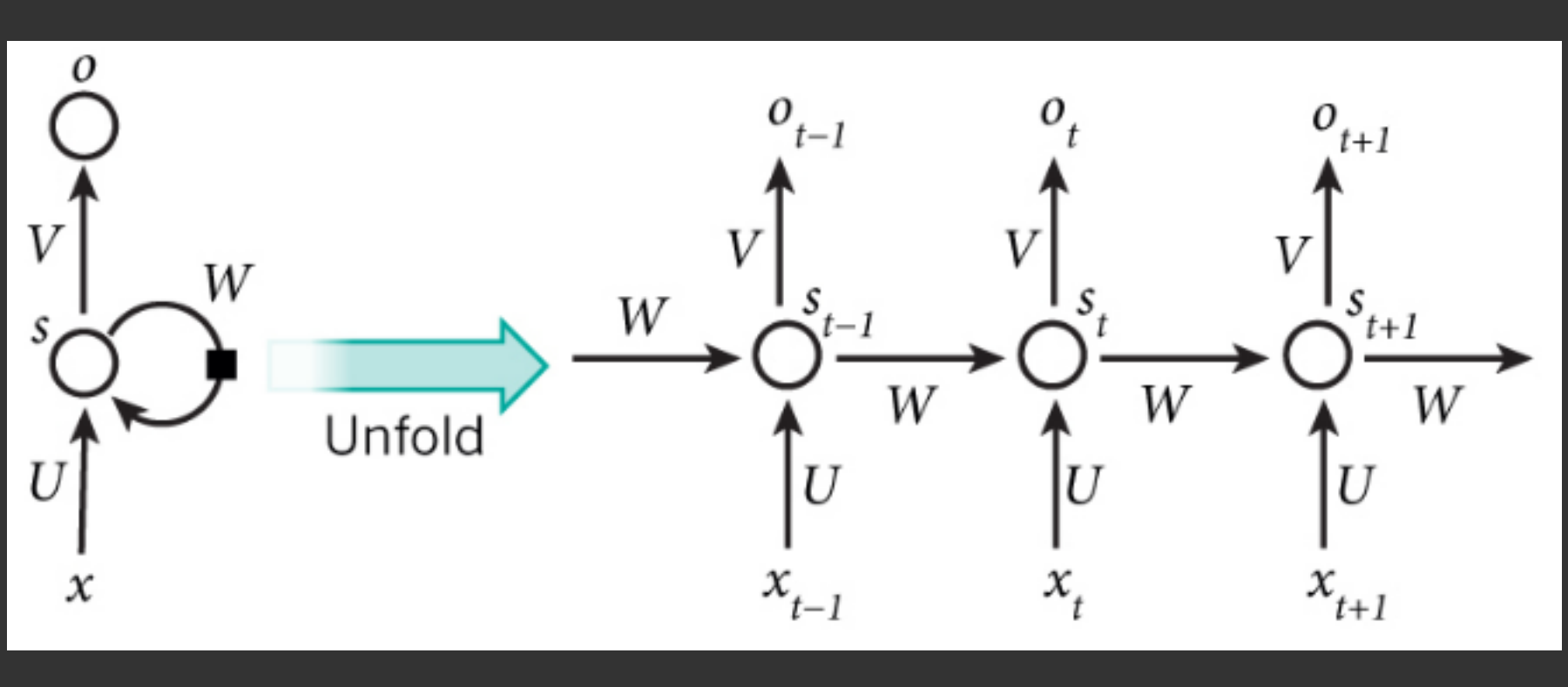
RNN

1. 什么是RNNs

RNNs的目的使用来处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNNs之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。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。理论上，RNNs能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关，下图便是一个典型的RNNs：



