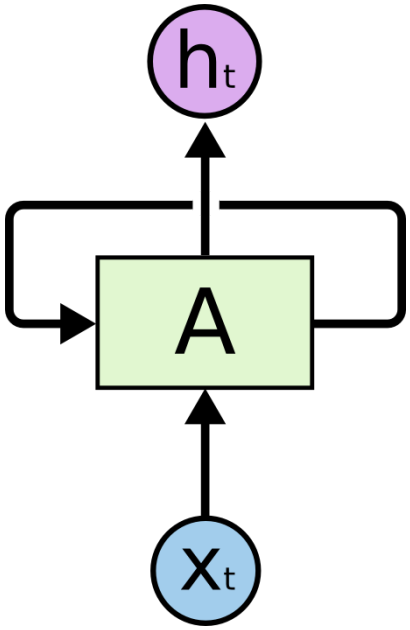
RNN和LSTM以及处理梯度消失和爆炸

1、理解RNN

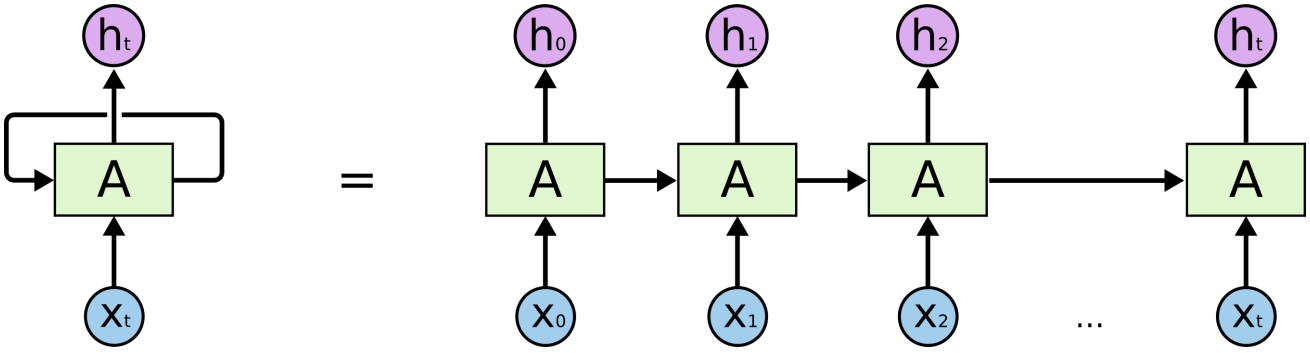
循环神经网络(RNN)，具有保持信息的能力。人类并非每一秒都在从头开始思考问题。当你阅读这篇文章时，你是基于之前的单词来理解每个单词。你并不会把所有内容都抛弃掉，然后从头开始理解。你的思考具有持久性。

传统的神经网络并不能做到这一点，这似乎是其一个主要的缺点。例如，想象你要把一部电影里面每个时间点所正在发生的事情进行分类。并不知道传统神经网络怎样才能把关于之前事件的推理运用到之后的事件中去。



RNN可以表示如上图，神经网络的模块 A 输入为 x_t，输出为 h_t。模块 A 的循环结构使得信息从网络的上一步传到了下一步。

这个循环使循环神经网络看起来有点神秘。然而，如果你仔细想想就会发现它与普通的神经网络并没有太大不同。循环神经网络可以被认为是相同网络的多重复制结构，每一个网络把消息传给其继承者。如果我们把循环体展开就是这样，如图所示：



这种链式属性表明，循环神经网络与序列之间有着紧密的联系。这也是运用这类数据最自然的结构。取得这项成功的一个要素是『LSTMs』，这是一种非常特殊的循环神经网络，对于许多任务，比标准的

