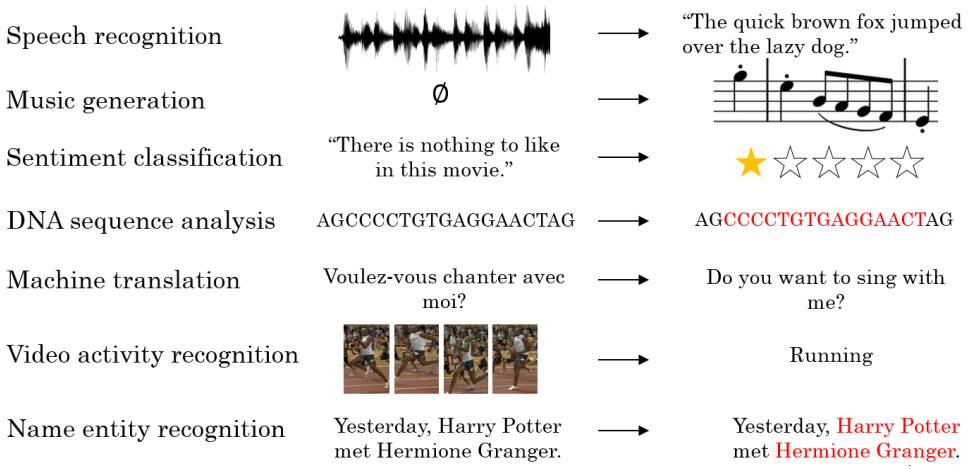
1 循环神经网络

随着深度学习技术的发展，使用**循环神经网络（Recurrent Neural Network，RNN）**建立的各种**序列模型**，使语音识别、机器翻译及自然语言理解等应用成为可能。

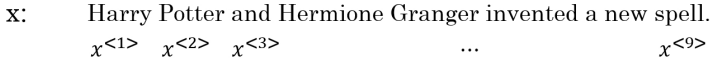


注意：上图中的数据为序列数据

1.1 符号表示

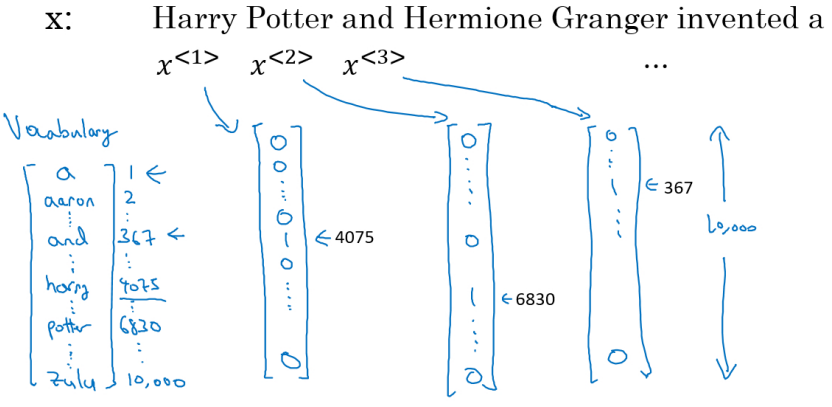
自然语言、音频等数据都是前后相互关联的数据，比如理解一句话要通过一整句而不是其中的几个词，分析一段音频要通过连续的片段而不是其中的几帧。前面所学的 DNN 以及 CNN 处理的都是前后毫无关联的一个个单独数据，对于这些序列数据则需要采用 RNN 进行处理。

用循环神经网络处理时间序列时，首先要对时间序列进行**标记化**，其实就是表示成一种计算机可以识别的语言，便于计算处理。对一个序列数据 x，用符号 $x^{\langle t \rangle}$ 来表示这个序列数据中的第 t 个元素。用 $y^{\langle t \rangle}$ 来表示第 t 个标签，用 T_x 和 T_y 来表示输入和输出的长度。这个元素的类型因数据类型而异，对一段音频，它可能其中的几帧，对一句话，它可能是一个或几个单词，如下图所示。



第 i 个序列数据的第 t 个元素用符号 $x^{(i)\langle t \rangle}$，第 t 个标签即为 $y^{(i)\langle t \rangle}$。对应即有 $T_x^{(i)}$ 和 $T_y^{(i)}$。

想要表示一个词语，需要先建立一个**词汇表（Vocabulary）**，或者叫**字典（Dictionary）**。将需要表示的所有词语变为一个列向量，可以根据字母顺序排列，然后根据单词在向量中的位置，用 **one-hot 向量（one-hot vector）**来表示该单词的标签：将每个单词编码成一个 $R^{|V|\times 1}$ 向量，其中 $|V|$ 是词汇表中单词的数量。一个单词在词汇表中的索引在该向量对应的元素为 1，其余元素均为 0。部分表示如下图：



如果遇到了一个不在所构建的词表中的单词，就创建一个新的标记，也就是一个叫做 **Unknow Word** 的伪造单词，用 **<UNK>** 作为标记，来表示不在词表中的单词。

补充：one-hot 向量是最简单的词向量。它的**缺点**是，由于每个单词被表示为完全独立的个体，因此单词间的相似度无法体现。例如单词 hotel 和 motel 意思相近，而与 cat 不相似，但是：

