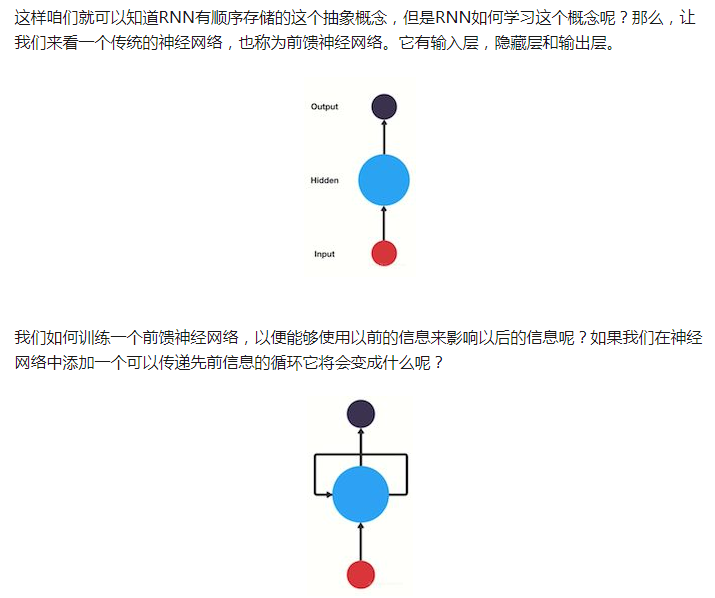
RNN

<https://zhuanlan.zhihu.com/p/45289691?utm_source=wechat_session&utm_medium=social&utm_oi=685930719541465088>

<https://zhuanlan.zhihu.com/p/28054589?utm_source=wechat_session&utm_medium=social&utm_oi=685930719541465088>

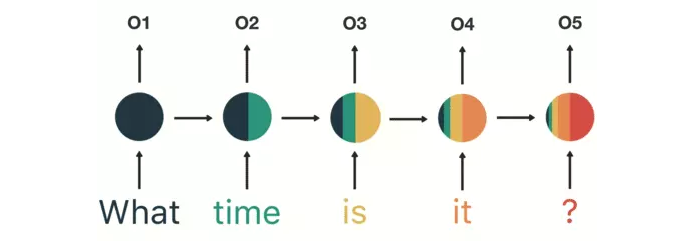
RNN是神经网络中的一种，它擅长对序列数据进行建模处理,音频是一种自然的序列，你可以将音频频谱图分成块并将其馈入RNN,文本也是一种形式的序列，你可以将文本分成一系列字符或一系列单词。

递归神经网络



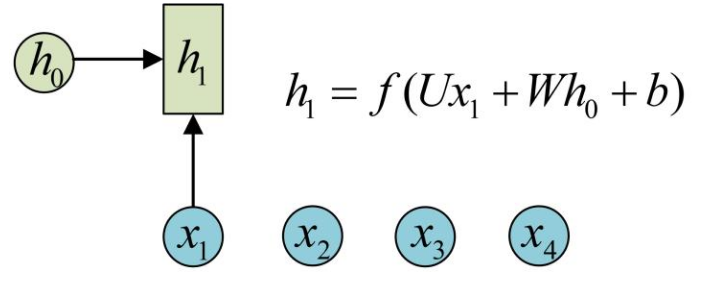
假设用户输入：what time is it？我们需要预测用户的意图，首先，我们将句子分解为单个单词。RNN按顺序工作，所以我们一次只能输入一个字。

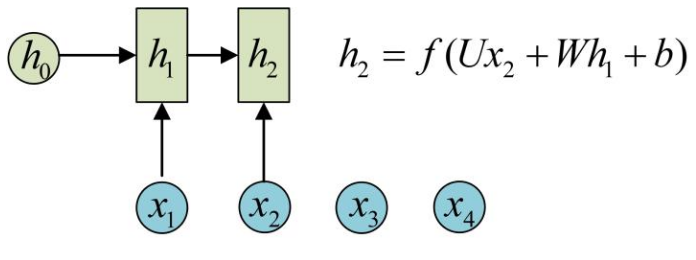
1. 第一步是将“What”输入RNN，RNN编码“what”并产生输出。
2. 对于下一步，我们提供单词“time”和上一步中的隐藏状态。RNN现在有关于“what”和“time”这两个词的信息。
3. 我们重复这个过程，直到最后一步。你可以通过最后一步看到RNN编码了前面步骤中所有单词的信息。