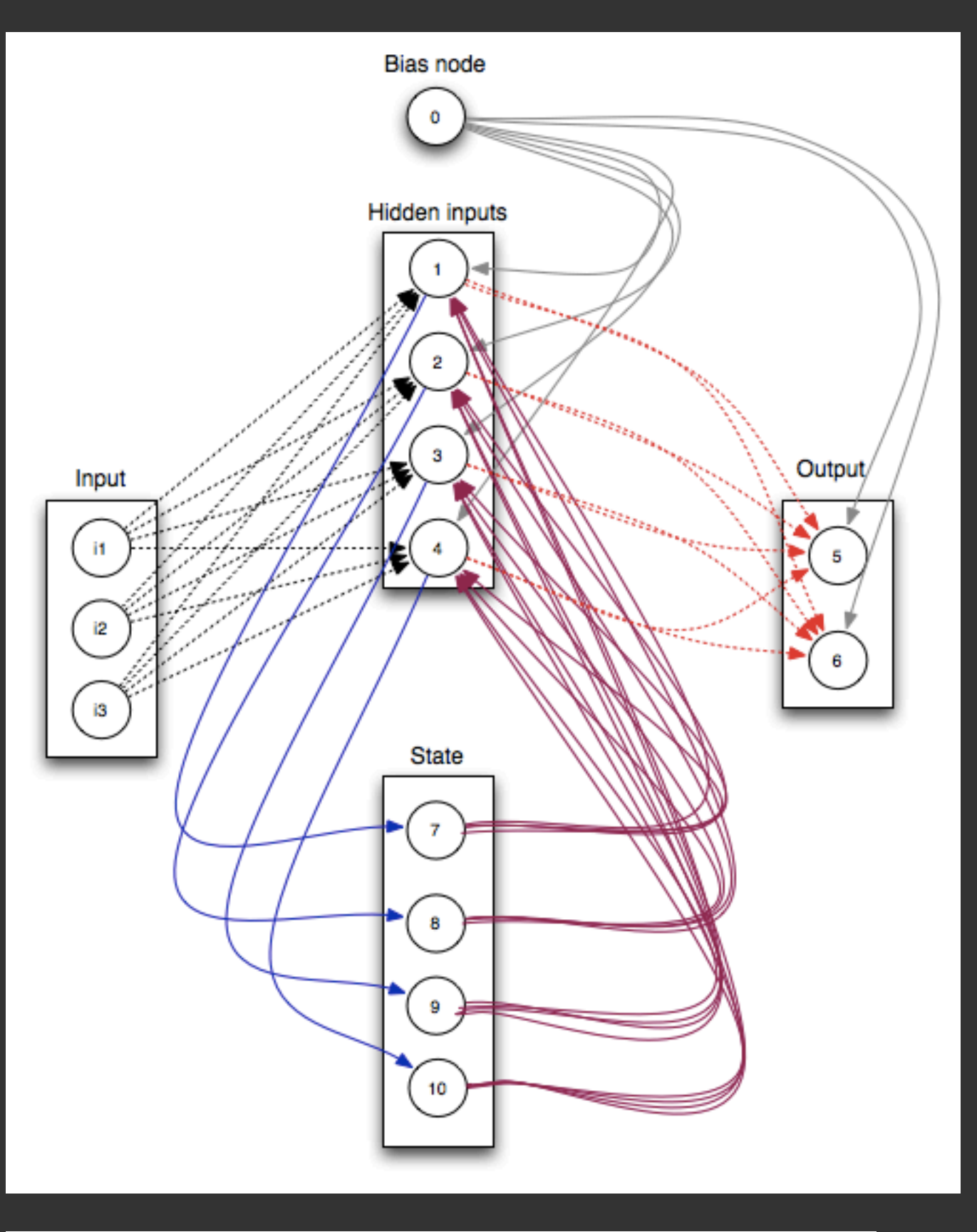
在图中：有一条单向流动的信息流是从输入单元到达隐藏单元的，与此同时另一条单向流动的信息流从隐藏单元到达输出单元。在某些情况下，**RNNs会打破后者的限制，引导信息从输出单元返回隐藏单元，这些被称为“Back Projections”，并且隐藏层的输入还包括上一隐藏层的状态，**即隐藏层内的节点可以自连也可以互连。（这实际上就是LSTM）

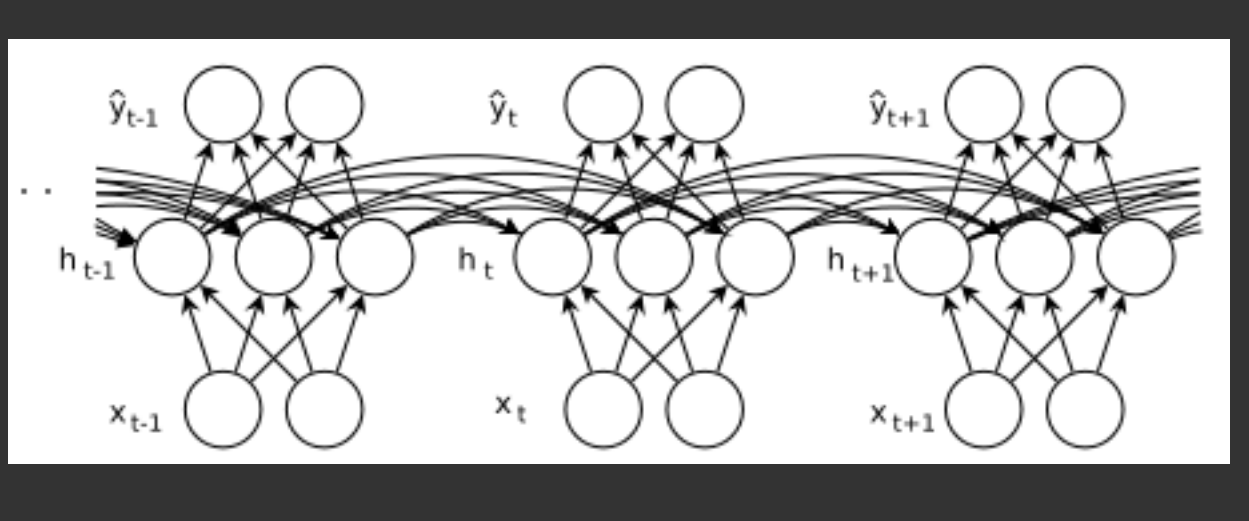
右侧为计算时便于理解记忆而产开的结构。简单说，x为输入层，o为输出层，s为隐含层，而t指第几次的计算；V,W,U为权重，其中计算第t次的隐含层状态时为：

$$St=f(U*Xt+W*St-1)$$St=f(U∗Xt+W∗St−1)