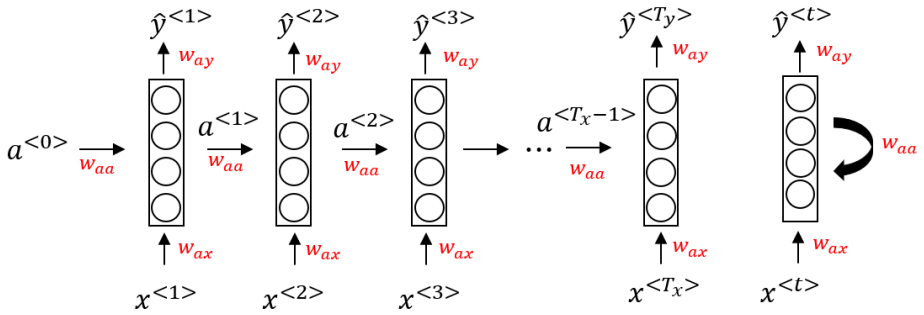
$$(w^{hotel})^T w^{motel} = (w^{hotel})^T w^{cat} = 0 \tag{1}$$

1.2 循环神经网络模型

对于序列数据，使用标准神经网络存在以下问题：

- 对于不同的示例，输入和输出可能有不同的长度，因此输入层和输出层的神经元数量无法固定。
- 从输入文本的不同位置学到的同一特征无法共享。
- 模型中的参数太多，计算量太大。

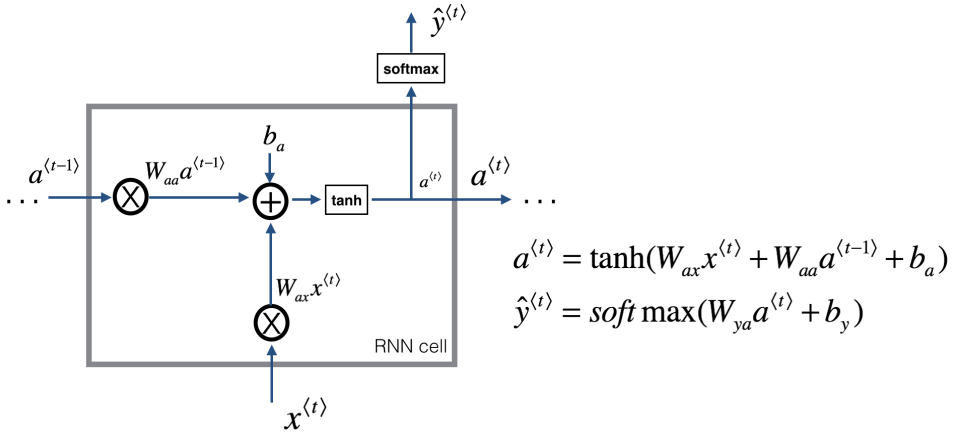
为了解决这些问题，引入**循环神经网络（Recurrent Neural Network，RNN）**。一种循环神经网络的结构如下图所示：



左边的网络可简单表示为右图的结构，其中元素 $x^{\langle t \rangle}$ 输入对应时间步（Time Step）的隐藏层的同时，该隐藏层也会接收上一时间步的隐藏层激活值 $a^{\langle t-1 \rangle}$，其中 $a^{\langle 0 \rangle}$ 一般直接初始化为零向量。一个时间步输出一个对应的预测结果 $\hat{y}^{\langle t \rangle}$。

循环神经网络从左向右扫描数据，同时每个时间步的参数也是共享的，输入、激活、输出的参数对应为 W_{ax}、W_{aa}、W_{ay}。

下图是一个 RNN 神经元的结构：



1.2.1 前向传播过程

$$\begin{aligned} a^{\langle 0 \rangle} &= \vec{0} \\ a^{\langle t \rangle} &= g_1(W_{aa}a^{\langle t-1 \rangle} + W_{ax}x^{\langle t \rangle} + b_a) \\ \hat{y}^{\langle t \rangle} &= g_2(W_{ya}a^{\langle t \rangle} + b_y) \end{aligned} \tag{2}$$

其中 b_a、b_y 是两个偏差参数，激活函数 g_1 通常选择 tanh，有时也用 ReLU；g_2 可选 sigmoid 或 softmax，取决于需要的输出类型。

为了进一步简化公式以方便运算，可以将 W_{ax}、W_{aa} **水平并列**为一个矩阵 W_a，同时 $a^{\langle t-1\rangle}$ 和 $x^{\langle t\rangle}$ **堆叠**成一个矩阵。则有：

$$\begin{aligned} W_a &= [W_{ax}, W_{aa}] \\ a^{\langle t\rangle} &= g_1(W_a[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_a) \\ \hat{y}^{\langle t\rangle} &= g_2(W_y a^{\langle t\rangle} + b_y) \end{aligned} \tag{3}$$

