# Understanding LSTM Networks

*Posted on August 27, 2015*

## Recurrent Neural Networks 循环神经网络

Humans don’t start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.

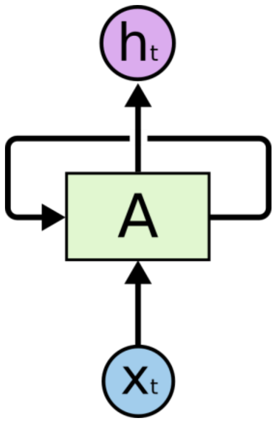
Traditional neural networks can’t do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

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。当阅读文章时候，我们都是基于之前的词来推断当前词的真实含义。我们不会每次都抛开所有的东西，然后用空白的大脑开始思考。我们的思想具有持久性。

传统的神经网络做不到这点，这是它们的主要弊端。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类问题——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。它是内部包含有循环的网络，这允许了信息的持续性。

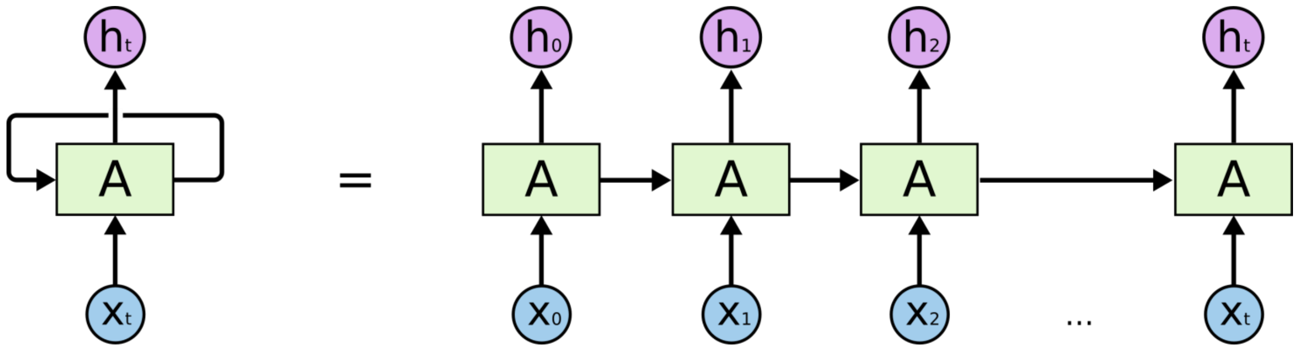


**Recurrent Neural Networks have loops.** 具有循环的 RNN

In the above diagram, a chunk of neural network, \(A\), looks at some input \(x_t\) and outputs a value \(h_t\). A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren’t all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

在上面的示例图中，神经网络块 A，它从某个输入 $x_t$ 读取信息，并输出一个值 $h_t$。循环(loop)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，它并也不比一个标准的神经网络更难于理解。RNN 可以被看作对同一神经网络的多次拷贝，每个神经网络块会把消息传递给下一个。所以，如果我们将这个循环展开：



**An unrolled recurrent neural network.** 一个展开的循环神经网络

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on. I'll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy's excellent blog post, The Unreasonable Effectiveness of Recurrent Neural Networks. But they really are pretty amazing.

Essential to these successes is the use of "LSTMs," a very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

展开的 RNN

这连锁的性质表明，循环神经网络与序列和列表密切相关。它们是用于这类数据的神经网络的自然结构。而实际上，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，机器翻译，图片字幕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。