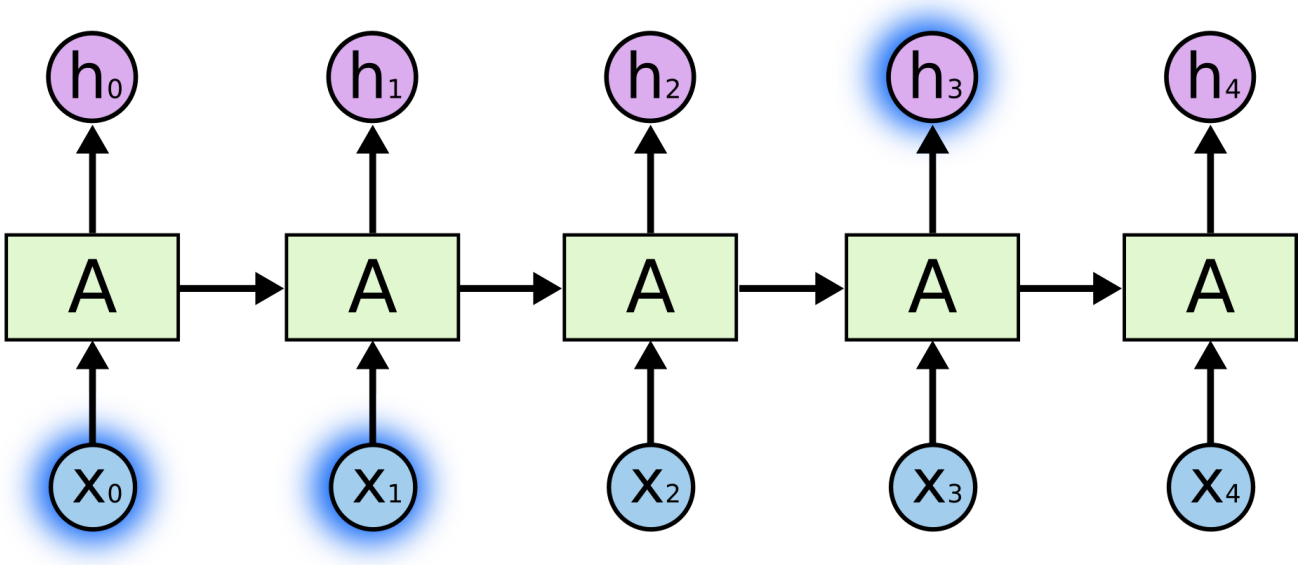
版本要有效得多。几乎所有基于循环神经网络的好成果都使用了它们。

2、长期依赖问题

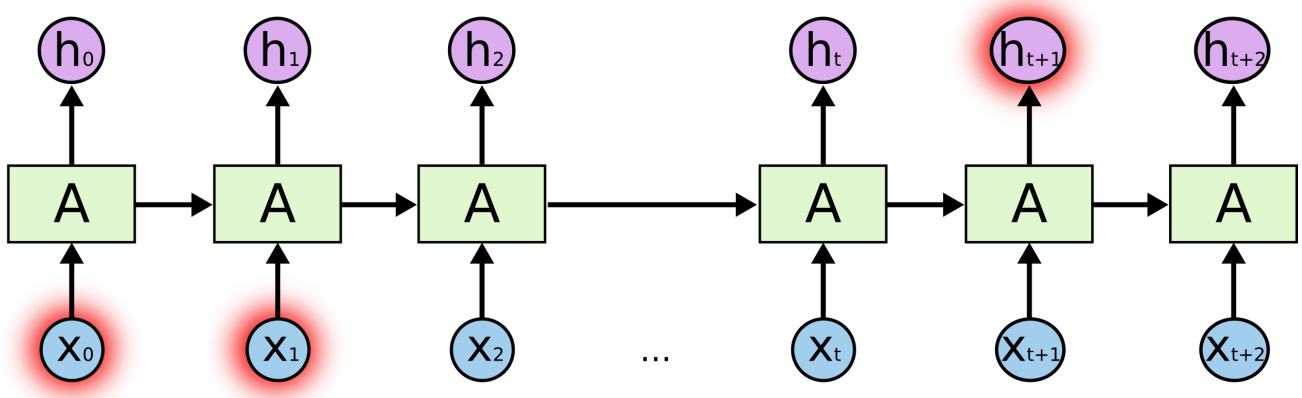
RNNs的一个想法是，它们可能会能够将之前的信息连接到现在的任务之中。例如用视频前一帧的信息可以用于理解当前帧的信息。如果RNNs能够做到这些，那么将会非常使用。但是它们可以吗？这要看情况。

有时候，我们处理当前任务仅需要查看当前信息。例如，设想又一个语言模型基于当前单词尝试着去预测下一个单词。如果我们尝试着预测『the clouds are i n the sky』的最后一个单词，我们并不需要任何额外的信息了-很显然下一个单词就是『天空』。这样的话，如果目标预测的点与其相关信息的点之间的间隔较小，RNNs可以学习利用过去的信息。

但是也有时候我们需要更多的上下文信息。设想预测这句话的最后一个单词：『I grew up in France… I speak fluent French』。最近的信息表明下一个单词似乎是一种语言的名字，但是如果我们希望缩小确定语言类型的范围，我们需要更早之前作为France 的上下文。而且需要预测的点与其相关点之间的间隔非常有可能变得很大，如图所示：



不幸的是，随着间隔增长，RNNs变得难以学习连接之间的关系了，主要原因是梯度消失问题，如图所示：



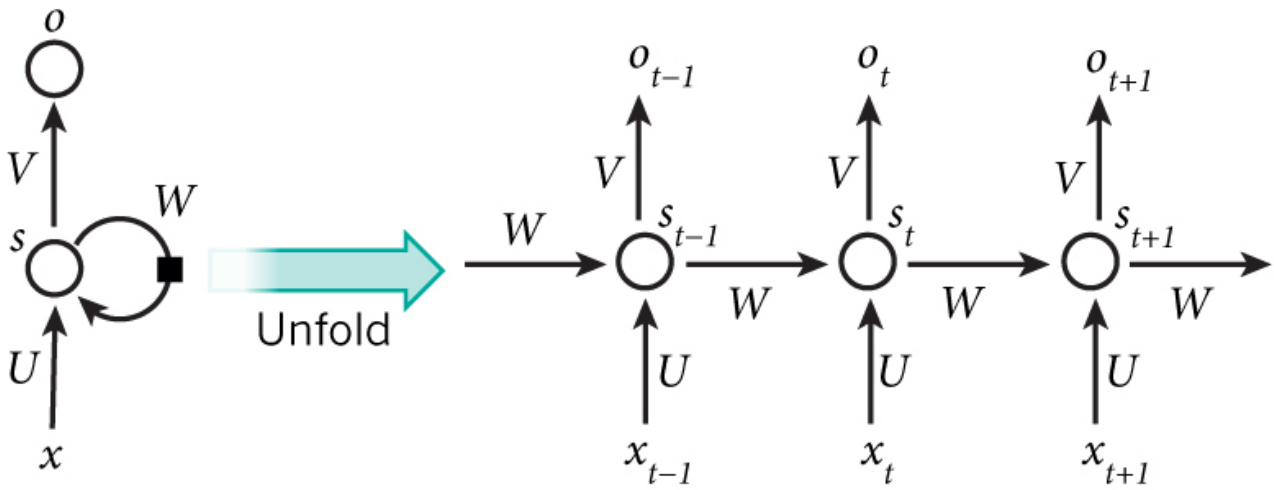
理论上来说，RNNs绝对能够处理这种『长期依赖』。人们可以小心选取参数来解决这种类型的小模型。悲剧的是，事实上，RNNs似乎并不能学习出来这些参数。参数不能学习，手工选择参数非常复杂