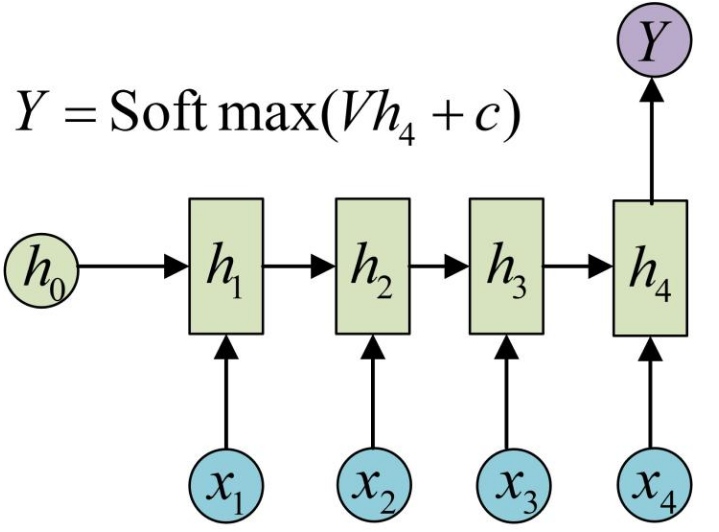
## N vs 1 结构

对于x1, x2, x3, x4 四个单词来说





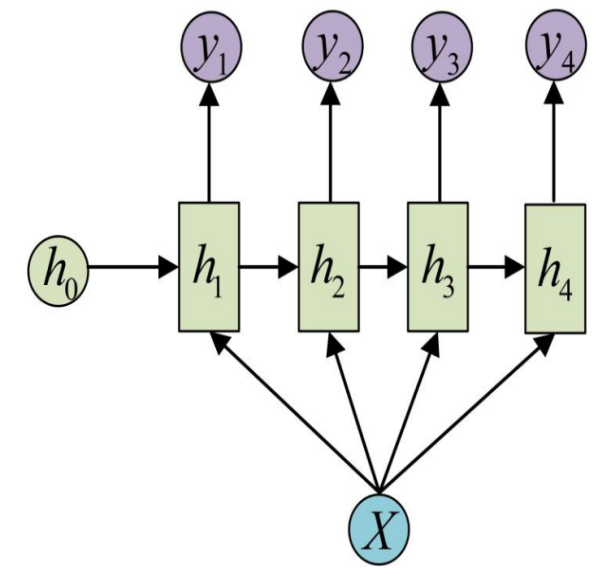
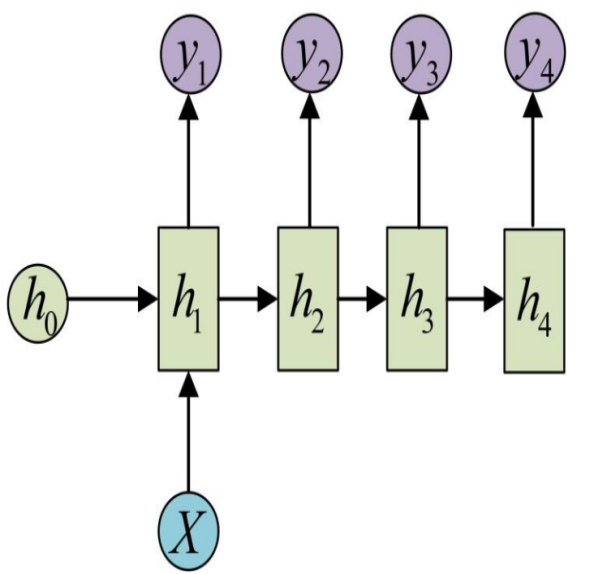
对于预测某个时间点的序列任务时，输出是一个单独的值而不是序列



