# 深度解析LSTM神经网络的设计原理

原创 夕小瑶 夕小瑶的卖萌屋 2017-10-25

来自专辑
卖萌屋@自然语言处理

**引人入胜的开篇：**
想要搞清楚LSTM中的每个公式的每个细节为什么是这样子设计吗?想知道simple RNN是如何一步步的走向了LSTM吗？觉得LSTM的工作机制看不透？恭喜你打开了正确的文章！

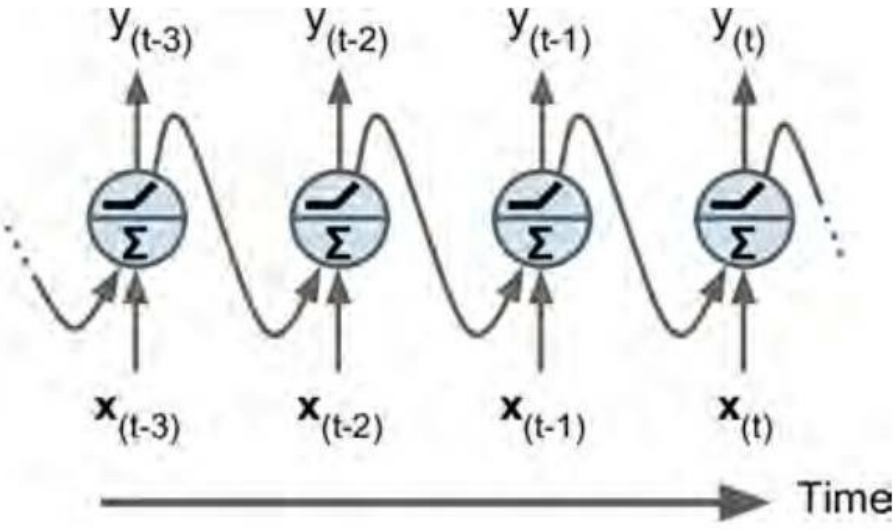
**前方核弹级高能预警！本文信息量非常大，文章长且思维连贯性强，建议预留20分钟以上的时间进行阅读。**

**前置知识1:**
在上一篇文章《前馈到反馈：解析RNN》中，小夕从最简单的无隐藏层的前馈神经网络引出了简单的循环神经网络：

$$y(t) = f(X(t) * W + y(t-1) * V + b)$$

它就是**无隐藏层的**循环神经网络，起名叫**“simple RNN”**。

这种方式即在每个时刻做决策的时候都考虑一下上一个时刻的决策结果。画出图来就是酱的：




（其中圆球里的下半球代表两向量的内积，上半球代表将内积结果激活）

虽然通过这种简单反馈确实可以看出**每个时间点的决策会受前一时间点决策的影响**，但是似乎很难让人信服这竟然能跟记忆扯上边！

想一下，人的日常行为流程是这样的。比如你在搭积木，那么每个时间点你的行为都会经历下面的子过程：

1、眼睛看到现在手里的积木。
2、回忆一下目前最高层的积木的场景。

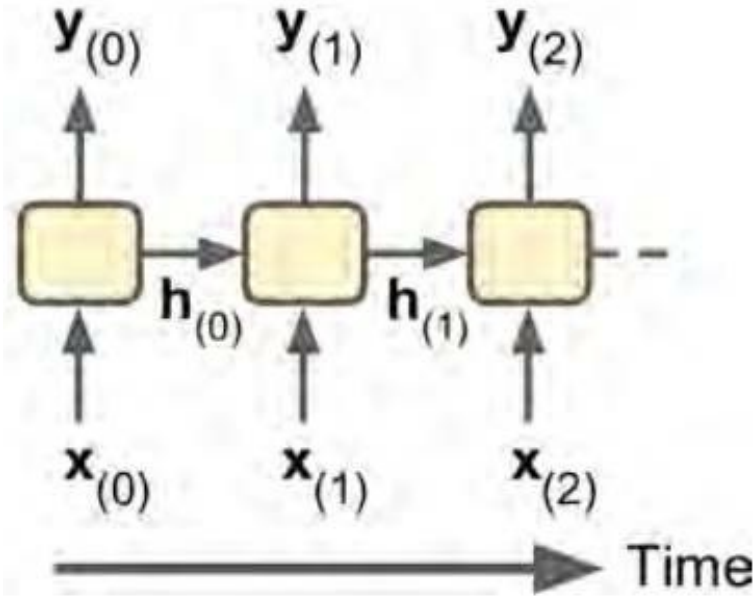
3、结合1和2的信息来做出当前时刻积木插到哪里的决策。

相信聪明的小伙伴已经知道我要表达的意思啦。第1步手里的积木就是当前时刻的外部输入X；第2步就是调用历史时刻的信息/记忆；第3步就是融合X和历史记忆的信息来推理出决策结果，即RNN的一步前向过程的输出y(t)。