实现当前输入结果与之前的计算挂钩的目的。对RNN想要更深入的了解可以戳这里。

表达得更直观的图有：





按照上图所示，可知道RNN网络前向传播过程中满足下面的公式（参考文献Learning Recurrent Neural Networks with Hessian-Free Optimization）：

$$\begin{aligned} t_i &= W_{hx}x_i + W_{hh}h_{i-1} + b_h \\ h_i &= e(t_i) \\ s_i &= W_{yh}h_i + b_y \\ \hat{y}_i &= g(s_i) \end{aligned}$$

其代价函数可以是重构的误差：

$$\sum_i \|\hat{y}_i - y_i\|^2/2$$

也可以是交叉熵：

$$-\sum_i \sum_j y_{ij}\log(\hat{y}_{ij})+(1-y_{ij})\log(1-\hat{y}_{ij})$$

2. RNN用途

RNNs已经被在实践中证明对NLP是非常成功的。如词向量表达、语句合法性检查、词性标注等。在RNNs中，目前使用最广泛最成功的模型便是LSTMs(Long Short-Term Memory，长短时记忆模型)模型，该模型通常比vanilla RNNs能够更好地对长短时依赖进行表达，该模型相对于一般的RNNs，只是在隐藏层做了手脚。对于LSTMs，后面会进行详细地介绍。RNNs在NLP中的应用有：

语言模型与文本生成(Language Modeling and Generating Text)，机器翻译(Machine Translation)，语音识别(Speech Recognition)，图像描述生成 (Generating Image Descriptions)。

3.RNN劣势

由于RNN模型如果需要实现长期记忆的话需要将当前的隐含态的计算与前n次的计算挂钩，即St = f(U*Xt + W1*St-1 + W2*St-2 + … + Wn*St-n)，那样的话计算量会呈指数式增长，导致模型训练的时间大幅增加，因此RNN模型一般不直接用来进行长期记忆计算。另外，传统RNN处理不了**长期依赖**问题，这是个致命伤。但之后的LSTM就解决了这问题。

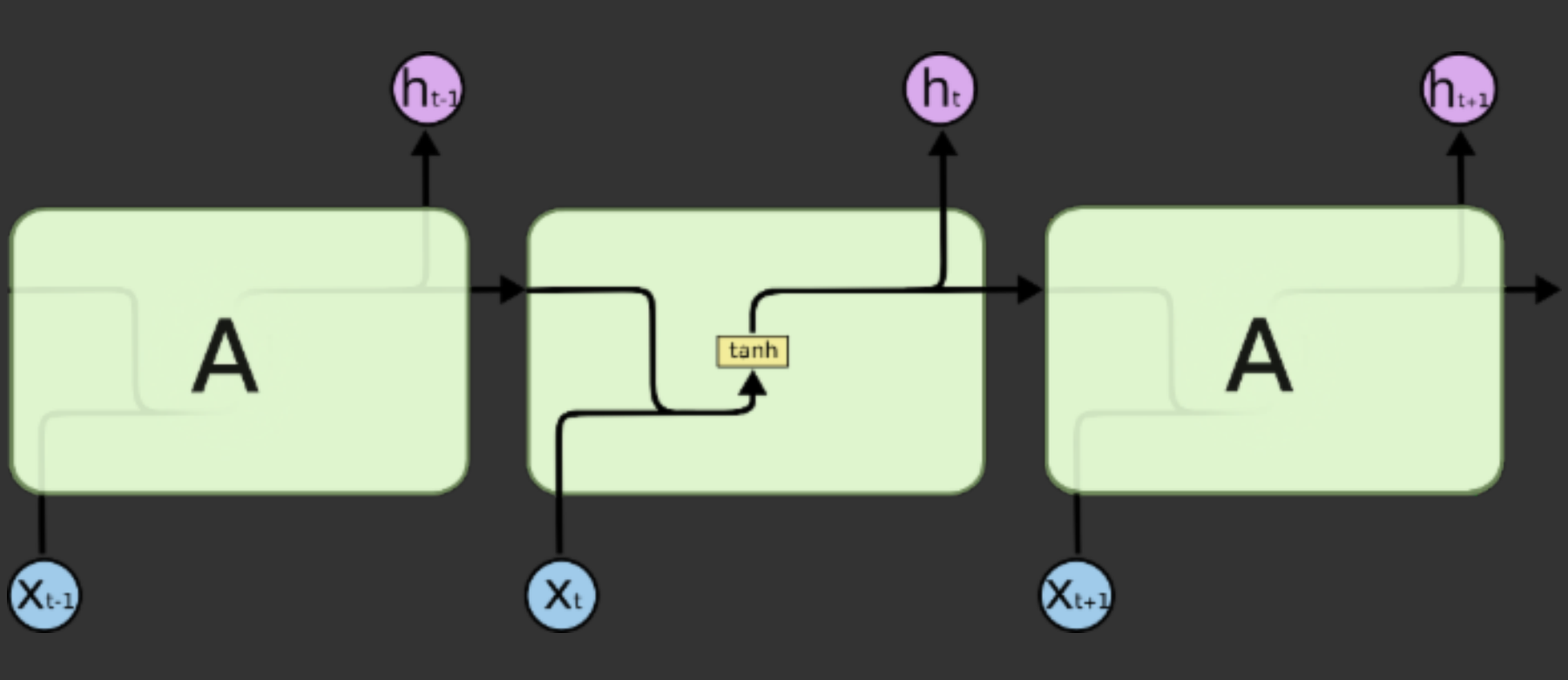
LSTM

1. LSTM是什么

Long Short Term 网络—— 一般就叫做 LSTM ——是一种 RNN 特殊的类型，可以学习长期依赖信息。LSTM 由 Hochreiter & Schmidhuber (1997) 提出，并在近期被 Alex Graves 进行了改良和推广。在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛的使用。