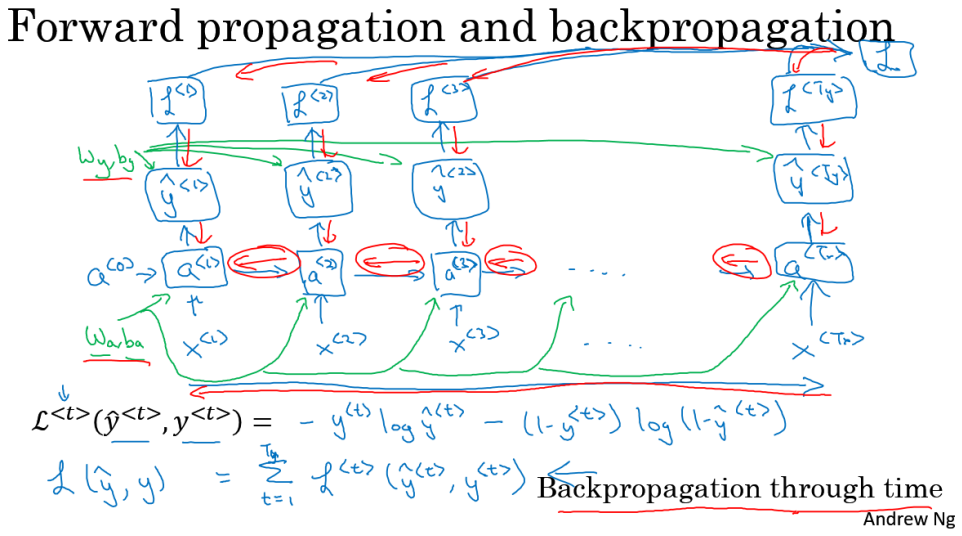
1.2.2 反向传播过程

为了计算反向传播过程，需要先定义一个损失函数。单个位置上（或者说单个时间步上）某个单词的预测值的损失函数采用**交叉熵损失函数**，如下所示：

$$L^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle}) = -y^{\langle t\rangle} log\hat{y}^{\langle t\rangle} - (1 - y^{\langle t\rangle})log(1 - \hat{y}^{\langle t\rangle}) \tag{4}$$

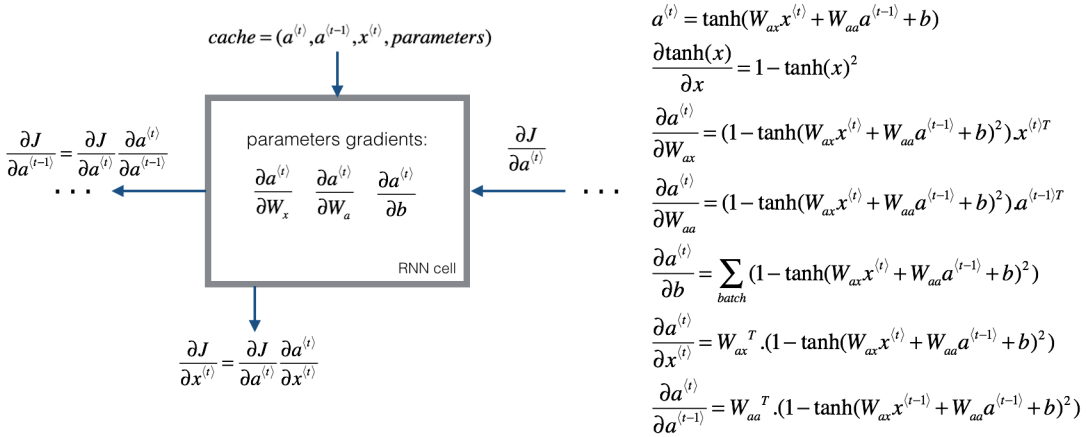
将单个位置上的损失函数相加，得到整个序列的成本函数如下：

$$J = L(\hat{y}, y) = \sum_{t=1}^{T_x} L^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle}) \tag{5}$$



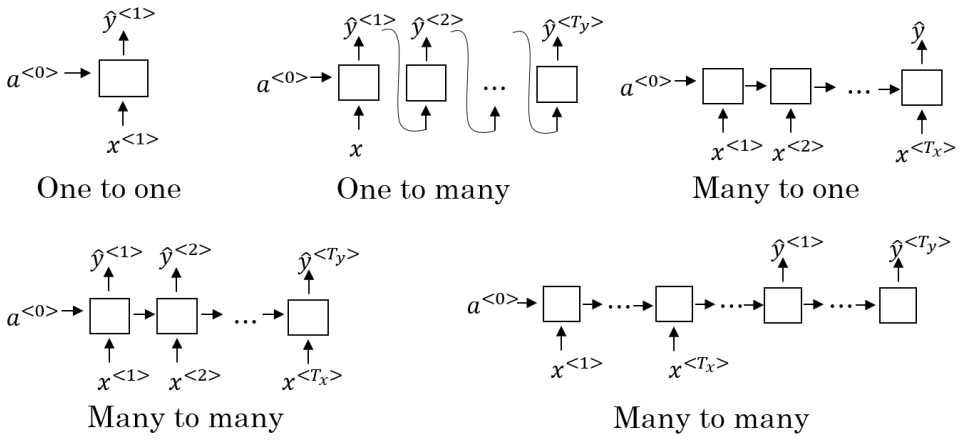
循环神经网络的反向传播被称为**通过时间反向传播（Backpropagation through time）**，因为从右向左计算的过程就像是时间倒流。

