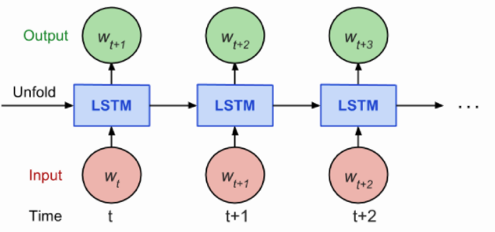
**深度学习RNN实现股票预测实战**

<https://blog.csdn.net/buptgshengod/article/details/78880941>

RNN是一种深度学习的网络结构，RNN的优势是它在训练的过程中会考虑数据的上下文联系，非常适合股票的场景，因为某一时刻的波动往往跟之前的走势蕴含某种联系。

RNN是由一个个神经元cell组成，然而传统的RNN当网络过于复杂的时候，后方节点对于前方的感知力会下降，LSTM（Long-short Term Memory）是一种变型，从名字就可以看出来，LSTM可以增加记忆力，解决上面提到的问题。对于股票这个场景，我们就可以通过LSTM来实现股票的走势的预测。



在股票这个场景下，通过上面这个图可以看出来，输入的是时间t、t+1、t+2的股票信息，可以返回t+1、t+2、t+3的股票信息，而且上下节点前后依赖，通过LSTM模型对于这样的股票序列进行预测，所以股票预测的关键就是首先构建股票序列化数据，然后训练LSTM模型，最终通过这个模型对于股票进行预测，以上就是大体的一些思路。

RNN是一个序列到序列的模型，假设xt−1,xt,xt+1是一个输入：“我是中国“，那么ot−1,ot就应该对应”是”，”中国”这两个，预测下一个词最有可能是什么？就是ot+1应该是”人”的概率比较大。

因此，我们可以做这样的定义：

Xt: 表示t时刻的输入，ot: 表示t时刻的输出，St: 表示t时刻的记忆。

因为我们当前时刻的输出是由记忆和当前时刻的输入决定的，就像你现在大四，你的知识是由大四学到的知识（当前输入）和大三以及大三以前学到的东西的（记忆）的结合，RNN在这点上也类似，神经网络最擅长做的就是通过一系列参数把很多内容整合到一起，然后学习这个参数，因此就定义了RNN的基础：

St=f(U∗Xt+W∗St−1)

代码参考：

https://github.com/lilianweng/stock-rnn

<https://github.com/jimenbian/stock-rnn>

**Predict Stock Prices Using RNN**

<https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html>

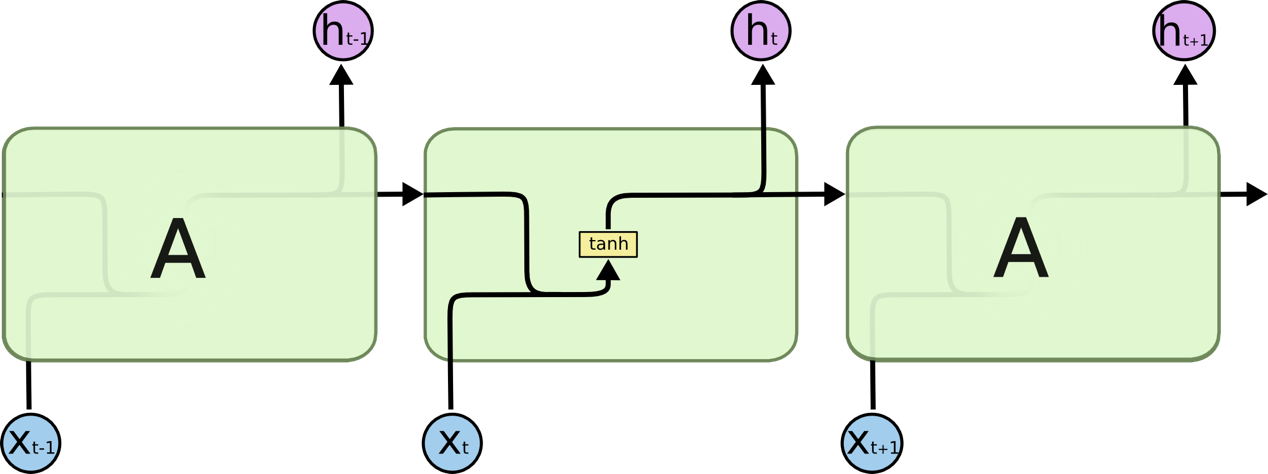
<https://lilianweng.github.io/lil-log/2017/07/22/predict-stock-prices-using-RNN-part-2.html>