3、RNN反向传播及其问题

让我们先迅速回忆一下RNN的基本公式，注意到这里在符号上稍稍做了改变（o 变成 $\hat{y}$），这只是为了和我参考的一些资料保持一致。其中 U, V, W 参数共享，这样能够保证整个模型的泛化能力更强，计算速度更快。

$$s_t = tanh(Ux_t + Ws_{t-1})$$

$$\hat{y}_t = softmax(Vs_t)$$

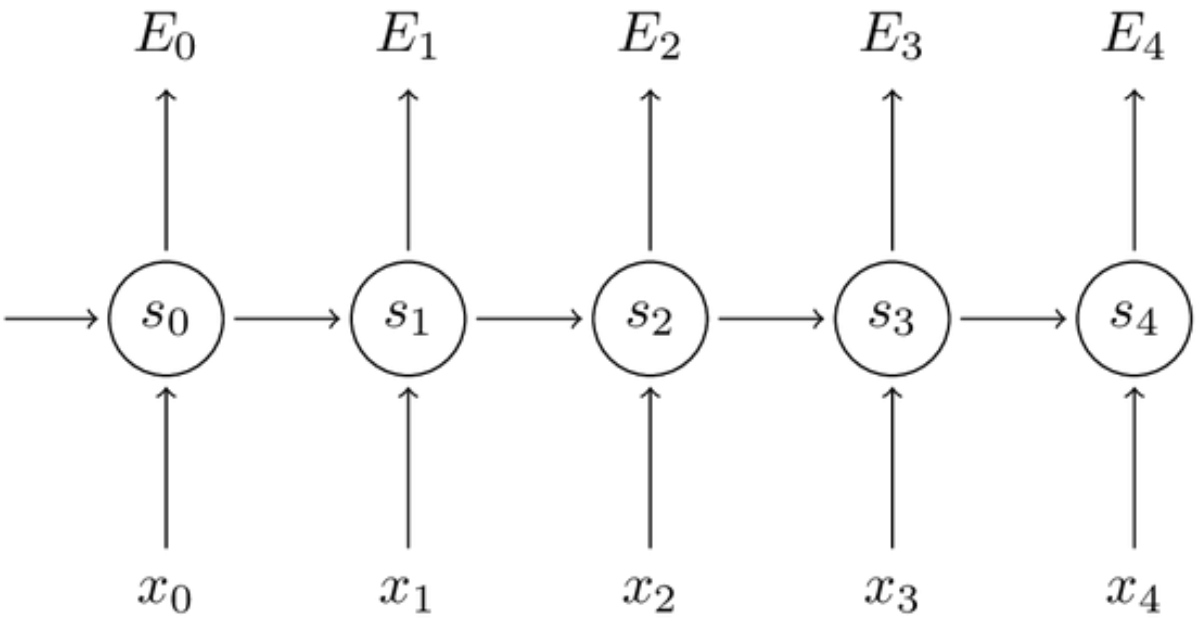


在这里我们将损失函数定义如下：

$$E_t(y_t, \hat{y}_t) = -y_t log(\hat{y}_t)$$

$$E_t(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t log(\hat{y}_t)$$

这里，y_t 表示时刻 t 正确的词，$\hat{y}_t$ 是我们的预测。通常我们会把整个句子作为一个训练样本，所以总体错误是每一时刻的错误的加和。



我们的目标是计算错误值相对于参数 U, V, W 的梯度，以及用随机梯度下降学习好的参数。就像我们要把所有错误相加一样，我们同样会把每一时刻针对每个训练样本的梯度值相加：$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$。为了计算梯度，我们使用链式求导法则，主要是用反向传播算法往后传播错误。下文使用 E_3 作为例子，主要是为了描述方便。

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} = (\hat{y}_3 - y_3) \otimes s_3$$

其中 $y_t, \hat{y}_t$ 分别表示输出对应的标签，真实输出。$z_3 = Vs_t$，$\otimes$ 表示向量的外积。如果你不理解上面的公式，不要担心，我在这里跳过了一些步骤，你可以自己尝试来计算这些梯度值。这里我想说明的一点是梯度值只依赖于当前时刻的结果 $\hat{y}_3, y_3, s_3$ 。根据这些，计算 V 的梯度就只剩下简单的矩阵乘积了。

