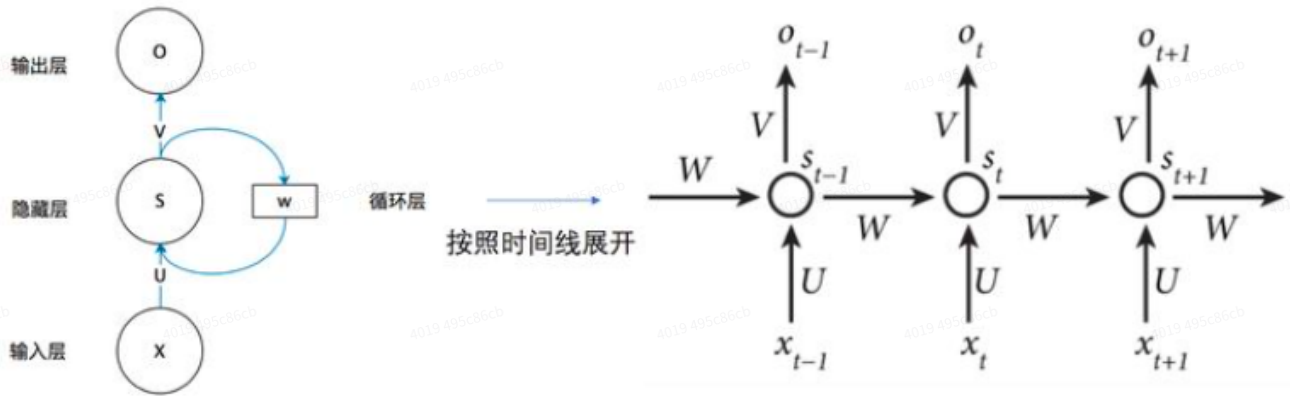
3.RNN& LSTM& GRU

1.RNN

循环神经网络(Recurrent Neural network, RNN) 是一种用于**处理序列数据**的神经网络。相比与一般的神经网络，其很适合用于处理序列变化的数据。

1.1RNN的基本结构

由于本质是处理序列数据（一般按时间顺序，也有可能按照文本顺序）。其基本结构如下：



RNN时间线展开图

现在看上去就比较清楚了，这个网络在t时刻接收到输入xt之后，**隐藏层的值（这个时刻下对应的状态值）是St , 输出值是Ot**。

$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$

$$O_t = g(V \cdot S_t)$$

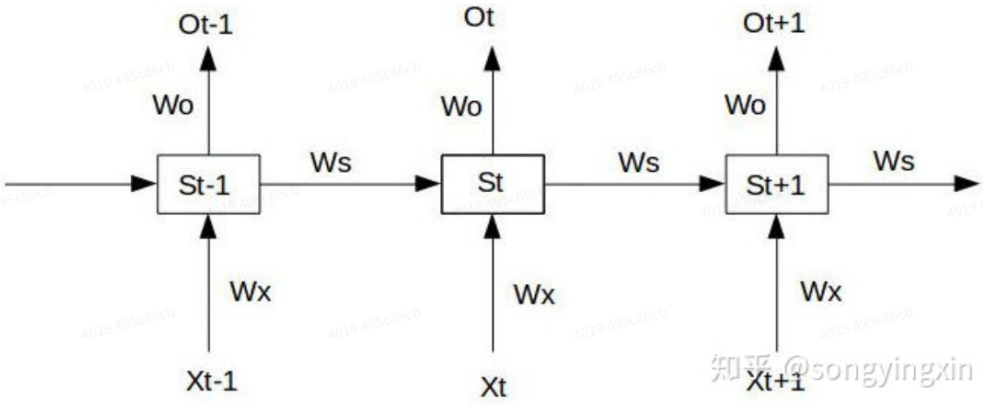
从公式中，可以看出，St的值不仅仅取决于xt,也取决于St-1。

同时，每个神经元都会接收上一个神经元的输出（其实这些神经元**都是相同的，只有三个参数矩阵（U，W, V），每一层的参数相同，只是使用的状态不一样**）。神经元的输出重新作为输入，因此将其称为循环神经网络。