有没有更加聪明的小伙伴惊奇的注意到第2步！！！我们在回忆历史的时候，一般不是简单的回忆上一个积木的形状，而是去回忆一个更加**模糊而宏观**的场景。在这个例子中，这个场景就是最近几次行为所产生出的抽象记忆——即“积木最高层的地形图”！

也就是说，人们在做很多时序任务的时候，尤其是稍微复杂的时序任务时，潜意识的做法**并不是直接将上个时刻的输出y(t-1)直接连接进来，而是连接一个模糊而抽象的东西进来**！这个东西是什么呢？

当然就是神经网络中的**隐结点h**啊！也就是说，人们潜意识里直接利用的是一段历史记忆融合后的东西h，而不单单是上一时间点的输出。而网络的输出则取自这个隐结点。所以更合理的刻画人的潜意识的模型应该是这样的：




（记忆在隐单元中存储和流动，输出取自隐单元）

这种加入了隐藏层的循环神经网络就是经典的RNN神经网络！即“standard RNN”。

RNN从simple到standard的变动及其意义对于本文后续内容非常重要哦。

**前置知识2:**

在上一篇文章《从前馈到反馈：循环神经网络（RNN）》中简单讲解和证明过，由于在误差反向传播时，算出来的梯度会随着往前传播而发生**指数级**的衰减或放大！而且这是在数学上板上钉钉的事情。因此，**RNN的记忆单元是短时的**。

好啦，那我们就借鉴前辈设计RNN的经验，从**simple**版本开始，即无隐藏层的、简单完成输出到输入反馈的网络结构开始，去设计一个全新的、**可以解决梯度爆炸消失问题从而记住长距离依赖关系**的神经网络吧！

那么**如何让梯度随着时间的流动不发生指数级消失或者爆炸呢？**

好像想起来挺难的，但是这个问题可能中学生会解答！那就是让算出来的梯度恒为1！**因为1的任何次方都是1嘛**(￣▽￣)

所以按照这个搞笑的想法，我们把要设计的长时记忆单元**记为c（以下全部用c指代长时记忆单元）**，那么我们设计出来的长时记忆单元的数学模型就是这样子喽：

c(t) = c(t-1)

这样的话，误差反向传播时的导数就恒定为1啦～误差就可以一路无损耗的向前传播到网络的前端，从而学习到遥远的前端与网络末端的远距离依赖关系。


路人：Excuse me？

不要急不要急，反正假设我们的c中存储了信息，那么c就能把这个信息一路带到输出层没问题吧？在T时刻算出来的梯度信息存储在c里后，它也能把梯度一路带到时刻0而无任何损耗也没问题吧？对吧(￣▽￣)

所以信息的**运输问题解决了**，那么就要解决对信息进行**装箱**和**卸车**的问题。

先来看装箱问题，即**如何把新信息写入c里面去呢？**

当然要先定义一下新信息是什么。不妨直接拿来simple RNN中对新信息的定义，即当前时刻的外部输入x(t)与前一时刻的网络输出（即反馈单元）y(t-1)联合得到网络在当前这一时刻get到的新信息，记为$\hat{c}(t)$。即：

$$\hat{c}(t) = f(W \cdot x(t-1) + V * \cdot y(t-1))$$

好，新信息$\hat{c}(t)$定义完成。下面考虑把$\hat{c}(t)$加到c里面去。如果把这个问题拿去问小学生的话，那么可能会兵分两路：

**1、乘进去！**
**2、加进去！**

那么这两种哪种可行呢？

其实稍微一想就很容易判断：乘法操作更多的是作为一种对信息进行某种控制的操作（比如任意数与0相乘后直接消失，相当于关闭操作；任意数与大于1的数相乘后会被放大规模等），而加法操作则是新信息叠加旧信息的操作。