这种结构通常用来处理序列分类问题。如输入一段文字判别它所属的类别，输入一个句子判断其情感倾向，输入一段视频并判断它的类别,股票预测等等。

## 1 vs N 结构

输入不是序列而输出为序列的情况怎么处理？我们可以只在序列开始进行输入计算，还有一种结构是把输入信息X作为每个阶段的输入，



这种1 VS N的结构可以处理的问题有：

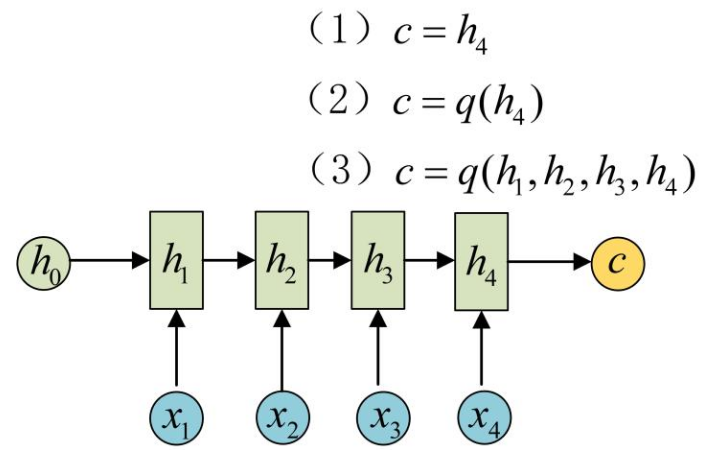
从图像生成文字（image caption），此时输入的X就是图像的特征，而输出的y序列就是一段句子

从类别生成语音或音乐等。

## N vs M 结构

下面我们来介绍RNN最重要的一个变种：N vs M。这种结构又叫Encoder-Decoder模型，也可以称之为Seq2Seq模型。

原始的N vs N RNN要求序列等长，然而我们遇到的大部分问题序列都是不等长的，如机器翻译中，源语言和目标语言的句子往往并没有相同的长度。为此，Encoder-Decoder结构先将输入数据编码成一个上下文向量c



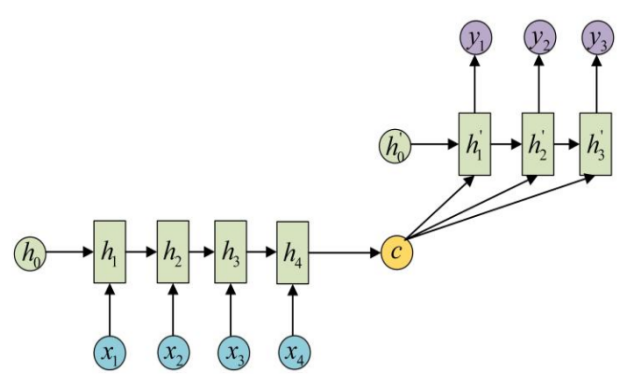
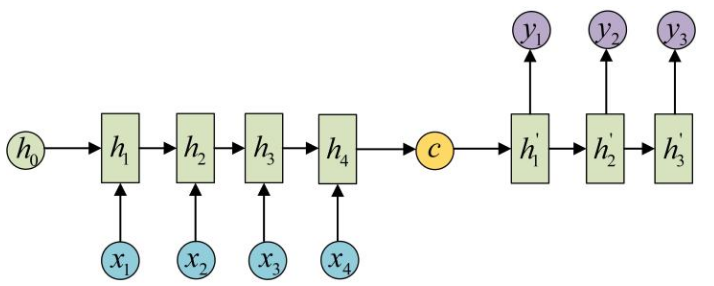
这一个过程就是Encoder，得到c有多种方式，

最简单的方法就是把Encoder的最后一个隐状态赋值给c，c=h4

还可以对最后的隐状态做一个变换得到c，c=q(h4)

也可以对所有的隐状态做变换。 c=q(h1,h2,h3,h4)

拿到c之后，就用另一个RNN网络对其进行解码，这部分RNN网络被称为Decoder。具体做法就是将c当做之前的初始状态h0输入到Decoder中，或者将c当做每一步的输入