1.2RNN的问题（梯度消失与梯度爆炸）

我们给定一个三个时间的RNN单元，如下：

我们给定一个三个时间的RNN单元，如下：



我们假设最左端的输入 S_0 为给定值， 且神经元中没有激活函数（便于分析）， 则前向过程如下：

$$
\begin{aligned}
S_1 &= W_x X_1 + W_s S_0 + b_1 \qquad & O_1 &= W_o S_1 + b_2 \\
S_2 &= W_x X_2 + W_s S_1 + b_1 & O_2 &= W_o S_2 + b_2 \\
S_3 &= W_x X_3 + W_s S_2 + b_1 & O_3 &= W_o S_3 + b_2
\end{aligned}
$$

在 $t=3$ 时刻， 损失函数为 $L_3 = \frac{1}{2}(Y_3 - O_3)^2$ ，那么如果我们要训练RNN时， 实际上就是是对 W_x, W_s, W_o, b_1, b_2 求偏导， 并不断调整它们以使得 L_3 尽可能达到最小（参见反向传播算法与梯度下降算法)。

那么我们得到以下公式：

$$
\frac{\delta L_3}{\delta W_0} = \frac{\delta L_3}{\delta O_3}\frac{\delta O_3}{\delta W_0}
$$

$$
\begin{aligned}
\frac{\delta L_3}{\delta W_x} &= \frac{\delta L_3}{\delta O_3}\frac{\delta O_3}{\delta S_3}\frac{\delta S_3}{\delta W_x} + \frac{\delta L_3}{\delta O_3}\frac{\delta O_3}{\delta S_3}\frac{\delta S_3}{\delta S_2}\frac{\delta S_2}{\delta W_x} + \frac{\delta L_3}{\delta O_3}\frac{\delta O_3}{\delta S_3}\frac{\delta S_3}{\delta S_2}\frac{\delta S_2}{\delta S_1}\frac{\delta S_1}{\delta W_x} \\
\frac{\delta L_3}{\delta W_s} &= \frac{\delta L_3}{\delta O_3}\frac{\delta O_3}{\delta S_3}\frac{\delta S_3}{\delta W_s} + \frac{\delta L_3}{\delta O_3}\frac{\delta O_3}{\delta S_3}\frac{\delta S_3}{\delta S_2}\frac{\delta S_2}{\delta W_s} + \frac{\delta L_3}{\delta O_3}\frac{\delta O_3}{\delta S_3}\frac{\delta S_3}{\delta S_2}\frac{\delta S_2}{\delta S_1}\frac{\delta S_1}{\delta W_s}
\end{aligned}
$$

将上述偏导公式与第三节中的公式比较，我们发现， 随着神经网络层数的加深对 W0 而言并没有什么影响（W0直接作用与O，与之间状态无关）， **而对 Wx,Ws 会随着时间序列的拉长而产生梯度消失和梯度爆炸问题。**

根据上述分析整理一下公式可得， 对于任意时刻t对 W_x, W_s 求偏导的公式为：

$$
\begin{aligned}
\frac{\delta L_t}{\delta W_x} &= \sum_{k=0}^{t} \frac{\delta L_t}{\delta O_t}\frac{\delta O_t}{\delta S_t}\left(\prod_{j=k+1}^{t}\frac{\delta S_j}{\delta S_{j-1}}\right)\frac{\delta S_k}{\delta W_x} \\
\frac{\delta L_t}{\delta W_s} &= \sum_{k=0}^{t} \frac{\delta L_t}{\delta O_t}\frac{\delta O_t}{\delta S_t}\left(\prod_{j=k+1}^{t}\frac{\delta S_j}{\delta S_{j-1}}\right)\frac{\delta S_k}{\delta W_s}
\end{aligned}
$$

我们发现， 导致梯度消失和爆炸的就在于 $\prod_{j=k+1}^{t}\frac{\delta S_j}{\delta S_{j-1}}$ ， 而加上激活函数后的S的表达式为：

$$
S_j = tanh(W_x X_j + W_s S_{j-1} + b_1)
$$

那么则有：

$$\prod_{j=k+1}^{t} \frac{\delta S_j}{\delta S_{j-1}} = \prod_{j=k+1}^{t} tanh' W_s$$

而在这个公式中， tanh的导数总是小于1 的， 如果 W_s 也是一个大于0小于1的值， 那么随着t的增大， 上述公式的值越来越趋近于0， 这就导致了梯度消失问题。 那么如果 W_s 很大， 上述公式会越来越趋向于无穷， 这就产生了梯度爆炸。

面试回答：

在RNN的参数更新过程中，利用的是反向传播的链式法则，但是由于**网络结构太深，梯度反向传播中的连乘效应随着**网络层数的增加而不断加剧，就带来了梯度消失与梯度爆炸的问题。