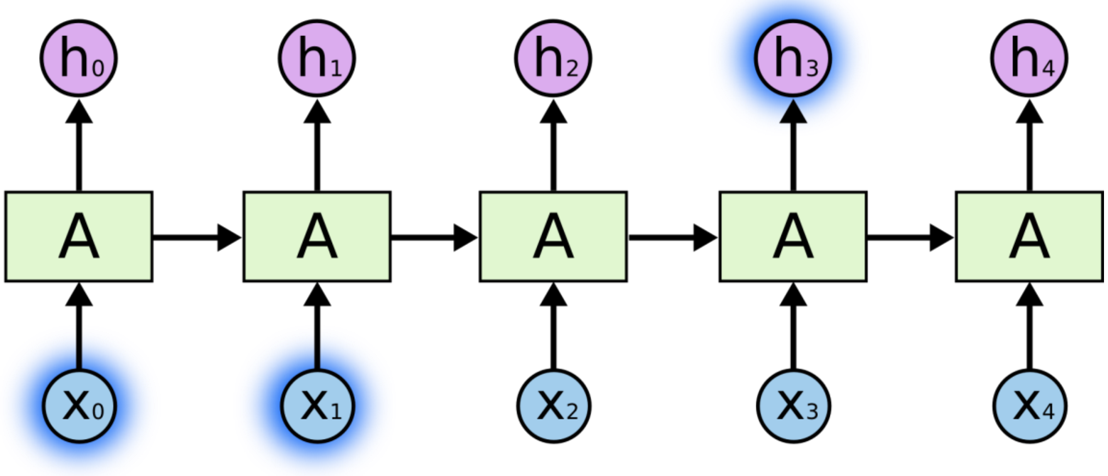
而其成功应用的关键之处在于 LSTM 的使用，这是一种特别的 RNN，比标准的 RNN 在很多的任务上都表现得更好。几乎所有令人振奋的 RNN 的结果都是通过 LSTM 实现的。这篇博文也会就 LSTM 进行展开。

## The Problem of Long-Term Dependencies 长程依赖问题

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they'd be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need any further context – it’s pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.

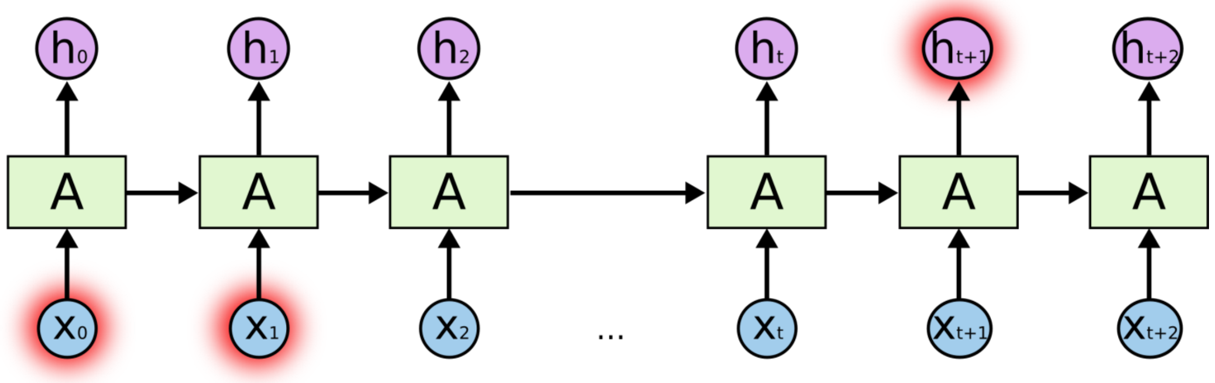
RNN 的关键点之一就是可以把先前的信息连接到当前的任务上，就像使用先前的视频帧来推测理解当前帧段。如果 RNN 可以做到这个，他们就变得非常有用。但是真的可以么？答案是，还有很多依赖因素。有时候，我们仅仅需要知道最近的信息就能执行当前的任务。例如，我考虑一个语言模型，它试图基于先前的词来预测下一个词。如果我们试着预测 “the clouds are in the sky” 最后的词，我们并不需要任何其他的上下文 —— 因此下一个词很显然就应该是 sky。在这样的场景中，相关的信息和预测的词位置之间的间隔是非常小的，RNN 能够学会使用先前的信息。



But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

然而，还有一些情况需要使用更加复杂的上下文。假设我们试着去预测 “I grew up in France... I speak fluent *French*” 最后的词。最近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。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。



In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.

Thankfully, LSTMs don't have this problem!

在理论上，RNN 绝对可以处理这样的 "长程依赖" 问题。人们可以通过挑选适当的参数来解决这种类型的问题。不幸的是，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，他们发现一些使 RNN 训练变得非常困难的，但是又相当根本的原因（梯度爆炸、梯度消失）。然而，幸运的是，LSTM 并没有这个问题！

## LSTM Networks LSTM 网络

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.[1] They work tremendously well on a large variety of problems, and are now widely used.

