Understanding LSTM Networks（LSTM原理）

Understanding LSTM Networks

Recurrent Neural Networks

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.

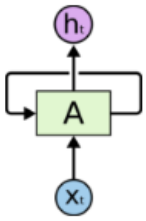
Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later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

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

人类不会每秒钟从头开始他们的思想。 当你阅读这篇文章时，你基于你对以前的话的理解来理解每个单词。 你不要把一切都丢掉，从头开始思考。 你的想法有坚持。

传统的神经网络不能做到这一点，它似乎是一个主要的缺点。 例如，假设您想要分类在电影中的每个点发生什么样的事件。 目前尚不清楚传统神经网络如何使用其对电影中以前事件的推理来告知后者。

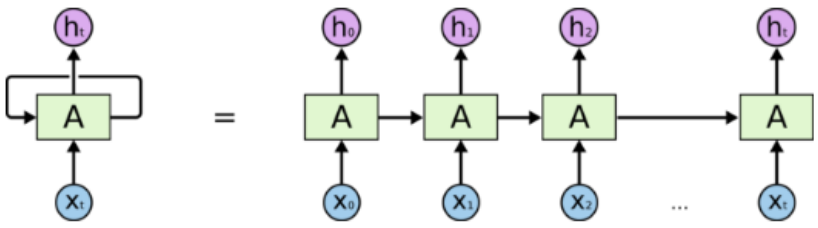
循环神经网络解决这个问题。 它们是在其中具有循环的网络，允许信息持久。



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. 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:



An unrolled recurrent neural network.

在上面的图中，神经网络的一个块A观察一些输入xt并输出值ht。 循环允许信息从网络的一个步骤传递到下一个步骤。

这些循环使得循环神经网络看起来是神秘的。 然而，如果你多想一点，结果是他们不是一个正常的神经网络不同。 循环神经网络可以被认为是同一网络的多个副本，每个副本将消息传递给后继者。 考虑如果我们展开循环会发生什么：

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.

这种链状的性质揭示了复发性神经网络与序列和列表密切相关。 它们是用于这种数据的神经网络的自然结构。

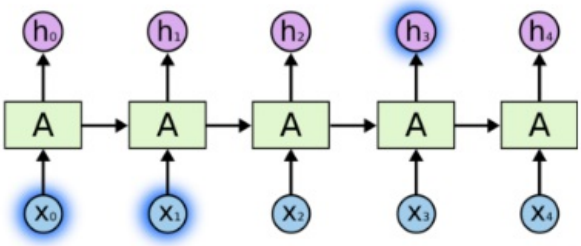
他们肯定是使用！ 在过去几年中，将RNN应用于各种问题取得了令人难以置信的成功：语音识别，语言建模，翻译，图像字幕...。 我将讨论使用RNNs对Andrej Karpathy的优秀博文“反复神经网络的不合理的有效性”所能达到的惊人的专长。 但他们真的是非常惊人。

这些成功的关键在于使用“LSTM”，这是一种非常特殊的循环神经网络，对于许多任务来说，它比标准版本好得多。 几乎所有令人兴奋的结果基于循环神经网络是与他们实现。 这篇文章将探讨这些LSTM。

The Problem of Long-Term Dependencies

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.

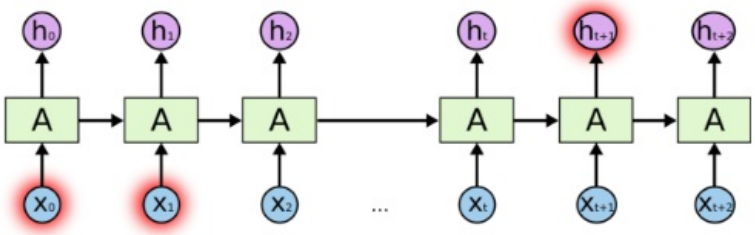


RNN的呼吁之一是这样的想法，即他们可能能够将先前的信息连接到当前任务，例如使用先前的视频帧可能通知对当前帧的理解。 如果RNN可以做到这一点，他们将是非常有用的。 但他们能吗？ 这取决于。

有时，我们只需要查看最近的信息来执行当前的任务。 例如，考虑一种基于先前的语言模型来尝试预测下一个单词的语言模型。 如果我们试图预测“云在天空中”的最后一个词，我们不需要任何进一步的上下文 - 很明显，下一个词将是天空。 在这种情况下，如果相关信息和需要的地方之间的差距很小，RNN可以学习使用过去的信息。

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.