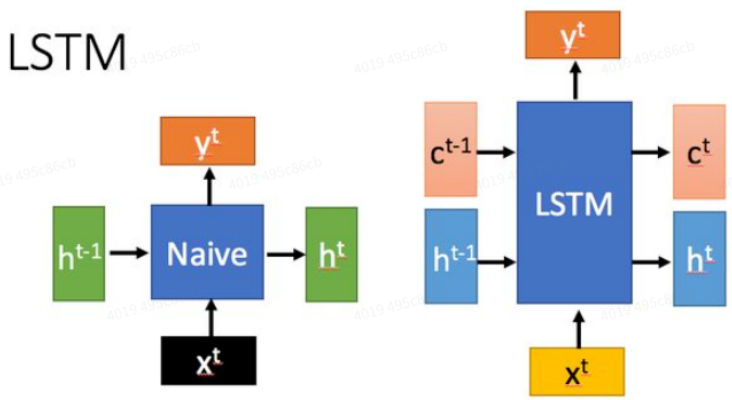
1.3为什么LSTM能解决梯度问题?

因为LSTM的门机制原因来解决了RNN中的梯度消失问题。

因为门值的激活函数是sigmoid,使得三个门的输出要么接近0，要么接近1。当门为 1 时，梯度可以很好的在LSTM中进行传播，当门值为0的时候，说明上一时刻的信息对当前时刻无影响，也就没有必要进行梯度回传与更新。

2.LSTM的结构详解

2.1粗对比



c change slowly ➡ c^t is c^{t-1} added by something

h change faster ➡ h^t and h^{t-1} can be very different

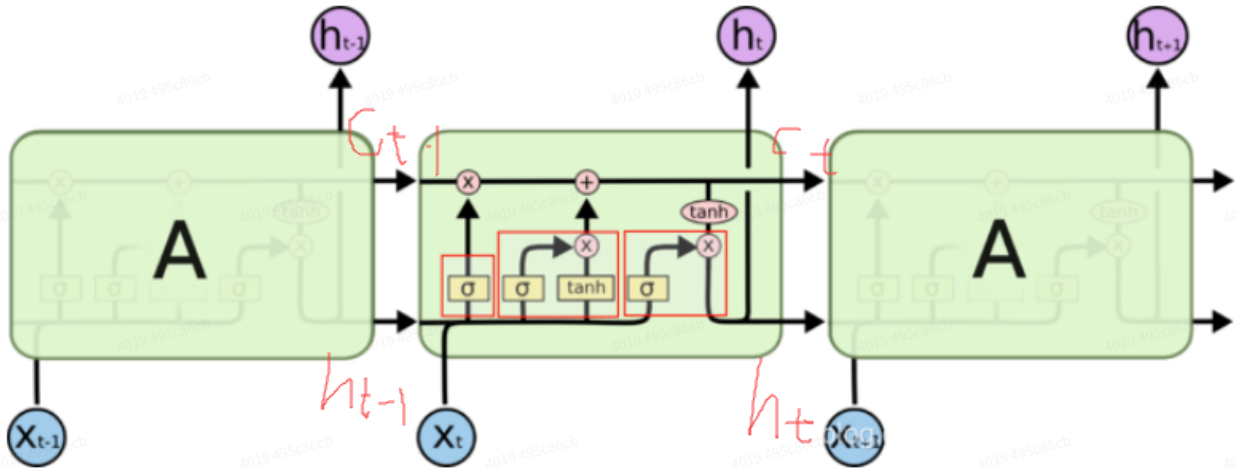
相比RNN只有一个传递状态ht,LSTM有两个传输状态，一个Ct(cell state)和一个ht(hidden state)。Tips,RNN中的ht相对于LSTM中的Ct。

其中对于传递下去的ct改变的很慢，通常输出的Ct是上一个状态传过来的C(t-1)加上一些数值。

ht在不同节点下往往会有很大的区别。

2.2LSTM详解

为了解决梯度消失和爆炸以及更好的预测和分类序列数据等问题，rnn逐渐转变为lstm。



$$i^{(t)} = \sigma(W^{(i)}x^{(t)} + U^{(i)}h^{(t-1)}) \quad \text{(Input gate)}$$

$$f^{(t)} = \sigma(W^{(f)}x^{(t)} + U^{(f)}h^{(t-1)}) \quad \text{(Forget gate)}$$

$$o^{(t)} = \sigma(W^{(o)}x^{(t)} + U^{(o)}h^{(t-1)}) \quad \text{(Output/Exposure gate)}$$