RNN-TensorFlow小项目

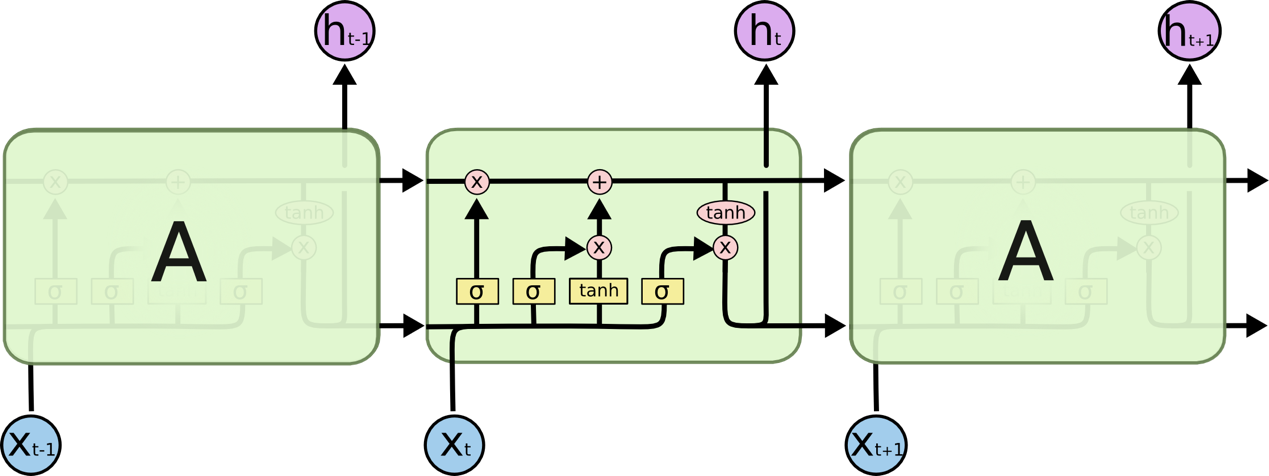
<https://github.com/hzy46/Char-RNN-TensorFlow>

# LSTM网络结构

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



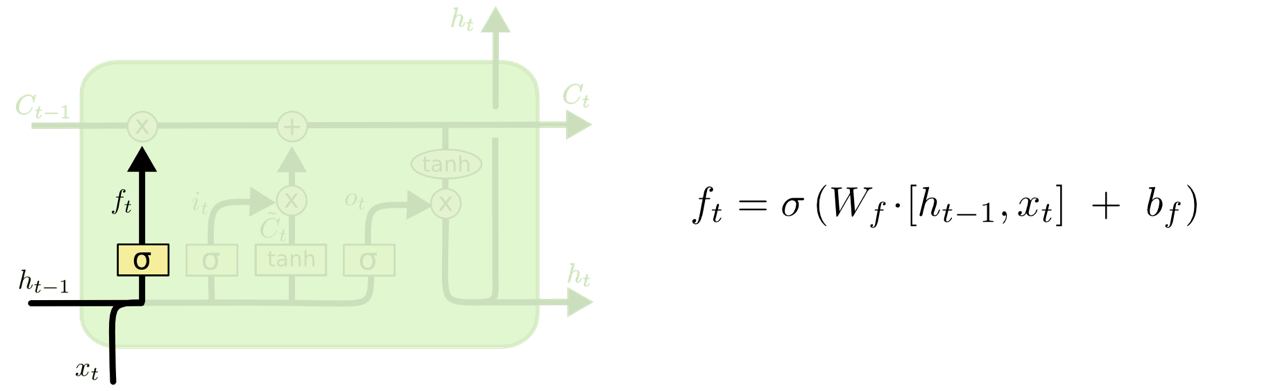
**The repeating module in a standard RNN contains a single layer.**



