（五）通俗易懂理解——BiLSTM

Future has arrived. It commences now.

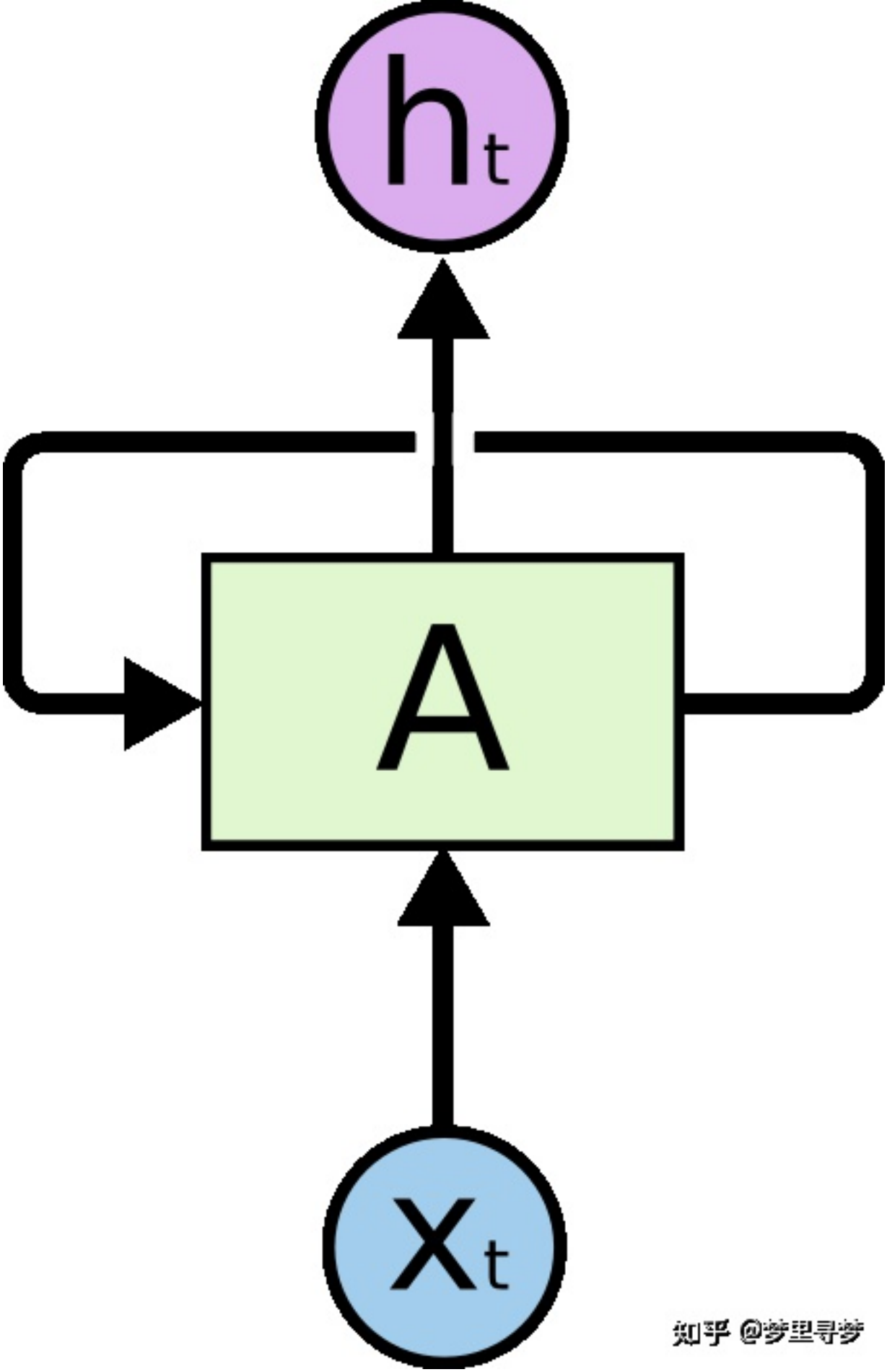
网上资料真的是非常丰富（杂乱），每次想要很清楚地了解一些算法原理都找不到比较有代表性，通俗易懂的。另外，自己以前很多内容看过理解了，当时更多地操作是收藏，但是结果就呵呵了（收藏过的东西基本上很少再去翻看了）。现在决定看过一些比较好的，易于理解的内容尽可能地放进专栏里，虽然很大部分是直接整合别人的文章转载过来，希望可以帮助自己以及跟我一样需要了解这方面内容的朋友。在此也感谢原作者的贡献，也希望自己以后也可以写一些原创。