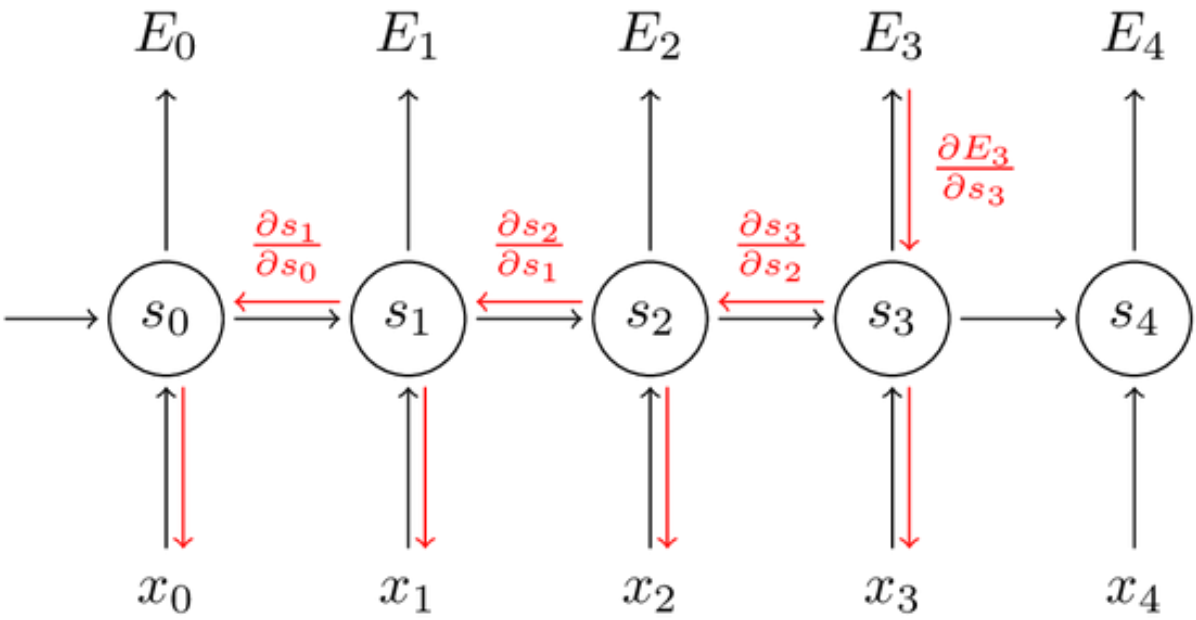
但是对于梯度 $\frac{\partial E_3}{\partial W}$ 情况就不同了，我们可以像上面一样写出链式法则。

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

注意到这里的 $s_3 = tanh(Ux_t + Ws_2)$ 依赖于s_2，s_2 依赖于 W 和 s_1，等等。所以为了得到 W 的梯度，我们不能将 s_2 看作常量。我们需要再次使用链式法则，得到的结果如下：

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

我们把每一时刻得到的梯度值加和，换句话说，W 在计算输出的每一步中都使用了，所以每一个 s 都与 W 有关，需要使用累加求导的方式。我们需要通过将 $t = 3$ 时刻的梯度反向传播至 $t = 0$ 时刻。



注意到这里和我们在深度前向神经网络中使用的标准反向传播算法是一致的，关键不同在于我们把每一时刻针对 W 的不同梯度做了加和。在传统神经网络中，不需要在层之间共享参数，就不需要做任何加和。在我看来，BPTT是应用于展开的RNN上的标准反向传播的另一个名字。就像反向传播一样，你也可以定义一个反向传递的 δ 向量，例如，$\delta_2^{(3)} = \frac{\partial E_3}{z2} = \frac{\partial E_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial z_2}$，其中 $z_2 = Ux_2 + Ws_1$。

这会让你明白为什么标准RNN很难训练：序列会变得很长，可能有20个词或更多，因而就需要反向传播很多层。实践中，很多人会把发现传播截断至几步。

4、梯度消失问题

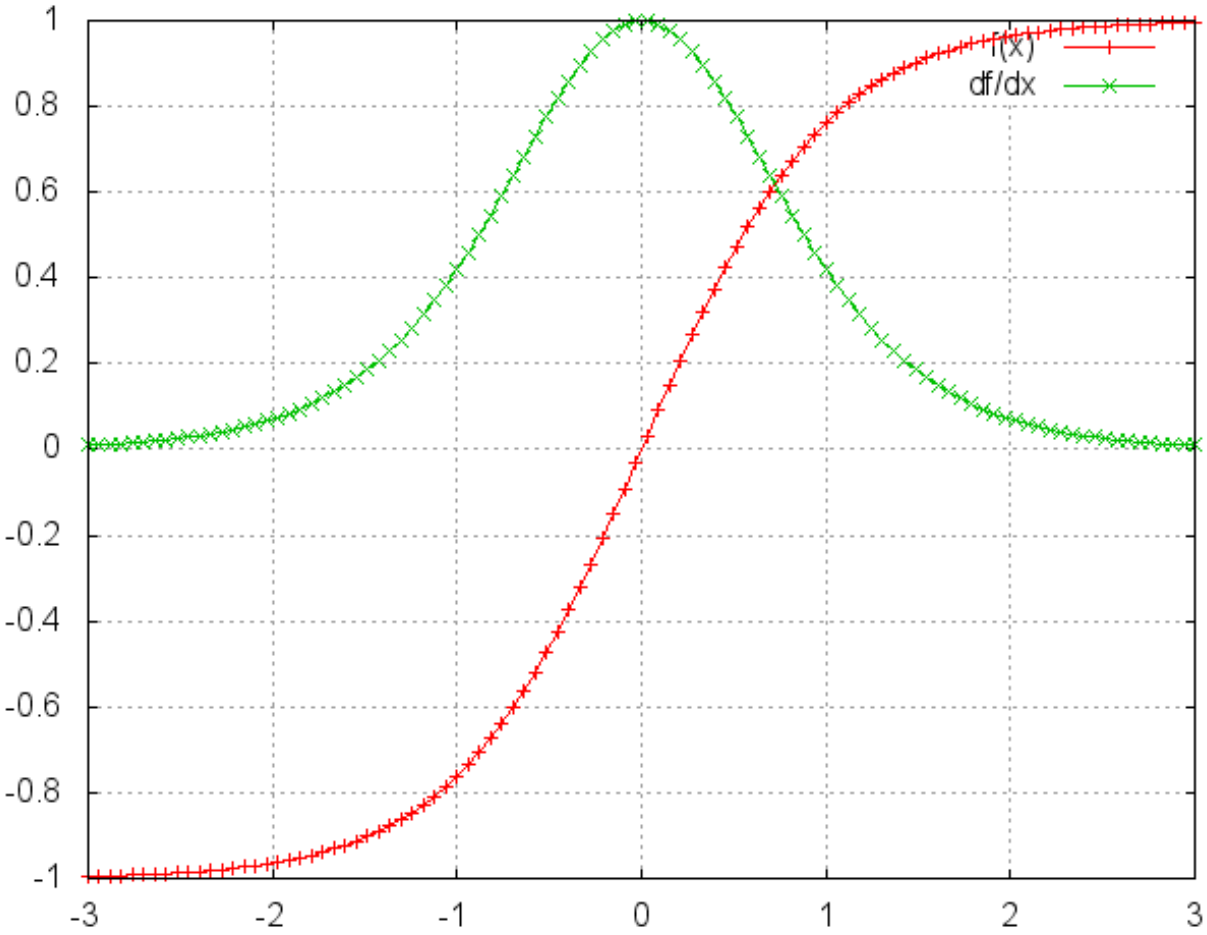
在教程前一部分，我提到RNN很难学到长范围的依赖——相隔几步的词之间的交互。这是有问题的，因为英语中句子的意思通常由相距不是很近的词来决定：“The man who wore a wig on his head went inside”。这个句子讲的是一个男人走了进去，而不是关于假发。但是普通的RNN不可能捕捉这样的信息。要理解为什么，让我们先仔细看一下上面计算的梯度：