由于这种Encoder-Decoder结构不限制输入和输出的序列长度，因此应用的范围非常广泛，比如：

机器翻译。Encoder-Decoder的最经典应用，事实上这一结构就是在机器翻译领域最先提出的

文本摘要。输入是一段文本序列，输出是这段文本序列的摘要序列。

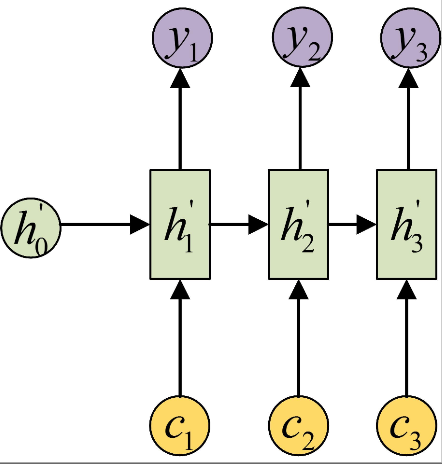
阅读理解。将输入的文章和问题分别编码，再对其进行解码得到问题的答案。

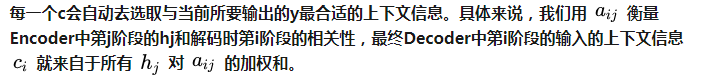
语音识别。输入是语音信号序列，输出是文字序列。

## Attention机制

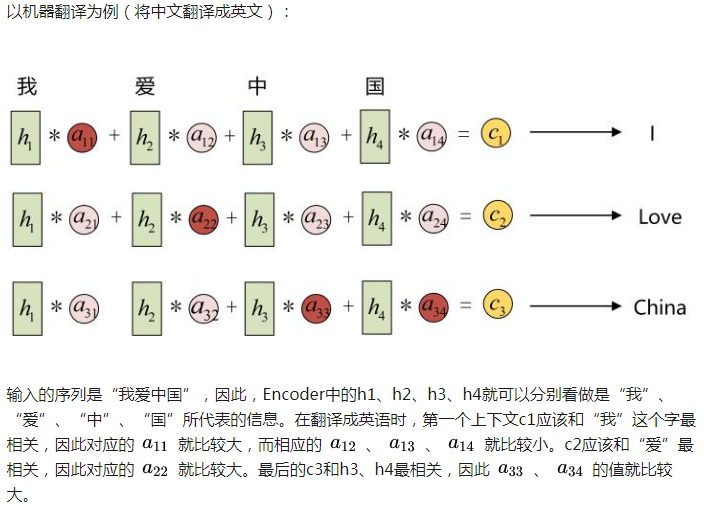
在Encoder-Decoder结构中，Encoder把所有的输入序列都编码成一个统一的语义特征c再解码，因此， c中必须包含原始序列中的所有信息，它的长度就成了限制模型性能的瓶颈。如机器翻译问题，当要翻译的句子较长时，一个c可能存不下那么多信息，就会造成翻译精度的下降。