Recurrent Neural Networks

人类并不是每时每刻都从一片空白的大脑开始他们的思考。在你阅读这篇文章时候，你都是基于自己已经拥有的对先前所见词的理解来推断当前词的真实含义。我们不会将所有的东西都全部丢弃，然后用空白的大脑进行思考。我们的思想拥有持久性。

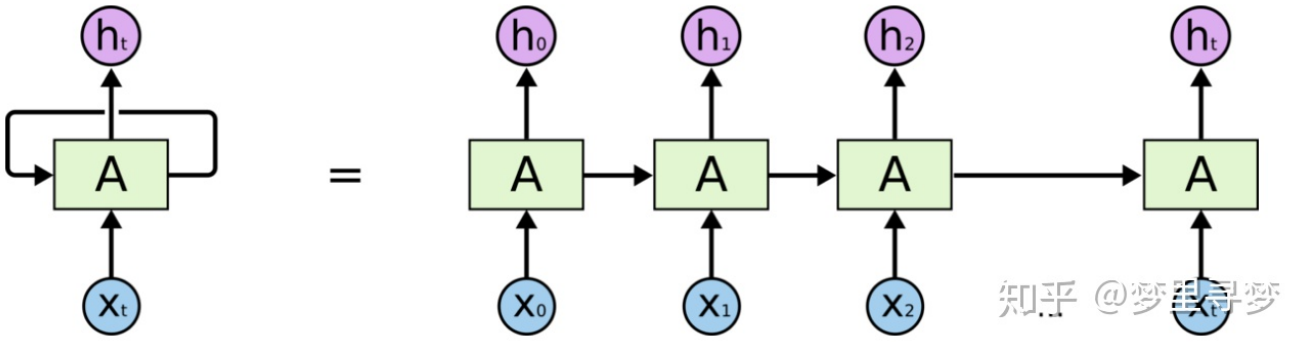
传统的神经网络并不能做到这点，看起来也像是一种巨大的弊端。例如，假设你希望对电影中的每个时间点的时间类型进行分类。传统的神经网络应该很难来处理这个问题——使用电影中先前的事件推断后续的事件。

RNN 解决了这个问题。RNN 是包含循环的网络，允许信息的持久化。



在上面的示例图中，神经网络的模块，A，正在读取某个输入 x_i，并输出一个值 h_i。循环可以使得信息可以从当前步传递到下一步。

这些循环使得 RNN 看起来非常神秘。然而，如果你仔细想想，这样也不比一个正常的神经网络难于理解。RNN 可以被看做是同一神经网络的多次复制，每个神经网络模块会把消息传递给下一个。所以，如果我们将这个循环展开：



链式的特征揭示了 RNN 本质上是与序列和列表相关的。他们是对于这类数据的最自然的神经网络架构。

并且 RNN 也已经被人们应用了！在过去几年中，应用 RNN 在语音识别，语言建模，翻译，图片描述等问题上已经取得一定成功，并且这个列表还在增长。我建议大家参考 Andrej Karpathy 的博客文章——The Unreasonable Effectiveness of Recurrent Neural Networks来看看更丰富有趣的 RNN 的成功应用。

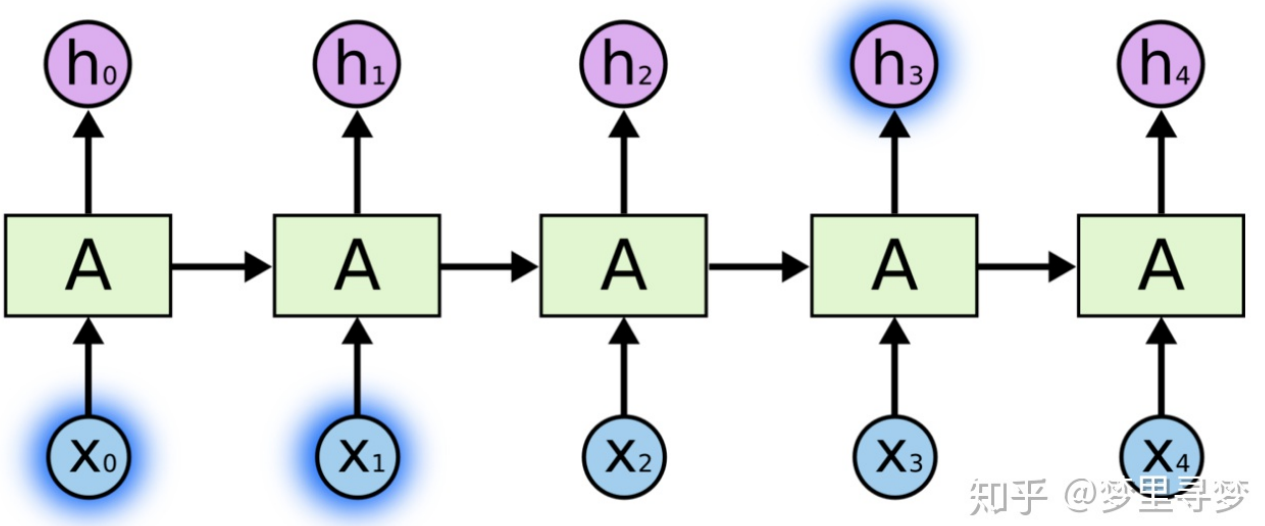
而这些成功应用的关键之处就是 LSTM 的使用，这是一种特别的 RNN，比标准的 RNN 在很多的任务上都表现得更好。几乎所有的令人振奋的关于 RNN 的结果都是通过 LSTM 达到的。这篇博文也会就 LSTM 进行展开。