更详细的计算公式如下：



1.2.3 多种结构类型

某些情况下，输入长度和输出长度不一致。根据所需的输入及输出长度，循环神经网络可分为“一对一”、“多对一”、“多对多”等结构：



目前我们看到的模型的问题是，只使用了这个序列中之前的信息来做出预测，即后文没有被使用。可以通过**双向循环神经网络（Bidirectional RNN，BRNN）**来解决这个问题。

1.3 语言模型

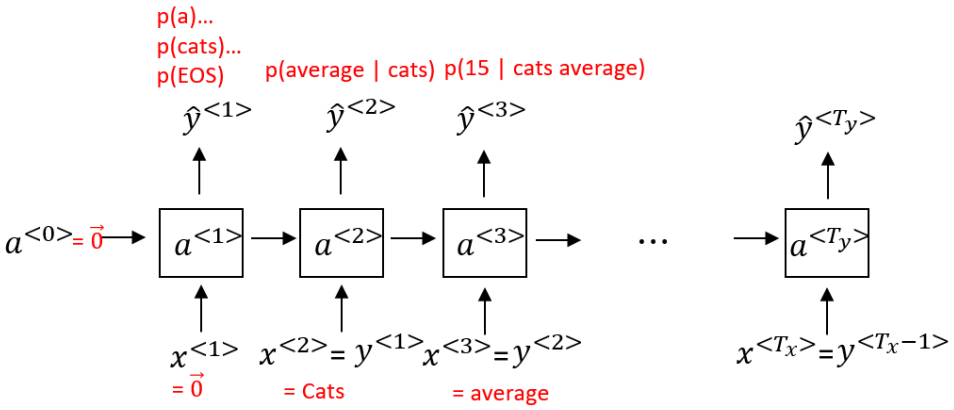
1.3.1 模型介绍

语言模型（Language Model）是根据语言客观事实而进行的语言抽象数学建模，能够估计某个序列中各元素出现的可能性。例如，在一个语音识别系统中，语言模型能够计算两个读音相近的句子为正确结果的概率，以此为依据作出准确判断。

建立语言模型所采用的训练集是一个大型的**语料库（Corpus）**，指数量众多的句子组成的文本。建立过程的第一步是**标记化（Tokenize）**，即建立字典；然后将语料库中的每个词表示为对应的 one-hot 向量。另外，需要增加一个额外的标记 EOS（End of Sentence）来表示一个句子的结尾。标点符号可以忽略，也可以加入字典后用 one-hot 向量表示。

对于语料库中部分特殊的、不包含在字典中的词汇，例如人名、地名，可以不必针对这些具体的词，而是在词典中加入一个 UNK（Unique Token）标记来表示。

将标志化后的训练集用于训练 RNN，过程如下图所示：



Cats average 15 hours of sleep a day. <EOS>

在第一个时间步中，输入的 $a^{\langle 0\rangle}$ 和 $x^{\langle 1\rangle}$ 都是零向量，$\hat{y}^{\langle 1\rangle}$ 是通过 softmax 预测出的字典中每个词作为第一个词出现的概率；在第二个时间步中，输入的 $x^{\langle 2\rangle}$ 是训练样本的标签中的第一个单词 $y^{\langle 1\rangle}$（即 “cats”）和上一层的激活项 $a^{\langle 1\rangle}$，输出的 $y^{\langle 2\rangle}$ 表示的是通过 softmax 预测出的、单词 “cats” 后面出现字典中的其他每个词，特别是 “average” 的条件概率。以此类推，最后就可以得到整个句子出现的概率。

定义损失函数为：

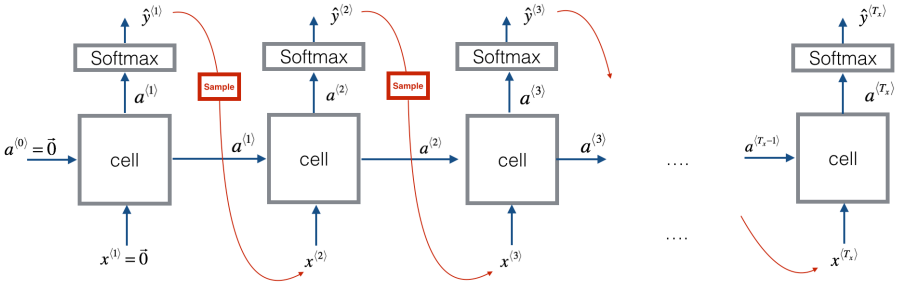
$$\mathcal{L}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) = -\sum_t y_i^{\langle t \rangle} log\ \hat{y}^{\langle t \rangle} \tag{6}$$

则成本函数为：

$$J = \sum_t L^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) \tag{7}$$

1.3.2 采样

在训练好一个语言模型后，可以通过采样（Sample）新的序列来了解这个模型中都学习到了一些什么。



在第一个时间步输入 $a^{\langle 0 \rangle}$ 和 $x^{\langle 1 \rangle}$ 为零向量，输出预测出的字典中每个词作为第一个词出现的概率，根据 softmax 的分布进行随机采样 （np.random.choice） ，将采样得到的 $\hat{y}^{\langle 1 \rangle}$ 作为第二个时间步的输入 $x^{\langle 2 \rangle}$。以此类推，直到采样到 EOS，最后模型会自动生成一些句子，从这些句子中可以发现模型通过语料库学习到的知识。

这里建立的是基于词汇构建的语言模型。根据需要也可以构建基于字符的语言模型，其优点是不必担心出现未知标识（UNK），其缺点是得到的序列过多过长，并且训练成本高昂。因此，基于词汇构建的语言模型更为常用。

1.4 梯度消失

1.4.1 问题来源

> The cat,which already ate a bunch of food, was full.
> The cats,which already ate a bunch of food, were full.

