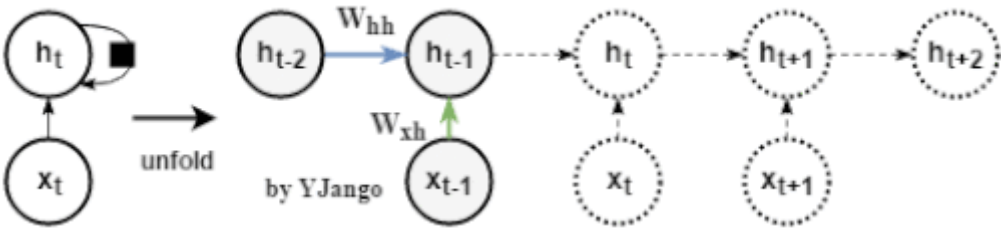
梯度消失和梯度爆炸

循环神经网络用相同的方式处理每个时刻的数据。

- 动态图：



- 数学公式：

$$h_t = \phi(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b)$$

实际问题：但普通的RNN结构却难以传递相隔较远的信息。

考虑：若只看上图蓝色箭头线的、隐藏状态的传递过程，不考虑非线性部分，那么就会得到一个简化的式子(1)：

$$h_t = W_{hh} \cdot h_{t-1}$$

如果将起始时刻的隐藏状态信息h0向第t时刻传递，会得到式子(2)

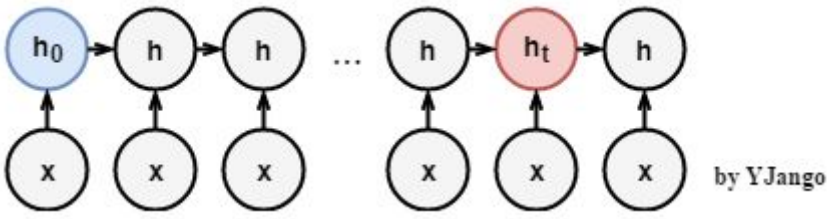
$$h_t = (W_{hh})^t \cdot h_0$$

Whh会被乘以多次，若允许权重矩阵Whh进行特征分解式子(2)会变成

$$h_t = Q \cdot \Lambda^t \cdot Q^T \cdot h_0$$

当特征值小于1时，不断相乘的结果是特征值的t次方趋近于0； 当特征值大于1时，不断相乘的结果是特征值的t次方向趋近于无穷大。 这时想要传递的h0

中的信息会被掩盖掉，无法传递到ht



传统的RNN网路结构

Long Short Term Memory (LSTM)

上面的现象可能并不意味着无法学习，但是即便可以，也会非常非常的慢。为了有效的利用梯度下降法学习，我们希望使不断相乘的梯度的积(the product of derivatives)保持在接近1的数值。

一种实现方式是建立线性自连接单元(linear self-connections)和在自连接部分数值接近1的权重，叫做leaky units。但Leaky units的线性自连接权重是手动设置或设为参数，而目前最有效的方式gated RNNs是通过gates的调控，允许线性自连接的权重在每一步都可以自我变化调节。LSTM就是gated RNNs中的一个实现。

LSTM的初步理解 LSTM(或者其他gated RNNs)是在标准RNN

$$h_t = \phi(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b)$$

的基础上装备了若干个控制数级(magnitude)的gates。可以理解成神经网络(RNN整体)中加入其他神经网络(gates)，而这些gates只是控制数级，控制信息的流动量。

$$\begin{aligned}
i_t &= sigmoid(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= sigmoid(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= sigmoid(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}$$

门(gate)的理解

理解Gated RNNs的第一步就是明白gate到底起到什么作用。

- 物理意义：gate本身可看成是十分有物理意义的一个神经网络。
    - 输入：gate的输入是控制依据；
    - 输出：gate的输出是值域为（0，1）的数值，表示该如何调节其他数据的数级的控制方式。

- 使用方式：gate所产生的输出会用于控制其他数据的数级，相当于过滤器的作用。

例如：当用gate来控制向量【20，5， 7， 8】时，若gate的表示为【0.1，0.2，0.9，0.5】时，原来的向量就会被对应元素相乘(element-wise)后变成：