长期依赖（Long-Term Dependencies）问题

RNN 的关键点之一就是他们可以用来连接先前的信息到当

前的任务上，例如使用过去的视频段来推测对当前段的理解。如果 RNN 可以做到这个，他们就变得非常有用。但是真的可以么？答案是，还有很多依赖因素。

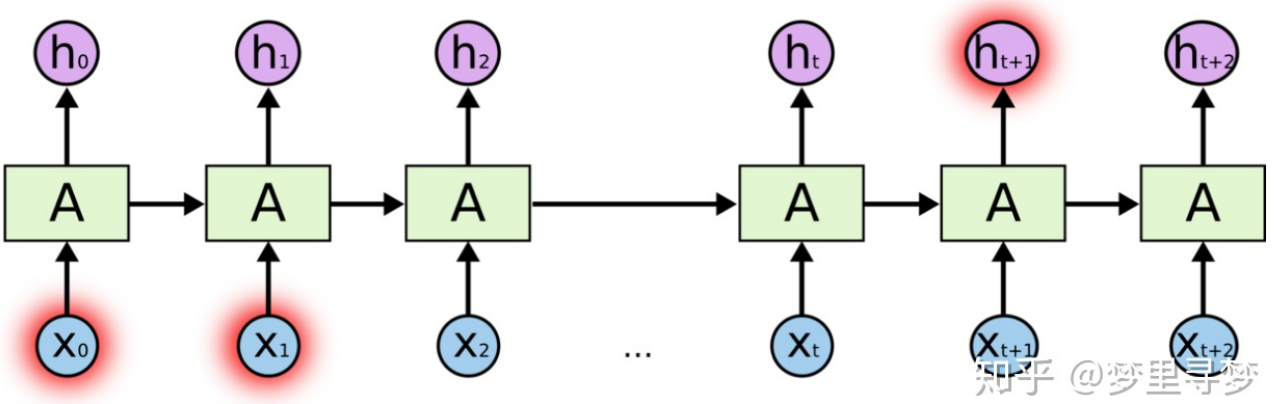
有时候，我们仅仅需要知道先前的信息来执行当前的任务。例如，我们有一个语言模型用来基于先前的词来预测下一个词。如果我们试着预测“the clouds are in the sky”最后的词，我们并不需要任何其他的上下文 —— 因此下一个词很显然就应该是 sky。在这样的场景中，相关的信息和预测的词位置之间的间隔是非常小的，RNN 可以学会使用先前的信息。



不太长的相关信息和位置间隔

但是同样会有一些更加复杂的场景。假设我们试着去预测“I grew up in France... I speak fluent French”最后的词。当前的信息建议下一个词可能是一种语言的名字，但是如果我们需要弄清楚是什么语言，我们是需要先前提到的离当前位置很远的 France 的上下文的。这说明相关信息和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。



在理论上，RNN 绝对可以处理这样的 长期依赖 问题。人们可以仔细挑选参数来解决这类问题中的最初级形式，但在实践中，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，他们发现一些使训练 RNN 变得非常困难的相当根本的原因。

