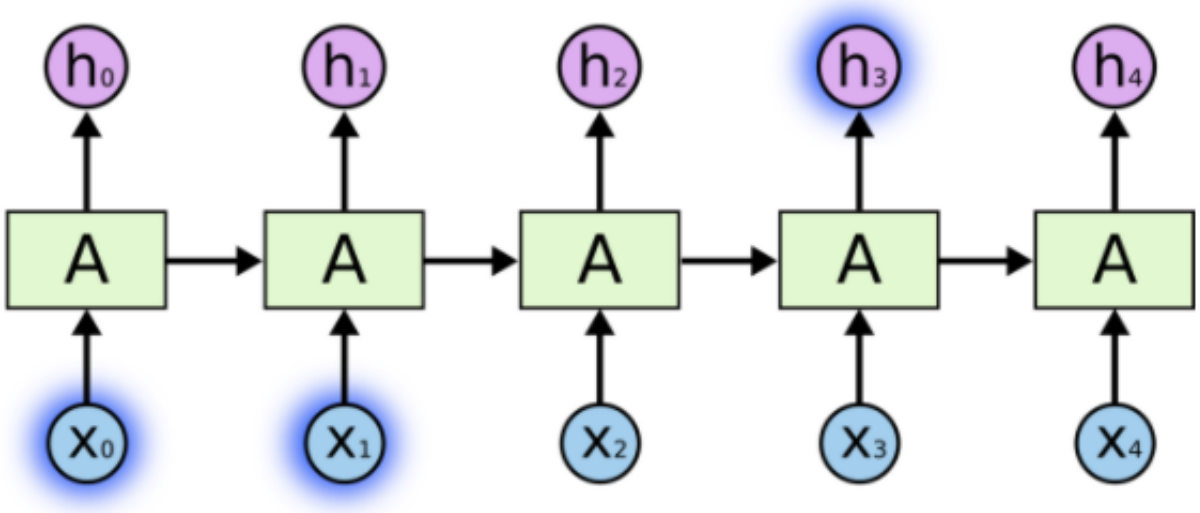
深度学习系列（5）：长短时记忆网络（LSTM）

一、长期依赖问题（Long-Term Dependencies）

循环神经网络（RNN）在实际应用中很难处理长距离依赖的问题。

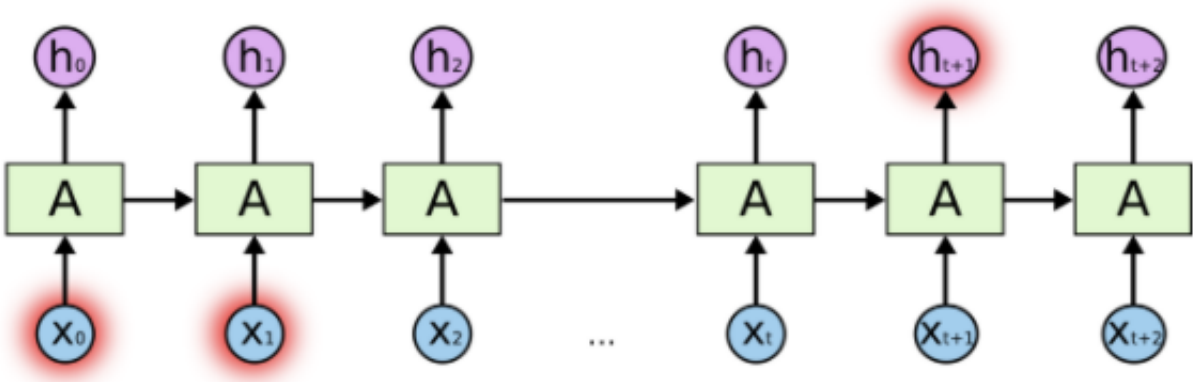
有的时候，我们仅仅需要知道先前的信息来完成预测任务。例如，我们有一个语言模型用来基于先前的词来预测下一个词，比如我们预测“the clouds are in the sky”最后的词的时候，我们不需要任何其他的上下文，很显然下一个词就是sky。在这种情况下，相关的信息与需要预测的词位置之间的间隔很小，而RNN可以学会使用较近距离的信息。



不太长的相关信息和位置间隔

但是到了一个更加复杂的场景，假设我们试着预测“I grew up in France.....I speak fluent French”中最后的词，从这句话的信息来看，下一个词很有可能是一种语言的名字，但具体到是哪种语言，我们就需要在与之距离较远的“I grew up in France”中得到。这说明相关信息与当前预测位置之间的间隔就肯定变得相当的大。