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

注意到 $\frac{\partial s_3}{\partial s_k}$ 也需要使用链式法则，例如，$\frac{\partial s_3}{\partial s_1} = \frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}$。注意到因为我们是用向量函数对向量求导数，结果是一个矩阵（称为Jacobian Matrix），矩阵元素是每个点的导数。我们可以把上面的梯度重写成：

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\left(\prod_{j=k+1}^{3}\frac{\partial s_j}{\partial s_{j-1}}\right)\frac{\partial s_k}{\partial W}$$

可以证明上面的Jacobian矩阵的二范数（可以认为是一个绝对值）的上界是1。这很直观，因为激活函数tanh把所有制映射到-1和1之间，导数值得界限也是1：



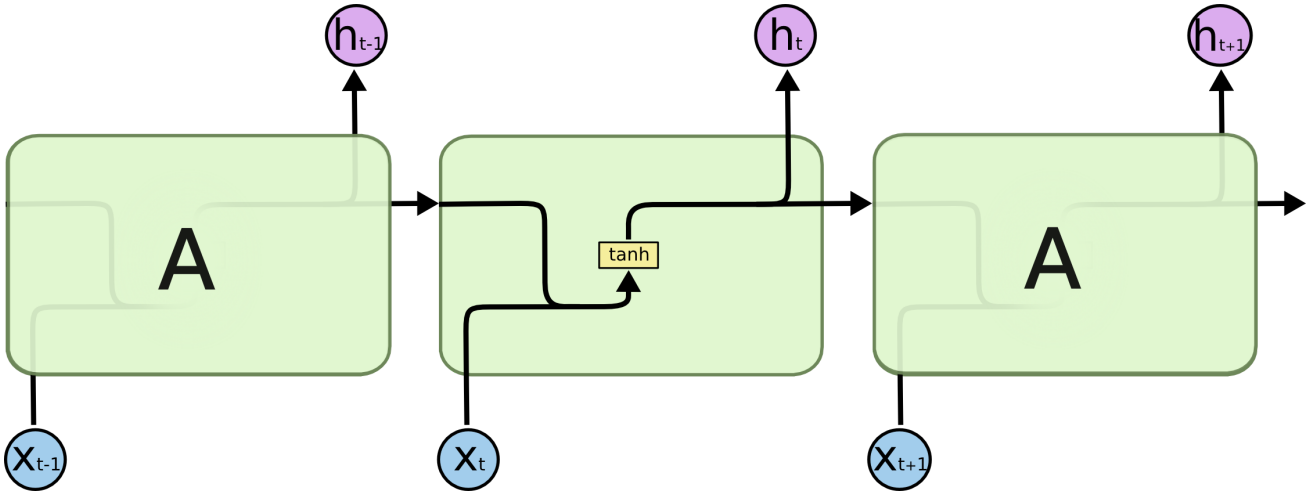
你可以看到 *tanh* 和 *sigmoid* 函数在两端的梯度值都为0，接近于平行线。当这种情况出现时，我们就认为相应的神经元饱和了。它们的梯度为**0**，使得前面层的梯度也为**0**。矩阵中存在比较小的值，多个矩阵相乘会使梯度值以指数级速度下降，最终在几步后完全消失。比较远的时刻的梯度值为**0**，这些时刻的状态对学习过程没有帮助，导致你无法学习到长距离依赖。消失梯度问题不仅出现在RNN中，同样也出现在深度前向神经网中。只是RNN通常比较深（例子中深度和句子长度一致），使得这个问题更加普遍。

很容易想到，依赖于我们的激活函数和网络参数，如果**Jacobian**矩阵中的值太大，会产生梯度爆炸而不是梯度消失问题。梯度消失比梯度爆炸受到了更多的关注有两方面的原因。其一，梯度爆炸容易发现，梯度值会变成**NaN**，导致程序崩溃。其二，用预定义的阈值裁剪梯度可以简单有效的解决梯度爆炸问题。梯度消失出现的时候不那么明显而且不好处理。有很多优化方法，例如使用 relu 激活函数，等等，但是不能从根本上解决梯度消失问题。