但也有一些情况下我们需要更多的上下文。 考虑尝试预测文本中的最后一个单词“我在法国长大...我说流利的法语”。最近的信息表明，下一个单词可能是一种语言的名称，但如果我们想缩小哪种语言，我们需要 法国的背景，从后面。 完全可能的是，相关信息和需要变得非常大的点之间的差距。

不幸的是，随着这种差距的增长，RNN无法学会连接信息。

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!
理论上，RNN绝对能够处理这种"长期依赖性"。人们可以仔细挑选参数来解决这种形式的玩具问题。遗憾的是，在实践中，RNN似乎不能够学习它们。Hochreiter（1991）[德国]和Bengio等人深入探讨了这个问题。（1994），他们发现了一些相当根本的原因，为什么它可能是困难的。

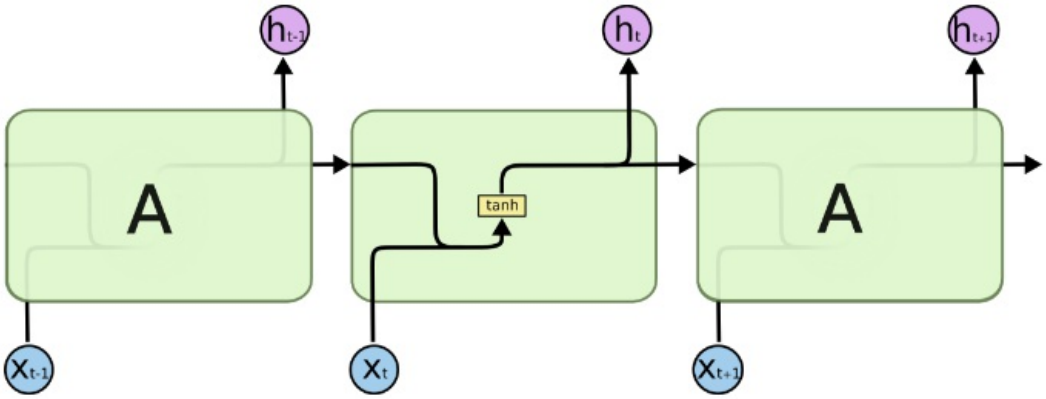
幸运的是，LSTM没有这个问题！

LSTM Networks

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.



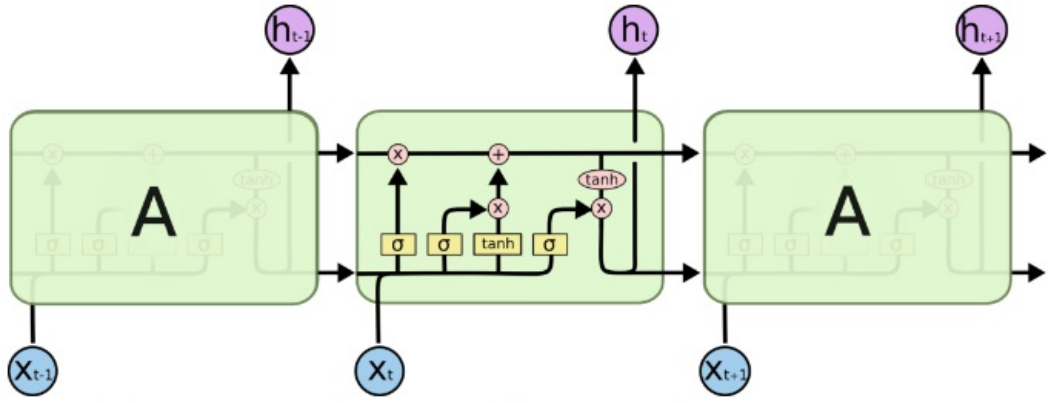
The repeating module in a standard RNN contains a single layer.