不幸的是，在这个间隔不断增大时，RNN会丧失学习到连接如此远的信息的能力。



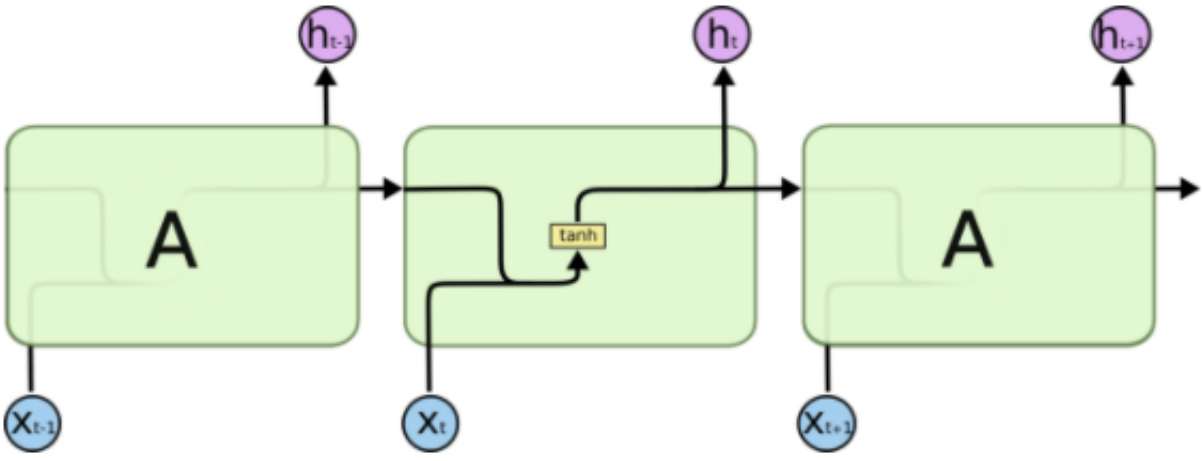
相当长的相关信息和位置间隔

当然，在理论上，RNN绝对可以处理这样的长期依赖问题。人们可以通过调参来解决，但是在实践中，RNN肯定不能够成功学习到这些知识。Bengio, et al. (1994)等人对该问题进行了深入的研究，它们发现一些使训练RNN变得非常困难的相当根本的原因。

既然找到了问题的原因，那我们就能解决它。从问题的定位到解决，科学家们大概花了7、8年的时间。终于有一天，Hochreiter和Schmidhuber两位科学家发明出长短时记忆网络，一举解决了这个问题。

二、LSTM的核心思想

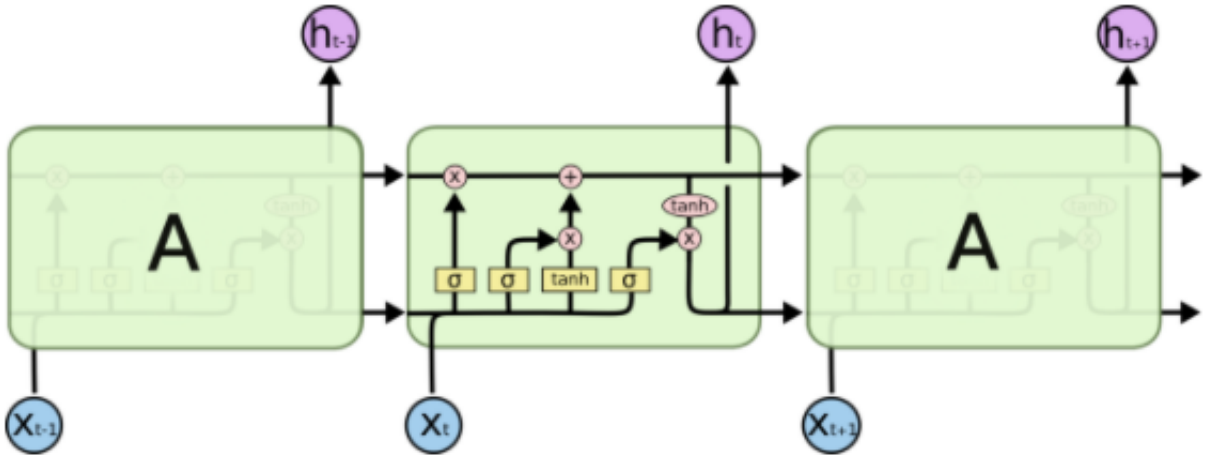
Long Short Term网络，一般就叫做LSTM，是一种特殊的RNN变体，它可以学习长期依赖信息。LSTM由Hochreiter和Schmidhuber在1997年提出，并在近期被Alex Graves进行了改良和推广。在很多问题上，LSTM都取得了相当巨大的成功，并得到了广泛的使用。
LSTM通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是LSTM的默认属性，而非需要付出很大的代价才能获得的能力！
所有的RNN都具有一种重复神经网络模块的链式的形式。在标准的RNN中，这个重复的模块只有一个非常简单的结构，例如一个tanh层。