Attention机制通过在每个时间输入不同的c来解决这个问题，下图是带有Attention机制的Decoder：

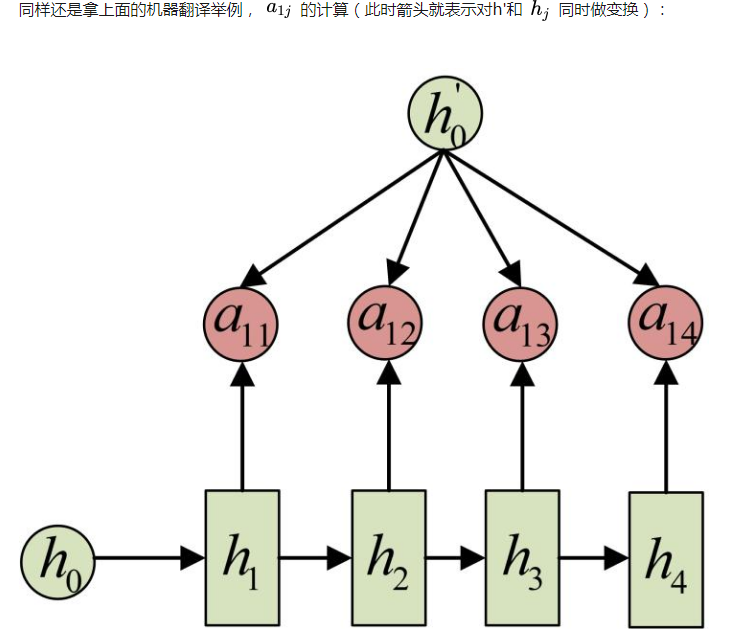




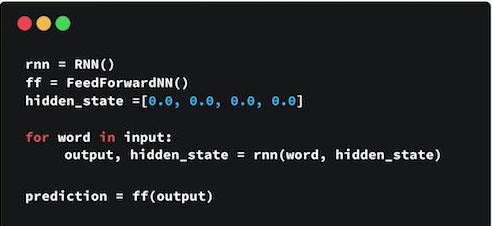
也就是说，在decode层输入Ci不在是全局信息而是与该节点最相关的信息。



事实上， aij 同样是从模型中学出的，它实际和Decoder的第i-1阶段的隐状态、Encoder第j个阶段的隐状态有关。



使用python展示了控制流程应该是最好的方式。



## 梯度消失

你可能已经注意到隐藏状态中奇怪的颜色分布。这是为了说明RNN被称为短期记忆的问题。



短期记忆问题是由臭名昭着的梯度消失问题引起的，这在其他神经网络架构中也很普遍。由于RNN处理很多步骤，因此难以保留先前步骤中的信息。正如你所看到的，在最后的时间步骤中，“what”和“time”这个词的信息几乎不存在。短期记忆和梯度消失是由于反向传播的性质引起的，反向传播是用于训练和优化神经网络的算法。为了理解这是为什么，让我们来看看反向传播对深度前馈神经网络的影响。

训练神经网络有三个主要步骤。

首先，它进行前向传递并进行预测。

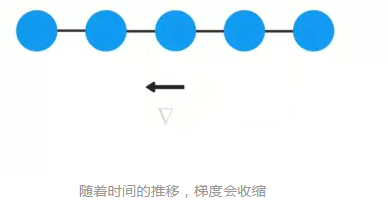
其次，它使用损失函数将预测与基础事实进行比较。损失函数输出一个错误值，该错误值是对网络执行得有多糟糕的估计。

最后，它使用该误差值进行反向传播，计算网络中每个节点的梯度。

梯度是用于调整网络内部权重的值从而更新整个网络。梯度越大，调整越大，反之亦然，这也就是问题所在。在进行反向传播时，图层中的每个节点都会根据渐变效果计算它在其前面的图层中的渐变。因此，如果在它之前对层的调整很小，那么对当前层的调整将更小。

这会导致渐变在向后传播时呈指数级收缩。由于梯度极小，内部权重几乎没有调整，因此较早的层无法进行任何学习。这就是消失的梯度问题。

让我们看看这如何适用于递归神经网络。你可以将循环神经网络中的每个时间步骤视为一个层，也就是说时间序列数据中，每一个样本都创建了一个神经元。为了训练一个递归神经网络，你使用了一种称为通过时间反向传播的方法。这样梯度值在每个时间步长传播时将呈指数级收缩。



由于梯度消失，RNN不会跨时间步骤学习远程依赖性。这意味着在尝试预测用户的意图时，有可能不考虑“what”和“time”这两个词。然后网络就可能作出的猜测是“is it？”。这很模糊，即使是人类也很难辨认这到底是什么意思。因此，无法在较早的时间步骤上学习会导致网络具有短期记忆。

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。

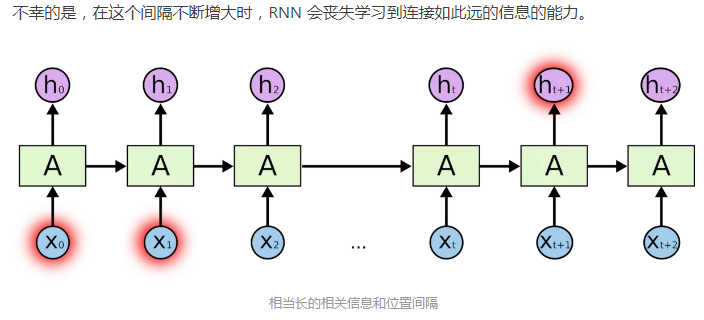
而这些成功应用的关键之处就是 LSTM 的使用，这是一种特别的 RNN，比标准的 RNN 在很多的任务上都表现得更好。几乎所有的令人振奋的关于 RNN 的结果都是通过 LSTM 达到的。这篇博文也会就 LSTM 进行展开。