下面我们深入的讨论一下**乘性操作**和**加性操作**，这在理解LSTM里至关重要。当然，首先，你要掌握偏导的概念和方

法、复合函数的求导法则、链式求导法则。有了这三点微积分基础后才能看懂哦。

（害怕数学和基础不够的童鞋可以跳过这里的论乘法和论加法小节。）

**论乘法：**

乘法时即令长时记忆添加信息时的数学模型为：

$$c(t) = c(t-1) * \hat{c}(t)$$

因此网络完整数学模型如下：

$$c(t) = c(t-1) * \hat{c}(t) \quad \text{公式【0.1】}$$
$$\hat{c}(t) = f(W \cdot x(t) + V \cdot y(t-1)) \quad \text{公式【0.2】}$$
$$y(t) = f(c(t)) \quad \text{公式【0.3】}$$

为了计算方便，还是像之前一样**假设激活函数为线性激活**（即没有激活函数。实际上tanh在小值时可以近似为线性，relu在正数时也为线性，这个假设还是很无可厚非的），这时网络模型简化为：

$$y(t) = y(t-1) * (W \cdot x(t) + V \cdot y(t-1)) \quad \text{【1】}$$

假如网络经过了T个时间步到了loss端，这时若要更新t=0时刻下网络参数V的权重，则即对t=0时刻的参数V求偏导，即计算

$$\frac{\partial loss(t=T)}{\partial V(t=0)}$$

其中

$$loss(t=T) = f_{loss}(y(t=T))$$

（其中的f_loss(·)为损失函数）

好，稍微一算，发现$\frac{\partial loss(t=T)}{\partial V(t=0)} = f'_{loss} * y'(t=T)$中的f_loss'的值就是我们要往前传的梯度（参数更新信息），则我们的目标就是讨论y'(t=T)，写全了就是

$$\frac{\partial y(t=T)}{\partial V(t=0)} \quad \text{【2】}$$

对V求偏导时其他变量（就是说的W和x）自然也就成了常量，这里我们再做一个过分简化，直接删掉$W \cdot x(t-1)$项！（在y二阶乘方存在的情况下忽略一阶乘方），这时就可以直接展开公式【1】：

$$y(T) = y(T-1)^2 \cdot v(T-1)$$
$$= v(0)^{2^T-1} \cdot y(0)^{2^{T-2}}$$

对v(0)求导的话，会得到

$$y' = y(0)^{2^{T-2}} \cdot (2^T - 1) \cdot v(0)^{2^T-2}$$
$$= a \cdot v(0)^{2^T-2}$$

如果说RNN的$v^T$是音速级的梯度爆炸和消失，那这$v^{2^T-2}$简直是光速级爆炸和消失了呐！～

**所以说直接将历史记忆乘进长时记忆单元只会让情况更糟糕，**导致当初c(t)=c(t-1)让导数恒为1的构想完全失效，**这也说明了乘性更新并不是简单的信息叠加，而是控制和scaling。**

**论加法：**

如果改成加性规则呢？此时添加信息的数学模型为

$$c(t) = c(t-1) + \hat{c}(t)$$

与前面的做法一样，假设线性激活并代入网络模型后得到

$$\begin{aligned} &y(T) = y(T-1) + \hat{c}(T) \\ &= y(T-1) + x * w + v(T-1) * y(T-1) \\ &= (v(0)+1)^T * y(0) + T * x * w \end{aligned}$$

噫？也有指数项～不过由于v加了一个偏置1，导致爆炸的可能性远远大于消失。不过通过做梯度截断，也能很大程度的缓解梯度爆炸的影响。

嗯～梯度消失的概率小了很多，梯度爆炸也能勉强缓解，看起来比RNN靠谱多了，毕竟控制好爆炸的前提下，梯度消失的越慢，记忆的距离就越长嘛。

因此，在往长时记忆单元添加信息方面，加性规则要显著优于乘性规则。**也证明了加法更适合做信息叠加，而乘法更适合做控制和scaling。**

由此，我们就确定应用加性规则啦，至此我们设计的网络应该是这样子的：

$$c(t) = c(t-1) + \hat{c}(t) \quad 【3.1】$$
$$\hat{c}(t) = f(W \cdot x(t) + V \cdot y(t-1)) \quad 【3.2】$$
$$y(t) = f(c(t)) \quad 【3.3】$$