$$\tilde{c}^{(t)} = \tanh(W^{(c)}x^{(t)} + U^{(c)}h^{(t-1)}) \quad \text{(New memory cell)}$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)} \quad \text{(Final memory cell)}$$

$$h^{(t)} = o^{(t)} \circ \tanh(c^{(t)})$$

看的不是特别懂，下面就来逐一分析。

2.2.1遗忘门（第一个框）

这个阶段主要是对上一个节点传进来的输入进行选择性忘记。简单来说就是会 “忘记不重要的，记住重要的”。具体来说是通过计算得到的f(t)表示forget来作为忘记门控，来控制上一个状态的 C(t－1) 哪些需要留, 哪些需要忘。(第二个公式)

2.2.2输入门（第二个框）

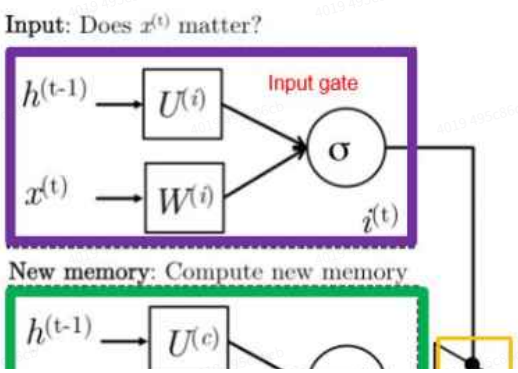
这个阶段将这个阶段的输入有选择性地进行“记忆”。主要是会对输入x(t)进行选择记忆。哪些重要则着重记录下来，哪些不重要，则少记一些。而选择的门控信号则是由 i代表(information)来进行控制。(第一个公式)

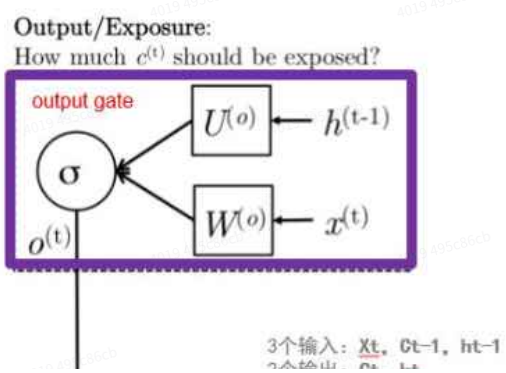
2.2.3输出门（第三个框）

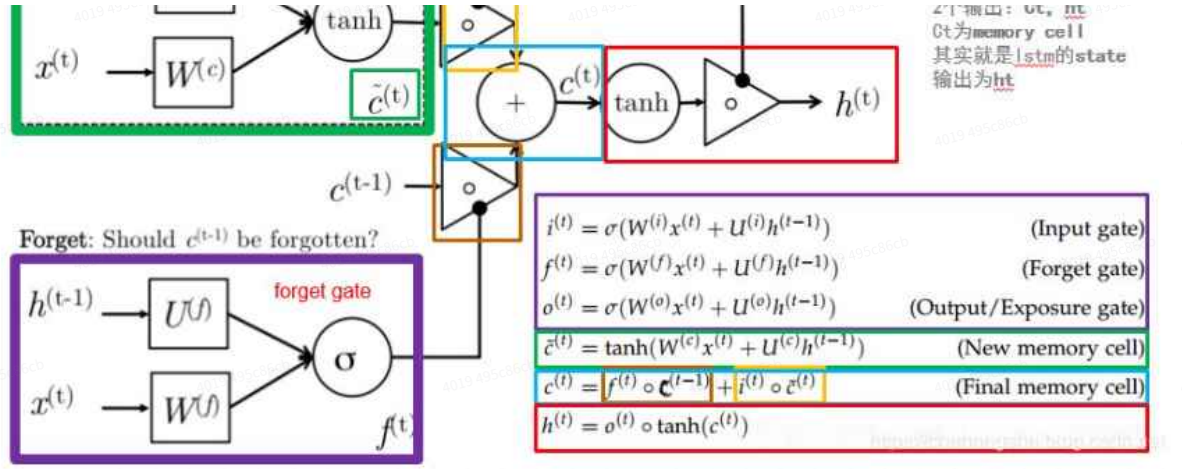
这个阶段将决定哪些将会被当成当前状态的输出。

主要是通过o(t)来进行控制的(第三个公式)。并且还对上一阶段得到c的进行了放缩（通过一个tanh激活函数进行变化）。（第四个公式）

2.3LSTM各部分展开图







1. 首先输入为三个值，一个是此刻的输入值xt，另一个是上一时刻的状态值c(t-1)，最后一个是上一个单元的输出h(t-1)。
2. 最终输出为两个值，一个是此刻产生的状态值ct和输出ht.