【20，5， 7， 8】*【0.1，0.2，0.9，0.5】 = 【2，1， 6.3， 4】

明白了gate之后再回过头来看LSTM的数学公式数学公式：

$$
\begin{aligned}
i_t &= sigmoid(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= sigmoid(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= sigmoid(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
\\
c_t &= f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$

gates：先将前半部分的三个式子统一理解。在LSTM中，网络首先构建了3个gates来控制信息的流通量。

注： 虽然gates的式子构成方式一样，但是注意3个gates式子w和b的下角标并不相同。它们有各自的物理意义，在网络学习过程中会产生不同的权重。

门是控制开闭的，全开时值为0，全闭值为1。有开有闭时，值在0到1之间。如果选择的激活函数得到的值不在0，1之间时，通常来说是没有意义的。对于求值时的激活函数，选取时我认为与深层网络中激活函数选取是一样的，没有行与不行，只有好与不好。所以，总结来说，门的激活函数只能是值域为0到1的，对于求值的激活函数无特殊要求

从信号处理的方式说，要保证系统稳定。类似线性系统极点要在单位圆里，非线性直接加个激活卡住。

所以简而言之：Relu不行，越界了；sigmoid差一半平面；只有tanh刚好。tanh还有个好处是零点梯度为1，这个性质比sigmoid好，relu在右半平面也是1，但越界不稳定，然并卵了。

总的流程是：

1.输入经过 sigmoid+tanh >>得到非线性变化的输入It和C^t

2.输入经过 sigmoid乘以细胞状态>>得到留存的细胞信息ft

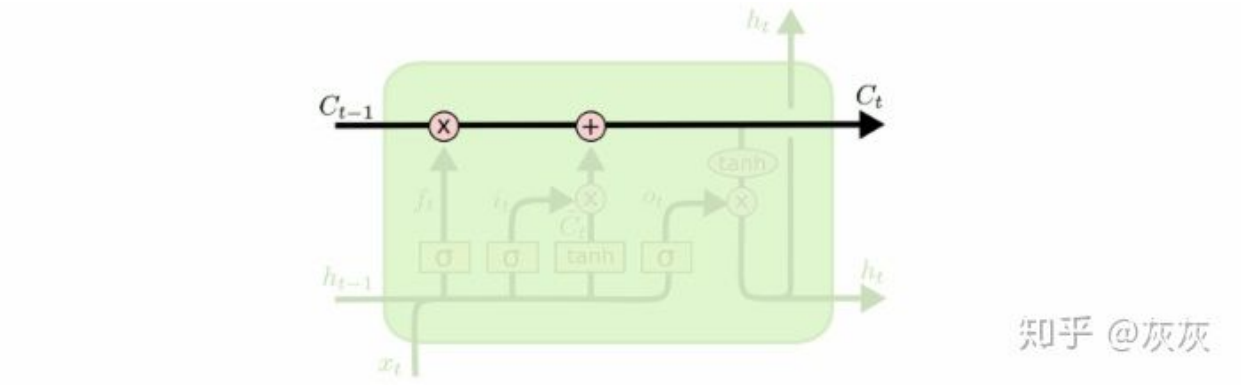
3.1+2》得到更新后的细胞状态

4. 更新后的细胞状态>>tanh激活得到合适的新的细胞状态

5.新的细胞状态 * 经过sigmoid的输入得到输出

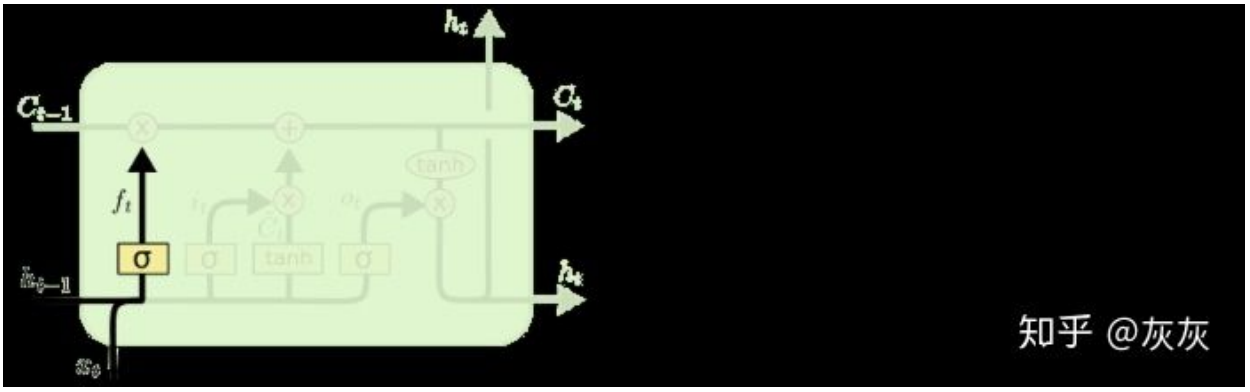
1.细胞状态(Ct)

t时刻的记忆信息，用来保存重要信息。就好像我们的笔记本一样，保存了我们以前学过的知识点。如下图的水平线从图上方贯穿运行，直接在整个链上运行，使得信息在上面流传保持不变会很容易。