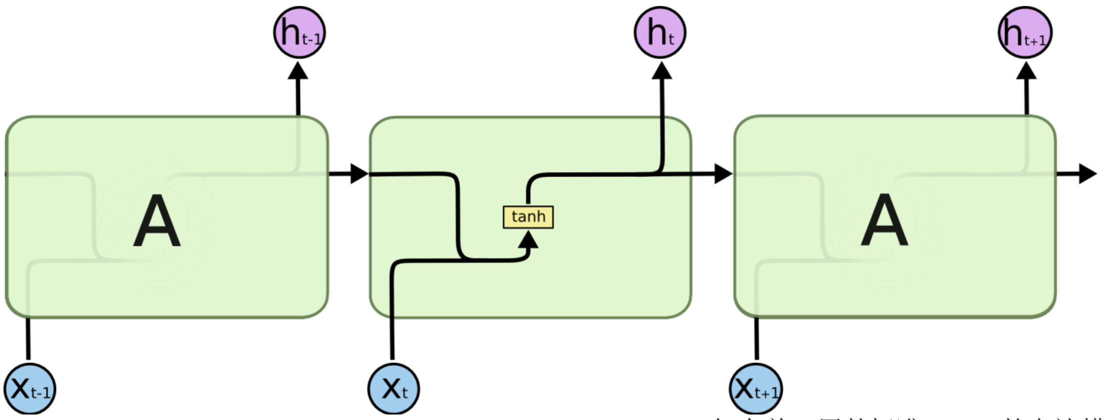
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

Long Short Term(距离)Memory(存储)网络——一般就叫做 LSTM——是一种 RNN 特殊的类型，可以学习长程依赖信息。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程依赖问题。记住长程信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力！

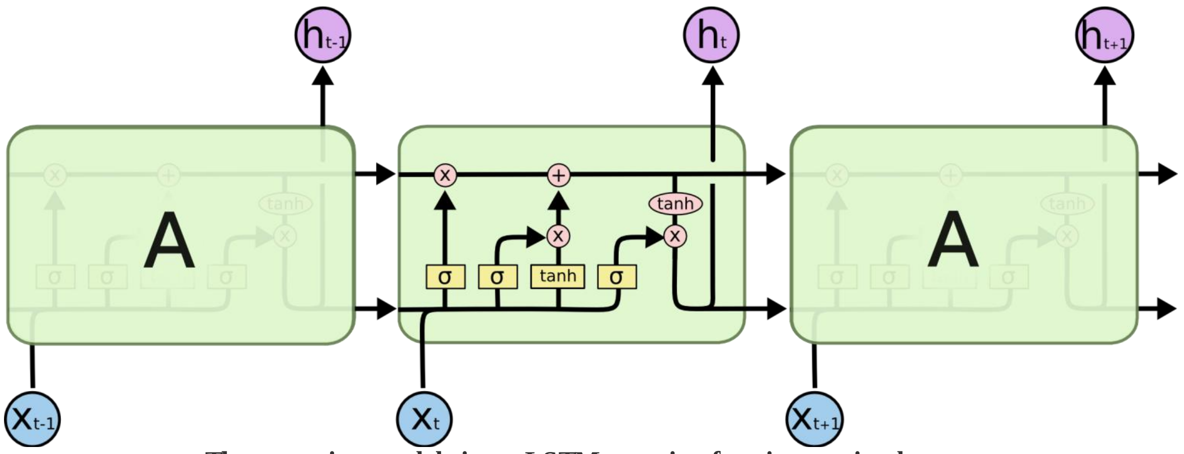
所有 RNN 都具有一种神经网络重复模型（相同）的链式形式。在标准的 RNN 中，这个重复模型（相同）只有一个非常简单的结构，例如一个 tanh 层。



**The repeating module in a standard RNN contains a single layer.** 包含单一层的标准 RMM 的表达模型

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

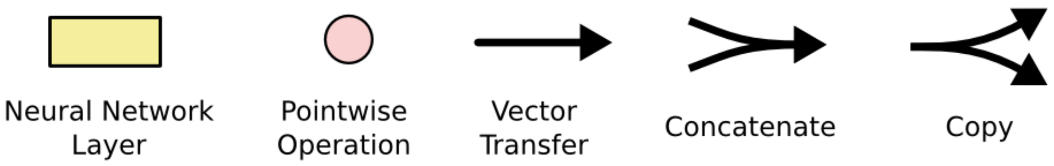
LSTM 也具有类似链式的结构，但是重复模型拥有一个不同的结构。区别于仅有一层的神经网络，如图是有四个，以一种非常特殊的方式进行交互。



**The repeating module in an LSTM contains four interacting layers.**

Don’t worry about the details of what’s going on. We’ll walk through the LSTM diagram step by step later. For now, let’s just try to get comfortable with the notation we’ll be using.