那么有没有办法**让信息装箱和运输同时存在的情况下，让梯度消失的可能性变的更低，让梯度爆炸的可能性和程度也更低呢？**

你想呀，我们往长时记忆单元添加新信息的频率肯定是很低的，现实生活中只有很少的时刻我们可以记很久，大部分时刻的信息没过几天就忘了。因此现在这种模型一股脑的试图永远记住每个时刻的信息的做法肯定是不合理的，**我们应该只记忆该记的信息。**

显然，对新信息选择记或者不记是一个控制操作，应该使用乘性规则。因此在新信息前加一个控制阀门，只需要让公式【3.1】变为

$$c(t) = c(t-1) + g_{in} * \hat{c}(t)$$

这个$g_{in}$我们就叫做**“输入门”**啦，取值0.0～1.0。

为了实现这个取值范围，我们很容易想到使用**sigmoid函数**作为输入门的激活函数，毕竟sigmoid的输出范围一定是

在0.0到1.0之间嘛。因此以输入门为代表的控制门的激活函数均为sigmoid，因此控制门：

$$g_x = sigmoid(...)$$

当然，这是对一个长时记忆单元的控制。我们到时候肯定要设置很多记忆单元的，要不然脑容量也太低啦。因此每个长时记忆单元都有它专属的输入门，在数学上我们不妨**使用⊗来表示这个按位相乘的操作，用大写字母C来表示长时记忆单元集合。**即：

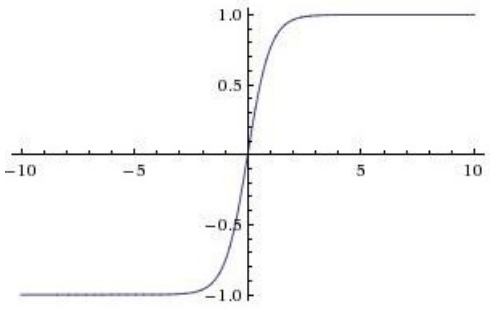
$$C(t) = C(t-1) + g_{in} \otimes \hat{C}(t) \quad 【4】$$

嗯～由于输入门只会在必要的时候开启，因此大部分情况下公式【4】可以看成
C(t)=C(t-1)，也就是我们最理想的状态。由此加性操作带来的梯度爆炸也大大减轻啦，梯度消失更更更轻了。

等等，爱思考的同学可能会注意到一个问题。万一神经网络读到一段信息量很大的文本，以致于这时**输入门欣喜若狂，一直保持大开状态，狼吞虎咽的试图记住所有这些信息，会发生什么呢？**

显然就会导致c的值变的非常大！

要知道，我们的网络要输出的时候是要把c激活的（参考公式【0.3】），当c变的很大时，sigmoid、tanh这些常见的激活函数的输出就完全饱和了！比如如图tanh：




当c很大时，tanh趋近于1，这时c变得再大也没有什么意义了，因为饱和了！脑子记不住这么多东西！

这种情况怎么办呢？显然relu函数这种正向无饱和的激活函数是一种选择，但是我们总不能将这个网络输出的激活函数限定为relu吧？那也设计的太失败啦！

那怎么办呢？

其实想想我们自己的工作原理就知道啦。我们之所以既可以记住小时候的事情，也可以记住一年前的事情，也没有觉得脑子不够用，不就是因为我们。。。**爱忘事嘛**。所以还需要加一个门用来忘事！这个门就叫做**“遗忘门”**吧。**这样每个时刻到来的时候，记忆要先通过遗忘门忘掉一些事情再考虑要不要接受这个时刻的新信息。**

显然，遗忘门是用来控制记忆消失程度的，因此也要用乘性运算，即我们设计的网络已进化成：

$$c(t) = g_{forget} c(t-1) + g_{in} * \hat{c}(t)$$

或者向量形式的：

$$C(t) = g_{forget}C(t-1) + g_{in} \otimes \hat{C}(t)$$

好啦～解决了如何为我们的长时记忆单元可控的添加新信息的问题，又贴心的考虑到并优雅解决了信息输入太过丰富导致输入控制门“合不拢嘴”的尴尬情况，那么是时候考虑我们的长时记忆单元如何输出啦～