## 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。

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。



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。

然而，幸运的是，LSTM 并没有这个问题

## LSTM 网络

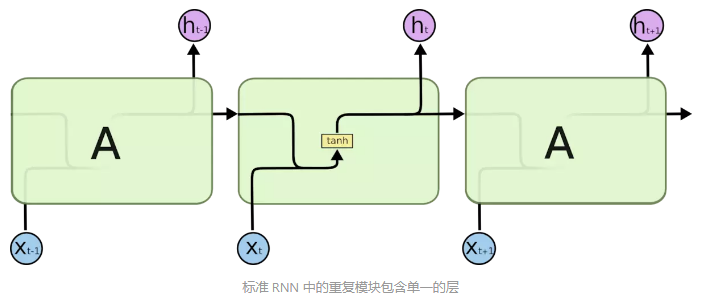
<https://www.jianshu.com/p/9dc9f41f0b29>

<https://blog.csdn.net/wangyanbeilin/article/details/81350683>

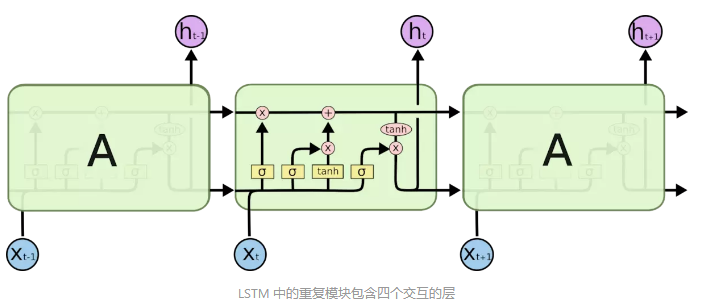
Long Short Term 网络—— 一般就叫做 LSTM ——是一种 RNN 特殊的类型，可以学习长期依赖信息。LSTM 由Hochreiter & Schmidhuber (1997)提出，并在近期被Alex Graves进行了改良和推广。在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛的使用。

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



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需要引入cell的概念来解释，每一个输入元素都有一个cell，每个cell里面有4个神经元来决定cell的状态，反观RNN只有一个简单的非线性的神经元。





Neural network layer：神经网络层，可以看作一个个的非线性处理模块。

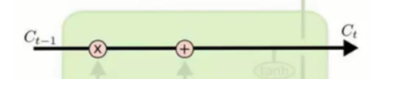
Pointwise operation：逐点运算，逐点运算举个逐点乘法的例子就是：0.5×[1., 2., 3.] = [0.5, 1.0, 1.5]。

Vector transfer：信息传递方向。

Concatenate：这个很好理解，信息的汇合。

Copy：与上面对应，信息的复制。

LSTM的关键：cell state 即细胞的状态。先讲它如何运作，再讲如何控制。如何运作？我们只截取这一小部分，就是从上一时刻的记忆到这一时刻的记忆的一个过程。



就是它记忆的信息就在这条传送带上从后往前传，传送的时候会发生一些信息的交互，信息就在这上面一直保存。我可以在这条传送带上取值，也可以在上面输入值。 Ct是memory，也就是记忆，是所学到的全部记忆，只不过在不同的cell其更新的内容不同。需要注意Ct和Ht的不同。Ct是全局的记忆，贯穿整个学习过程，Ht是当前cell输出的状态

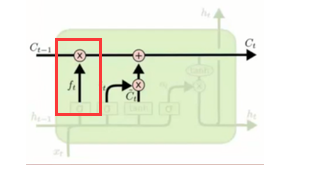
如何控制cell state？我们会通过一个叫‘门’gate的东西来处理它，会给信息进行一个选择性的放行，来去除或者增加信息。先放结论：它包含着什么？包含一个sigmoid+一个pointwise。

先看sigmoid，它的输出结果会得出一个概率p，是0-1的一个值。我要做的就是一个信息的变更，到底让不让这部分信息，（前提：我的脑容量就这么大，只能记忆这么多东西，再多我记不住了）让他们接着往下存下去，还是说这部分记忆就没用了就更新了？（这里补充一下，忘记信息之后要更新）肯定要有个东西来控制它，这里我们控制的就是一个sigmoid，描述每个部分有多少量可以通过。这个概率可以和任何一个东西相乘，表示我允许你多大的量可以通过。0表示不需任何量通过，1表示允许任何量通过。