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。



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

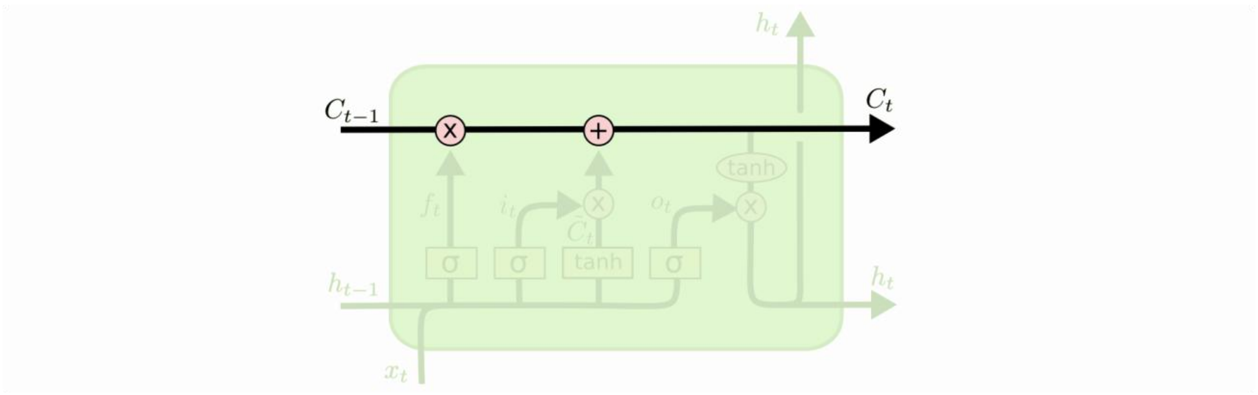
在上面的图例中，线条表示携带一个完整的向量，从一个节点输出再输入到另一个节点。粉色的圈代表逐点（Pointwise）的运算，诸如向量求和，而黄盒子表示学习到的神经网络层。合并（Concatenate）的线条表示向量的连接，分开的线表示内容被复制，然后拷贝到不同的位置。

## The Core Idea Behind LSTMs LSTM 的核心思想

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

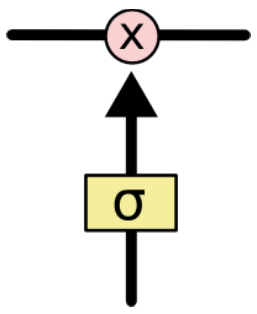
LSTM 的关键就是细胞（Cell）状态，在图上方一条水平线贯穿运行。细胞状态有点类似于一个传送带。它在整个链条上运行，只有一些少量的线性交互。信息在上面保持不变的流传是很容易的。



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

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影响细胞的状态。门是一种让信息有选择的通过或阻止的方法。他们包含一个 sigmoid 神经网络层和一个 逐点（pointwise）乘法操作。



The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.

Sigmoid 层输出 0~1 之间的数值，这个值描述了每个组成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。

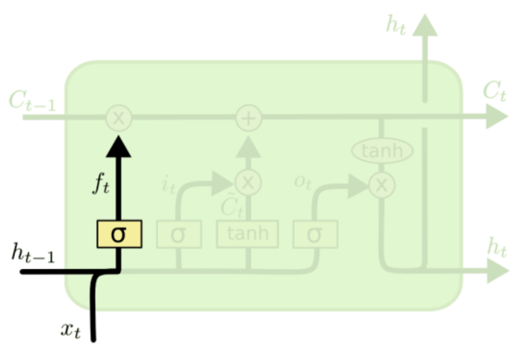
## Step-by-Step LSTM Walk Through 分步详解 LSTM 的流程

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at \(h_{t-1}\) and \(x_t\), and outputs a number between \(0\) and \(1\) for each number in the cell state \(C_{t-1}\). A \(1\) represents "completely keep this" while a \(0\) represents "completely get rid of this."

Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

在 LSTM 中的第一步是确定从细胞状态中丢弃哪些信息。这个决策通过一个称为 "**忘记门**" **层**来完成。该门读取来自 $h_{\{t-1\}}$和 $x_t$ 的输出，为在细胞状态 $C_{\{t-1\}}$中的每个值输出一个 0~1 之间的值。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，我们基于所有之前的结果的预测出下一个词。在这类问题中，细胞状态可能包含当前的主题词，以便正确的 **代词** 可以被选择出来。当我们看到新的主题词，我们希望能够忘记旧的主题词。



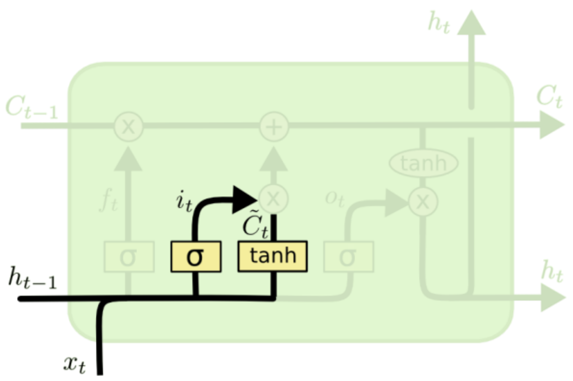
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, \(\tilde{C}_t\), that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

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。第一部分，sigmoid 层称 "输入门层" 决定我们将要更新的值。接下来，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。在下一步，我们将联合这两个信息来产生状态的更新。

在这个例子中，我们希望增加新的主题词类别到细胞状态中，来替代旧的需要忘记的主题词。



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

It's now time to update the old cell state, \(C_{t-1}\), into the new cell state \(C_t\). The previous steps already decided what to do, we just need to actually do it.

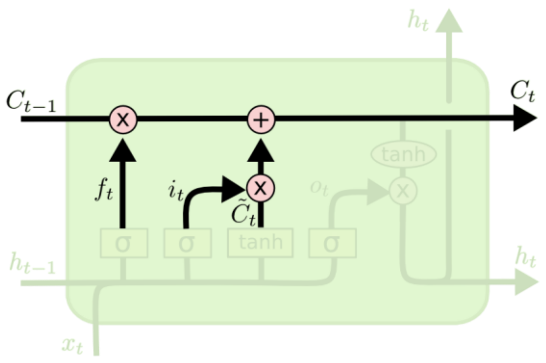
We multiply the old state by \(f_t\), forgetting the things we decided to forget earlier. Then we add \(i_t*\tilde{C}_t\). This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

现在到了更新旧细胞状态的时候，$C_{\{t-1\}}$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它。

我们把旧状态与 ft 相乘， “忘掉”我们确定需要忘掉的信息。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，丢弃旧主题词的类别信息，并添加新的信息的位置。



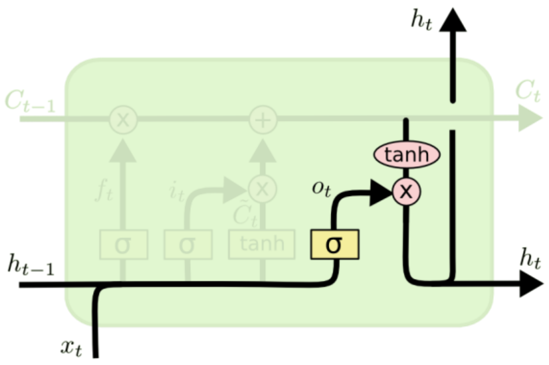
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through \(\tanh\) (to push the values to be between \(-1\) and \(1\)) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

最终，我们需要确定输出的值。输出将会基于我们的细胞状态，但是也有一个过滤后的版本。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，因为看到了一个 **主题词** ，可能需要输出与一个 **动词** 相关的信息。例如，可能主语的输出是单数还是复数，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。



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

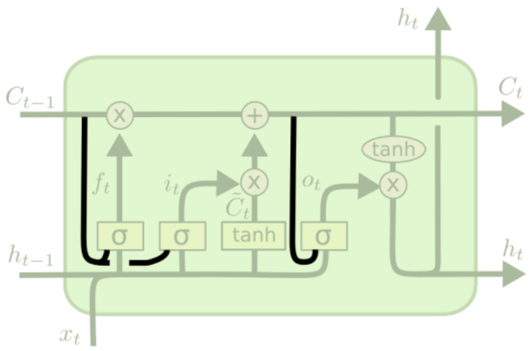
## Variants on Long Short Term Memory LSTM 的变体

What I've described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them.

