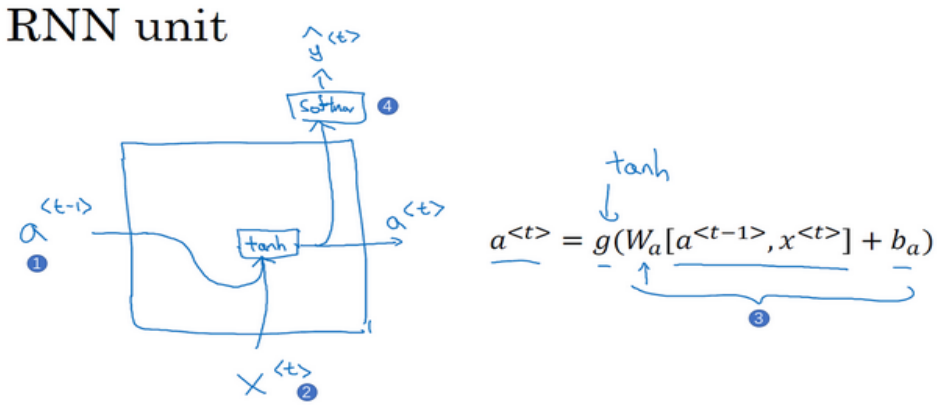
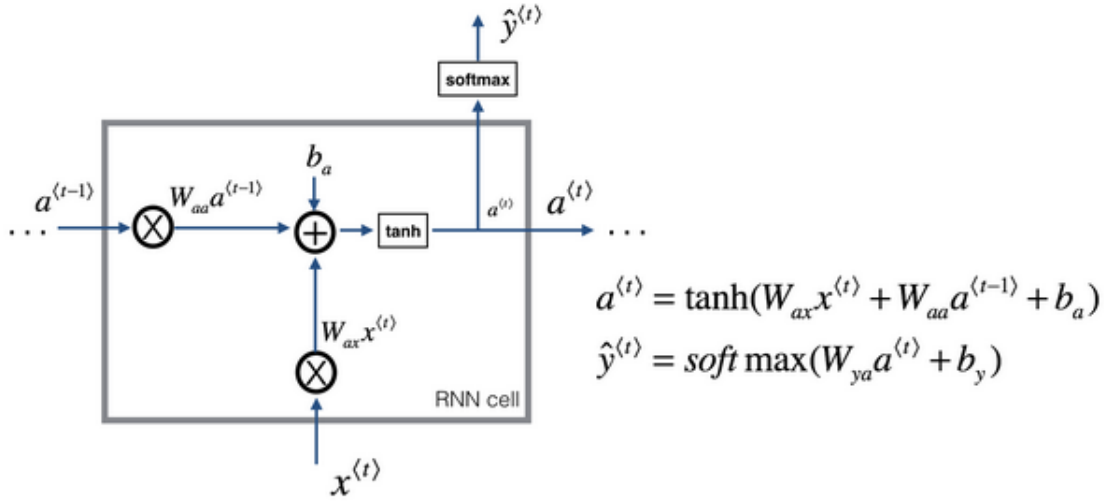
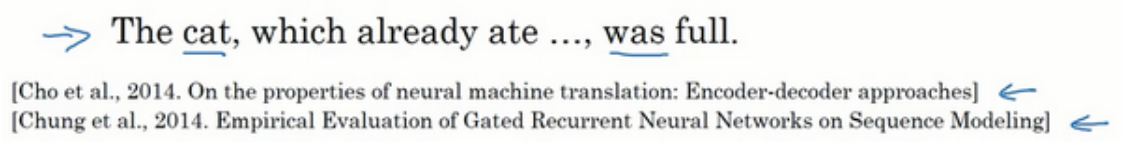
1.9 GRU 单元（Gated Recurrent Unit（GRU））

你已经了解了基础的 **RNN** 模型的运行机制，在本节视频中你将会学习门控循环单元，它改变了 **RNN** 的隐藏层，使其可以更好地捕捉深层连接，并改善了梯度消失问题，让我们看一看。



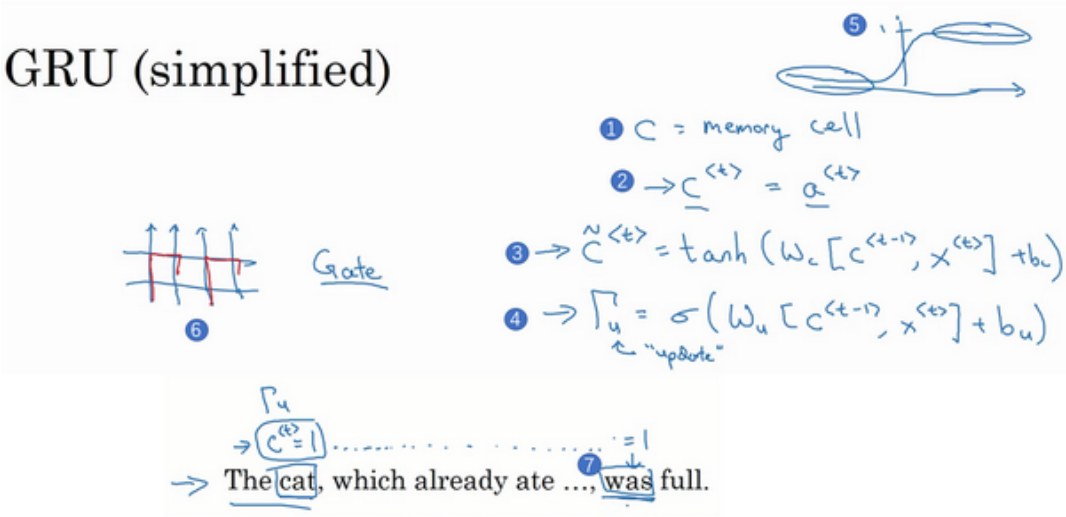
你已经见过了这个公式，$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$，在 **RNN** 的时间$t$处，计算激活值。我把这个画个图，把 **RNN** 的单元画个图，画一个方框，输入$a^{<t-1>}$（上图编号 1 所示），即上一个时间步的激活值，再输入$x^{<t>}$（上图编号 2 所示），再把这两个并起来，然后乘上权重项，在这个线性计算之后（上图编号 3 所示），如果g是一个 **tanh** 激活函数，再经过 **tanh** 计算之后，它会计算出激活值$a^{<t>}$。然后激活值$a^{<t>}$将会传 **softmax** 单元（上图编号 4 所示），或者其他用于产生输出$y^{<t>}$的东西。就这张图而言，这就是 **RNN** 隐藏层的单元的可视化呈现。我向展示这张图，因为我们将使用相似的图来讲解门控循环单元。