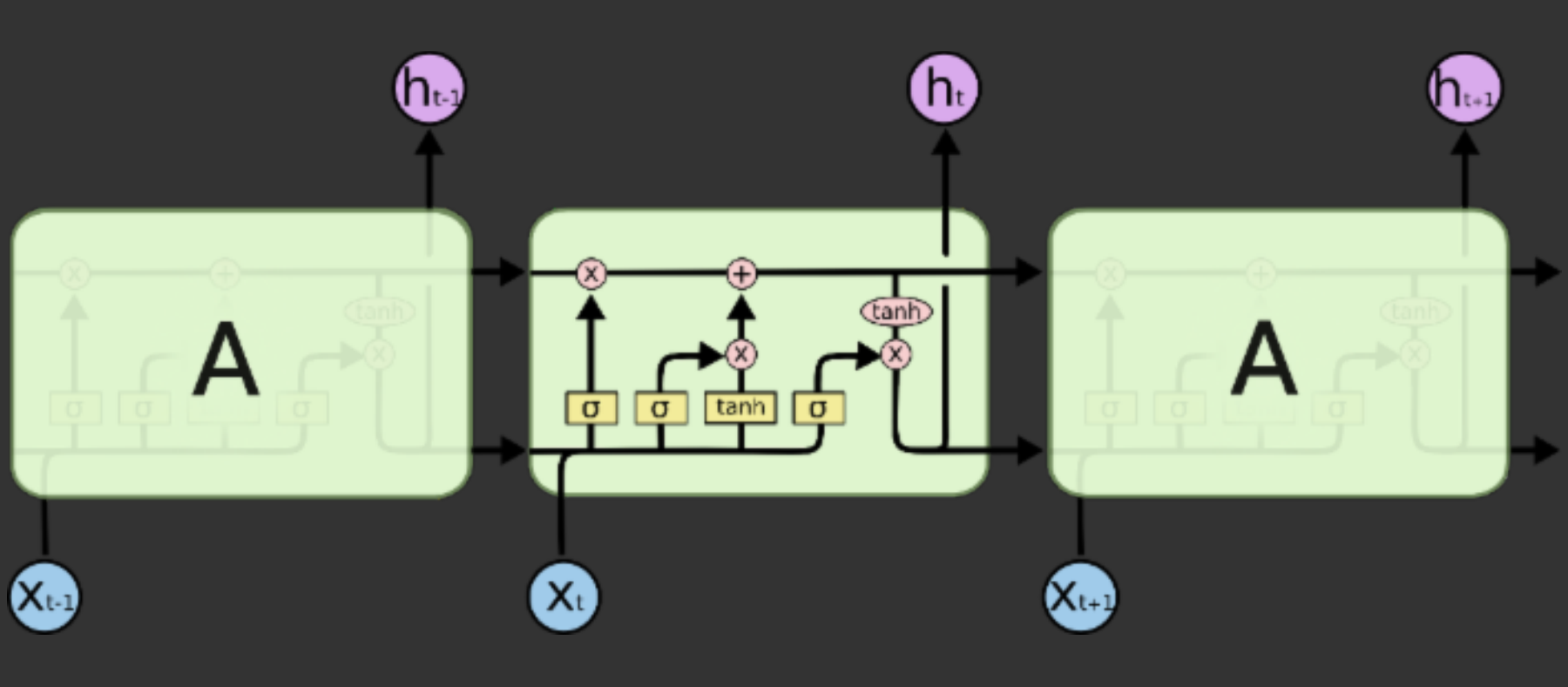
LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力！