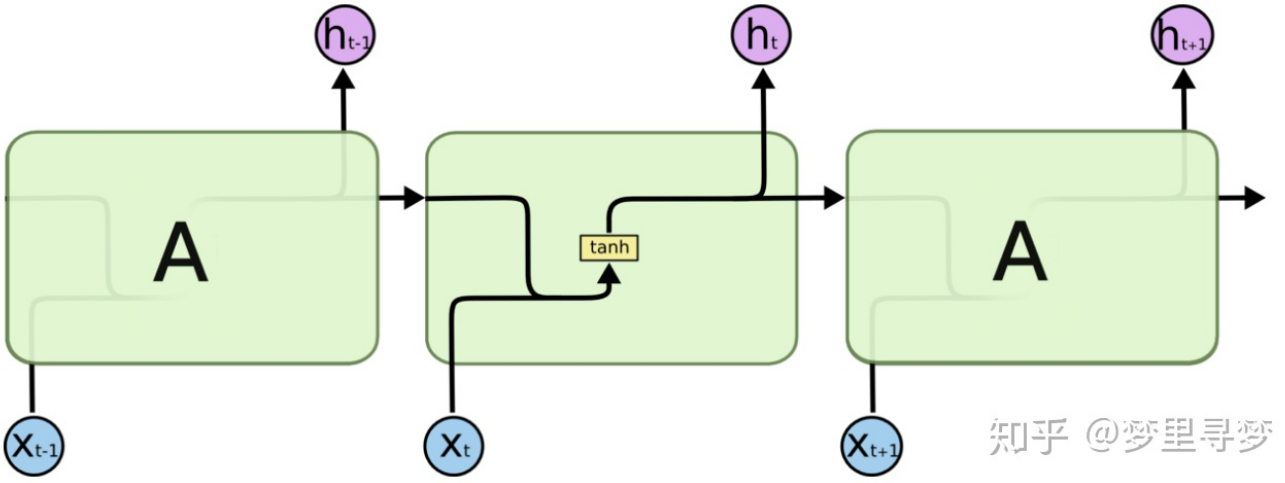
然而，幸运的是，LSTM 并没有这个问题！

LSTM 网络

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。

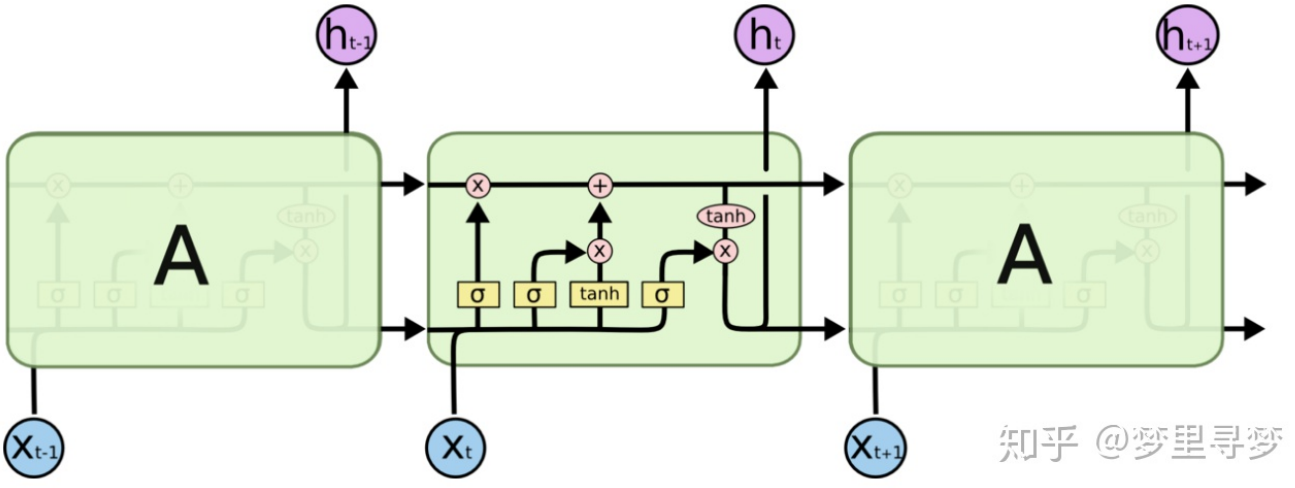
LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力！

