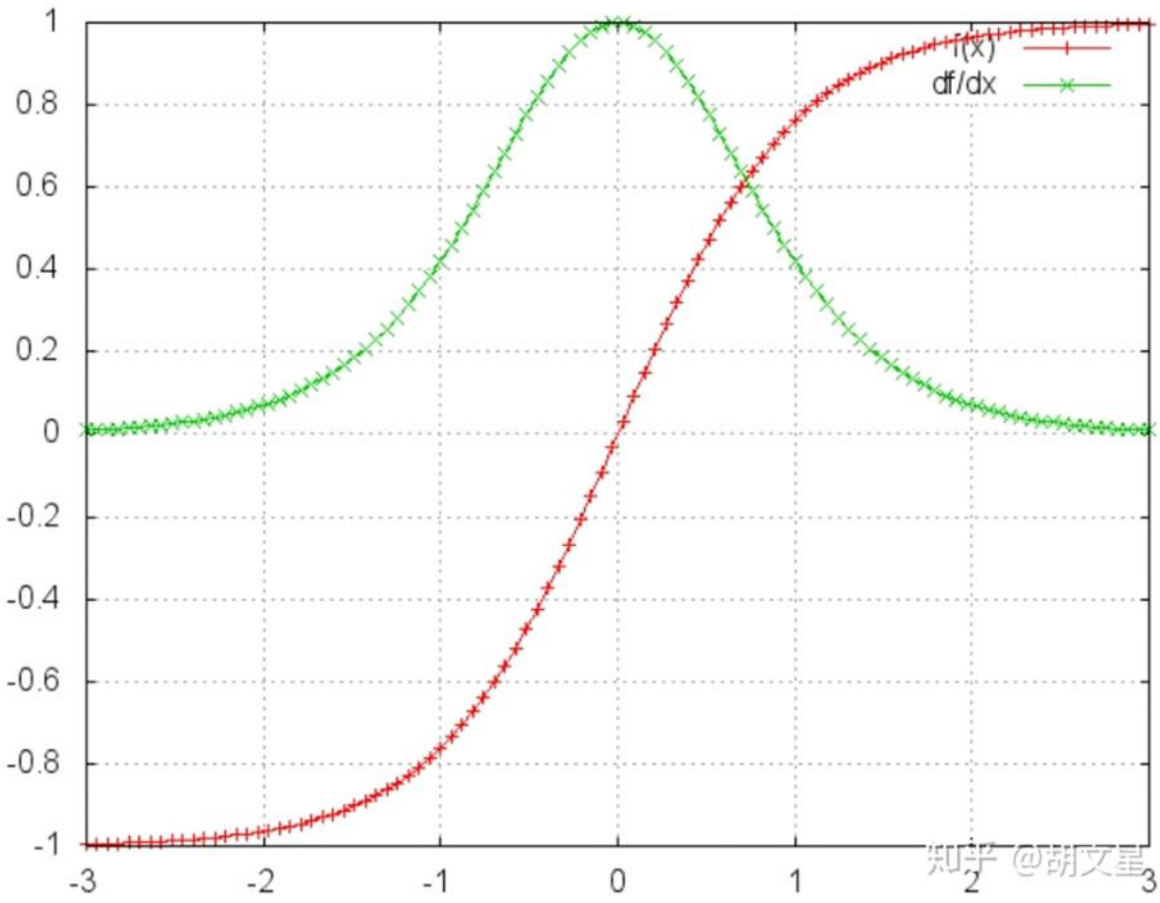
因为在更新 W 偏导数即求 W 梯度的步骤中存在:

$$\frac{\partial L_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial O_t} \frac{\partial O_t}{\partial S_t} \left(\prod_{j=k+1}^{t} \frac{\partial S_j}{\partial S_{j-1}} \right) \frac{\partial S_j}{\partial W} \qquad (3)$$

把 S_j 展开: $s_j = tanh(Ux_j + Ws_{j-1} + b)$

则式(3)中的 $\prod_{j=k+1}^{t} \frac{\partial S_j}{\partial S_{j-1}}$ 就变成了: $\prod_{j=k+1}^{t} tanh' W$

激活函数tanh和它的导数图像如下:



可以看到 tanh 的导函数在几乎训练过程中都是 < 1的。

再回过头来看这个式子:

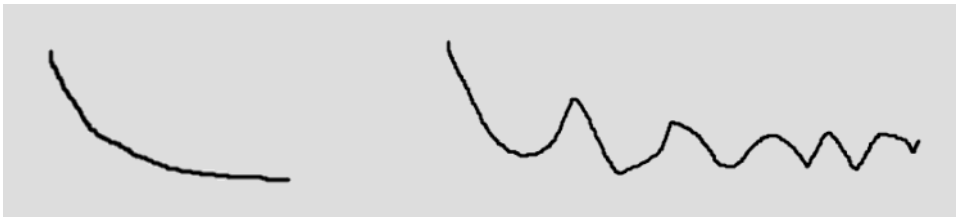
$$\prod_{j=k+1}^{t} tanh' W$$

随着时间 t 的增长，tanh' 需要被连乘的次数变多，导致这个式子趋近于 0，这便是梯度消失。

而当权重 W 过大时，且时间增长，W 需要被连乘的次数变多，导致这个式子趋近于无穷，这便是梯度爆炸。

梯度消失带来的影响是，该网络对于经过了长时间的‘记忆’，会逐渐‘遗忘’，即长时记忆失效的问题。

梯度爆炸带来的影响是，会导致权重变化地非常大，代价函数无法收敛。这有点抽象，这里抽象地表达一下这种感觉。



正常　　异常（梯度爆炸，变化剧烈）

可以发现问题的根本就在于：

$$\prod_{j=k+1}^{t} \frac{\partial S_j}{\partial S_{j-1}}$$

存在这么一个连乘项。那么如何解决这个问题呢？百度告诉我要用 LSTM。这个打算放在另一篇博客单独讲。连同实现明天搞定吧。

参阅的博客链接

https://zhuanlan.zhihu.com/p/26892413

https://zhuanlan.zhihu.com/p/30844905

https://blog.csdn.net/lzw66666/article/details/113132149