标准 RNN 中的重复模块包含单一的层

LSTM同样是这样的结构，但是其中重复的模块拥有一个不同的结构。不同于单一神经网络层，

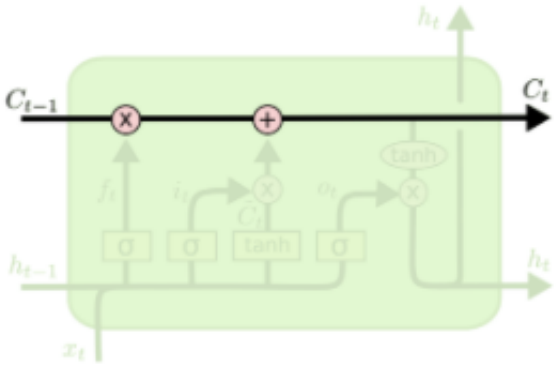
这里有四个以非常特殊的方式进行交互的小器件。



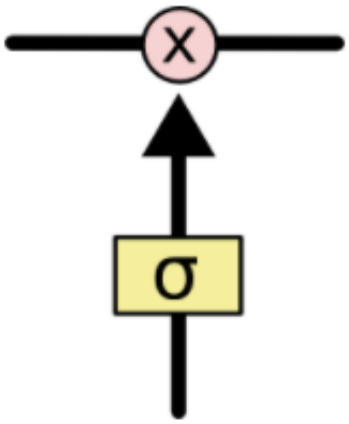
LSTM 中的重复模块包含四个交互的层

图中每一条黑线传输着一整个向量，从一个节点的输出到其他节点的输入。粉色的圈代表pointwise的操作，比如向量的和，而黄色的矩阵就是学习到的神经网络层。

LSTM的关键在于细胞（Cell），水平线在细胞内贯穿运行。细胞类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在水平线上很容易保持不变。



LSTM通过精心设计“门”结构来去除或者增加信息到Cell上。门是一种让信息选择式通过的方法（过滤器）。它们包含一个sigmoid神经网络层和一个pointwise乘法操作。



Sigmoid层输出0到1之间的数值，描述每个部分有多少量可以通过。0代表“不许任何量通过”，1就指“允许任意量通过”

三、LSTM的前向计算

LSTM用两个门来控制单元状态Cell的内容，一个是遗忘门（forget gate），它决定了上一时刻的单元状态$c_t - 1$有多少保留到当前时刻c_t；另一个是输入门（input gate），他决定了当前时刻网络的输入x_t有多少保存到单元状态c_t。LSTM用输出门（output gate）来控制单元状态c_t有多少输出到LSTM的当前输出值h_t。

3.1 遗忘门

我们先看一下遗忘门：

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

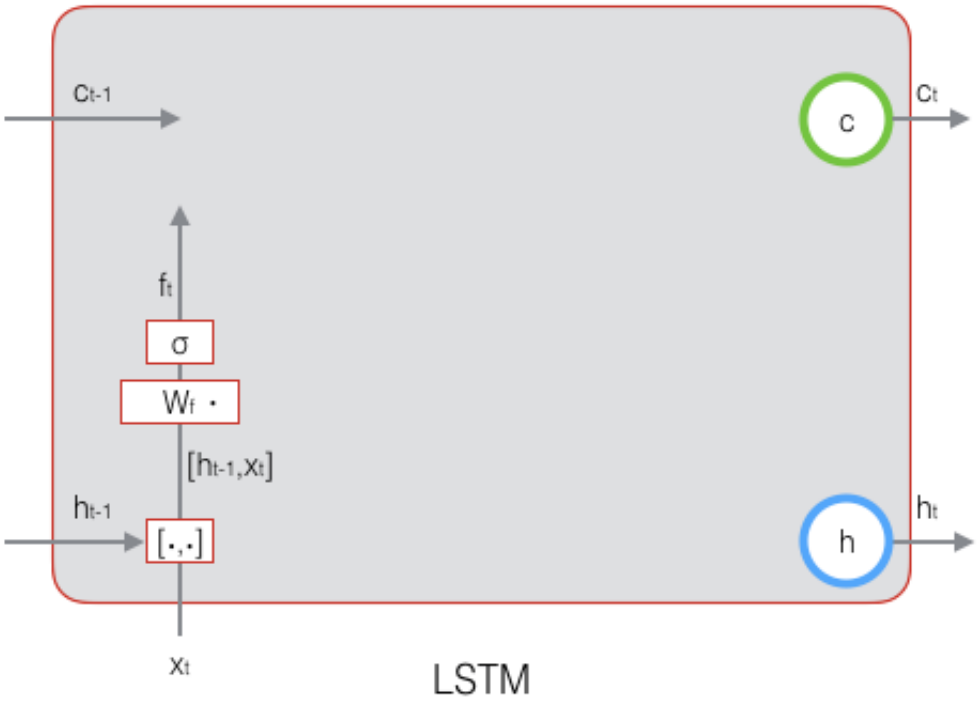
上式中，W_f是遗忘门的权重矩阵，$[h_{t-1}, x_t]$表示把两个向量连接成一个更长的向量，b_f是遗忘门的偏置项，σ是sigmoid函数。若输入的维度是d_x，隐藏层的维度是d_h，单元状态的维度是d_c（通常$d_c = d_h$），则遗忘门的权重矩阵W_f维度是$d_c \times (d_h + d_x)$。事实上，权重矩阵W_f都是两个矩阵拼接而成的：一个是W_{fh}，它对应着输入项h_{t-1}，其维度为$d_c \times d_h$；一个是W_{fx}，它对应着输入项x_t，其维度为$d_c \times d_x$。W_f可以写为：