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.

我们到目前为止都还在介绍标准的 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。

其中一个流行的 LSTM 变体，就是由 Gers & Schmidhuber (2000) 提出的，增加了 "peephole connection"。是说，我们让 门层 观察细胞的状态。



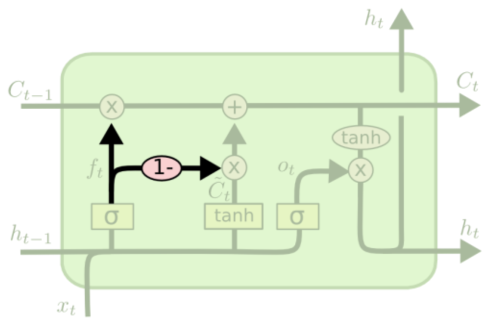
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t]\ +\ b_o\right)$$

The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.

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

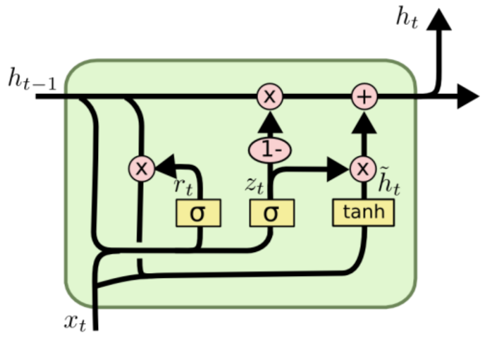
另一个变体是通过使用 成对儿的 忘记门和输入门。代替之前的相互分离的遗忘门和输入门。这种门使忘记旧信息和添加新信息同时进行。当要输入新信息时，才会执行忘记。仅仅输入新值到状态中，以忘记旧的值 。



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

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

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，并且做出了一些其他改动。最终的模型比标准的 LSTM 模型要简单，也是非常流行的变体。



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

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015). There’s also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014).

Which of these variants is best? Do the differences matter? Greff, et al. (2015) do a nice comparison of popular variants, finding that they’re all about the same. Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

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 万中 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。

## Conclusion 结论

Earlier, I mentioned the remarkable results people are achieving with RNNs. Essentially all of these are achieved using LSTMs. They really work a lot better for most tasks!

Written down as a set of equations, LSTMs look pretty intimidating. Hopefully, walking through them step by step in this essay has made them a bit more approachable.

LSTMs were a big step in what we can accomplish with RNNs. It's natural to wonder: is there another big step? A common opinion among researchers is: "Yes! There is a next step and it's attention!" The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, *et al.* (2015) do exactly this – it might be a fun starting point if you want to explore attention! There's been a number of really exciting results using attention, and it seems like a lot more are around the corner…

Attention isn't the only exciting thread in RNN research. For example, Grid LSTMs by Kalchbrenner, *et al.* (2015) seem extremely promising. Work using RNNs in generative models – such as Gregor, *et al.* (2015), Chung, *et al.* (2015), or Bayer & Osendorfer (2015) – also seems very interesting. The last few years have been an exciting time for recurrent neural networks, and the coming ones promise to only be more so!

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 看起来也是很有钱途。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！