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。



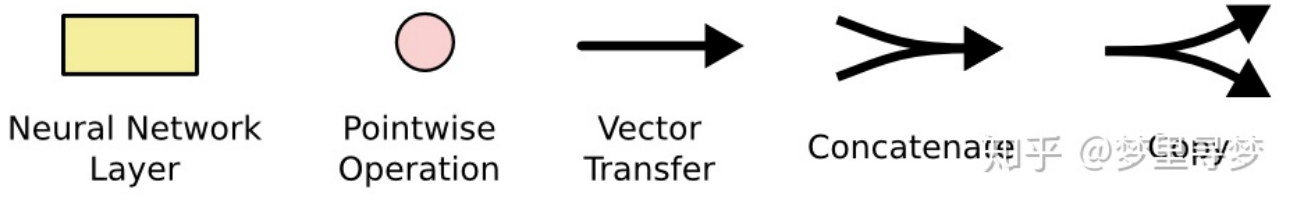
标准 RNN 中的重复模块包含单一的层

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。

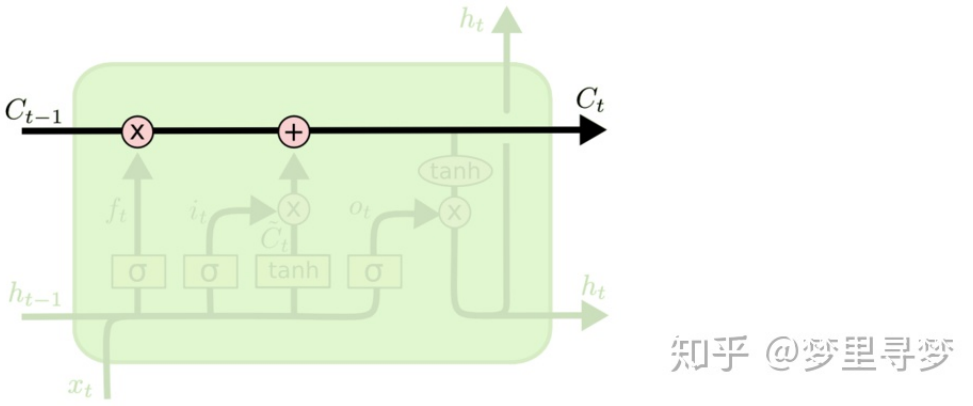


LSTM 中的图标

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。

LSTM 的核心思想

LSTM 的关键就是细胞状态，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。

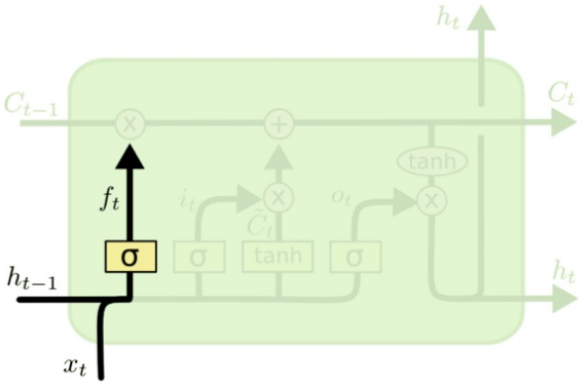
Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 就指“允许任意量通过”！

LSTM 拥有三个门，来保护和控制细胞状态。

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，输出一个在 0 到 1 之间的数值给每个在细胞状态c_{t-1}中的数字。1 表示“完全保留”，0 表示“完全舍弃”。

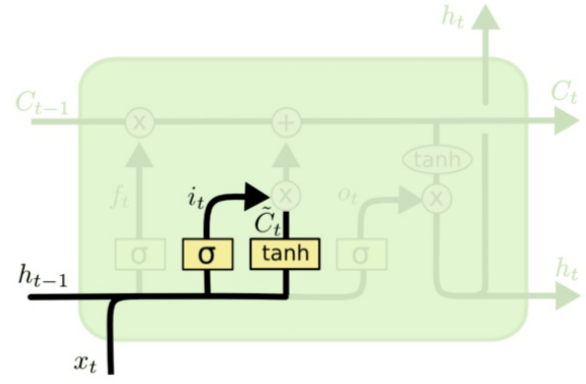
让我们回到语言模型的例子中来基于已经看到的预测下一个词。在这个问题中，细胞状态可能包含当前**主语**的性别，因此正确的**代词**可以被选择出来。当我们看到新的**主语**，我们希望忘记旧的**主语**。



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

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`，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，我们希望增加新的主语的性别到细胞状态中，来替代旧的需要忘记的主语。

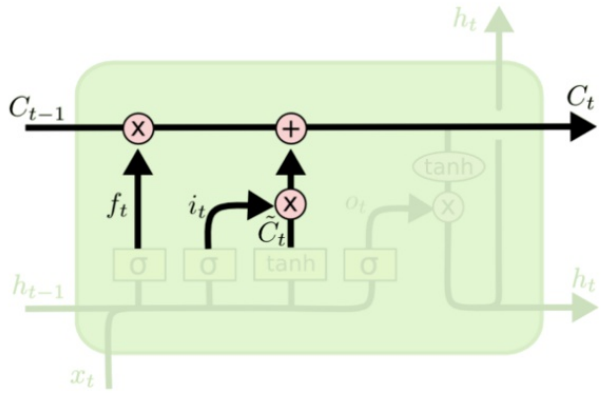


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

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_t`。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`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`。这就是新的候选值，

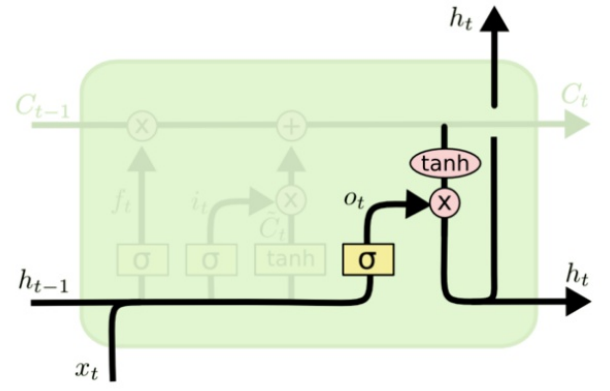
根据我们决定更新每个状态的程度进行变化。
在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的性别信息并添加新的信息的地方。