$$\left[W_f\right] \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} = \begin{bmatrix} W_{fh} & W_{fx} \end{bmatrix} \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} = W_{fh} \cdot h_{t-1} + W_{fx} x_t$$

所以总结一下，遗忘门的作用为控制有多少上一时刻的memory cell中的信息可以累积到当前时刻的memory cell中。其数学公式可以写作：

$$f_t = sigmoid(W_{fx} \cdot x_t + W_{fh} \cdot h_{t-1} + b_i)$$

其计算图示如下：

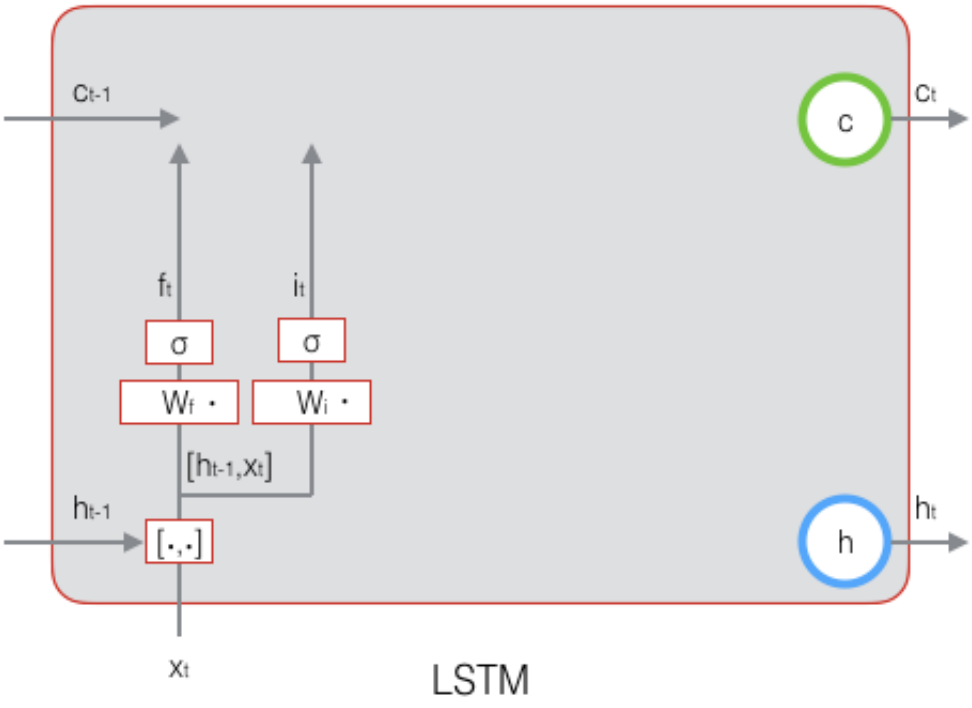


3.2 输入门

接下来看输入门：