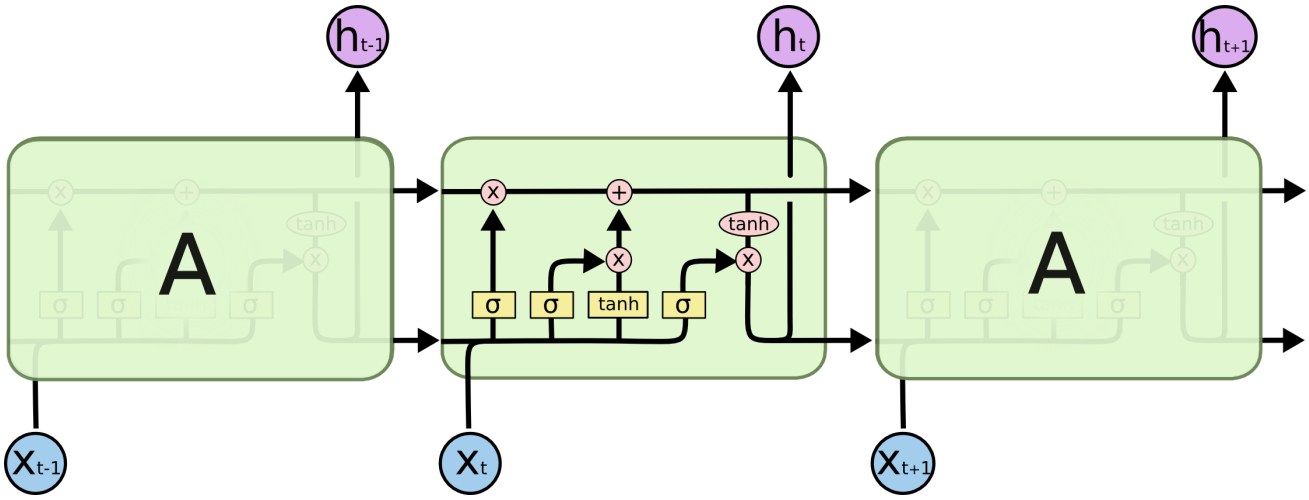
5、理解LSTM

长短期记忆网络（Long Short Term Memory networks）——通常成为『LSTMs』——是一种特殊的RNN，它能够学习长期依赖。它们由Hochreiter & Schmidhuber (1997)提出，后来由很多人加以改进和推广。他们在大量的问题上都取得了巨大成功，现在已经被广泛应用。

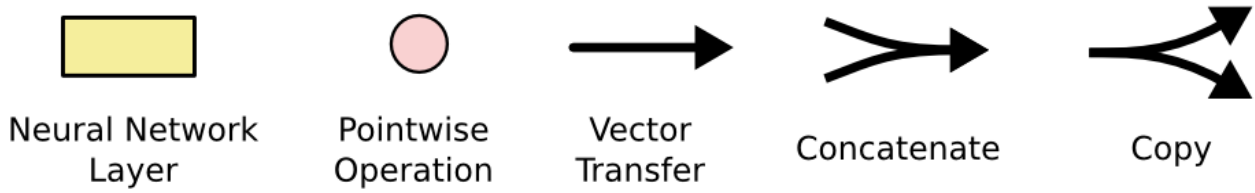
LSTMs是专门设计用来避免长期依赖问题的。记忆长期信息是**LSTMs**的默认行为，而不是它们努力学习的东西！所有的循环神经网络都具有链式的重复模块神经网络。在标准的RNNs中，这种重复模块具有非常简单的结构，比如是一个 *tanh* 层，如图所示：



LSTMs同样具有链式结构，但是其重复模块却有着不同的结构。不同于单独的神经网络层，它具有4个以特殊方式相互影响的神经网络层，如图所示：



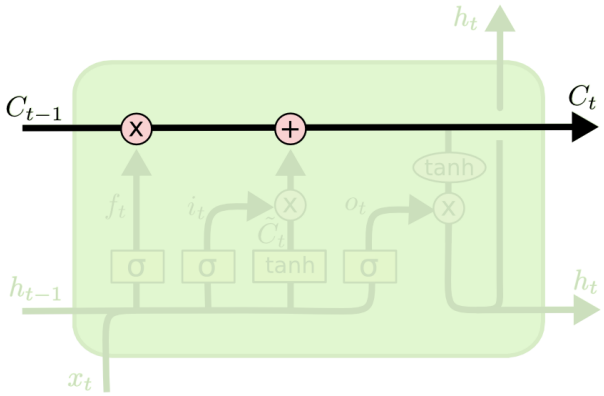
不要担心接下来涉及到的细节。我们将会一步步讲解LSTM的示意图。下面是我们将要用到的符号，如图所示：



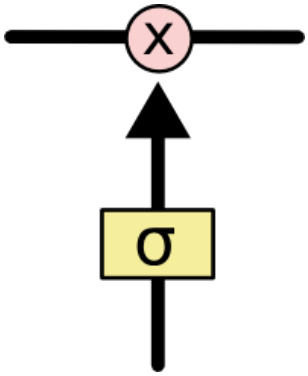
在上图中，每一条线代表一个完整的向量，从一个节点的输出到另一个节点的输入。粉红色圆形代表了逐点操作，例如向量求和；黄色方框代表学习出得神经网络层。聚拢的线代表了串联，而分叉的线代表了内容复制去了不同的地方。

5.1、LSTMs背后的核心思想

LSTMs的关键在于细胞状态，在图中以水平线表示。细胞状态就像一个传送带。它顺着整个链条从头到尾运行，中间只有少许线性的交互。信息很容易顺着它流动而保持不变，细胞状态中，包含以前所有的信息，通过加法对信息累加，避免了梯度消失。如图所示：