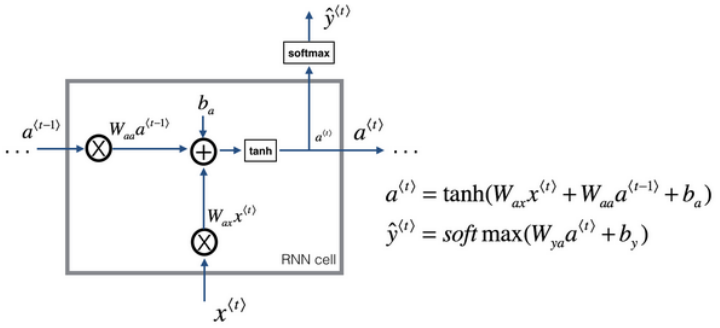
对于以上两个句子，后面的动词单复数形式由前面的名词的单复数形式决定。但是基本的 RNN 不擅长捕获这种**长期依赖关系**。究其原因，由于梯度消失，而使后面输入的序列难以受到早先输入序列的影响，在反向传播时，后面层的输出误差很难影响到较靠前层的计算，网络很难调整靠前的计算。

在反向传播时，随着层数的增多，梯度不仅可能指数型下降，也有可能指数型上升，即梯度爆炸。不过梯度爆炸比较容易发现，因为参数会急剧膨胀到数值溢出（可能显示为 NaN）。这时可以采用**梯度修剪（Gradient Clipping）**来解决：观察梯度向量，如果它大于某个阈值，则缩放梯度向量以保证其不会太大。相比之下，梯度消失问题更难解决。GRU 和 LSTM 都可以作为缓解梯度消失问题的方案。

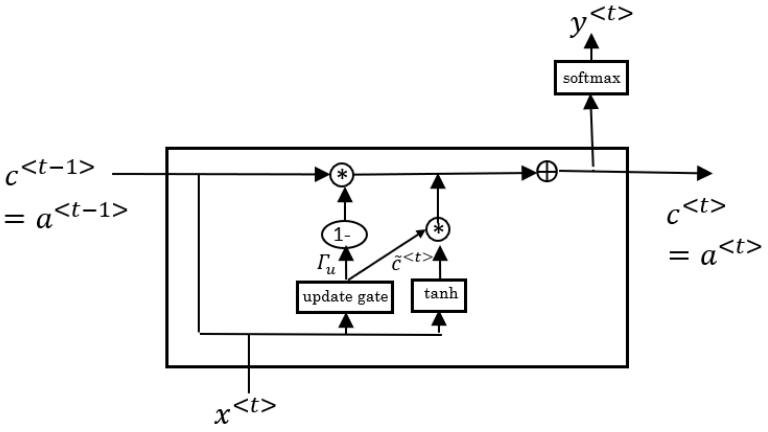
1.4.2 门控循环单元（GRU）

GRU（Gated Recurrent Units, 门控循环单元）改善了 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。

RNN 隐藏层的单元的可视化呈现如下图：



GRU 结构图如下所示：



为了理解 GRU，我们看下面这个句子：

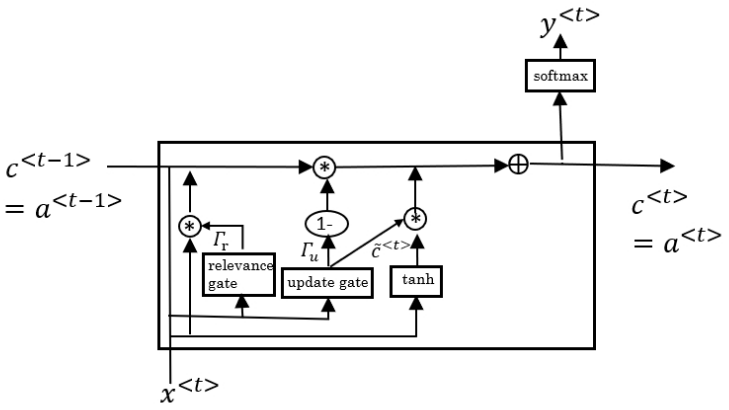
> The cat,which already ate a bunch of food, was full.