然后再看，LSTM是怎么样用这些门一起去串出来这样一个东西的。

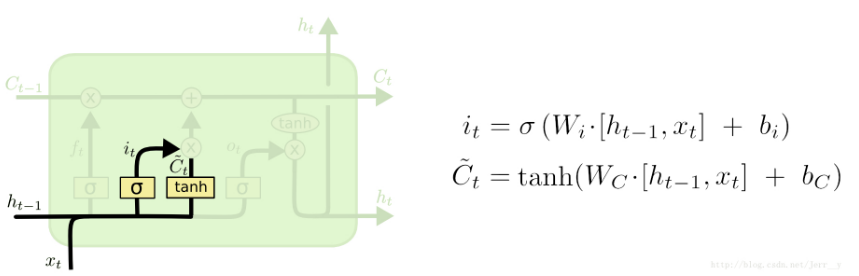
第一步，我会决定从cell state中丢掉什么信息，这里有个‘遗忘门’。比如我一开始填我导的职称是副教授，后面我导升成教授了，我好像下次填职称的时候要填教授了，所以我需要用忘记门来忘记旧的信息（忘记她是副教授），更新新的信息（下次再填的时候就填教授）。





所以我会用上一时刻的输出ht-1（旧的记忆）和现在的输入xt（新的知识），来一起去决定我以多大的程度来忘记这个内容。б是个sigmoid。输出一个0-1的值，也就是ft要以多大的概率留下这个信息，这个式子产出一个概率值。W和b都是参数，会训练得到的。

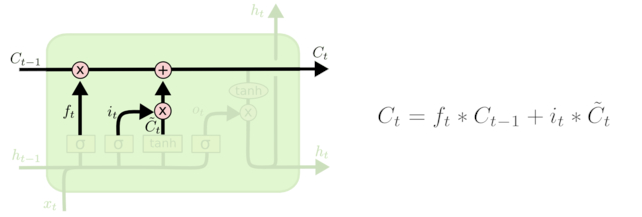
第二步，输入门，我们把一部分信息用忘记门过滤掉了，就应该补一部分新的信息过来到cell state中，也就是刚才说的传送带上加什么信息



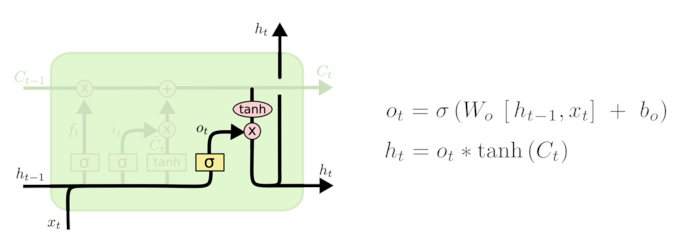
需要两个值，it和Ct，它们一个是概率，一个是记忆。具体解释一下：it也是范围0-1的一个概率值p，it做的事就是对目前为止学到的所有信息做一个过滤，它的概率值表明，现在学到的哪部分新知识可以更新我之前的记忆的，也就是拿它过滤本次记忆。而本次学到的所有知识就是Ct,他的形式和RNN一样的。要把本次学到的信息，放到之前学到的所有信息中。所以用it这个概率，对Ct做一个过滤，补充到之前学习到的信息中。

从旧的概率里，以一个0-1的概率值，去保留下来我要的一部分信息，同时我把本次学习的信息筛选出一部分补到我之前学习的知识体系里，形成了现在的知识体系

遗忘门和更新门主要的作用目标是全局记忆Ct-1，Ct-1是旧的知识，ft是旧的记忆的通过率，就是我们的gate门。it是本次信息的筛选器，Ct的是本次更新后的知识