3. 首先是输入值x和上一个单元的输出h，分别两个输入都有对应的权重，在经过sigmoid激活作用下得到0-1的值，也就是三个门值。**（得到紫色框中的三个门值）**

$$i^{(t)} = \sigma(W^{(i)}x^{(t)} + U^{(i)}h^{(t-1)}) \quad \text{(Input gate)}$$
$$f^{(t)} = \sigma(W^{(f)}x^{(t)} + U^{(f)}h^{(t-1)}) \quad \text{(Forget gate)}$$
$$o^{(t)} = \sigma(W^{(o)}x^{(t)} + U^{(o)}h^{(t-1)}) \quad \text{(Output/Exposure gate)}$$

4. 和3差不多，依然还是 输入值x和上一个单元的输出h，两个值有对应的权重和3中的描述一模一样，**唯一的区别在于有一个tanh激活函数**，最后相当于得到此时输入得到的当前state，也就是new memory。

   这里可以理解为输入其实是近似的x和h的concatenate操作，经过正常的神经网络的权重，最后经过tanh激活函数得到此时输入的当前的state，**x相当于此刻的输入，h为前面历史的输入，合在一起就是整个序列的信息，也就是此时的new memory**。

$$\tilde{c}^{(t)} = \tanh(W^{(c)}x^{(t)} + U^{(c)}h^{(t-1)}) \quad \text{(New memory cell)}$$

5. 最后输出的state，也就是final memory的计算利用了input gate和forget gate，output gate只与输出有关。

   final memory的计算自然而然和**上一步算得此时的记忆state相关**并且和**上一个输出的final memory相关**，故为忘记门和Ct-1的乘积加上上一步算出来的此时单元的C和输入门的乘积为最终的state。

   忘记门和Ct-1的乘积：选择遗忘哪一些之前的信息

   单元的C和输入门的乘积：选择保留当前状态的那些信息

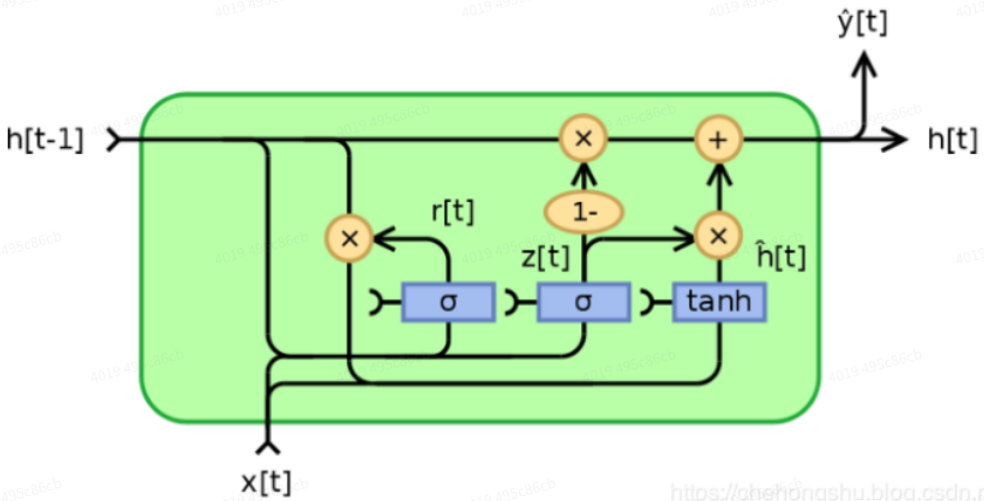
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)} \qquad \text{(Final memory cell)}$$