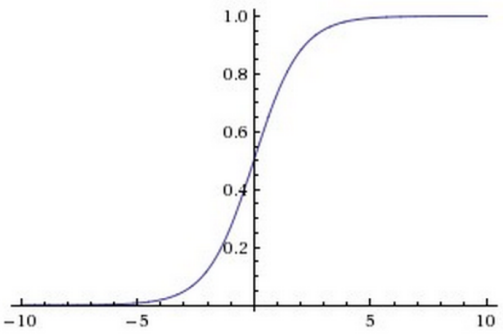


接下来要考虑的就是如何用它们装备在普通的RNN上来控制信息流，而根据它们所用于控制信息流通的地点不同，它们又被分为：

2遗忘门ft：控制上一时刻的memory cell中的信息有多少可以累积到当前时刻的memory cell中。



控制遗忘上一层细胞状态的内容，根据上一序列的ht-1和本序列的Xt为输入，通过sigmoid激活函数，得到上一层细胞状态内容哪些需要去除，那些需要保留。值得注意的是，该输入是以向量的形式，**我们希望遗忘门输出的值大多为0或1，即对向量中的每个值是完全忘记或者完全记住，因此我们使用的是sigmoid函数作为激活函数，因为该函数在许多取值范围内的值都接近于0或1(这里不能用阶跃函数作为激活函数，因为它在所有位置的梯度都为0，无法作为激活函数，，而tanh函数输出范围是【-1， 1】不符合遗忘的意思，输出为负值，怎么遗忘？)**。其他门使用sigmoid函数同理。因此，虽然在其他神经网络可以变换激活函数，但并不建议变换LSTM的激活函数。

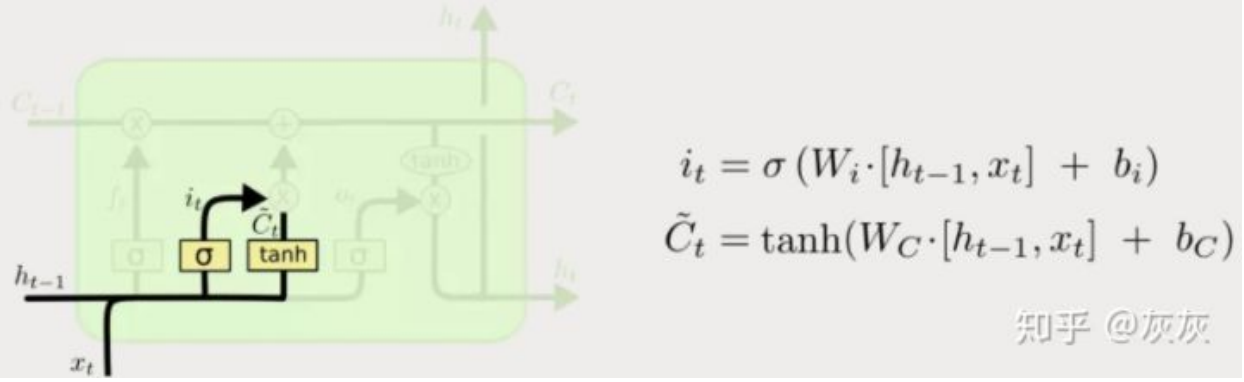


sigmoid

以一个例子来说明遗忘门的作用：在语言模型中，细胞状态可能保存着这样的重要信息：当前主语为单数或者复数等。如当前的主语

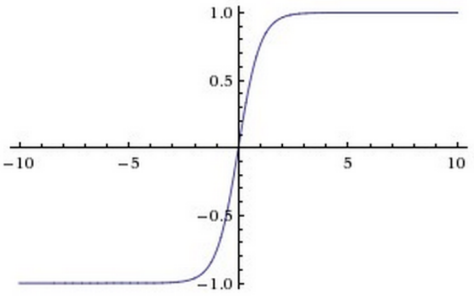
为“小明”，当输入为“同学们”，此时遗忘门就要开始“干活”了，将“小明”遗忘，主语为单数形式遗忘。