有人说，输出有什么好考虑的，**当前的输出难道不就仅仅是激活当前的记忆吗？**难道不就是最前面说的y(t)=f(c(t))？（其中f(·)为激活函数）

试想，假如人有1万个长时记忆的脑细胞，每个脑细胞记一件事情，那么我们在处理眼前的事情的时候是每个时刻都把这1万个脑细胞里的事情都回忆一遍吗？显然不是呀，**我们只会让其中一部分跟当前任务当前时刻相关的脑细胞输出**，即应该给我们的长时记忆单元添加一个输出阀门！也就是说应该输出：

$$y(t) = g_{out} * f(c(t))$$

嗯～终于看起来好像没有什么问题了。

那么我们最后再**定义一下控制门们（输入门、遗忘门、输出门）受谁的控制就可以啦。**

这个问题也很显然，当然就是让各个门受当前时刻的外部输入x(t)和上一时刻的输出y(t-1)啦，即
$g_x(t) = f(W \cdot x(t) + V \cdot y(t-1))$。。。。。。？

好像这样的思维在RNN中并不会有什么问题，但！是！不要忘了在我们这个新设计的网络中，多了一堆阀门！尤其注意到输出门，**一旦输出门关闭，就会导致其控制的记忆f(c(t))被截断，下一时刻各个门就仅仅受当前时刻的外部输入x(t)控制了**！这显然不符合我们的设计初衷（尽可能的让决策考虑到尽可能久的历史信息）。怎么办呢？

最简单的做法就是再把长时记忆单元接入各个门，即把上一时刻的长时记忆c(t-1)接入遗忘门和输入门，把当前时刻的长时记忆c(t)接入输出门（当信息流动到输出门的时候，当前时刻的长时记忆已经被计算完成了）。即

$$g_{in}(t) = sigm(W \cdot x(t) + V \cdot y(t-1) + U \cdot c(t-1))$$
$$g_{forget}(t) = sigm(W \cdot x(t) + V \cdot y(t-1) + U \cdot c(t-1))$$
$$g_{out}(t) = sigm(W \cdot x(t) + V \cdot y(t-1) + U \cdot c(t))$$

当然，这个让各个门考虑长时记忆的做法是后人打的补丁，这些从长时记忆单元到门单元的连接被称为**"peephole（猫眼）"**。

至此还有什么问题吗？看起来真没有问题啦～我们设计的**simple版**的网络就完成啦，总结一下，即：

$$C(t) = g_{forget}C(t-1) + g_{in} \otimes \hat{C}(t)$$
$$\hat{C}(t) = f(W \cdot x(t) + V \cdot y(t-1))$$
$$y(t) = g_{out} \otimes f(C(t))$$

$$g_{in}(t) = sigm(W \cdot x(t) + V \cdot y(t-1) + U \cdot C(t-1))$$
$$g_{forget}(t) = sigm(W \cdot x(t) + V \cdot y(t-1) + U \cdot C(t-1))$$
$$g_{out}(t) = sigm(W \cdot x(t) + V \cdot y(t-1) + U \cdot C(t))$$

就起名叫“**门限simple RNN**”吧！（非学术界认可）

然而，作为伟大的设计者，怎么能止步于simple呢！**我们要像simple　RNN推广出standardRNN的做法那样，推广出我们的standard版本！即加入隐藏层！**

为什么要加隐藏层已经在本文开头提到了，这也是simpleRNN到standardRNN的核心区别，这也是RNN及其变种可以作为深度学习的主角之一的原因。模仿RNN的做法，我们直接用隐藏层单元h来代替最终输出y：

$$C(t) = g_{forget}C(t-1) + g_{in} \otimes \hat{C}(t)$$
$$\hat{C}(t) = f(W \cdot x(t) + V \cdot h(t-1))$$
$$y(t) = g_{out} \otimes f(C(t))$$

$$g_{in}(t) = sigm(W \cdot x(t) + V \cdot h(t-1) + U \cdot C(t-1))$$
$$g_{forget}(t) = sigm(W \cdot x(t) + V \cdot h(t-1) + U \cdot C(t-1))$$
$$g_{out}(t) = sigm(W \cdot x(t) + V \cdot h(t-1) + U \cdot C(t))$$