许多 **GRU** 的想法都来分别自于 **Yu Young Chang, Kagawa， Gaza Hera, Chang Hung Chu** 和 **Jose Banjo** 的两篇论文。我再引用上个视频中你已经见过的这个句子，**“The cat, which already ate……, was full.”**，你需要记得猫是单数的，为了确保你已经理解了为什么这里是 **was** 而不是 **were**，**“The cat was full.”**或者是**“The cats were full”**。当我们从左到右读这个句子，**GRU 单元将会有个新的变量称为c，代表细胞（cell），即记忆细胞**（下图编号 1 所示）。记忆细胞的作用是提供了记忆的能力，比如说一只猫是单数还是复数，所以当它看到之后的句子的时候，它仍能够判断句子的主语是单数还是复数。于是在时间t处，有记忆细胞$c^{<t>}$，然后我们看的是，**GRU** 实际上输出了激活值$a^{<t>}$，$c^{<t>} = a^{<t>}$（下图编号 2 所示）。于是我们想要使用不同的符号c和a来表示记忆细胞的值和输出的激活值，即使它们是一样的。我现在使用这个标记是因为当我们等会说到 **LSTMs** 的时候，这两个会是不同的值，但是现在对于 **GRU**，$c^{<t>}$的值等于$a^{<t>}$的激活值。

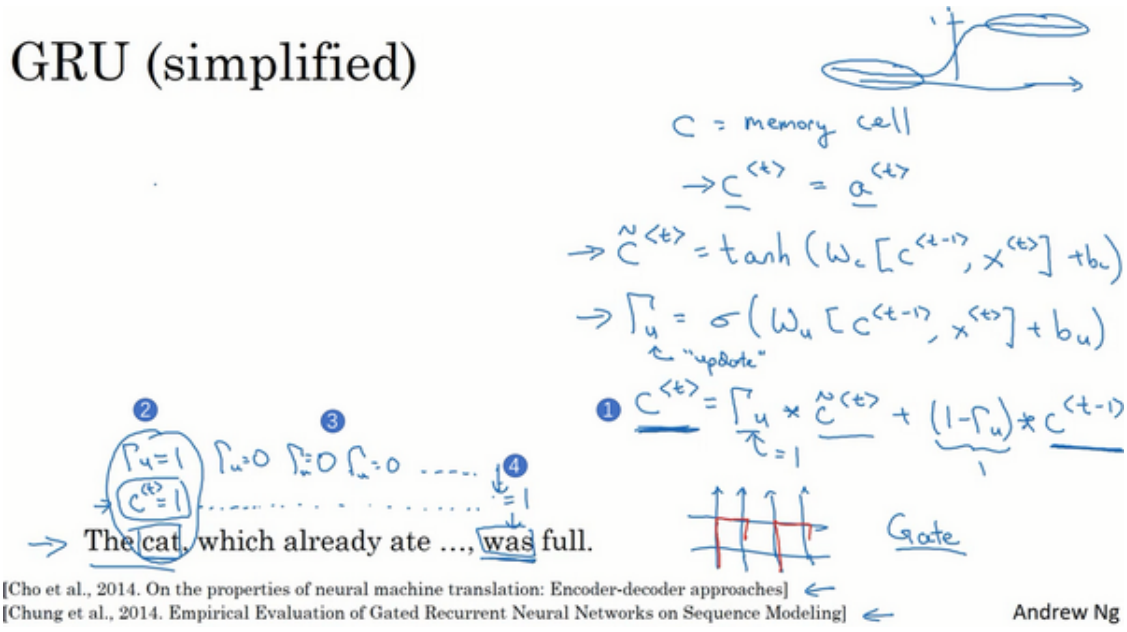
所以这些等式表示了 **GRU** 单元的计算，**在每个时间步，我们将用一个候选值重写记忆细胞，即$\tilde{c}^{<t>}$的值，所以它就是个候选值，替代了$c^{<t>}$的值。**然后我们用 **tanh** 激活函数来计算，$\tilde{c}^{<t>} = tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$，所以$\tilde{c}^{<t>}$的值就是个替代值，代替表示$c^{<t>}$的值（下图编号 3 所示）。



重点来了，在 **GRU** 中真正重要的思想是我们有一个门，我先把这个门叫做**Γ_u**（上图编号 4 所示），这是个下标为u的大写希腊字母Γ，**u代表更新门**，这是一个 0 到 1 之间的值。为了让你直观思考 **GRU** 的工作机制，先思考Γ_u，这个一直在 0 到 1 之间的门值，实际上这