当我们从左到右读上面这个句子时，GRU 单元有一个新的变量称为 c，代表**记忆细胞（Memory Cell）**，其作用是提供记忆的能力，记住例如前文主语是单数还是复数等信息。在时间 t，记忆细胞的值 $c^{\langle t\rangle}$ 等于输出的激活值 $a^{\langle t\rangle}$；$\tilde{c}^{\langle t\rangle}$ 代表下一个 c 的候选值。Γ_u 代表更新门（Update Gate），用于决定什么时候更新记忆细胞的值。以上结构的具体公式为：

$$\begin{aligned}\tilde{c}^{\langle t\rangle} &= tanh(W_c[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_c) \\ \Gamma_u &= \sigma(W_u[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u) \\ c^{\langle t\rangle} &= \Gamma_u \times \tilde{c}^{\langle t\rangle} + (1 - \Gamma_u) \times c^{\langle t-1\rangle} \\ a^{\langle t\rangle} &= c^{\langle t\rangle}\end{aligned} \tag{8}$$

当使用 sigmoid 作为激活函数 σ 来得到 Γ_u 时，Γ_u 的值在 0 到 1 的范围内，且大多数时间非常接近于 0 或 1。当 $\Gamma_u = 1$ 时，$c^{\langle t\rangle}$ 被更新为 $\tilde{c}^{\langle t\rangle}$，否则保持为 $c^{\langle t-1\rangle}$。因为 Γ_u 可以很接近 0，因此 $c^{\langle t\rangle}$ 几乎就等于 $c^{\langle t-1\rangle}$。在经过很长的序列后，c 的值依然被维持，从而实现 “记忆” 的功能。

上面所述的是简化后的GRU，完整的GRU结构如下：



新的**相关门（Relevance Gate）**Γ_r，表示 $\tilde{c}^{\langle t \rangle}$ 和下一个 c 的候选值$c^{\langle t \rangle}$ 的相关性。

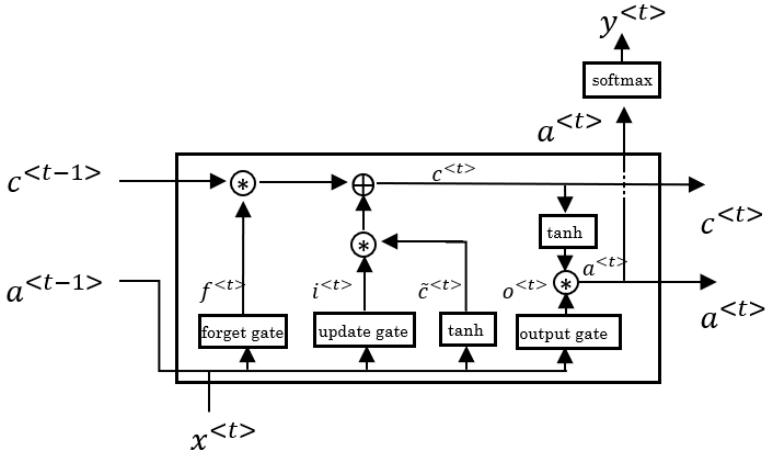
因此，表达式改为如下所示：

$$\begin{aligned} \tilde{c}^{\langle t \rangle} &= tanh(W_c[\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c) \\ \Gamma_u &= \sigma(W_u[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u) \\ \Gamma_r &= \sigma(W_r[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_r) \\ c^{\langle t \rangle} &= \Gamma_u \times \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) \times c^{\langle t-1 \rangle} \\ a^{\langle t \rangle} &= c^{\langle t \rangle} \end{aligned} \tag{9}$$

GRU其实只是一种LSTM的流行变体，其相关概念来自于 2014 年 Cho 等人发表的论文 On the properties of neural machine translation: Encoder-decoder approaches (https://arxiv.org/pdf/1409.1259.pdf) 以及 Chung 等人的 Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling (https://arxiv.org/pdf/1412.3555.pdf)。

1.4.3 长短期记忆（LSTM）

1997 年 Hochreiter 和 Schmidhuber 共同在论文 Long short-term memory (https://www.researchgate.net/publication/13853244_Long_Short-term_Memory) 中提出的 LSTM（Long Short Term Memory，长短期记忆）网络比 GRU 更加通用及强大，其结构如下：

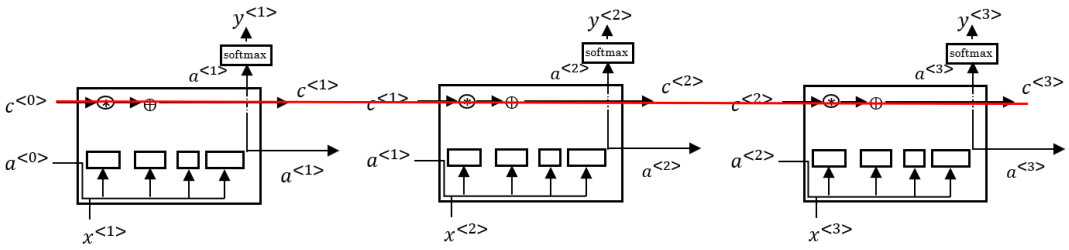


相比之前的简化版 GRU，LSTM 中多了**遗忘门（Forget Gate）**Γ_f **和输出门（Output Gate）Γ_o**，具体表达式如下：

$$\tilde{c}^{\langle t\rangle} = \tanh(W_c[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_c) \tag{10}$$
$$\Gamma_u = \sigma(W_u[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u)$$
$$\Gamma_f = \sigma(W_f[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_f)$$
$$\Gamma_o = \sigma(W_o[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_o)$$
$$c^{\langle t\rangle} = \Gamma_f^{\langle t\rangle} \times c^{\langle t-1\rangle} + \Gamma_u^{\langle t\rangle} \times \tilde{c}^{\langle t\rangle}$$
$$a^{\langle t\rangle} = \Gamma_o^{\langle t\rangle} \times \tanh(c^{\langle t\rangle})$$