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

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。



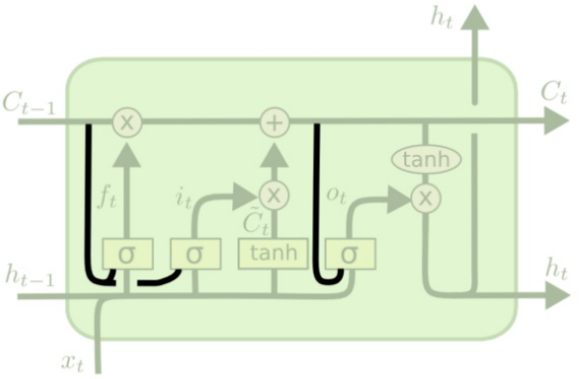
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

输出信息

LSTM 的变体

我们到目前为止都还在介绍正常的 LSTM。但是不是所有的 LSTM 都长成一个样子的。实际上，几乎所有包含 LSTM 的论文都采用了微小的变体。差异非常小，但是也值得拿出来讲一下。

其中一个流形的 LSTM 变体，就是由 Gers & Schmidhuber (2000) 提出的，增加了“peephole connection”。是说，我们让 门层 也会接受细胞状态的输入。

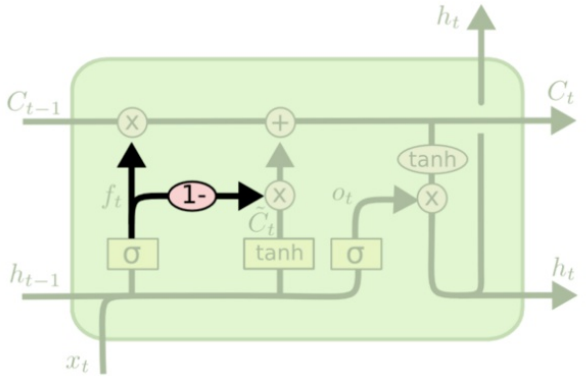


$$f_t = \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [C_t, h_{t-1}, x_t] + b_o\right)$$

peephole 连接

上面的图例中，我们增加了 peephole 到每个门上，但是许多论文会加入部分的 peephole 而非所有都加。

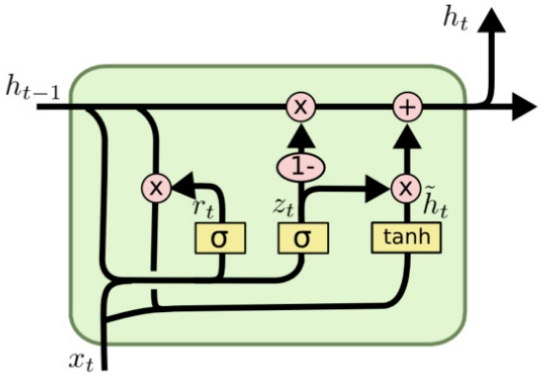
另一个变体是通过使用 coupled 忘记和输入门。不同于之前是分开确定什么忘记和需要添加什么新的信息，这里是一同做出决定。我们仅仅会当我们将要输入在当前位置时忘记。我们仅仅输入新的值到那些我们已经忘记旧的信息的那些状态 。



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

coupled 忘记门和输入门

另一个改动较大的变体是 Gated Recurrent Unit (GRU)，这是由 Cho, et al. (2014) 提出。它将忘记门和输入门合成了一个单一的 更新门。同样还混合了细胞状态和隐藏状态，和其他一些改动。最终的模型比标准的 LSTM 模型要简单，也是非常流行的变体。



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

GRU

这里只是部分流行的 LSTM 变体。当然还有很多其他的，如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。

要问哪个变体是最好的？其中的差异性真的重要吗？Greff, et al. (2015) 给出了流行变体的比较，结论是他们基本上是一样的。Jozefowicz, et al. (2015) 则在超过 1 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。