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

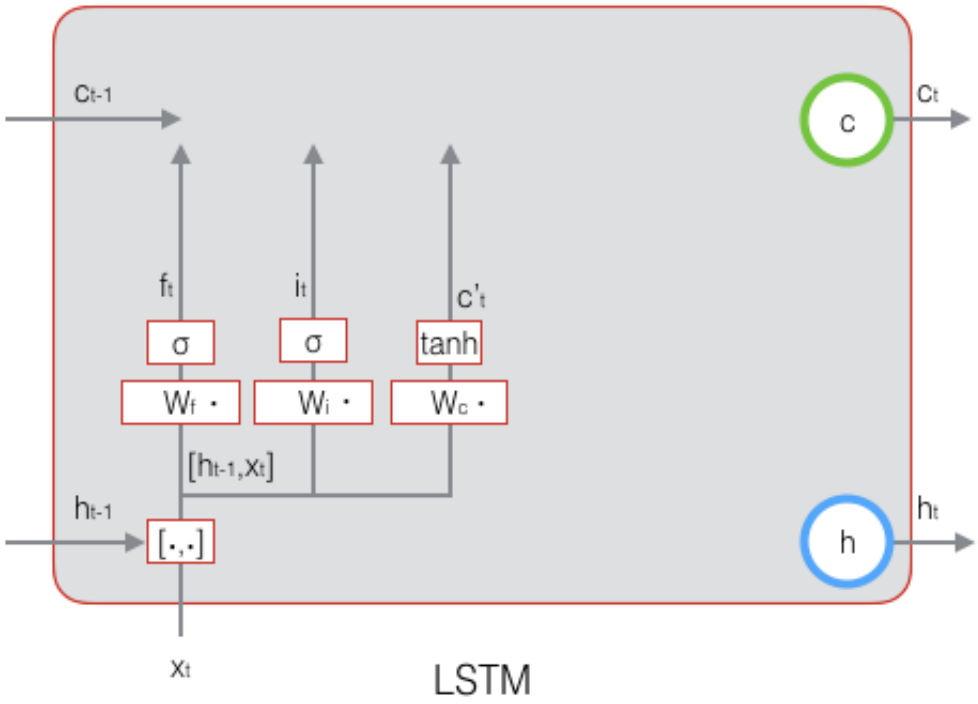
上式中，W_i是输入们的权重矩阵，b_i是输入门的偏置项。下图表示了输入门的计算：



接下来，我们计算用于描述当前输入的单元状态$\tilde{c}_t$，它是根据上一次的输出和本次输入来计算的：

$$\tilde{c}_t = \tan \mathrm{h}\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$

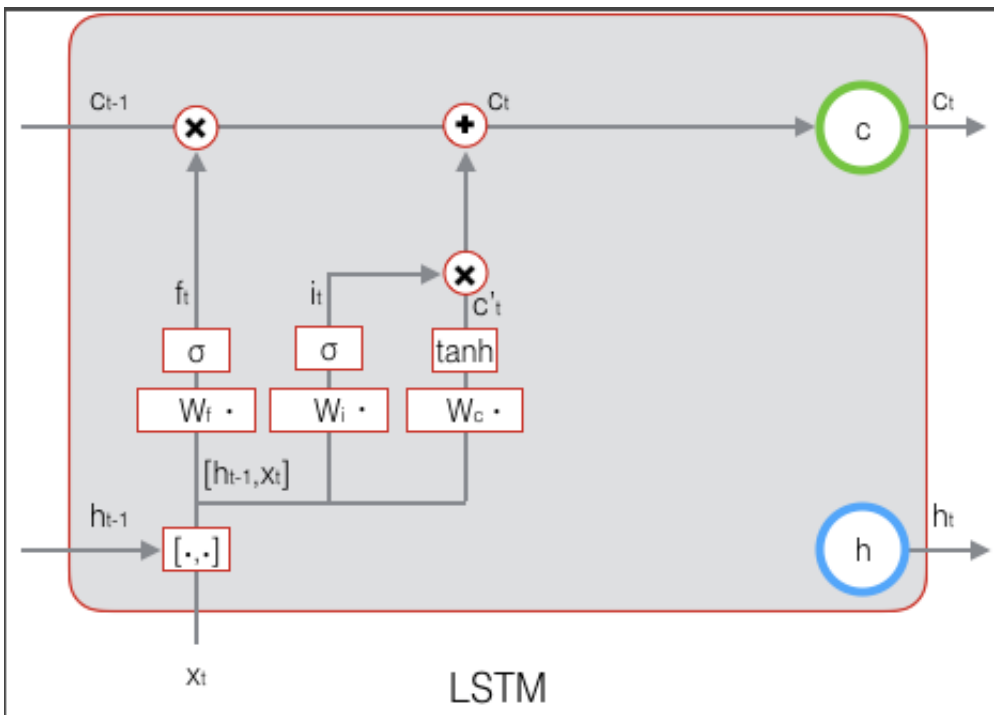
下图是$\tilde{c}_t$的计算：



现在，我们计算当前时刻的单元状态c_t。它是由上一次的单元状态c_{t-1}按元素乘以遗忘门f_t，再用当前输入的单元状态$\tilde{c}_t$按元素乘以输入门i_t，再将两个积加和产生的：