3输入门It：控制有多少信息可以流入memory cell（第四个式子ct）

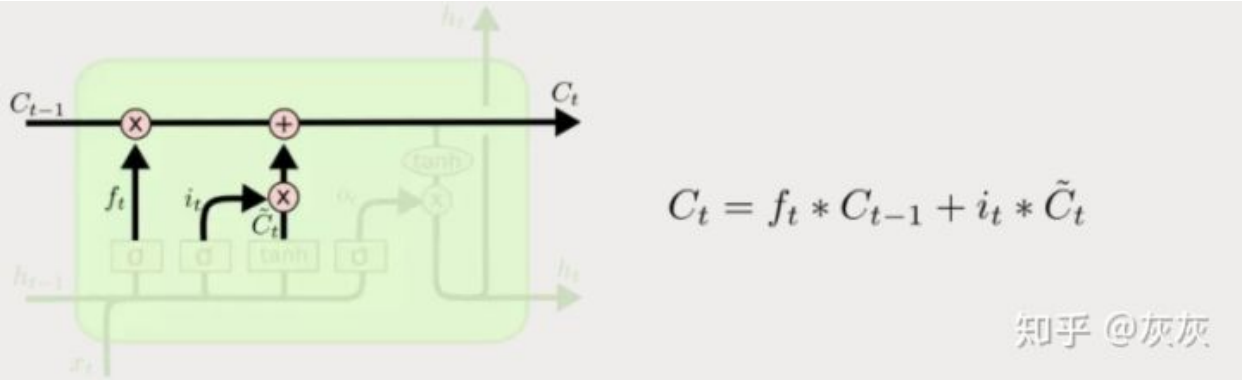


处理当前序列位置的输入，确定需要更新的信息，去更新细胞状态。**此过程分为两部分，3.1一部分是使用包含sigmoid层的输入门决定哪些新信息该被加入到细胞状态；3.2确定了哪些新信息要加入后，需要将新信息转换成能够加入到细胞状态的形式。所以另一部分是使用tanh函数产生一个新的候选向量。**(可以这么理解，LSTM的做法是对信息都转为能加入细胞状态的形式，然后再通过第一部分得到的结果确定其中哪些新信息加入到细胞状态。)

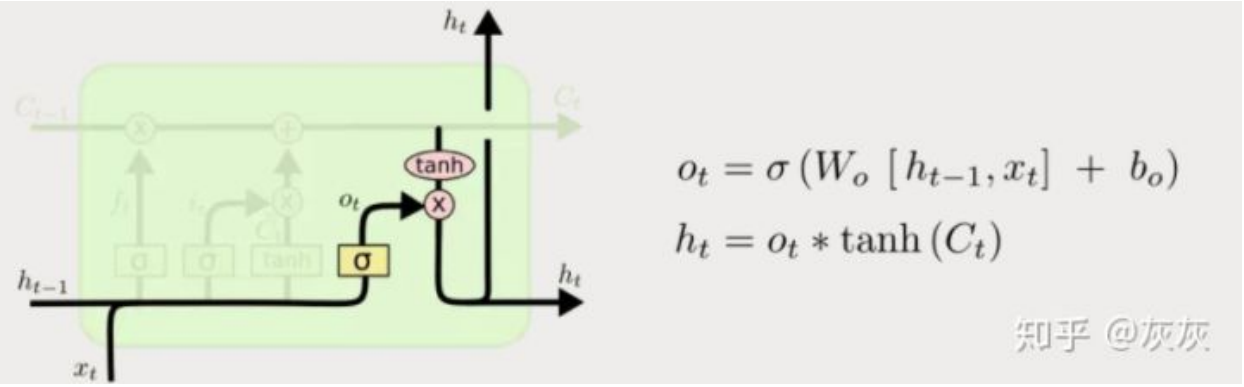
我的理解是既然sigmoid决定加入那些新信息，但是新信息需要激活一下（**即需要将新信息转换成能够加入到细胞状态的形式，也就是神经网络的里面的激活形式，这样经过tanh转换后的信息更具有泛化性**）**输入门和下面的输出门的原理类似**



4.更新细胞状态：有了遗忘门和输入门，现在我们就能把细胞状态Ct−1更新为Ct了。如下图所示，其中ft×Ct−1表示希望删除的信息，it×Ct表示新增的信息。



5输出门ot：控制有多少当前时刻的memory cell中的信息可以流入当前隐藏状态ht中。



最后要基于细胞状态保存的内容来确定输出什么内容。**即选择性的输出细胞状态保存的内容。类似于输入门两部分实现更新一样**，输出门也是需要使用sigmoid激活函数确定哪个部分的内容需要输出，**然后再使用tanh激活函数对细胞状态的内容进行处理(因为通过上面计算得到的Ct每个值不是在tanh的取值范围-1~1中，需要调整)**，将这两部分相乘就得到了我们希望输出的那部分。