所有 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。



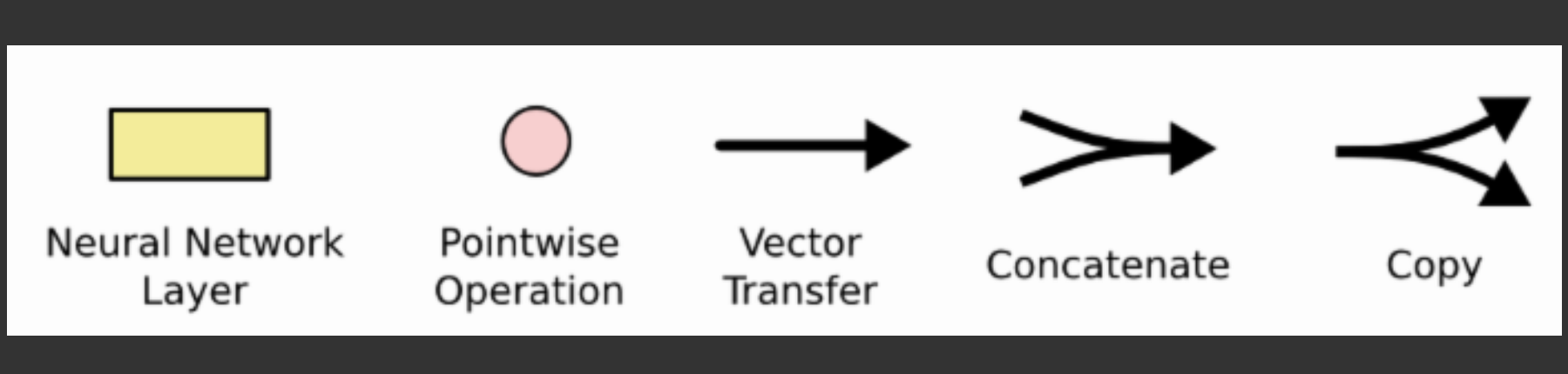
标准 RNN 中的重复模块包含单一的层

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，这里是有四个，以一种非常特殊的方式进行交互。



LSTM 中的重复模块包含四个交互的层

不必担心这里的细节。我们会一步一步地剖析 LSTM 解析图。现在，我们先来熟悉一下图中使用的各种元素的图标。

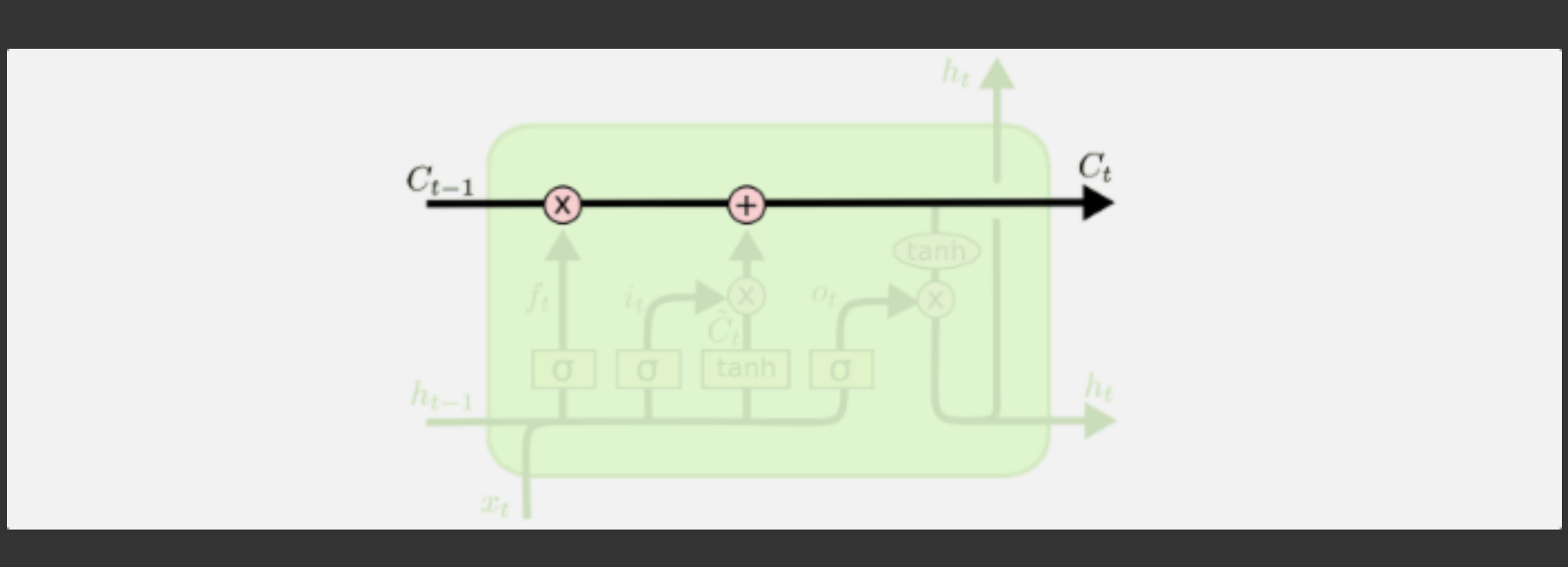


LSTM 中的图标

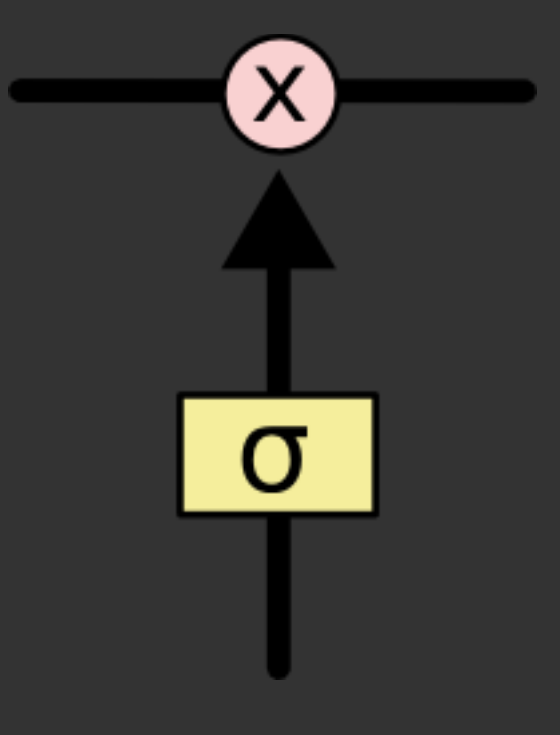
在上面的图例中，每一条黑线传输着一整个向量，从一个节点的输出到其他节点的输入。粉色的圈代表 pointwise 的操作，诸如向量的和，而黄色的矩阵就是学习到的神经网络层。合在一起的线表示向量的连接，分开的线表示内容被复制，然后分发到不同的位置。

