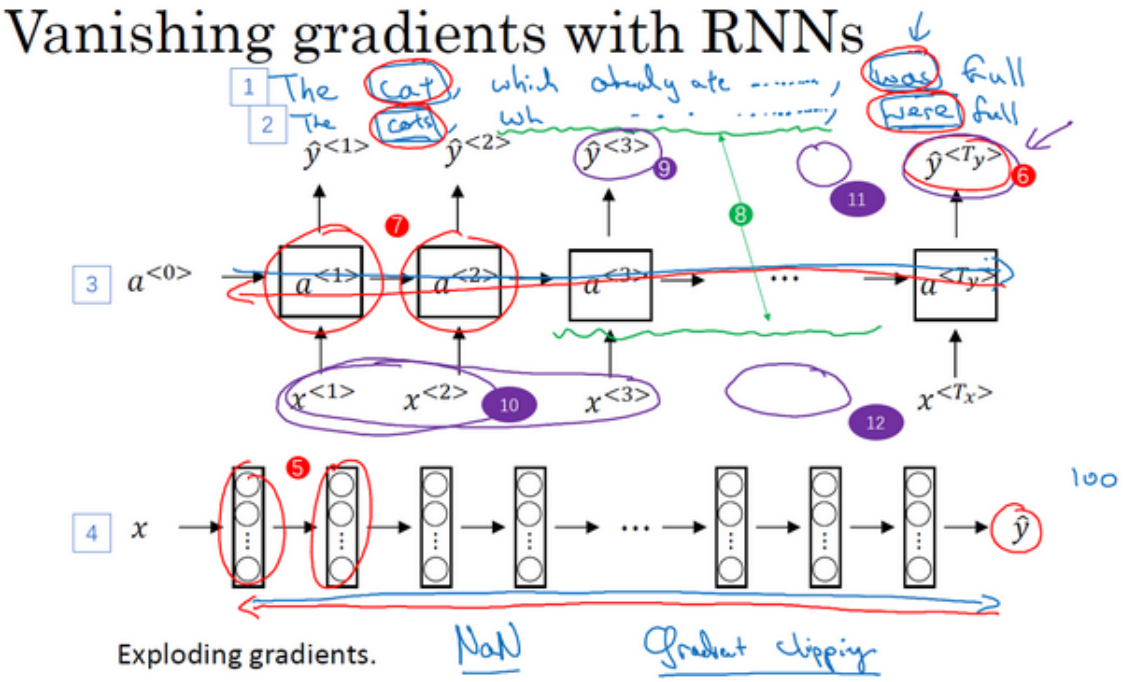
1.8 循环神经网络的梯度消失（Vanishing gradients with RNNs）

你已经了解了 **RNN** 是如何工作的了，并且知道如何应用到具体问题上，比如命名实体识别，比如语言模型，你也看到了怎么把反向传播用于 **RNN**。其实，基本的 **RNN** 算法还有一个很大的问题，就是**梯度消失**的问题。这节课我们会讨论，在下几节课我们会讨论一些方法用来解决这个问题。



你已经知道了 **RNN** 的样子，现在我们举个语言模型的例子，假如看到这个句子（上图编号 1 所示），**“The cat, which already ate ……, was full.”**，前后应该保持一致，因为 **cat** 是单数，所以应该用 **was**。**“The cats, which ate ……, were full.”**（上图编号 2 所示），**cats** 是复数，所以用 **were**。这个例子中的句子有长期的依赖，最前面的单词对句子后面的单词有影响。**但是我们目前见到的基本的 RNN 模型（上图编号 3 所示的网络模型），不擅长捕获这种长期依赖效应，**解释一下为什么。

你应该还记得之前讨论的训练很深的网络，我们讨论了梯度消失的问题。比如说一个很深很深的网络（上图编号 4 所示），100 层，甚至更深，对这个网络从左到右做前向传播然后再反向传播。**我们知道如果这是个很深的神经网络，从输出$\hat{y}$得到的梯度很难传播回去，很难影响靠前层的权重，很难影响前面层（编号 5 所示的层）的计算。**

对于有同样问题的 **RNN**，首先从左到右前向传播，然后反向传播。但是反向传播会很困难，因为同样的梯度消失的问题，后面层的输出误差（上图编号 6 所示）很难影响前面层

（上图编号 7 所示的层）的计算。这就意味着，实际上很难让一个神经网络能够意识到它要记住看到的是单数名词还是复数名词，然后在序列后面生成依赖单复数形式的 **was** 或者 **were**。而且在英语里面，这中间的内容（上图编号 8 所示）可以任意长，对吧？所以你需要长时间记住单词是单数还是复数，这样后面的句子才能用到这些信息。也正是这个原因，所以基本的 **RNN** 模型会有很多局部影响，意味着这个输出$\hat{y}^{<3>}$（上图编号 9 所示）主要受$\hat{y}^{<3>}$附近的值（上图编号 10 所示）的影响，上图编号 11 所示的一个数值主要与附近的输入（上图编号 12 所示）有关，上图编号 6 所示的输出，基本上很难受到序列靠前的输入（上图编号 10 所示）的影响，这是因为不管输出是什么，不管是对的，还是错的，这个区域都很难反向传播到序列的前面部分，也因此网络很难调整序列前面的计算。这是基本的 **RNN** 算法的一个缺点，我们会在下几节视频里处理这个问题。**如果不管的话，RNN 会不擅长处理长期依赖的问题。**

Exploding gradients. NaN Gradient clipping

尽管我们一直在讨论梯度消失问题，但是，你应该记得我们在讲很深的神经网络时，我们也提到了梯度爆炸，我们在反向传播的时候，随着层数的增多，梯度不仅可能指数型的下降，也可能指数型的上升。**事实上梯度消失在训练 RNN 时是首要的问题**，尽管梯度爆炸也是会出现，但是梯度爆炸很明显，因为指数级大的梯度会让你的参数变得极其大，以至于你的网络参数崩溃。所以梯度爆炸很容易发现，因为参数会大到崩溃，你会看到很多 **NaN**，或者不是数字的情况，这意味着你的网络计算出现了数值溢出。如果你发现了梯度爆炸的问题，一个解决方法就是用梯度修剪。**梯度修剪**的意思就是观察你的梯度向量，如果它大于某个阈值，缩放梯度向量，保证它不会太大，这就是通过一些最大值来修剪的方法。所以如果你遇到了梯度爆炸，如果导数值很大，或者出现了 **NaN**，就用梯度修剪，这是相对比较鲁棒的，这是梯度爆炸的解决方法。然而梯度消失更难解决，这也是我们下几节视频的主题。

总结一下，在前面的课程，我们了解了训练很深的神经网络时，随着层数的增加，导数有可能指数型的下降或者指数型的增加，我们可能会遇到梯度消失或者梯度爆炸的问题。加入一个 **RNN** 处理 1,000 个时间序列的数据集或者 10,000 个时间序列的数据集，这就是一个 1,000 层或者 10,000 层的神经网络，这样的网络就会遇到上述类型的问题。梯度爆炸基本上用梯度修剪就可以应对，但梯度消失比较棘手。我们下节会介绍 **GRU**，门控循环单元网络，这个网络可以有效地解决梯度消失的问题，并且能够使你的神经网络捕获更长的长期依赖。