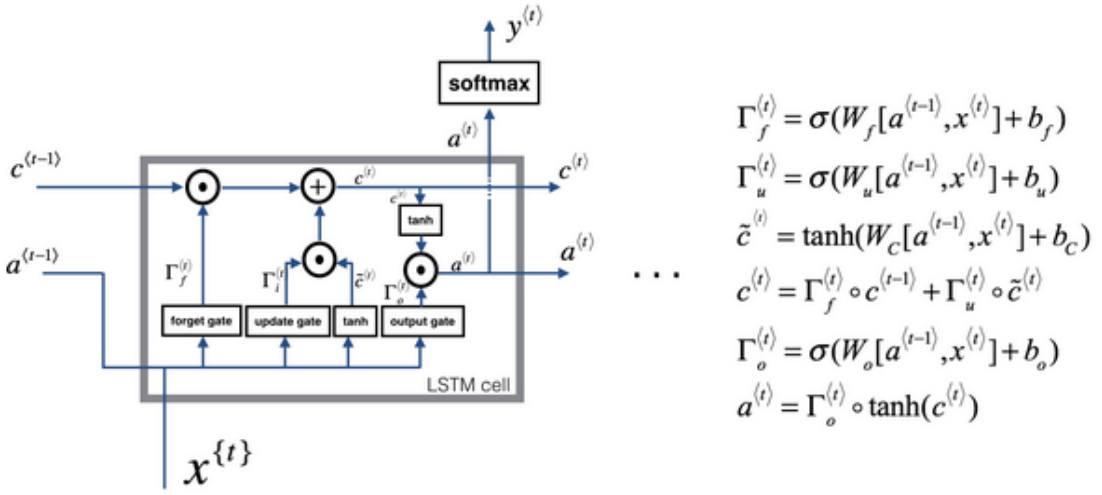
更为常用的 LSTM 版本中，几个门值的计算不只取决于输入 $a^{\langle t-1\rangle}$ 和 $x^{\langle t\rangle}$ 值，有时也可以偷窥上一个记忆细胞输入的 $c^{\langle t-1\rangle}$ 值，这叫**窥视孔连接（Peephole Connection)**。这时，和 GRU 不同，$c^{\langle t-1\rangle}$ 和门值是一对一的。c^0 常被初始化为零向量。

多个 LSTM 单元连接在一起，形成一个 LSTM 网络：

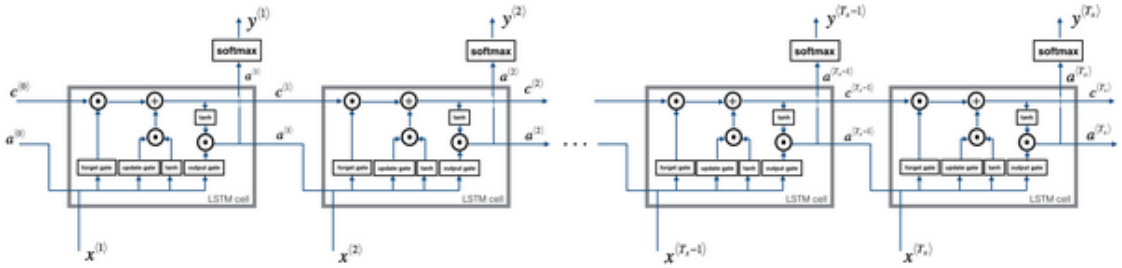


扩展

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：

$$d\Gamma_o^{\langle t\rangle} = da_{next} * \tanh(c_{next}) * \Gamma_o^{\langle t\rangle} * (1-\Gamma_o^{\langle t\rangle}) \tag{11}$$
$$d\tilde{c}^{\langle t\rangle} = dc_{next} * \Gamma_i^{\langle t\rangle} + \Gamma_o^{\langle t\rangle}(1-\tanh(c_{next})^2) * i_t * da_{next} * \tilde{c}^{\langle t\rangle} * (1-\tanh(\tilde{c})^2)$$
$$d\Gamma_u^{\langle t\rangle} = dc_{next} * \tilde{c}^{\langle t\rangle} + \Gamma_o^{\langle t\rangle}(1-\tanh(c_{next})^2) * \tilde{c}^{\langle t\rangle} * da_{next} * \Gamma_u^{\langle t\rangle} * (1-\Gamma_u^{\langle t\rangle})$$
$$d\Gamma_f^{\langle t\rangle} = dc_{next} * \tilde{c}_{prev} + \Gamma_o^{\langle t\rangle}(1-\tanh(c_{next})^2) * c_{prev} * da_{next} * \Gamma_f^{\langle t\rangle} * (1-\Gamma_f^{\langle t\rangle})$$