个值是把这个式子带入 **sigmoid** 函数得到的，$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$。我们还记得 **sigmoid** 函数是上图编号 5 所示这样的，它的输出值总是在 0 到 1 之间，对于大多数可能的输入，**sigmoid** 函数的输出总是非常接近 0 或者非常接近 1。在这样的直觉下，可以想到Γ_u在大多数的情况下非常接近 0 或 1。然后这个字母 **u** 表示“**update**”，我选了字母Γ是因为它看起来像门。还有希腊字母 **G**，**G** 是门的首字母，所以 **G** 表示门。

然后 **GRU** 的关键部分就是上图编号 3 所示的等式，我们刚才写出来的**用$\tilde{c}$更新c的等式。然后门决定是否要真的更新它。**于是我们这么看待它，记忆细胞$c^{<t>}$将被设定为 0 或者 1，这取决于你考虑的单词在句子中是单数还是复数，因为这里是单数情况，所以我们先假定它被设为了 1，或者如果是复数的情况我们就把它设为 0。然后 **GRU** 单元将会一直记住$c^{<t>}$的值，直到上图编号 7 所示的位置，$c^{<t>}$的值还是 1，这就告诉它，噢，这是单数，所以我们用 **was**。于是门Γ_u的作用就是决定什么时候你会更新这个值，特别是当你看到词组 **the cat**，即句子的主语猫，这就是一个好时机去更新这个值。然后当你使用完它的时候，“**The cat, which already ate……, was full.**”，然后你就知道，我不需要记住它了，我可以忘记它了。



所以我们接下来要给 **GRU** 用的式子就是

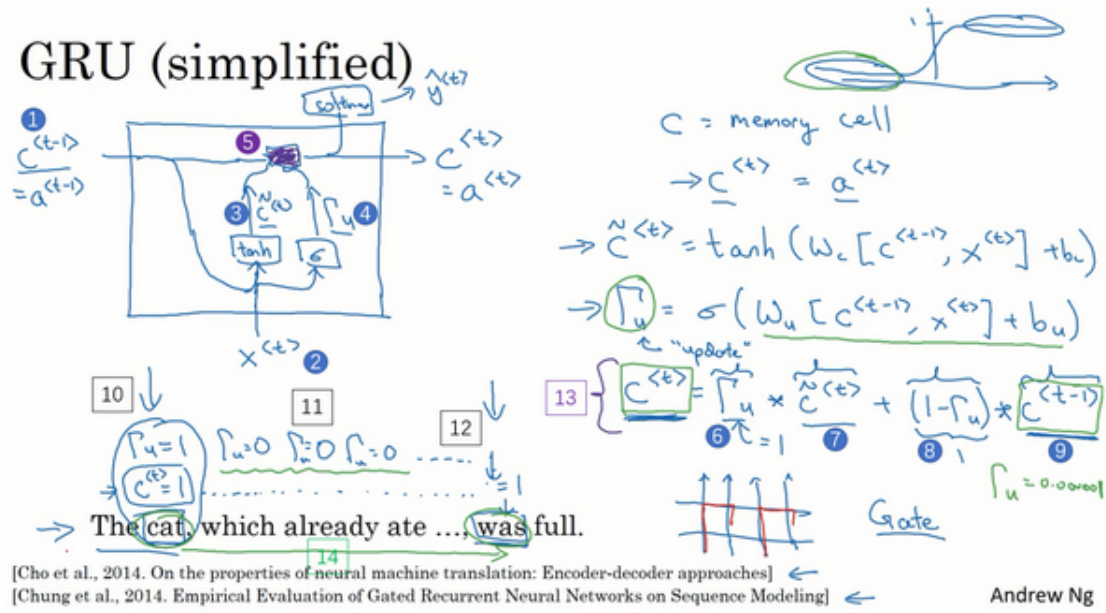
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$（上图编号 1 所示）

你应该注意到了，如果这个更新值$\Gamma_u = 1$，也就是说把这个新值，即$c^{<t>}$设为候选值（$\Gamma_u = 1$时简化上式，$c^{<t>} = \tilde{c}^{<t>}$）。将门值设为 1（上图编号 2 所示），然后往前再更新这个值。对于所有在这中间的值，你应该把门的值设为 0，即$\Gamma_u = 0$，意思就是说不更新它，就用旧的值。因为如果$\Gamma_u = 0$，则$c^{<t>} = c^{<t-1>}$，$c^{<t>}$等于旧的值。甚至你从左到右扫描这个句子，当门值为 0 的时候（上图编号 3 所示，中间$\Gamma_u = 0$一直为 0，表示一直不更新），就

是说不更新它的时候，不要更新它，就用旧的值，也不要忘记这个值是什么，这样即使你一直处理句子到上图编号 4 所示，$c^{<t>}$应该会一直等$c^{<t-1>}$，于是它仍然记得猫是单数的。