2.LSTM核心内容

LSTM 的关键就是细胞状态（cell），水平线在图上方贯穿运行。细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。



LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。



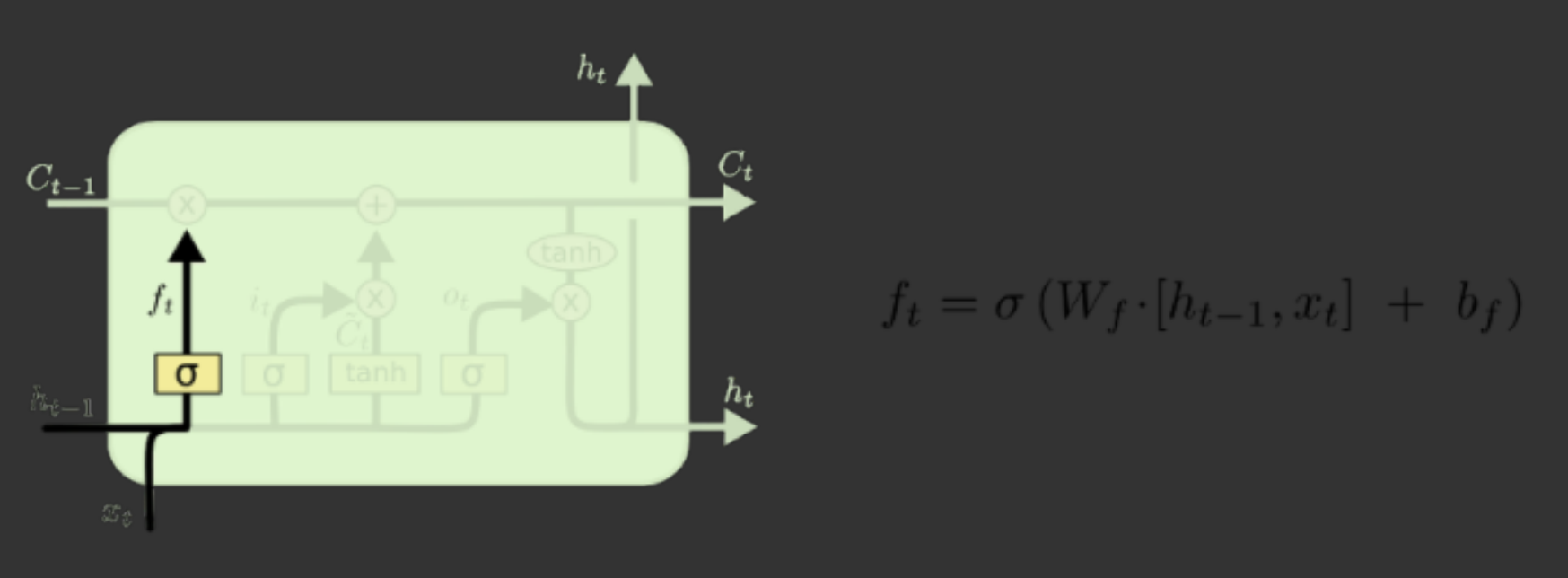
Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 就指“允许任意量通过”！

LSTM 拥有三个门，来保护和控制细胞状态。

3.逐步理解LSTM

在我们 LSTM 中的第一步是决定我们会从细胞状态中丢弃什么信息。这个决定通过一个称为忘记门层 完成。该门会读取h_{t-1}和$x_t$ht−1和xt，输出一个在 0 到 1 之间的数值给每个在细胞状态C_{t-1}Ct−1中的数字。1 表示“完全保留”，0 表示“完全舍弃”。