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

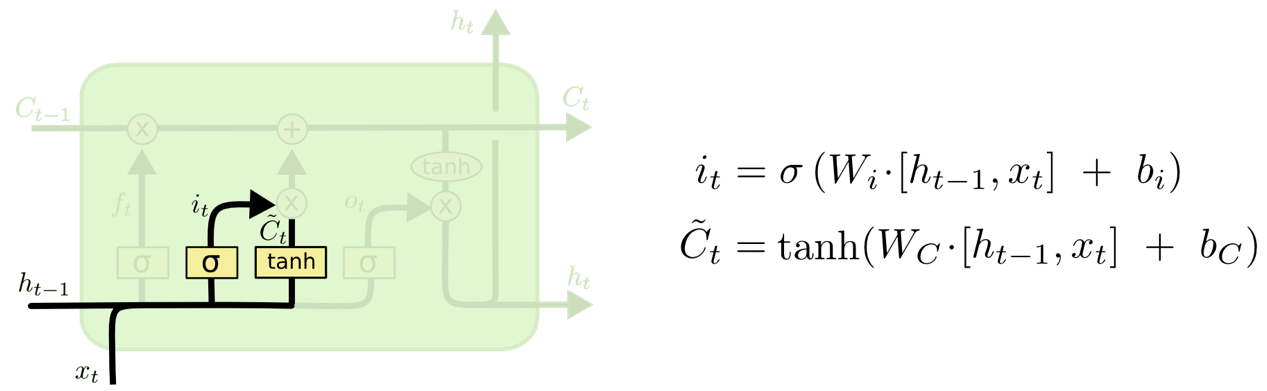


The Core Idea Behind LSTMs



The first step in our LSTM is to decide what information we’re going to throw away from the cell state.

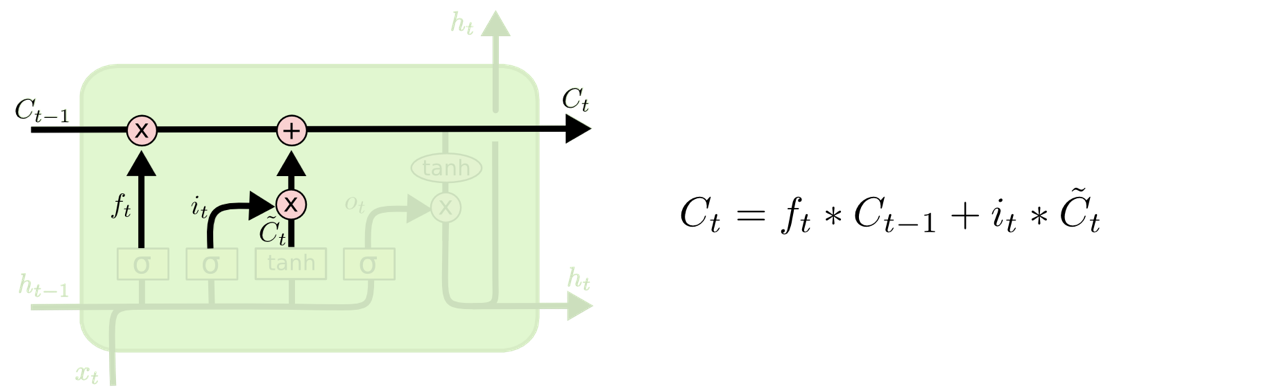
forget gate layer.