Jozefowicz等人论文截图

结论

刚开始，我提到通过 RNN 得到重要的结果。本质上所有这些都可以使用 LSTM 完成。对于大多数任务确实展示了更好的性能！

由于 LSTM 一般是通过一系列的方程表示的，使得 LSTM 有一点令人费解。然而本文中一步一步地解释让这种困惑消除了不少。

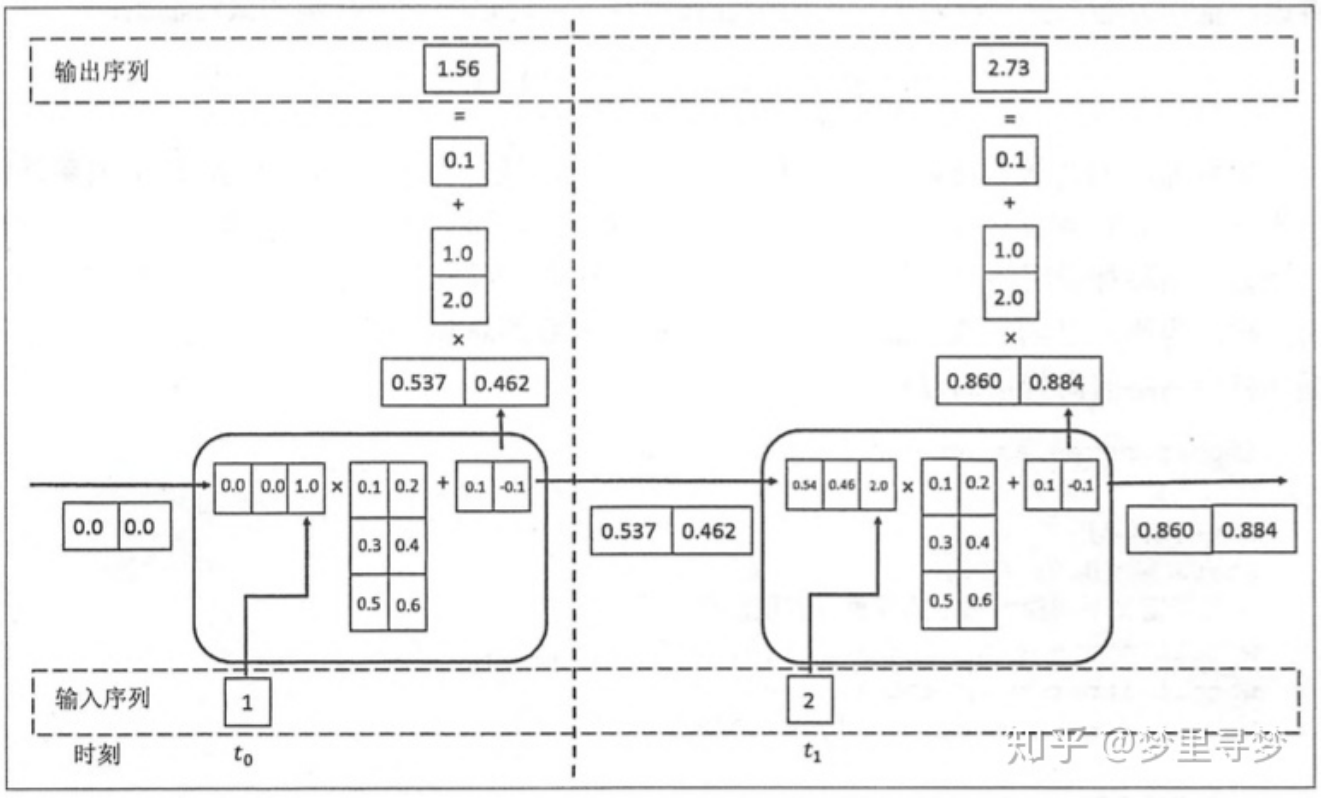
LSTM 是我们在 RNN 中获得的重要成功。很自然地，我们也会考虑：哪里会有更加重大的突破呢？在研究人员间普遍的观点是：“Yes! 下一步已经有了——那就是**注意力**！”这个想法是让 RNN 的每一步都从更加大的信息集中挑选信息。例如，如果你使用 RNN 来产生一个图片的描述，可能会选择图片的一个部分，根据这部分信息来产生输出的词。实际上，Xu, et al.(2015)已经这么做了——如果你希望深入探索**注意力**可能这就是一个有趣的起点！还有一些使用注意力的相当振奋人心的研究成果，看起来有更多的东西亟待探索.....

注意力也不是 RNN 研究领域中唯一的发展方向。例

如，Kalchbrenner, et al. (2015) 提出的 Grid LSTM 看起来也是很有前途。使用生成模型的 RNN，诸如Gregor, et al. (2015) Chung, et al. (2015) 和 Bayer & Osendorfer (2015) 提出的模型同样很有趣。在过去几年中，RNN 的研究已经相当的燃，而研究成果当然也会更加丰富！

以上是对于论文的翻译，现在用例子对双向进行解释！

一下是RNN前向传播过程，仔细看数据之间的一个传播过程可以很好地理解循环神经网络的运行过程。



以下部分是对于上图的解说，看得懂的可以直接跳到**双向循环神经网络（BRNN）** 部分。

如上图所示，假设节点状态的维度为2，节点的输入和输出维度为1，那么在循环体的全连接层神经网络的输入维度为3，**也就是将上一时刻的状态与当前时刻的输入拼接成一维向量作为循环体的全连接层神经网络的输入**，在这里t0时刻的节点状态初始化为[0.0, 0.0]，t0时刻的节点