## Acknowledgments

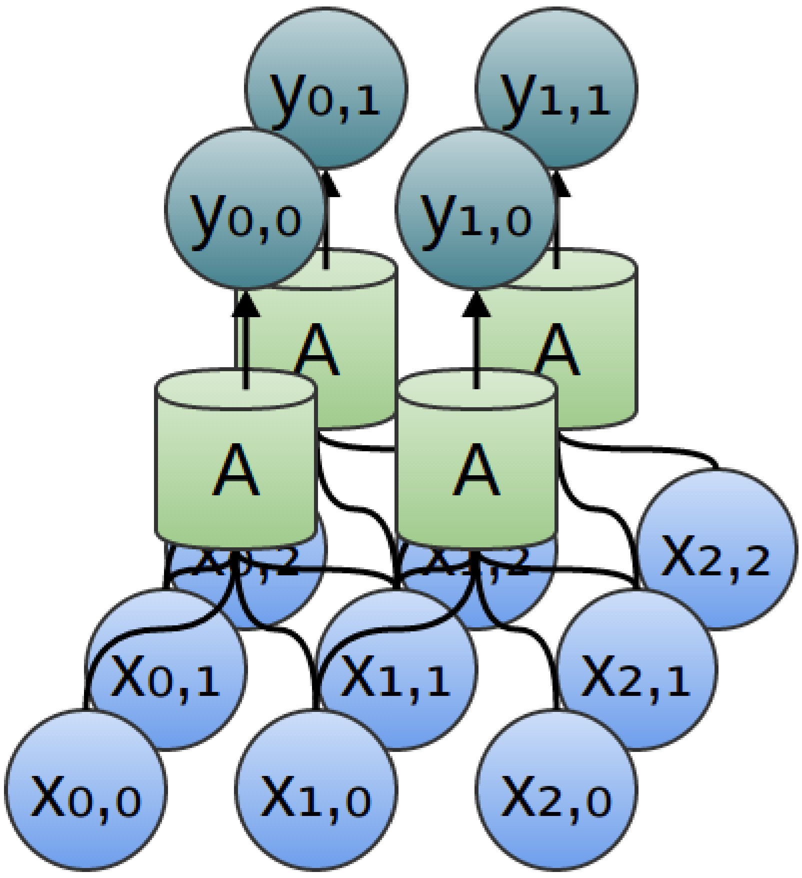
I'm grateful to a number of people for helping me better understand LSTMs, commenting on the visualizations, and providing feedback on this post.

I'm very grateful to my colleagues at Google for their helpful feedback, especially Oriol Vinyals, Greg Corrado, Jon Shlens, Luke Vilnis, and Ilya Sutskever. I'm also thankful to many other friends and colleagues for taking the time to help me, including Dario Amodei, and Jacob Steinhardt. I'm especially thankful to Kyunghyun Cho for extremely thoughtful correspondence about my diagrams.

Before this post, I practiced explaining LSTMs during two seminar series I taught on neural networks. Thanks to everyone who participated in those for their patience with me, and for their feedback.

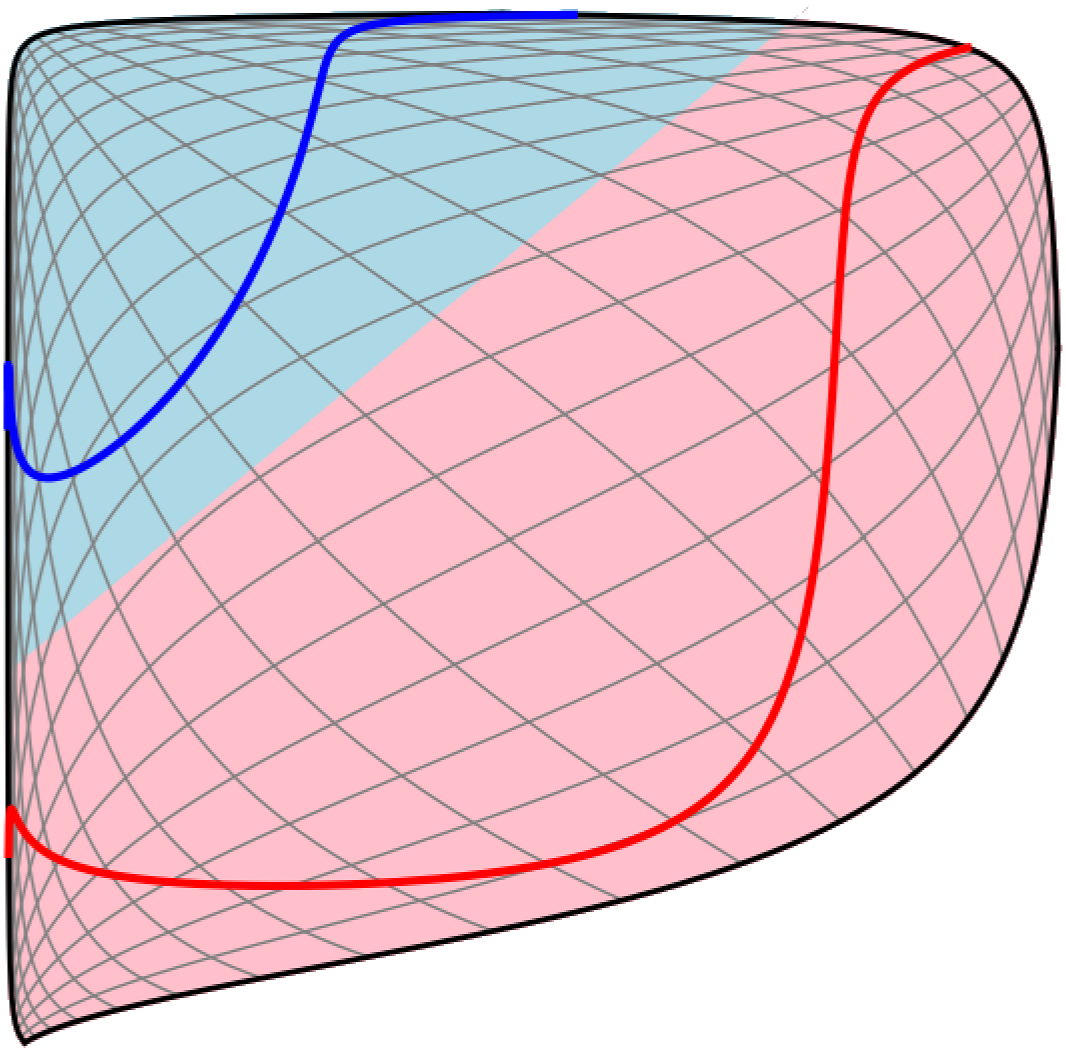
1. In addition to the original authors, a lot of people contributed to the modern LSTM. A non-comprehensive list is: Felix Gers, Fred Cummins, Santiago Fernandez, Justin Bayer, Daan Wierstra, Julian Togelius, Faustian Gomez, Matteo Gagliolo, and Alex Graves.↩