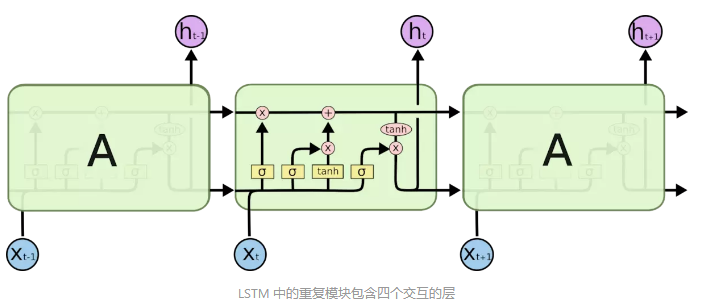
第三步，输出门，Ct是我更新以后之前所有学到的全部知识，（而我解决当前需解决的问题的时候，只需要某些知识，我只是要把我需要的知识筛选出来，于是我拿一个ot去筛选），ot依旧是0-1之间的概率p，它会从Ct所有知识里头Ct去筛选出来解决当前问题的信息，然后给出结果



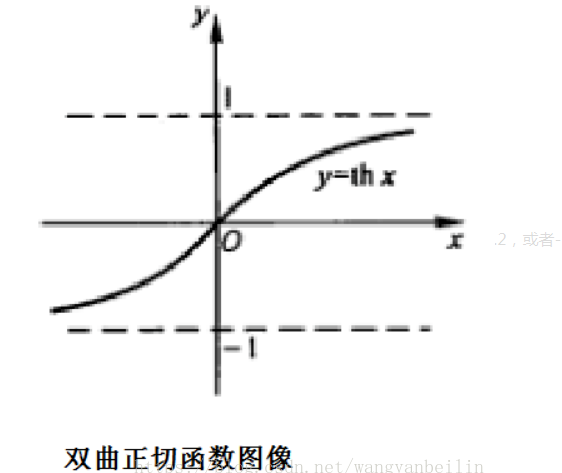
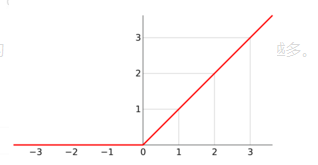
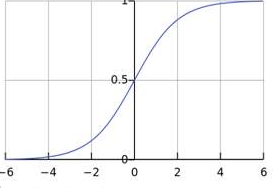
这一步与前面两步的不同在于，其目标是得出当前cell的输出状态Ht，Ht由全局记忆和上一个cell的状态Ht-1以及本次的输入Xt得到，就是说本次输出对比Rnn他考虑的最新的全局记忆信息Ct，这是LSTM与RNN最大的不同，前面两个门都是在更新全局记忆Ct，这一个门才算是真的学习了并输出了该cell的状态

讲完了LSTM的步骤，下面是要讨论的核心问题：

1. 为什么LSTM比RNN要有优势，能够缓解长时依赖的问题。



Sigmoid函数在学习过程主要的作用就是控制信息的流入流出的多少，将其限制在0到1的范围之内



Sigmoid会饱和，确实，可能会带来问题。注意看公式，0-1才可以起到筛选的作用，如果用relu信息就会全过去了，信息它可能会爆炸，每层都会变大，下一层更大，就做不到把每层的信息都压缩到一个共同的范围了。用tanh的话，它双曲正切你想一下图像，它可能会出现一个负值，如-0.2，或者-0.5之类的，就会否定掉之前的全部的

1. 为什么LSTM比RNN更能解决长时依赖？

RNN怎么修改St的？ St=tanh（WXt+USt-1）这是个复合函数，复合函数求偏导是连乘的形式，我用的是双曲正切，双曲正切在x偏大和偏小的时候斜率是接近于0的，所以接近于0会让反向传播的时候，St约等于0，rnn什么都学不到了，因为没有梯度传回来了，也就是没有梯度来校正参数W了。

而什么时候会出现≈0呢，就是链路非常长的时候，就可能带来梯度消失，也就是梯度弥散。这个时候求导的链式法则是让无数东西的梯度相乘，当有一个≈0时，整个式子的结果就会≈0，当我的链路越长时，越有可能出现≈0的情况。

看LSTM，LSTM不是复合函数，是两个函数求和，f(x)+g(x)求偏导的话，得到的是两个偏导的和，即使有一个≈0，它不会导致整体约等于0。所以我的梯度往回传的时候是沿着两条轴往回传的。梯度不是顺着一条轴往回传的，LSTM最大的变化就是把RNN的连乘变成了求和，因此，不再会严重地出现梯度消失问题，这意味着即使时间再远，我应该也是学得到的。

以上的公式都不是必须只能这样，有很多变种（基本上没差异），但这篇总结写的是标准的那一种。