长短期存储器网络（通常称为"LSTM"）是一种特殊类型的RNN，能够学习长期依赖性。它们由Hochreiter&Schmidhuber（1997）引入，并且在许多人的下列工作中被改进和普及.1它们在各种各样的问题上工作得非常好，并且现在被广泛使用。

LSTM被明确地设计以避免长期依赖性问题。记住长时间的信息实际上是他们的默认行为，而不是他们努力学习的东西！

所有循环神经网络具有神经网络的重复模块链的形式。在标准RNN中，该重复模块将具有非常简单的结构，例如单个tanh层。

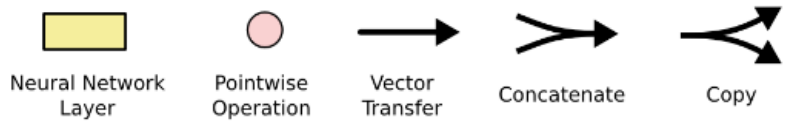
LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

LSTM也具有这种链状结构，但是重复模块具有不同的结构。 代替具有单个神经网络层，有四个，以非常特殊的方式交互。

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.

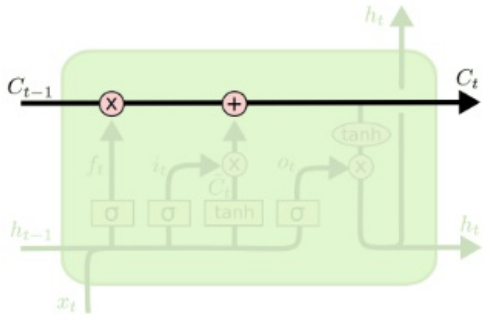


不要担心发生了什么的细节。 我们将稍后逐步介绍LSTM图。 现在，让我们尝试使用我们将使用的符号。
In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.
在上图中，每一行携带一个整个矢量，从一个节点的输出到其他节点的输入。 粉红色圆圈表示点向运算，如矢量加法，而黄色框是学习的神经网络层。 行合并表示并置，而分叉表示其内容正在复制，副本将转到不同位置。

The Core Idea Behind LSTMs

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.

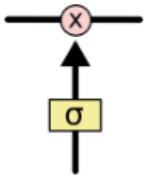


LSTM的关键是单元状态，水平线穿过图的顶部。

电池状态就像一条传送带。 它直接沿整个链运行，只有一些小的线性相互作用。 这是很容易的信息只是沿着它不变。

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确实具有去除或添加信息到细胞状态的能力，由称为门的结构仔细调节。

盖茨是一种可选的让信息通过的方式。 它们由S形神经网络层和点乘法运算组成。

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层输出零和一之间的数字，描述每个组件应该通过多少。 值为零意味着“不能通过”，而值为一意味着“让一切通过！

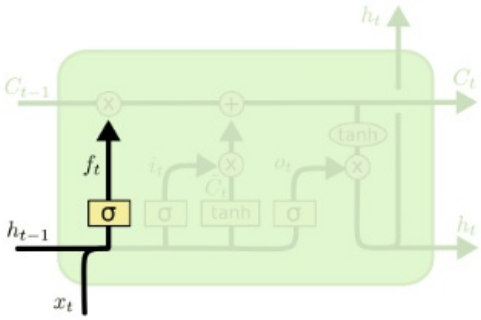
LSTM具有这些门中的三个，以保护和控制单元状态。

Step-by-Step LSTM Walk Through

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 ht−1 and xt, and outputs a number between 00 and 11 for each number in the cell state Ct−1Ct−1. A 11 represents “completely keep this” while a 00 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.



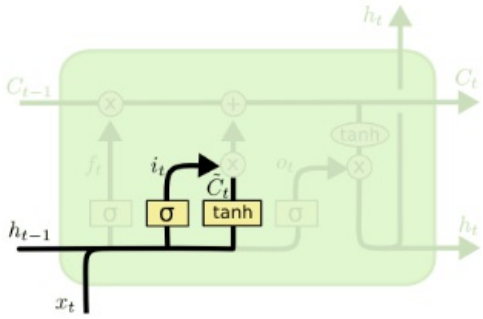
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

我们的LSTM的第一步是决定什么信息，我们将抛弃的单元格状态。 该决定由称为“忘记门层”的S形层进行。它查看ht-1和xt，并且对于单元状态Ct-1中的每个数字输出在0和1之间的数字。 A 1表示“完全保持这个”，而0表示“完全消除这个”。

让我们回到我们的例子，一个语言模型试图根据所有以前的语言模型预测下一个单词。 在这样的问题中，单元状态可以包括当前被摄体的性别，使得可以使用正确的代词。 当我们看到一个新的主题，我们想忘记旧主题的性别。

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, 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.