LSTM通过称之为门（gates）的结构来对细胞状态增加或者删除信息。门是选择性让信息通过的方式。它们的输出有一个 $sigmoid$ 层和逐点乘积操作，如图所示：

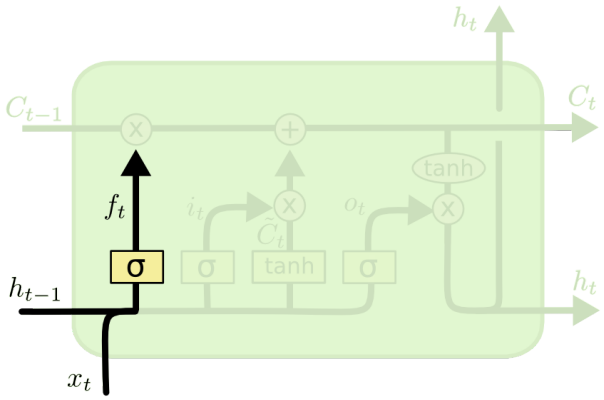


$Sigmoid$ 层的输出在0到1之间，是一个小数，定义了各成分被放行通过的程度。0值意味着『不让任何东西过去』；1值意味着『让所有东西通过』。一个LSTM具有3种门，用以保护和控制细胞状态。

5.2、逐步讲解LSTM

LSTM的第一步是决定我们要从细胞中抛弃何种信息。这个决定是由叫做『遗忘门』的 $sigmoid$ 层决定的。它以 h_{t-1} 和 x_t 为输入，输出一个介于0和1之间的数，再与 C_{t-1} 细胞作用，选择忘记一部分信息。其中1代表『完全保留』，0代表『完全遗忘』。

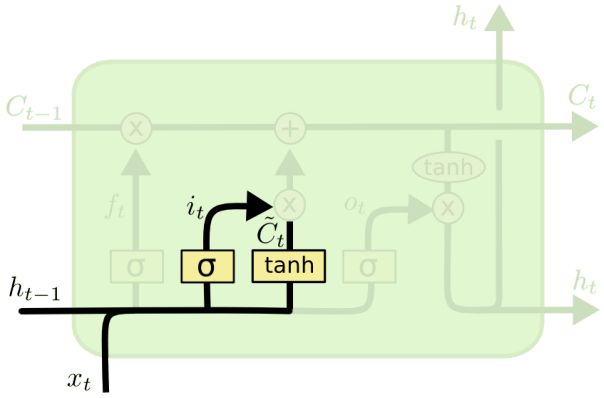
让我们回到之前那个语言预测模型的例子，这个模型尝试着根据之前的单词学习预测下一个单词。在这个问题中，细胞状态可能包括了现在主语的性别，因此能够使用正确的代词。当我们见到一个新的主语时，我们希望它能够忘记之前主语的性别。如图所示：



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

下一步是决定细胞中要存储何种信息，它有2个组成部分。首先，由一个叫做『输入门层』的 *sigmoid* 层决定我们将要更新哪些值。其次，一个 *tanh* 层创建一个新的候选向量 $\tilde{C}_t$，然后通过选择部分，选择一些向量加入到新状态中，它可以加在状态之中(也就是需要存储的新信息)。在下一步我们将结合两者来生成状态的更新。

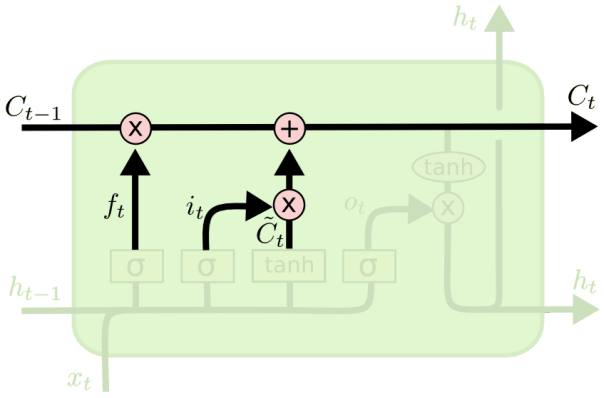
在语言模型的例子中，我们希望把新主语的性别加入到状态之中，从而取代我们打算遗忘的旧主语的性别，如图所示：



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

现在我们可以将旧细胞状态 C_{t-1} 更新为 C_t 了。之前的步骤已经决定了该怎么做，我们现在实际操作一下。我们把旧状态乘以 f_t，用以遗忘之前我们决定忘记的信息。然后我们加上 $i_t * \tilde{C}_t$，这是新的候选值，根据我们决定更新状态的程度来作为放缩系数，关键是这一个，避免了梯度消失问题，这里将梯度计算由乘法变成了加法。

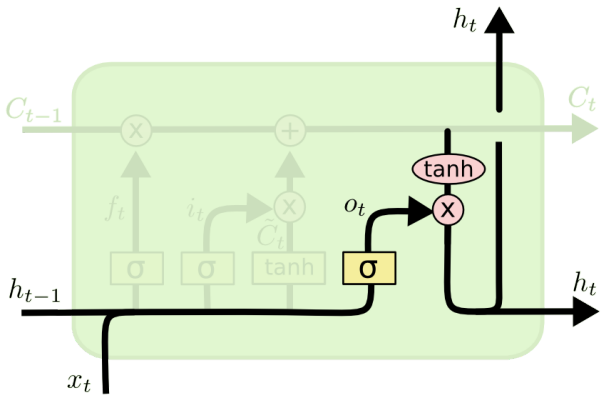
在语言模型中，这里就是我们真正丢弃关于旧主语性别信息以及增添新信息的地方，如图所示：