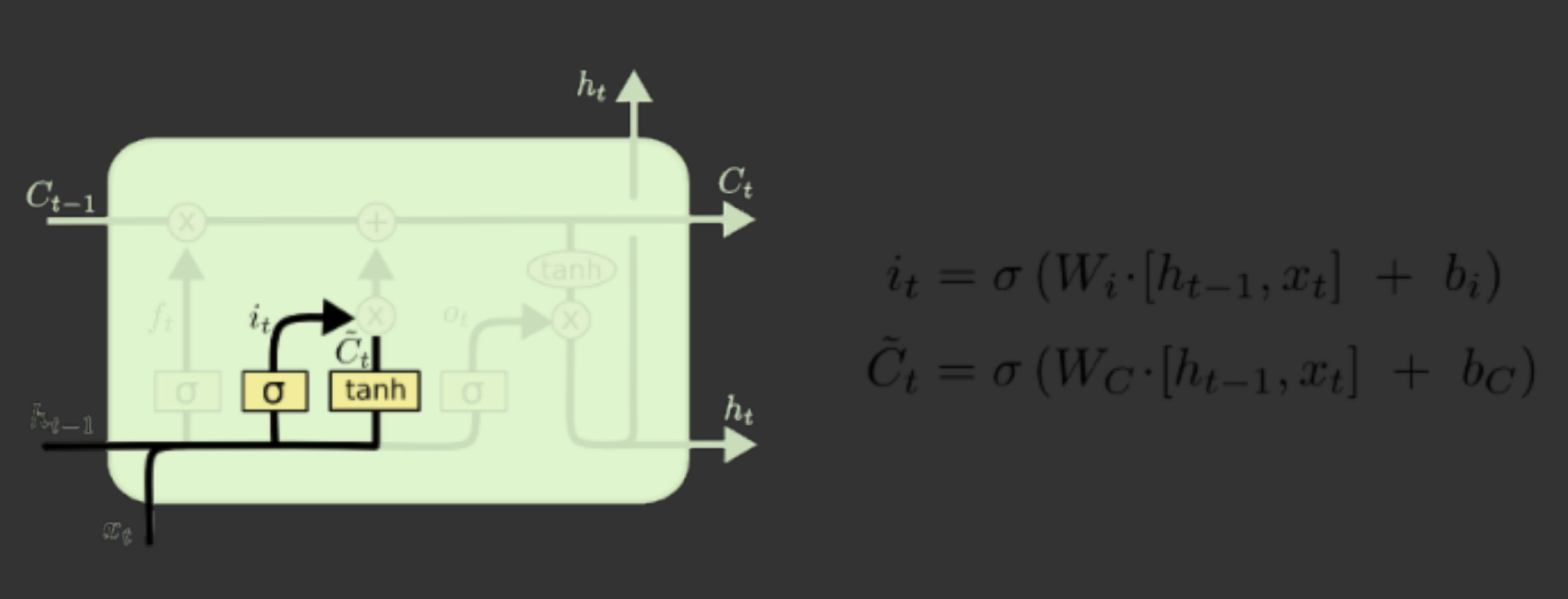
让我们回到语言模型的例子中来基于已经看到的预测下一个词。在这个问题中，细胞状态可能包含当前 主语 的类别，因此正确的 代词 可以被选择出来。当我们看到新的 代词 ，我们希望忘记旧的代词 。



决定丢弃信息

下一步是确定什么样的新信息被存放在细胞状态中。这里包含两个部分。第一，sigmoid 层称 “输入门层” 决定什么值我们将要更新。然后，一个 tanh 层创建一个新的候选值向量，$\tilde{C}_t$C~t，会被加入到状态中。下一步，我们会讲这两个信息来产生对状态的更新。