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

下图是c_t的计算图示：



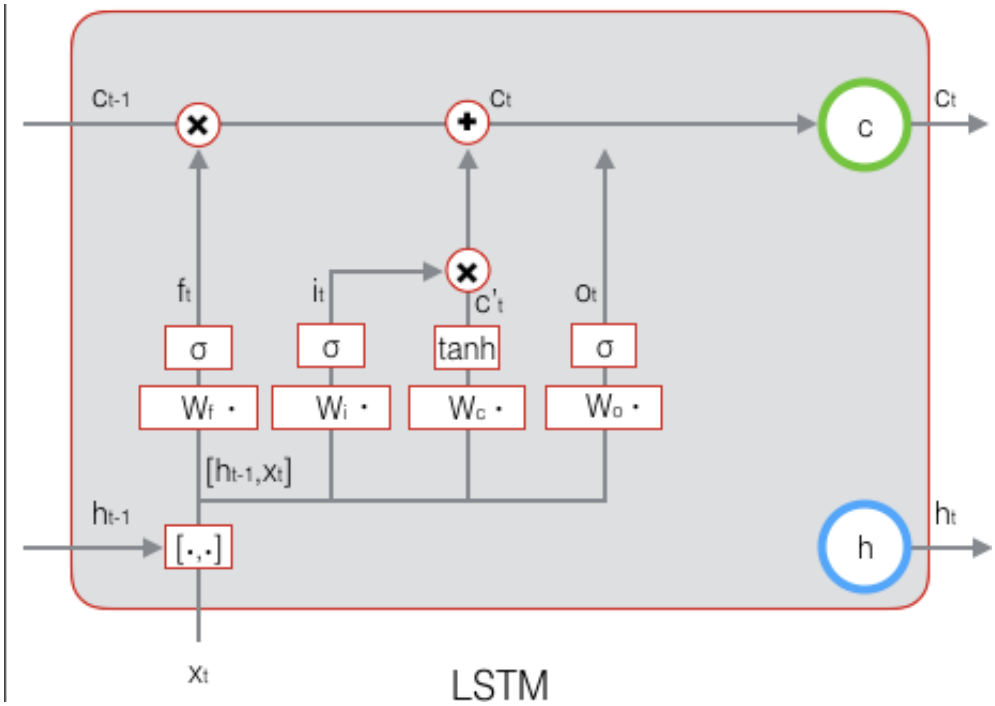
这样，我们就把LSTM关于当前的记忆$\tilde{c}_t$和长期的记忆c_{t-1}组合在一起，形成了新的单元状态c_t。由于遗忘门的控制，它可以保存很久很久之前的信息，由于输入门的控制，它又可以避免当前无关紧要的内容进入记忆。