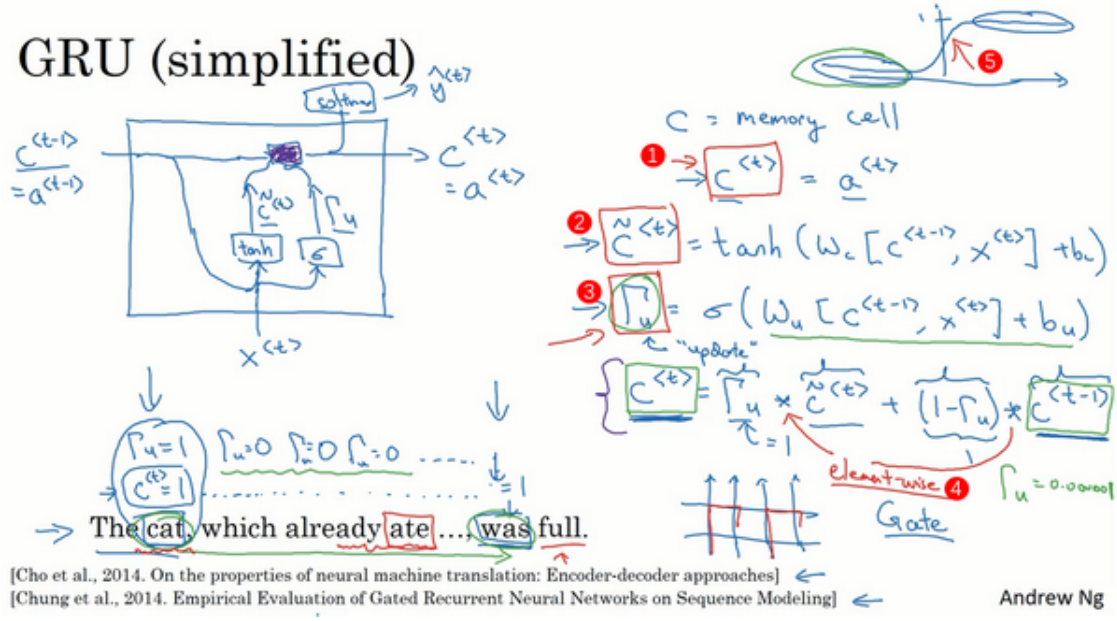
GRU 单元输入$c^{<t-1>}$（下图编号 1 所示），对于上一个时间步，先假设它正好等于$a^{<t-1>}$，所以把这个作为输入。然后$x^{<t>}$也作为输入（下图编号 2 所示），然后把这两个用合适权重结合在一起，再用$tanh$计算，算出$\tilde{c}^{<t>}$，$\tilde{c}^{<t>} = tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$，即$c^{<t>}$的替代值。

再用一个不同的参数集，通过 **sigmoid** 激活函数算出Γ_u，$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$，即更新门。最后所有的值通过另一个运算符结合，我并不会写出公式，但是我用紫色阴影标注的这个方框（下图编号 5 所示，其所代表的运算过程即下图编号 13 所示的等式），代表了这个式子。所以这就是紫色运算符所表示的是，它输入一个门值（下图编号 6 所示），新的候选值（下图编号 7 所示），这再有一个门值（下图编号 8 所示）和$c^{<t>}$的旧值（下图编号 9 所示），所以它把这个（下图编号 1 所示）、这个（下图编号 3 所示）和这个（下图编号 4 所示）作为输入一起产生记忆细胞的新值$c^{<t>}$，所以$c^{<t>}$等于$a^{<t>}$。如果你想，你也可以也把这个带入 **softmax** 或者其他预测$y^{<t>}$的东西。



这就是 **GRU** 单元或者说是一个简化过的 **GRU** 单元，它的优点就是通过门决定，当你从左（上图编号 10 所示）到右扫描一个句子的时候，这个时机是要更新某个记忆细胞，还是不更新，不更新（上图编号 11 所示，中间$\Gamma_u = 0$一直为 0，表示一直不更新）直到你到你真的需要使用记忆细胞的时候（上图编号 12 所示），这可能在句子之前就决定了。因为 **sigmoid** 的值，现在因为门很容易取到 0 值，只要这个值是一个很大的负数，再由于数值上的四舍五入，上面这些门大体上就是 0，或者说非常非常非常接近 0。所以在这样的情况下，这个更

新式子（上图编号 13 所示的等式）就会变成$c^{<t>} = c^{<t-1>}$，这非常有利于维持细胞的值。因为Γ_u很接近 0，可能是 0.000001 或者更小，这就不会有梯度消失的问题了。因为Γ_u很接近 0，这就是说$c^{<t>}$几乎就等于$c^{<t-1>}$，而且$c^{<t>}$的值也很好地被维持了，即使经过很多很多的时间步（上图编号 14 所示）。这就是缓解梯度消失问题的关键，因此允许神经网络运行在非常庞大的依赖词上，比如说 **cat** 和 **was** 单词即使被中间的很多单词分割开。

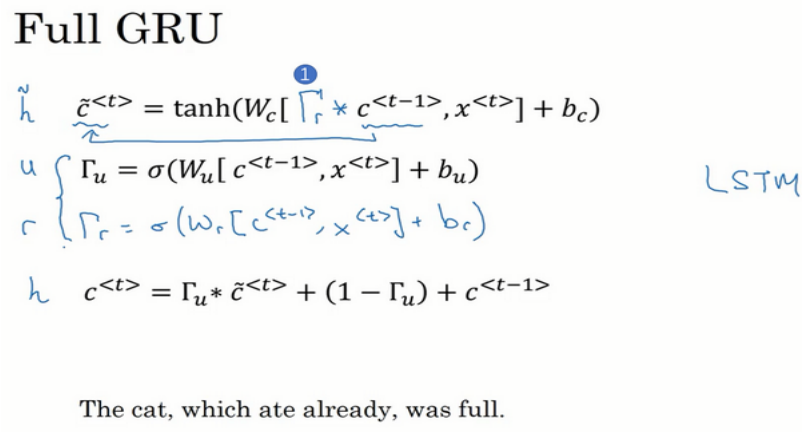


现在我想说下一些实现的细节，在这个我写下的式子中$c^{<t>}$可以是一个向量（上图编号 1 所示），如果你有 100 维的隐藏的激活值，那么$c^{<t>}$也是 100 维的，$\tilde{c}^{<t>}$也是相同的维度（$\tilde{c}^{<t>} = tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$），$\Gamma_u$也是相同的维度（$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$），还有画在框中的其他值。这样的话“*”实际上就是元素对应的乘积（$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$），所以这里的$\Gamma_u$：（$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$），即如果门是一个 100 维的向量，$\Gamma_u$也就 100 维的向量，里面的值几乎都是 0 或者 1，就是说这 100 维的记忆细胞$c^{<t>}$（$c^{<t>} = a^{<t>}$上图编号 1 所示）就是你要更新的比特。