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

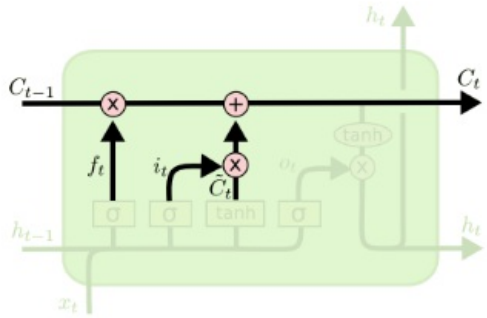
下一步是决定我们将要存储在单元状态中的新信息。 这有两个部分。 首先，称为“输入门层”的S形层决定我们将更新哪些值。 接下来，tanh层创建可以被添加到状态的新的候选值的向量C~tC~t。 在下一步中，我们将结合这两个来创建状态的更新。

在我们的语言模型的例子中，我们想要将新主题的性别添加到单元格状态，以替换我们忘记的旧主题。

It’s now time to update the old cell state, Ct−1Ct−1, into the new cell state CtCt. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by ftft, forgetting the things we decided to forget earlier. Then we add it∗C~tit∗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.



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

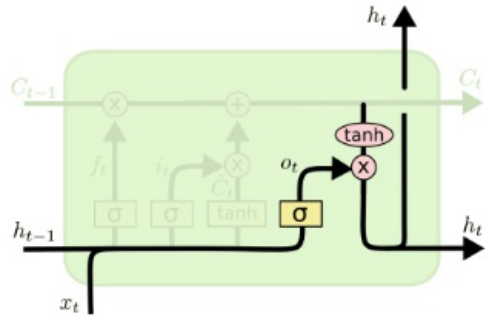
现在是时间更新旧的单元格状态，Ct-1，到新的单元格状态Ct。 以前的步骤已经决定了做什么，我们只需要实际做。

我们将旧状态乘以ft，忘记我们决定忘记的事情。 然后我们加上它* C⌢tit * C⌢t。 这是新的候选值，由我们决定更新每个状态值的程度进行调整。

在语言模型的情况下，这是我们实际删除关于旧主题的性别的信息，并添加新的信息，我们在前面的步骤中决定。

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 tanhtanh (to push the values to be between −1−1 and 11) 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.



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

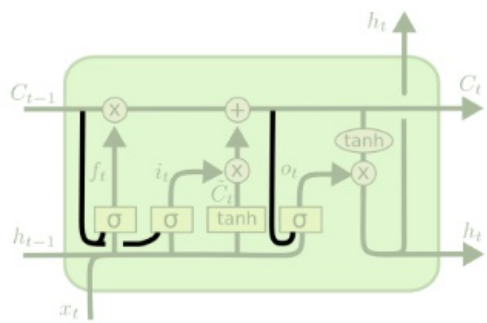
最后，我们需要决定我们要输出什么。 此输出将基于我们的单元格状态，但将是已过滤的版本。 首先，我们运行一个Sigmoid层，决定我们要输出的单元状态的哪些部分。 然后，我们将单元状态通过tanhtanh（推动值在-1-1和11之间），并乘以Sigmoid门的输出，以便我们只输出我们决定的部分。

对于语言模型示例，由于它只是看到一个主题，它可能想输出与动词相关的信息，以防万一这是下一步。 例如，它可以输出主语是单数还是复数，以便我们知道如果这是接下来的话，动词应该被结合到什么形式。

Variants on Long Short Term Memory

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.