3.3 输出门

下面，我们要看看输入们，它控制了长期记忆对当前输出的影响：

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

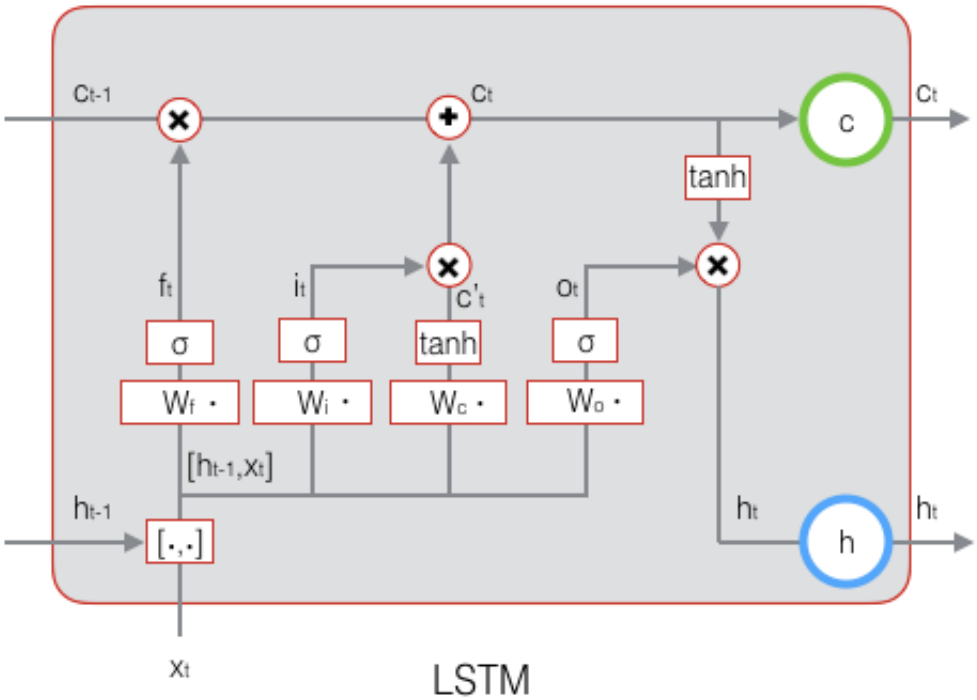
下图表示输出门的计算：



LSTM最终的输出，是由输出门和单元状态共同确定的：

$$h_t = o_t \circ \tan \mathrm{h}\,(c_t)$$

下图表示LSTM最终输出的计算：



四、LSTM的训练

LSTM的训练算法仍然是反向传播算法，它主要有下面三个步骤：

- 1）前向计算每个神经元的输出值，对于LSTM来说，即f_t、i_t、c_t、o_t、h_t五个向量的值。
- 2）反向计算每个神经元的误差项δ值。与循环神经网络一样，LSTM误差项的反向传播也是包括两个方向：一个是沿着时间的反向传播，即从当前t时刻开始，计算每个时刻的误差项；一个是将误差项向上一层传播。
- 3）根据相应的误差项，计算每个权重的梯度。

首先，我们队推导中用到的一些公式、符号做一下必要的说明。

接下来的推导中，我们设定gate的激活函数为sigmoid函数，输出的激活函数为tanh函数。它们的导数分别为：

$$\sigma(z) = y = \frac{1}{1+e^{-z}}$$

$$\sigma'(z) = y(1-y)$$

$$\tanh(z) = y = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

$$\tanh'(z) = 1-y^2$$

从上面可以看出，sigmoid和tanh函数的导数都是原函数的函数。这样，我们一旦计算原函数的值，就可以用它来计算出导数的值。

LSTM需要学习的参数共有8组，分别是：遗忘门的权重矩阵W_f和偏置项b_f、输入门的权重矩阵W_i和偏置项b_i、输出门的权重矩阵W_o和偏置项b_o，以及计算单元状态的权重矩阵W_c和偏置项b_c，因为权重矩阵的两部分在反向传播中使用不同的公式，因此在后续的推导中，权重矩阵W_f、W_i、W_c、W_o都会被写成分开的两个矩阵：
W_{fh}、W_{fx}、W_{ih}、W_{ix}、W_{oh}、W_{ox}、W_{ch}、W_{cx}。

我们解释一下按元素乘o符号。当o作用于两个向量时，运算如下：

$$a \circ b = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_n \end{bmatrix} \circ \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ \cdots \\ a_n b_n \end{bmatrix}$$

当o作用于一个向量和一个矩阵时，运算如下：

$$a \circ X = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_n \end{bmatrix} \circ \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} = \begin{bmatrix} a_1 x_{11} & a_1 x_{12} & \cdots & a_{1n} x_{1n} \\ a_2 x_{21} & a_2 x_{22} & \cdots & a_2 x_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_n x_{n1} & a_n x_{n2} & \cdots & a_n x_{nn} \end{bmatrix}$$

当o作用于两个矩阵时，两个矩阵对应位置的元素相乘。按元素乘可以再某些情况下简化矩阵和向量的运算。例如，当一个对角矩阵右乘一个矩阵时，相当于用对角矩阵的对角线组成的向量按元素乘那个矩阵：$diag[a] \cdot X = a \circ X$当一个行向量右乘一个对角矩阵时，相当于这个行向量按元素乘那个矩阵对角线组成的向量：

$$a^T \cdot diag[b] = a \circ b$$

上面这俩点，在后续推导中会多次用到。

在t时刻，LSTM的输出值为h_t。我们定义t时刻的误差项δ_t为：

$$\delta_t = \frac{\partial E}{\partial h_t}$$

注意，这里假设误差项是损失函数对输出值的导数，而不是对加权输入net^l的导数。因为LSTM有四个加权输入，分别对应f_t、i_t、c_t、o_t，我们希望往上一层传递一个误差项而不是四个。但我们仍然要定义出这四个加权输入，以及他们对应的误差项。

$$\$net_{f,t} = W_f[h_{t-1}, x_t] + b_f = W_{fh} h_{t-1} + W_{fx} x_t + b_f$$

$$net_{i,t} = W_i[h_{t-1}, x_t] + b_i = W_{ih} h_{t-1} + W_{ix} x_t + b_i$$

$$net_{\tilde{c},t} = W_c\,[h_{t-1}, x_t] + b_c = W_{ch} h_{t-1} + W_{cx} x_t + b_c$$

$$net_{o,t} = W_o\,[h_{t-1}, x_t] + b_o = W_{oh} h_{t-1} + W_{ox} x_t + b_o$$

$$\delta_{f,t} = \frac{\partial E}{\partial net_{f,t}}$$

$$\delta_{i,t} = \frac{\partial E}{\partial net_{i,t}}$$

$$\delta_{\tilde{c},t} = \frac{\partial E}{\partial net_{c,t}}$$

$$\delta_{o,t} = \frac{\partial E}{\partial net_{o,t}}$$

4.1 误差项沿时间的反向传播

沿时间反向传导误差项，就是要计算出$t-1$时刻的误差项δ_{t-1}。

$$\delta_{t-1}^T = \frac{\partial E}{\partial h_{t-1}}$$

$$= \frac{\partial E}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}$$

$$= \delta_t^T \frac{\partial h_t}{\partial h_{t-1}}$$

我们知道，$\frac{\partial h_t}{\partial h_{t-1}}$是一个jacobian矩阵。如果隐藏层$h$的维度是N的话，那么它就是一个$N \times N$矩阵。为了求出它，我们列出$h_t$的计算公式：