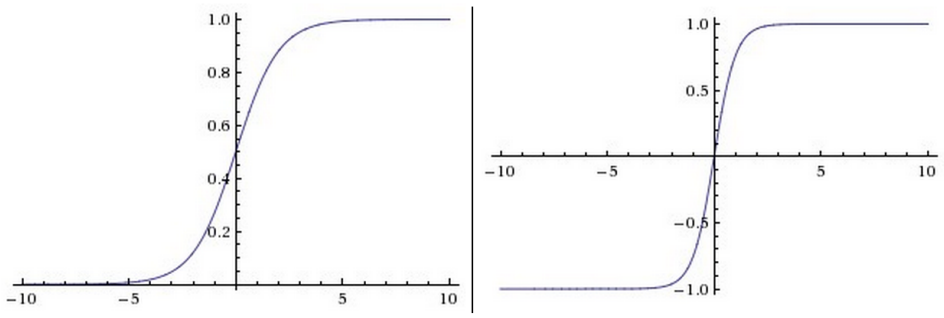
注：gates并不提供额外信息，gates只是起到限制信息的量的作用。因为gates起到的是过滤器作用，所以所用的激活函数是sigmoid而不是tanh。

LSTM 两个激励函数区别sigmoid 和tanh

1. sigmoid激活函数

sigmoid将一个实数输入映射到[0,1]范围内，如下图（左）所示。使用sigmoid作为激活函数存在以下几个问题：

- 梯度饱和。当**函数激活值接近于0或者1时，函数的梯度接近于0。**在反向传播计算梯度过程中，每层残差接近于0，计算出的梯度也不可避免地接近于0。这样在参数微调过程中，会引起参数弥散问题，传到前几层的梯度已经非常靠近0了，参数几乎不会再更新。
- 函数输出不是以0为中心的。我们更偏向于当激活函数的输入是0时，输出也是0的函数。



Left: Sigmoid non-linearity squashes real numbers to range between [0,1] **Right:** The tanh non-linearity squashes real numbers to range between [-1,1].

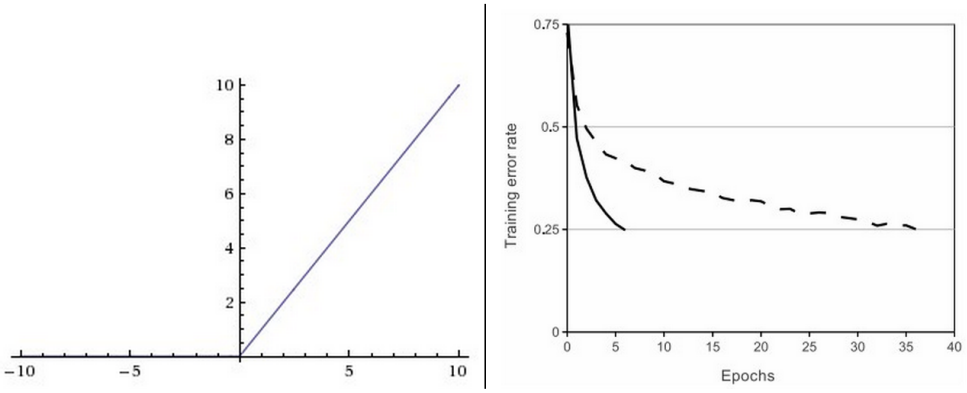
因为上面两个问题的存在，导致参数收敛速度很慢，严重影响了训练的效率。因此在设计神经网络时，很少采用sigmoid激活函数。

2. tanh激活函数

tanh函数将一个实数输入映射到[-1,1]范围内，如上图（右）所示。当输入为0时，tanh函数输出为0，符合我们对激活函数的要求。然而，tanh函数也存在梯度饱和问题，导致训练效率低下。

3.Relu激活函数

Relu激活函数（The Rectified Linear Unit）表达式为： 。如下图（左）所示：



Left: Rectified Linear Unit (ReLU) activation function, which is zero when x < 0 and then linear with slope 1 when x > 0. **Right:** A plot from Krizhevsky et al. (pdf) paper indicating the 6x improvement in convergence with the ReLU unit compared to the tanh unit.

相比sigmoid和tanh函数，Relu激活函数的优点在于：

- 梯度不饱和。梯度计算公式为：。因此在反向传播过程中，减轻了梯度弥散的问题，神经网络前几层的参数也可以很快的更新。
- 计算速度快。正向传播过程中，sigmoid和tanh函数计算激活值时需要计算指数，而Relu函数仅需要设置阈值。如果，如果。加快了正向传播的计算速度。

因此，Relu激活函数可以极大地加快收敛速度，相比tanh函数，收敛速度可以加快6倍（如上图（右）所示）。