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

最终，我们可以决定输出哪些内容。输出取决于我们的细胞状态，但是以一个过滤后的版本(里面包括所有的历史信息)。首先，我们使用 *sigmoid* 层来决定我们要输出细胞状态的哪些部分，然后，把用

tanh 处理细胞状态（将状态值映射到-1至1之间）。最后将其与sigmoid门的输出值相乘，从而我们能够输出我们决定输出的值。如图所示：



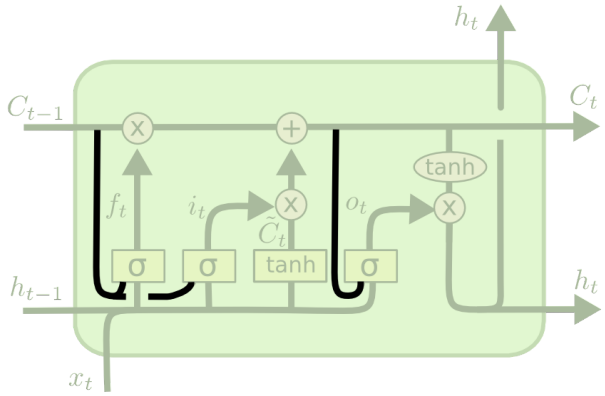
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

对于语言模型，在预测下一个单词的例子中，当它输入一个主语，它可能会希望输出相关的动词。例如，当主语是单数或复数时，它可能会以相应形式的输出。

5.3、各种LSTM的变化形式

目前我所描述的都是普通的LSTM。然而并非所有的LSTM都是一样的。事实上，似乎每一篇使用LSTMs的文章都有些细微差别。这些差别很小，但是有些值得一提。

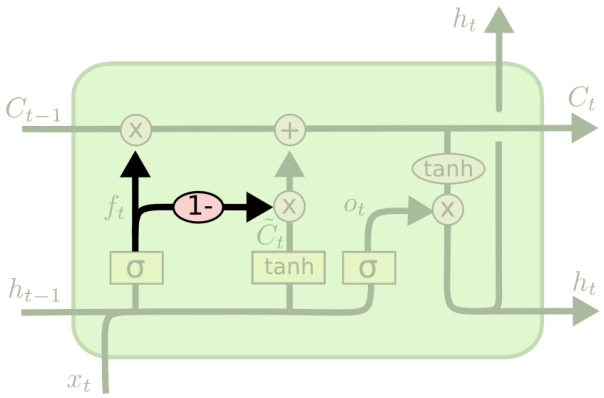
其中一个流行的LSTM变化形式是由Gers & Schmidhuber (2000)提出，增加了『窥视孔连接（peephole connections）』。如图所示：



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

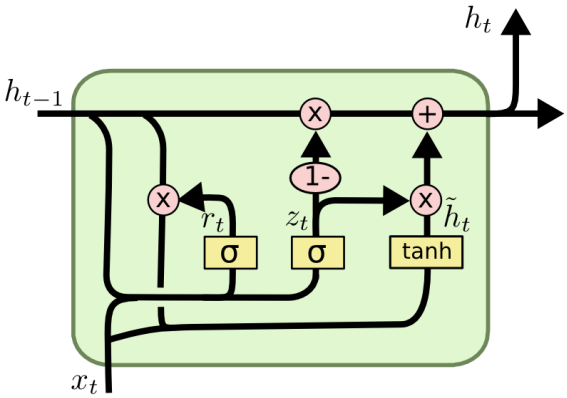
在上图中，所有的门都加上了窥视孔，但是许多论文中只在其中一些装了窥视孔。

另一个变种是使用了配对遗忘与输入门。与之前分别决定遗忘与添加信息不同，我们同时决定两者。只有当我们需要输入一些内容的时候我们才需要忘记。只有当早前信息被忘记之后我们才会输入。如图所示：



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

LSTM一个更加不错变种是 Gated Recurrent Unit（GRU），是由Cho, et al. (2014)提出的。这个模型将输入门与和遗忘门结合成了一个单独的『更新门』。而且同时还合并了细胞状态和隐含状态，同时也做了一下其他的修改。因此这个模型比标准**LSTM**模型要简单，并且越来越收到欢迎。如图所示：



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

6、LSTM与RNN比较

LSTM只能避免RNN的梯度消失（gradient vanishing）；梯度膨胀(gradient explosion)不是个严重的问题，一般靠裁剪后的优化算法即可解决，比如gradient clipping（如果梯度的范数大于某个给定值，将梯度同比收缩）。下面简单说说LSTM如何避免梯度消失.

RNN的本质是在网络内部维护了一个状态 S_t，其中 t 表示时间且 S_t 需要使用递归的方式求得:

- 传统的RNN总是用“覆写”的方式计算状态： $S_t = f(S_{t-1}, x_t)$，其中 $f(\cdot)$ 表示放射变换外面再套一个 $sigmoid$，x_t 表示输入序列在时刻 t 的值，根据求导的链式法则，这种形式直接导致梯度被表示为连乘的形式，以至于造成梯度消失，简单的来说，很多个小于1的项，连乘就很快逼近0.
- 现代的 RNN（包括但不限于LSTM单元的RNN）使用累加的形式计算状态：$S_t = \sum_{\tau=1}^{t} \Delta S_\tau$，其中的 ΔS_τ 显示依赖序列输入 x_t。稍加推导就可以发现，这种累加形式导致导数也是累加形式，避免了梯度消失。

于是简化来看，加法操作可以认为是两组信息的叠加，乘法操作可以看做一组控制信号对另一组信息的控制。