当然在实际应用中Γ_u不会真的等于 0 或者 1，有时候它是 0 到 1 的一个中间值（上图编号 5 所示），但是这对于直观思考是很方便的，就把它当成确切的 0，完全确切的 0 或者就是确切的 1。元素对应的乘积做的就是告诉 **GRU** 单元哪个记忆细胞的向量维度在每个时间步要做更新，所以你可以选择保存一些比特不变，而去更新其他的比特。比如说你可能需要一个比特来记忆猫是单数还是复数，其他比特来理解你正在谈论食物，因为你在谈论吃饭或者食物，然后你稍后可能就会谈论“**The cat was full.**”，你可以每个时间点只改变一些比特。

你现在已经理解 **GRU** 最重要的思想了，幻灯片中展示的实际上只是简化过的 **GRU** 单元，现在来描述一下完整的 **GRU** 单元。

对于完整的 GRU 单元我要做的一个改变就是在我们计算的第一个式子中给记忆细胞的新候选值加上一个新的项，我要添加一个门Γ_r（下图编号 1 所示），你可以认为r代表相关性（**relevance**）。这个Γ_r门告诉你计算出的下一个$c^{<t>}$的候选值$\tilde{c}^{<t>}$跟$c^{<t-1>}$有多大的相关性。计算这个门Γ_r需要参数，正如你看到的这个，一个新的参数矩阵W_r，$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$。



正如你所见，有很多方法可以来设计这些类型的神经网络，然后我们为什么有Γ_r？为什么不用上一张幻灯片里的简单的版本？这是因为多年来研究者们试验过很多很多不同可能的方法来设计这些单元，去尝试让神经网络有更深层的连接，去尝试产生更大范围的影响，还有解决梯度消失的问题，**GRU** 就是其中一个研究者们最常使用的版本，也被发现在很多不同的问题上也是非常健壮和实用的。你可以尝试发明新版本的单元，只要你愿意。但是 **GRU** 是一个标准版本，也就是最常使用的。你可以想象到研究者们也尝试了很多其他版本，类似这样的但不完全是，比如我这里写的这个。然后另一个常用的版本被称为 **LSTM**，表示长短时记忆网络，这个我们会在下节视频中讲到，但是 **GRU** 和 **LSTM** 是在神经网络结构中最常用的两个具体实例。

还有在符号上的一点，我尝试去定义固定的符号让这些概念容易理解，如果你看学术文章的话，你有的时候会看到有些人使用另一种符号$\tilde{x}$，u，r和h表示这些量。但我试着在 **GRU** 和 **LSTM** 之间用一种更固定的符号，比如使用更固定的符号Γ来表示门，所以希望这能让这些概念更好理解。

所以这就是 **GRU**，即门控循环单元，这是 **RNN** 的其中之一。这个结构可以更好捕捉非常长范围的依赖，让 **RNN** 更加有效。然后我简单提一下其他常用的神经网络，比较经典的是这个叫做 **LSTM**，即长短时记忆网络，我们在下节视频中讲解。

1.10 长短期记忆（LSTM（long short term memory）unit）

在上一个视频中你已经学了 **GRU**（门控循环单元）。它能够让你可以在序列中学习非常深的连接。其他类型的单元也可以让你做到这个，比如 **LSTM** 即长短时记忆网络，甚至比 **GRU** 更加有效，让我们看看。

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

这里是上个视频中的式子，对于 **GRU** 我们有$a^{<t>} = c^{<t>}$。

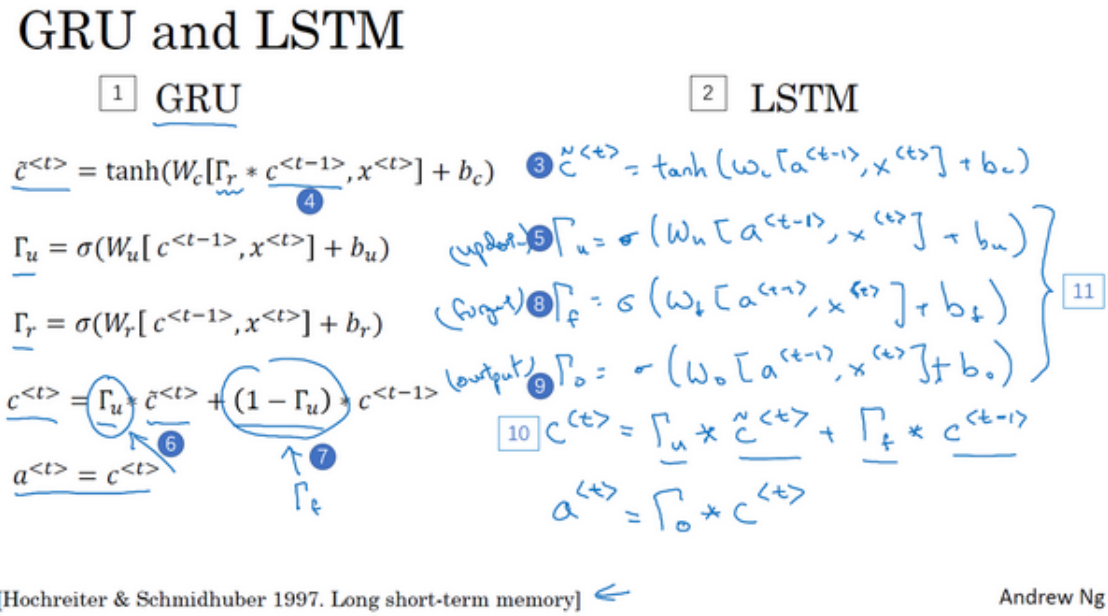
还有两个门:

更新门Γ_u（the update gate）

相关门Γ_r（the relevance gate）

$\tilde{c}^{<t>}$，这是代替记忆细胞的候选值，然后我们使用更新门Γ_u来决定是否要用$\tilde{c}^{<t>}$ 更新$c^{<t>}$。

LSTM 是一个比 **GRU** 更加强大和通用的版本，这多亏了 **Sepp Hochreiter** 和 **Jurgen Schmidhuber**，感谢那篇开创性的论文，它在序列模型上有着巨大影响。我感觉这篇论文是挺难读懂的，虽然我认为这篇论文在深度学习社群有着重大的影响，它深入讨论了梯度消失的理论，我感觉大部分的人学到 **LSTM** 的细节是在其他的地方，而不是这篇论文。



这就是 **LSTM** 主要的式子（上图编号 2 所示），我们继续回到记忆细胞 **c** 上面来，使用$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c$来更新它的候选值$\tilde{c}^{<t>}$（上图编号 3 所示）。注意了，在 **LSTM** 中我们不再有$a^{<t>} = c^{<t>}$的情况，这是现在我们用的是类似于左边这个式子（上图编号 4 所示），但是有一些改变，现在我们专门使用$a^{<t>}$或者$a^{<t-1>}$，而不是用$c^{<t-1>}$，我们也不用Γ_r，即相关门。虽然你可以使用 **LSTM** 的变体，然后把这些东西（左边所示的 **GRU** 公式）都放回来，但是在更加典型的 **LSTM** 里面，我们先不那样做。

我们像以前那样有一个更新门Γ_u和表示更新的参数W_u，$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$（上图编号 5 所示）。一个 **LSTM** 的新特性是不只有一个更新门控制，这里的这两项（上图编号 6，7 所示），我们将用不同的项来代替它们，要用别的项来取代Γ_u和$1 - \Gamma_u$，这里（上图编号 6 所示）我们用Γ_u。

然后这里（上图编号 7 所示）用遗忘门（the forget gate），我们叫它Γ_f，所以这个$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$（上图编号 8 所示）；

然后我们有一个新的输出门，$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$（上图编号 9 所示）；

于是记忆细胞的更新值$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$（上图编号 10 所示）；

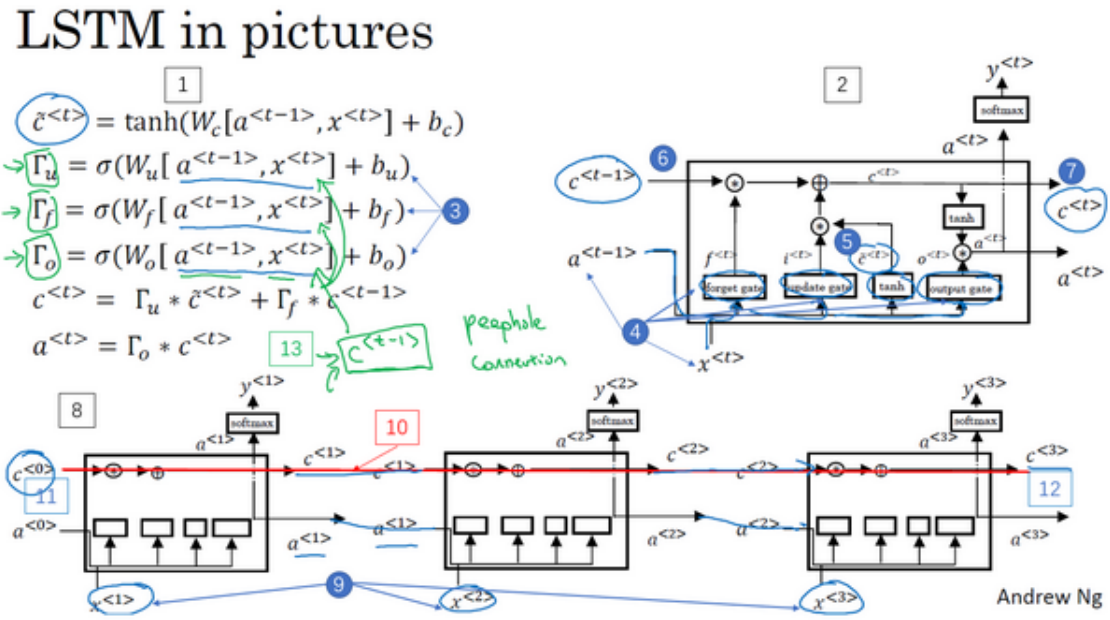
所以这给了记忆细胞选择权去维持旧的值$c^{<t-1>}$或者就加上新的值$\tilde{c}^{<t>}$，所以这里用了单独的更新门Γ_u和遗忘门Γ_f，

然后这个表示更新门（$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$上图编号 5 所示）；

遗忘门（$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$上图编号 8 所示）和输出门（上图编号 9 所示）。

最后$a^{<t>} = c^{<t>}$的式子会变成$a^{<t>} = \Gamma_o * c^{<t>}$。这就是 **LSTM** 主要的式子了，然后这里（上图编号 11 所示）有三个门而不是两个，这有点复杂，它把门放到了和之前有点不同