输入为[1.0]，拼接之后循环体的全连接层神经网络的输入为[0.0, 0.0, 1.0]，循环体中的全连接层的权重表示为二维矩阵[[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]]，偏置项为[0.1, -0.1]，我们可以看到权重矩阵和偏置项在t0和t1时刻的循环体中是一样的，**这也说明了RNN结构中的参数在不同时刻中也是共享的**。经过循环体中的全连接层神经网络后节点的状态改变为tanh([0.6, 0.5]) = [0.537, 0.462]，当前节点状态的输出作为下一个节点状态的输入。

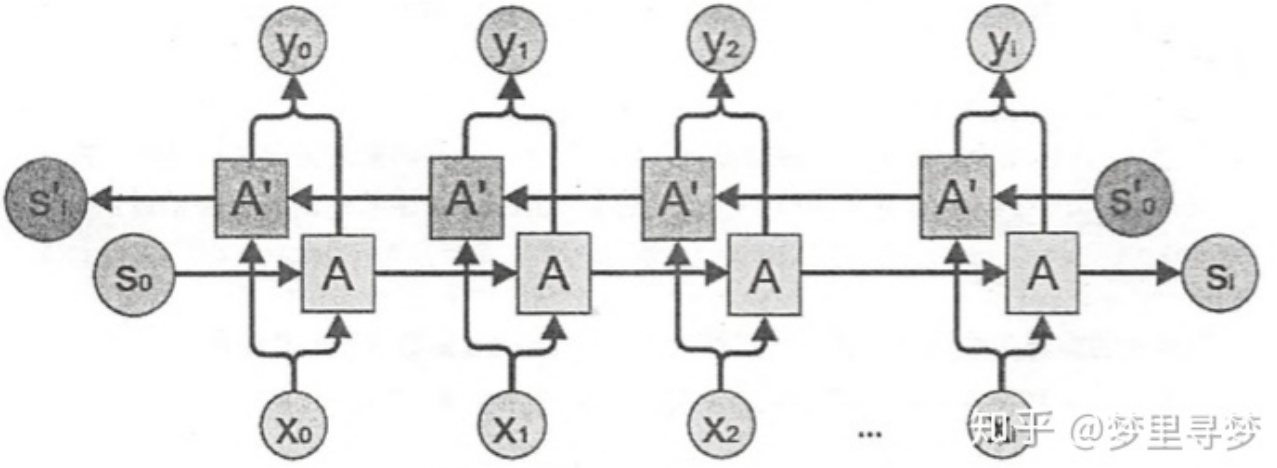
为了将当前时刻的状态转变为节点的最终输出，RNN中还有另外一个全连接神经网络来计算节点输出，在图2中被表示为[0.537, 0.462] * [1.0, 2.0] + [0.1] = [1.56]，用于输出的全连接层权重为[1.0, 2.0]，偏置项为[0.1]，1.56表示为t0时刻节点的最终输出。

得到RNN的前向传播结果之后，和其他神经网络类似，定义损失函数，使用反向传播算法和梯度下降算法训练模型，但RNN唯一的区别在于：**由于它每个时刻的节点都有一个输出，所以RNN的总损失为所有时刻（或部分时刻）上的损失和。**

双向循环神经网络（BRNN）

RNN和LSTM都只能依据之前时刻的时序信息来预测下一时刻的输出，但在有些问题中，**当前时刻的输出不仅和之前的状态有关，还可能和未来的状态有关系**。比如预测一句话中缺失的单词不仅需要根据前文来判断，还需要考虑它后面的内容，真正做到基于上下文判断。BRNN有两个RNN上下叠加在一起组成的，输出由这两个RNN的状态共

同决定。



对于每个时刻t，输入会同时提供给两个方向相反的RNN，输出由这两个单向RNN共同决定。

我的理解

由上图非常容易理解，在我看来，所谓的Bi-LSTM以及Bi-RNN，可以看成是两层神经网络，第一层从左边作为系列的起始输入，在文本处理上可以理解成从句子的开头开始输入，而第二层则是从右边作为系列的起始输入，在文本处理上可以理解成从句子的最后一个词语作为输入，反向做与第一层一样的处理处理。最后对得到的两个结果进行处理。

中文翻译作者博客：https://www.cnblogs.com/wangduo/p/6773601.html?utm_source=itdadao&utm_medium=referral

英文原文作者网址：http://colah.github.io/posts/2015-08-Understanding-LSTMs/

参考相关网址：https://www.imooc.com/article/23821