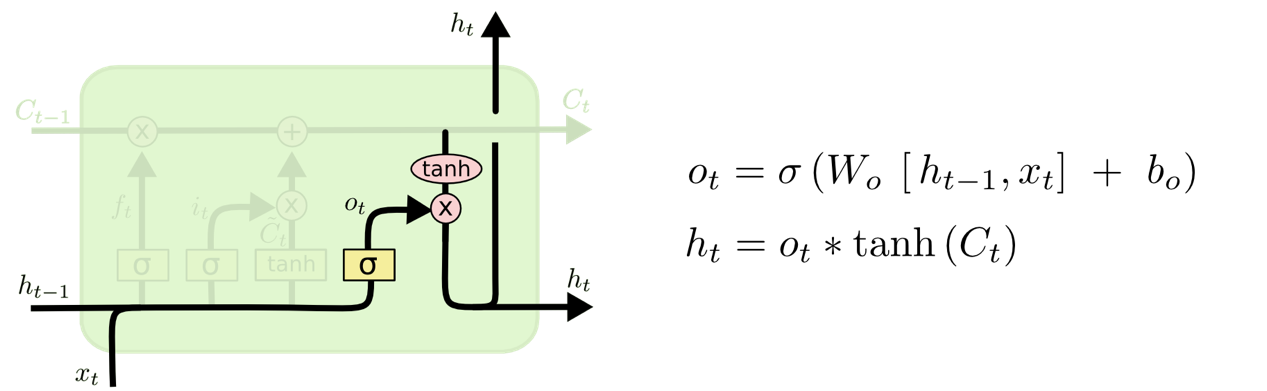
The next step is to decide what new information we’re going to store in the cell state :

1、input gate layer

2、tanh layer

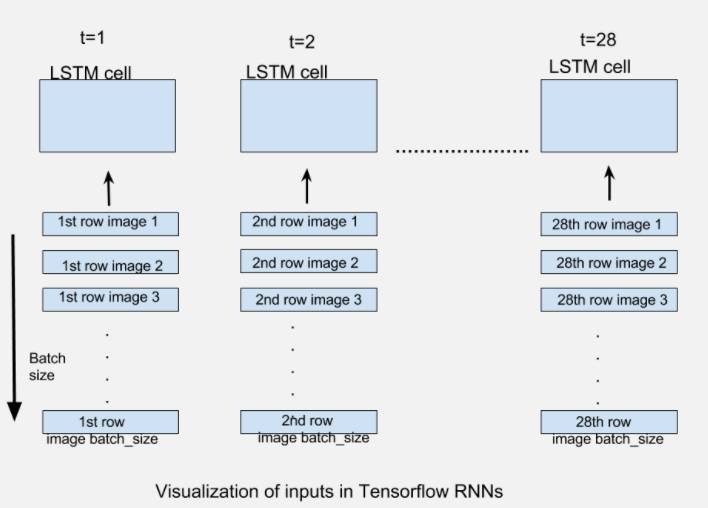


In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.



# LSTM输入输出形式

对于我们的MNIST图像的情况，我们有大小为28X28的图像。它们可以被推断为具有28行28像素的图像。我们将通过28个时间步骤展开我们的网络，使得在每个时间步长，我们可以输入一行28像素（input\_size），从而通过28个时间步长输入完整的图像。如果我们提供batch\_size图像的数量，每个时间步长将提供相应的batch\_size图像行。下图应该可以解释上述描述：



## 输入形式：

原始数据初始化 **raw\_data**

(shape:[batch\_size, num\_step, input\_size]) 形式，其中

batch\_size: 每个时间步长将提供相应的batch\_size个对象

num\_step: 时间步长数

input\_size: 输入的特征数量

转换成inputs 参数格式: **inputs = tf.unstack(raw\_data ,num\_step,1)**

inputs参数是为了接受形状张量列表[batch\_size,input\_size]。该列表的长度是网络展开的时间步长数num\_step，即该列表的每个元素对应于我们展开网络的相应时间步长的输入。