在我们语言模型的例子中，我们希望增加新的代词的类别到细胞状态中，来替代旧的需要忘记的代词。

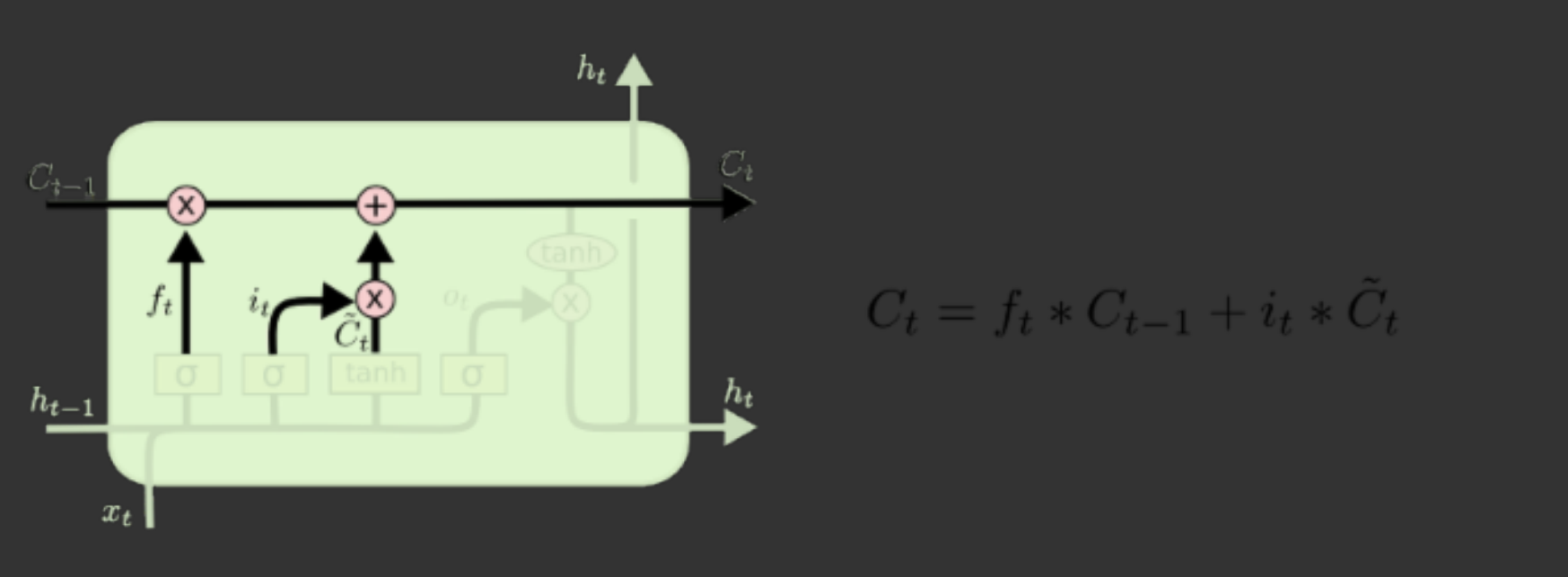


确定更新的信息

现在是更新旧细胞状态的时间了，C_{t-1}更新为C_tCt−1更新为Ct。前面的步骤已经决定了将会做什么，我们现在就是实际去完成。

我们把旧状态与$f_t$ft相乘，丢弃掉我们确定需要丢弃的信息。接着加上$i_t*\tilde{C}_t$it∗C~t。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。

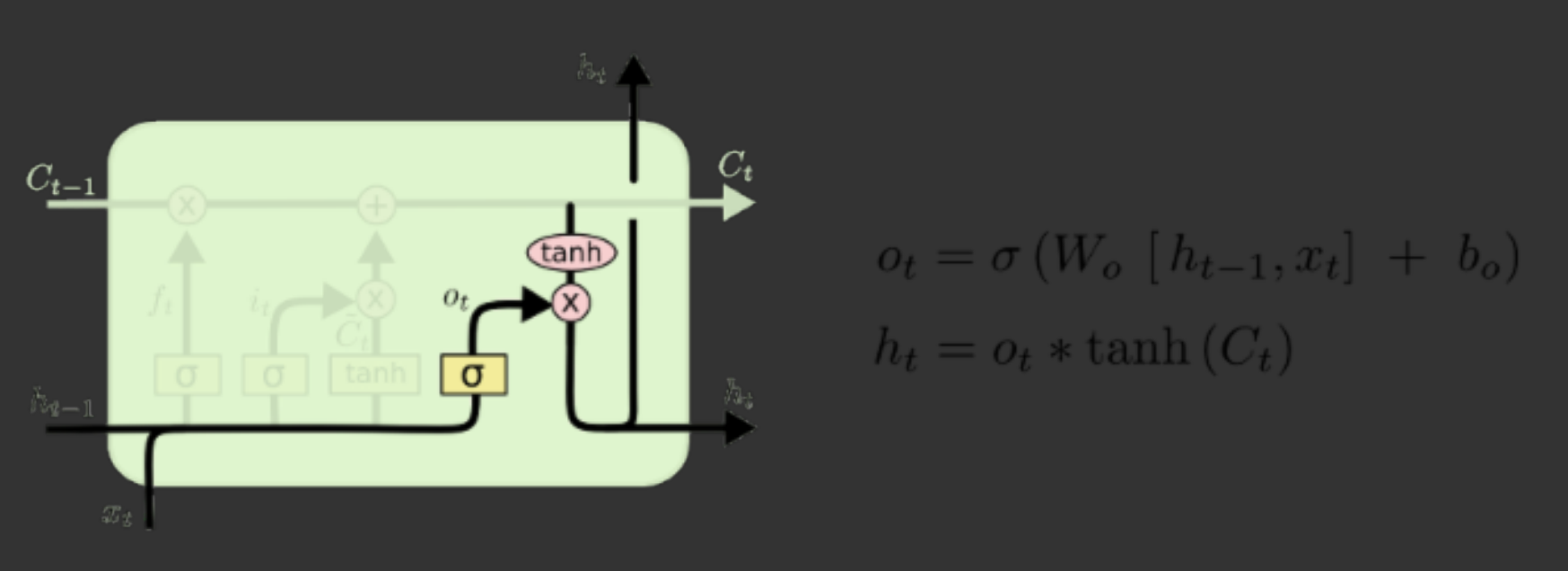
在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的类别信息并添加新的信息的地方。



更新细胞状态

最终，我们需要确定输出什么值。这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。

在语言模型的例子中，因为他就看到了一个 代词 ，可能需要输出与一个 动词 相关的信息。例如，可能输出是否代词是单数还是负数，这样如果是动词的话，我们也知道动词需要进行的词形变化。



输出信息

4.总结数学表达式

Gates：

输入变换：

状态更新：

使用图片描述类似下图：

此图对应于本文上面LSTM的结构图。

LSTM代码实现

具体可参考这篇文章，利用Torch7实现。以后有机会会更新python实现。

参考文献（排名分先后）：

http://blog.csdn.net/heyongluoyao8/article/details/48636251（详细）
http://blog.csdn.net/prom1201/article/details/52221822（易懂）
http://www.csdn.net/article/2015-09-14/2825693（LSTM Torch7实现）
http://www.cnblogs.com/arkenstone/p/5794063.html
http://www.cnblogs.com/tornadomeet/p/3439503.html