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

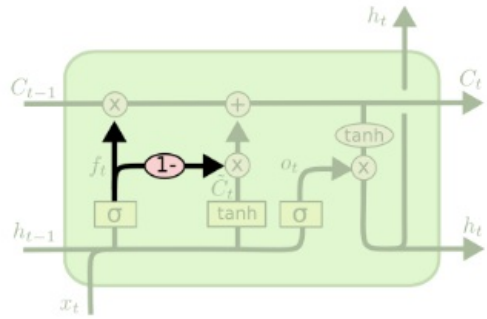
到目前为止，我描述的是一个非常正常的LSTM。 但并不是所有的LSTM都和上面的一样。 事实上，几乎每一篇涉及LSTM的论文都使用

略有不同的版本。 差异很小，但值得一提的是其中一些。

一个流行的LSTM变体，由Gers＆Schmidhuber（2000）介绍，增加了“窥视孔连接”。这意味着我们让栅极层看看单元状态。

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.

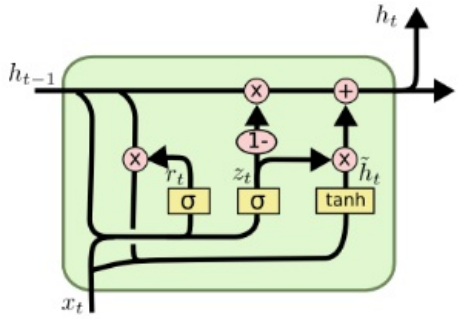


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

上面的图表添加窥视孔到所有的门，但许多文件将给一些窥视孔，而不是其他。

另一种变化是使用耦合的忘记和输入门。 而不是单独决定忘记什么，我们应该添加新的信息，我们一起做这些决定。 我们只会忘记当我们要在它的地方输入东西。 当我们忘记某些东西时，我们只向状态输入新的值。

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.



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

LSTM上稍微更显着的变化是由Cho等人介绍的门控循环单元（GRU）。（2014）。 它将忘记门和输入门组合成单个“更新门”。它还合并单元状态和隐藏状态，并进行一些其他改变。 结果模型比标准LSTM模型更简单，并且已经越来越流行。

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.
这些只是几个最显着的LSTM变体。 还有很多其他的，如Yao等人的Depth Gated RNNs。（2015）。 还有一些完全不同的解决长期依赖的方法，如Koutnik等人的Clockwork RNN。（2014）。

以下哪个变体是最好的？ 差异是否重要？ Greff，et al。（2015）做了一个很好的比较流行的变种，发现他们都是一样的。 Jozefowicz，et al。（2015）测试了一万多个RNN架构，发现一些在某些任务上比LSTM更好。

Conclusion

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!
早些时候，我提到了人们使用RNNs取得的显着成果。基本上所有这些都是使用LSTM实现的。他们真的为大多数任务更好地工作！

写成一组方程，LSTM看起来很吓人。希望，在这篇文章中一步一步地走过他们，使他们更加平易近人。

LSTM是我们可以用RNN完成的一个重要步骤。很自然地想知道：还有另一个大步吗？研究人员的一个共同看法是：“是的！有一个下一步，它的注意！”这个想法是让RNN的每一步都选择信息从一些更大的信息集合来看。例如，如果您使用RNN创建描述图像的字幕，它可能会选择图像的一部分来查看其输出的每个字。事实上，Xu，et al。（2015）做到这一点 - 这可能是一个有趣的起点，如果你想探索注意！有一些真正令人兴奋的结果，使用注意，它似乎更多的是在拐角处...

注意不是RNN研究中唯一令人兴奋的线索。例如，Kalchbrenner等人的Grid LSTMs（2015）似乎非常有前途。在生成模型中使用RNNs工作 - 如Gregor，et al。（2015），Chung，et al。（2015），或者Bayer和Osendorfer（2015） - 也似乎很有趣。最近几年一直是一个令人兴奋的时间，循环神经网络，而来的人承诺只有更多！

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.
我感谢一些人帮助我更好地了解LSTM，评论可视化，并提供对这篇文章的反馈。

我非常感谢Google的同事提供有用的反馈，特别是Oriol Vinyals，Greg Corrado，Jon Shlens，Luke Vilnis和Ilya Sutskever。 我也感谢许多其他朋友和同事花时间帮助我，包括Dario Amodei和Jacob Steinhardt。 我特别感谢Kyunghyun Cho非常周到的对我的图表。

在这篇文章之前，我在神经网络教授的两个系列研讨会上练习了解释LSTM。 感谢所有参与这些活动的人对我们的耐心，并感谢他们的反馈。