显然，由于h随时都可以被输出门截断，所以我们可以很感性的把h理解为短时记忆单元。

而从数学上看的话，更是短时记忆了，因为梯度流经h的时候，经历的是h(t)->c(t)->h(t-1)的连环相乘的路径（在输入输出门关闭前），显然如前边的数学证明中所述，这样会发生梯度爆炸和消失，而梯度消失的时候就意味着记忆消失了，即h为短时记忆单元。

同样的思路可以再证明一下，由于梯度只从c走的时候，存在一条无连环相乘的路径，可以避免梯度消失。又有遗忘门避免激活函数和梯度饱和，因此c为长时记忆单元。

好啦，我们standard版本的新型网络也完成了！有没有觉得信息量超级大，又乱掉了呢？不要急，贴心的小夕就再带你总结一下我们这个**网络的前馈过程：**

新时刻t刚刚到来的时候，
1、首先长时记忆单元c(t-1)通过遗忘门g_forget去遗忘一些信息。
2、其中g_forget受当前时刻的外部输入x(t)、上一时刻的输出(短时记忆)h(t-1)、上一时刻的长时记忆c(t-1)的控制。
3、然后由当前时刻外部输入x(t)和上一时刻的短时记忆h(t-1)计算出当前时刻的新信息$\hat{c}(t)$。
4、然后由输入门g_in控制，将当前时刻的**部分**新信息$\hat{c}(t)$写入长时记忆单元，产生新的长时记忆c(t)。
5、其中g_in受x(t)、h(t-1)、c(t-1)的控制。
6、激活长时记忆单元c(t)，准备上天（输出）。
7、然后由输出门g_out把控，将至目前积累下来的记忆c(t)选出部分相关的记忆生成这一时刻我们关注的记忆h(t)，再把这部分记忆进行输出y(t)。
8、其中输出门g_out受x(t)、h(t-1)和当前时刻的长时记忆c(t)的控制。

前馈的过程写完了，梯度反传的过程就让深度学习平台去自动求导来完成吧～有M倾向的同学可以尝试对上述过程进

行手动求导。

**好啦，最后对全文的设计过程总结一下：**

1、我们为了解决RNN中的梯度消失的问题，为了让梯度无损传播，想到了c(t)=c(t-1)这个朴素却没毛病的梯度传播模型，我们于是称c为“长时记忆单元”。
2、然后为了把新信息平稳安全可靠的**装入**长时记忆单元，我们引入了“输入门”。
3、然后为了解决新信息装载次数过多带来的**激活函数饱和**的问题，引入了“遗忘门”。
4、然后为了让网络能够**选择合适的记忆进行输出**，我们引入了“输出门”。
5、然后为了解决记忆被**输出门截断**后使得各个门单元**受控性降低**的问题，我们引入了“peephole”连接。
6、然后为了将神经网络的简单反馈结构升级成**模糊历史记忆**的结构，引入了隐单元h，并且发现h中存储的模糊历史记忆是短时的，于是记h为短时记忆单元。
7、于是该网络既具备长时记忆，又具备短时记忆，就干脆起名叫**“长短时记忆神经网络(Long Short Term Memory Neural Networks，简称LSTM)“**啦。

（呼～历时三天终于完稿了。将12000字的手稿强行压缩到了5600字，将初稿里5、6个啰哩啰嗦的故事全都删了。天，我只想说，我再也不抱着讲透彻的想法给别人讲解LSTM了！！！

**参考文献：**


1. Hochreiter S, Schmidhuber J. Long Short-Term Memory[J]. Neural Computation, 1997, 9(8): 1735-1780.
2. Gers F A, Schmidhuber J, Cummins F, et al.Learning to Forget: Continual Prediction with LSTM[J]. Neural Computation,2000, 12(10): 2451-2471.
3. Gers F A, Schraudolph N N, Schmidhuber J, etal. Learning precise timing with lstm recurrent networks[J]. Journal of MachineLearning Research, 2003, 3(1): 115-143.
4. A guide to recurrent neural networks and backpropagation. Mikael Bod én.
5. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
6. 《Supervised Sequence Labelling with Recurrent Neural Networks》Alex Graves
7. 《Hands on machine learning with sklearn and tf》Aurelien Geron
8. 《Deep learning》Goodfellow et.


# LSTM