6. 输出门只与输出相关，最终的输出h为输出门乘以tanh（c）

$$h^{(t)} = o^{(t)} \circ \tanh(c^{(t)})$$

3.GRU

因为LSTM的训练比较慢，而GRU在其上稍微修改，速度可以快很多，而精度基本不变，所以GRU也十分流行



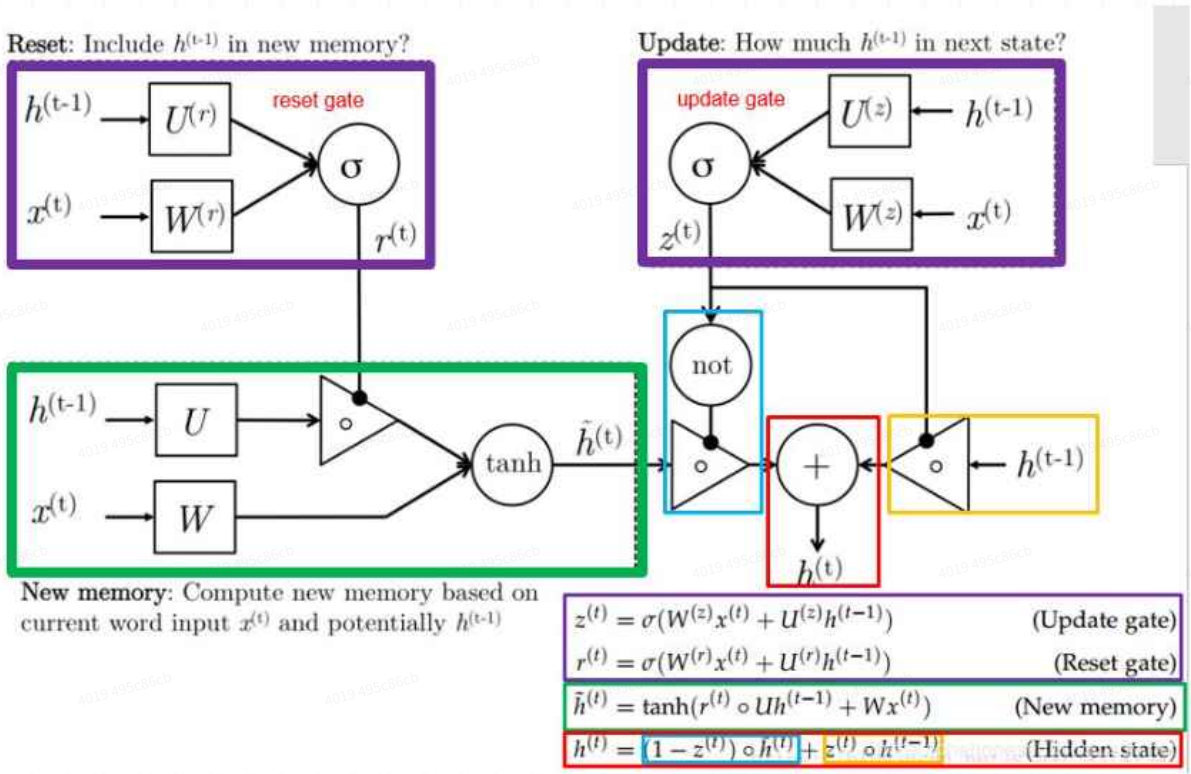
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

换个图看看：



1. 这里GRU只有两个gate，一个是reset gate， 一个是update gate，

   update gate的作用类似于input gate和forget gate，

   (1-z)相当于input gate， z相当于forget gate。

2. 输入为两个值，输出也为一个值，输入为输入此时时刻值x和上一个时刻的输出ht-1， 输出这个时刻的输出值ht。
3. 首先依然是利用xt和ht-1经过权重相乘通过sigmoid，得到两个0-1的值，即两个门值。

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

4. 接下来这里有一些不同，并且经常容易搞混淆。对于LSTM来说依然还是xt与ht-1分别权重相乘相加，之后经过tanh函数为此时的new memory。

而GRU为在这个计算过程中，在ht-1与权重乘积之后和reset gate相乘，之后最终得到new memory，**这里的reset gate的作用为让这个new memory包括之前的ht-1的信息的多少。**

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

5. 接下来和lstm得到final memory其实一样，只是GRU只有两个输入，一个输出，其实这里h即输出也是state，就是说GRU的输出和state是一个值，所以4步骤得到的是new_h，这步骤得到的是final_h，通过update gate得到。

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

4.GRU与LSTM之间的比较

4.1 结构上

1. lstm为三个输入xt，ht-1， ct-1，两个输出。gru为两个输入xt， ht-1，一个输出ht，输出即state。
2. lstm有三个门，输入输出忘记门。gru有两个门，reset，update 门。
3. update 类似于 input gate和forget gate.

4.2 功能上

4. GRU参数更少，训练速度更快，**相比之下需要的数据量更少。**
5. **如果有足够的数据，LSTM的效果可能好于GRU。**