**More Posts**

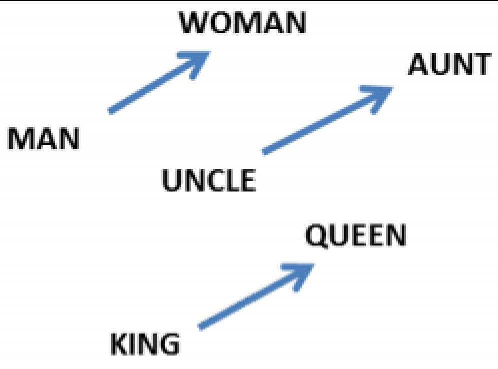


Conv Nets

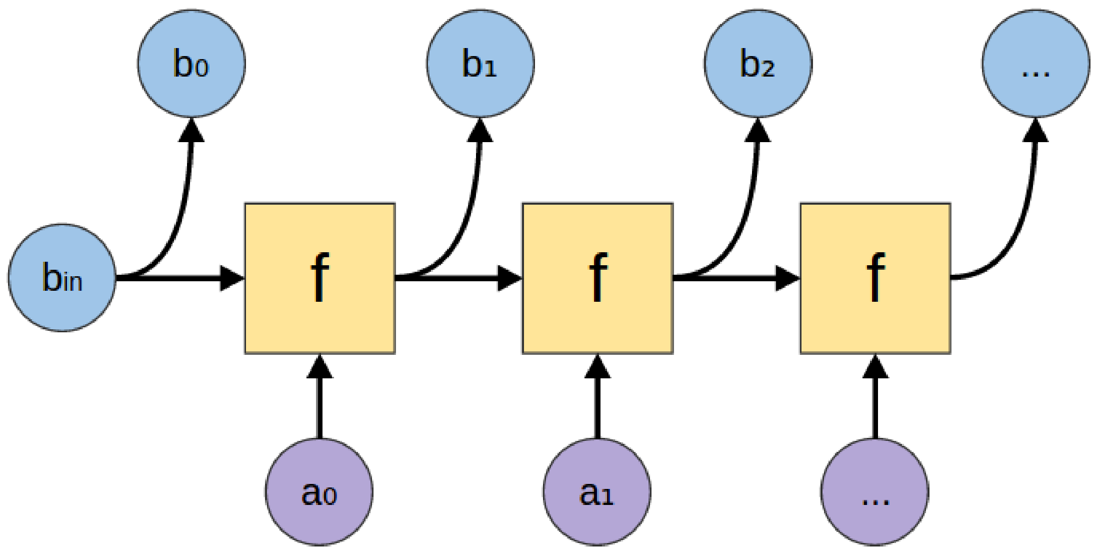
A Modular Perspective



Neural Networks, Manifolds, and Topology



Deep Learning, NLP, and Representations



Data.List Recursion Illustrated