的地方。



再提一下，这些式子就是控制 **LSTM** 行为的主要的式子了（上图编号 1 所示）。这个右上角的图的灵感来自于 **Chris Ola** 的一篇博客，标题是《理解 **LSTM** 网络》（**Understanding LSTM Network**），这里的这张图跟他博客上的图是很相似的，但关键的不同可能是这里的这张图用了$a^{<t-1>}$和$x^{<t>}$来计算所有门值（上图编号 3，4 所示），在这张图里是用$a^{<t-1>}$，$x^{<t>}$一起来计算遗忘门Γ_f的值，还有更新门Γ_u以及输出门Γ_o（上图编号 4 所示）。然后它们也经过 **tanh** 函数来计算$\tilde{c}^{<t>}$（上图编号 5 所示），这些值被用复杂的方式组合在一起，比如说元素对应的乘积或者其他的方式来从之前的$c^{<t-1>}$（上图编号 6 所示）中获得$c^{<t>}$（上图编号 7 所示）。

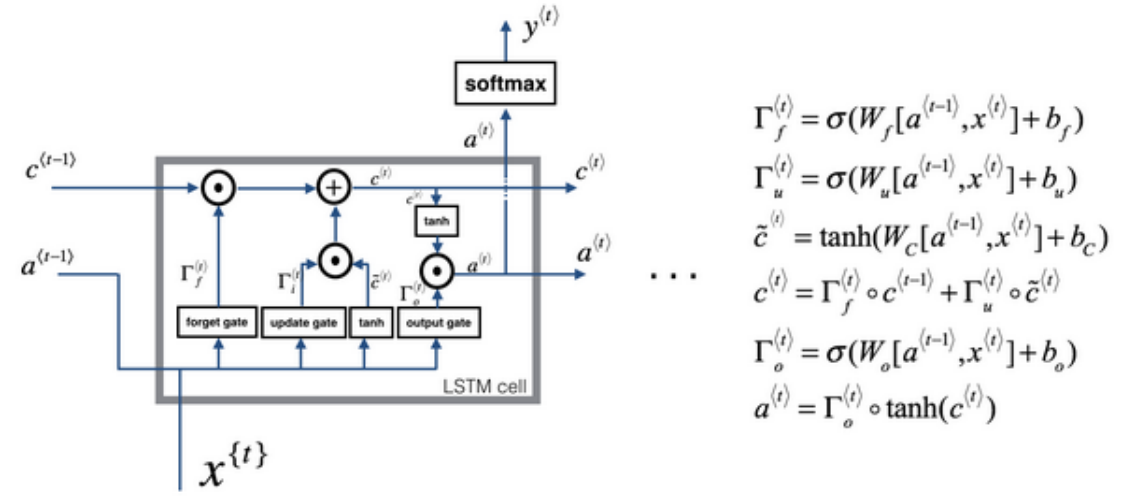
这里其中一个元素很有意思，如你在这一堆图（上图编号 8 所示的一系列图片）中看到的，这是其中一个，再把他们连起来，就是把它们按时间次序连起来，这里（上图编号 9 所示）输入$x^{<1>}$，然后$x^{<2>}$，$x^{<3>}$，然后你可以把这些单元依次连起来，这里输出了上一个时间的a，a会作为下一个时间步的输入，c同理。在下面这一块，我把图简化了一下（相对上图编号 2 所示的图有所简化）。然后这有个有意思的事情，你会注意到上面这里有条线（上图编号 10 所示的线），这条线显示了只要你正确地设置了遗忘门和更新门，**LSTM** 是相当容易把$c^{<0>}$的值（上图编号 11 所示）一直往下传递到右边，比如$c^{<3>} = c^{<0>}$（上图编号 12 所示）。这就是为什么 **LSTM** 和 **GRU** 非常擅长于长时间记忆某个值，对于存在记忆细胞中的某个值，即使经过很长很长的时间步。

这就是 **LSTM**，你可能会想到这里和一般使用的版本会有些不同，最常用的版本可能是门值不仅取决于$a^{<t-1>}$和$x^{<t>}$，有时候也可以偷窥一下$c^{<t-1>}$的值（上图编号 13 所示），这

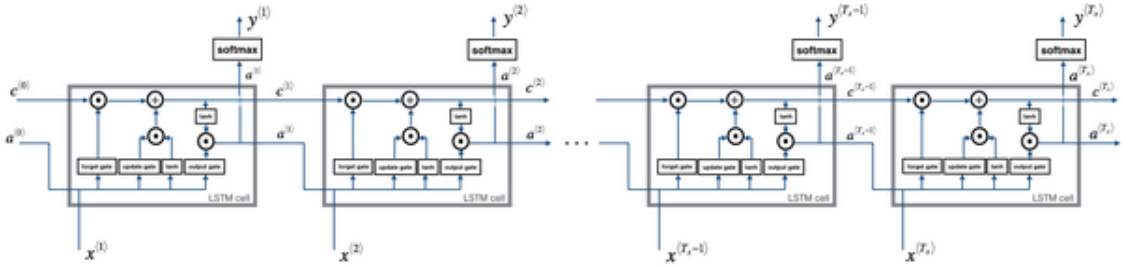
叫做**“窥视孔连接”（peephole connection）**。虽然不是个好听的名字，但是你想，**“偷窥孔连接”**其实意思就是门值不仅取决于$a^{<t-1>}$和$x^{<t>}$，也取决于上一个记忆细胞的值（$c^{<t-1>}$），然后“偷窥孔连接”就可以结合这三个门（Γ_u、Γ_f、Γ_o）来计算了。