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tan \mathrm{h}\,(c_t)$$

显然，o_t、f_t、i_t、$\tilde{c}_t$都是h_{t-1}的函数，那么，利用全导数公式可得：

4.2 将误差项传递到上一层

4.3 权重梯度的计算

五、LSTM的变体—GRU（Gated Recurrent Unit）

前面我们讲了一种最为普通的LSTM，事实上LSTM存在很多变体，许多论文中的LSTM都或多或少的不太一样。只要遵守几个关键点，就可以根据需求设计需要的Gated RNNS。在众多的LSTM变体中，GRU也许是最成功的一种。它对LSTM做了很多简化，同时却保持着和LSTM相同的效果。因此，GRU最近变得越来越流行。

GRU对LSTM做了两个大改动：

- 1）将输入门、遗忘门、输出门变为两个门：更新门（Update Gate）z_t和重置门（Reset Gate）r_t。
- 2）将单元状态与输出合并为一个状态：h

GRU的前向计算公式为：

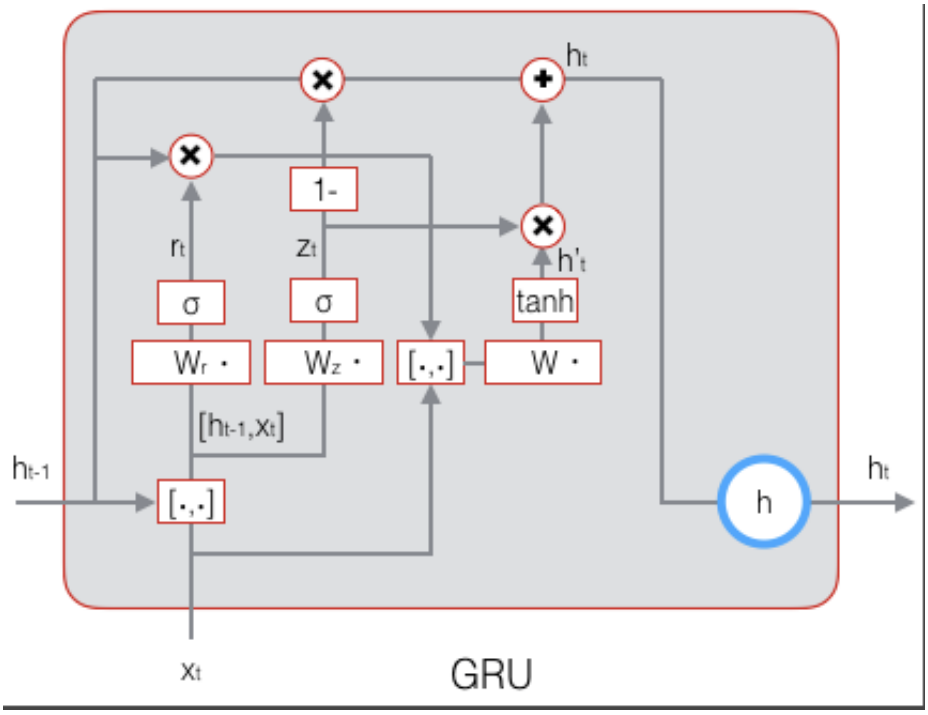
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tan h\,(W \cdot [r_t \circ h_{t-1}, x_t])$$

$$h = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

下图是GRU的示意图：



GRU的训练算法比LSTM相对也要简单一些

当然还有很多其他的变体，如 Gers & Schmidhuber (2000) 提出的LSTM变体增加了“peephole connection”；另一种变体使用coupled 遗忘和输入门对遗忘和需要的信息一同做出决定。Yao, et al. (2015) 提出的Depth Gated RNN。还有用一些完全不同的观点来解决长期依赖的问题，如Koutnik, et al. (2014) 提出的Clockwork RNN。

但Greff, et al. (2015)给出了流行变体的比较，结论是它们基本上是一样的。Jozefowicz, et al. (2015) 则在超过一万种RNN架构上进行了测试，发现一些架构在某些任务上也取得了比LSTM更好的结果。