如你所见 **LSTM** 主要的区别在于一个技术上的细节，比如这（上图编号 13 所示）有一个 100 维的向量，你有一个 100 维的隐藏的记忆细胞单元，然后比如第 50 个$c^{<t-1>}$的元素只会影响第 50 个元素对应的那个门，所以关系是一对一的，于是并不是任意这 100 维的$c^{<t-1>}$可以影响所有的门元素。相反的，第一个$c^{<t-1>}$的元素只能影响门的第一个元素，第二个元素影响对应的第二个元素，如此类推。但如果你读过论文，见人讨论**“偷窥孔连接”**，那就是在说$c^{<t-1>}$也能影响门值。

LSTM 前向传播图：



$$\Gamma_f^{\langle t\rangle}=\sigma(W_f[a^{\langle t-1\rangle},x^{\langle t\rangle}]+b_f)$$
$$\Gamma_u^{\langle t\rangle}=\sigma(W_u[a^{\langle t-1\rangle},x^{\langle t\rangle}]+b_u)$$
$$\tilde{c}^{\langle t\rangle}=\tanh(W_C[a^{\langle t-1\rangle},x^{\langle t\rangle}]+b_C)$$
$$c^{\langle t\rangle}=\Gamma_f^{\langle t\rangle}\circ c^{\langle t-1\rangle}+\Gamma_u^{\langle t\rangle}\circ\tilde{c}^{\langle t\rangle}$$
$$\Gamma_o^{\langle t\rangle}=\sigma(W_o[a^{\langle t-1\rangle},x^{\langle t\rangle}]+b_o)$$
$$a^{\langle t\rangle}=\Gamma_o^{\langle t\rangle}\circ\tanh(c^{\langle t\rangle})$$



LSTM 反向传播计算：

门求偏导：

$$d\Gamma_o^{\langle t\rangle}=da_{next}*\tanh(c_{next})*\Gamma_o^{\langle t\rangle}*(1-\Gamma_o^{\langle t\rangle})$$

$$d\cancel{6}^{\langle t\rangle}=dc_{next}*\Gamma_i^{\langle t\rangle}+\Gamma_o^{\langle t\rangle}(1-\tanh(c_{next})^2)*i_t*da_{next}*\cancel{6}^{\langle t\rangle}*(1-\tanh(\cancel{6}^2))$$

$$d\Gamma_u^{\langle t\rangle}=dc_{next}*\cancel{6}^{\langle t\rangle}+\Gamma_o^{\langle t\rangle}(1-\tanh(c_{next})^2)*\cancel{6}^{\langle t\rangle}*da_{next}*\Gamma_u^{\langle t\rangle}*(1-\Gamma_u^{\langle t\rangle})$$

$$d\Gamma_f^{\langle t\rangle} = dc_{next} * \%_{prev} + \Gamma_o^{\langle t\rangle}(1-\tanh(c_{next})^2) * c_{prev} * da_{next} * \Gamma_f^{\langle t\rangle} * (1-\Gamma_f^{\langle t\rangle})$$

参数求偏导 ：

$$dW_f = d\Gamma_f^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

$$dW_u = d\Gamma_u^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

$$dW_c = d\%_0^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

$$dW_o = d\Gamma_o^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

为了计算db_f, db_u, db_c, db_o 需要各自对$d\Gamma_f^{\langle t\rangle}, d\Gamma_u^{\langle t\rangle}, d\tilde{c}^{\langle t\rangle}, d\Gamma_o^{\langle t\rangle}$ 求和。

最后，计算隐藏状态、记忆状态和输入的偏导数：

$$da_{prev} = W_f^T * d\Gamma_f^{\langle t\rangle} + W_u^T * d\Gamma_u^{\langle t\rangle} + W_c^T * d\%_0^{\langle t\rangle} + W_o^T * d\Gamma_o^{\langle t\rangle}$$

$$dc_{prev} = dc_{next}\Gamma_f^{\langle t\rangle} + \Gamma_o^{\langle t\rangle} * (1-\tanh(c_{next})^2) * \Gamma_f^{\langle t\rangle} * da_{next}$$

$$dx^{\langle t\rangle} = W_f^T * d\Gamma_f^{\langle t\rangle} + W_u^T * d\Gamma_u^{\langle t\rangle} + W_c^T * d\%_t + W_o^T * d\Gamma_o^{\langle t\rangle}$$

这就是 **LSTM**，我们什么时候应该用 **GRU**？什么时候用 **LSTM**？这里没有统一的准则。而且即使我先讲解了 **GRU**，在深度学习的历史上，**LSTM** 也是更早出现的，而 **GRU** 是最近才发明出来的，它可能源于 **Pavia** 在更加复杂的 **LSTM** 模型中做出的简化。研究者们在很多不同问题上尝试了这两种模型，看看在不同的问题不同的算法中哪个模型更好，所以这不是个学术和高深的算法，我才想要把这两个模型展示给你。

GRU 的优点是这是个更加简单的模型，所以更容易创建一个更大的网络，而且它只有两个门，在计算性上也运行得更快，然后它可以扩大模型的规模。

但是 **LSTM** 更加强大和灵活，因为它有三个门而不是两个。如果你想选一个使用，我认为 **LSTM** 在历史进程上是个更优先的选择，所以如果你必须选一个，我感觉今天大部分的人还是会把 **LSTM** 作为默认的选择来尝试。虽然我认为最近几年 **GRU** 获得了很多支持，而且我感觉越来越多的团队也正在使用 **GRU**，因为它更加简单，而且还效果还不错，它更容易适应规模更加大的问题。

所以这就是 **LSTM**，无论是 **GRU** 还是 **LSTM**，你都可以用它们来构建捕获更加深层连接

的神经网络。