参数求偏导：

$$dW_f = d\Gamma_f^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T \tag{12}$$
$$dW_u = d\Gamma_u^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$
$$dW_c = d\tilde{c}^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$
$$dW_o = d\Gamma_o^{\langle t\rangle} * \begin{pmatrix} a_{prev} \\ x_t \end{pmatrix}^T$$

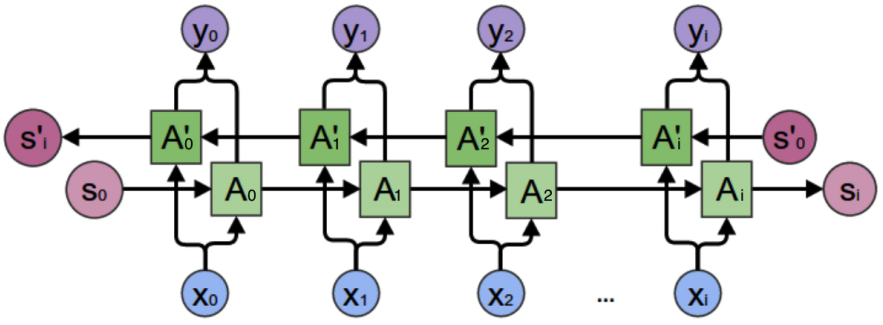
为了计算 db_f, db_u, db_c, db_o 需要各自对 $d\Gamma_f^{\langle t\rangle}, d\Gamma_u^{\langle t\rangle}, d\tilde{c}^{\langle t\rangle}, d\Gamma_o^{\langle t\rangle}$ 求和。

最后，计算隐藏状态、记忆状态和输入的偏导数：

$$da_{prev} = W_f^T * d\Gamma_f^{\langle t\rangle} + W_u^T * d\Gamma_u^{\langle t\rangle} + W_c^T * d\tilde{c}^{\langle t\rangle} + W_o^T * d\Gamma_o^{\langle t\rangle} \tag{13}$$
$$dc_{prev} = dc_{next}\Gamma_f^{\langle t\rangle} + \Gamma_o^{\langle t\rangle} * (1-\tanh(c_{next})^2) * \Gamma_f^{\langle t\rangle} * da_{next}$$
$$dx^{\langle t\rangle} = W_f^T * d\Gamma_f^{\langle t\rangle} + W_u^T * d\Gamma_u^{\langle t\rangle} + W_c^T * d\tilde{c}_t + W_o^T * d\Gamma_o^{\langle t\rangle}$$

1.5 双向循环神经网络（BRNN）

前面介绍的循环神经网络在结构上都是单向的，也提到过它们具有某一时刻的预测结果仅使用了该时刻之前输入的序列信息的缺陷，而**双向循环神经网络（Bidirectional RNN）**弥补了这一缺陷，可以在序列的任意位置使用之前和之后的数据。其工作原理是增加一个反向循环层。BRNN的结构图如下所示：



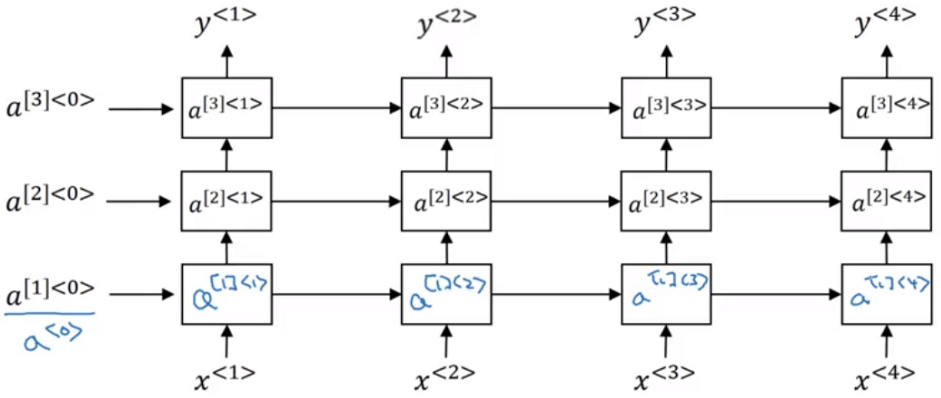
因此，有：

$$y^{\langle t\rangle} = g(W_y[\overrightarrow{a}^{\langle t\rangle}, \overleftarrow{a}^{\langle t\rangle}] + b_y) \tag{14}$$

这个改进的方法不仅能用于基本的 RNN，也可以用于 GRU 或 LSTM。通过这些改变，你就可以用一个用RNN或GRU或LSTM构建的模型，并且能够预测任意位置，即使在句子的中间，因为模型能够考虑整个句子的信息。**缺点**是需要完整的序列数据，才能预测任意位置的结果。例如构建语音识别系统，需要等待用户说完并获取整个语音表达，才能处理这段语音并进一步做语音识别。因此，实际应用会有更加复杂的模块。

1.6 深层循环神经网络（Deep RNNs）

循环神经网络的每个时间步上也可以包含多个隐藏层，形成**深度循环神经网络（Deep RNN)**。结构如下图所示：



以 $a^{[2]\langle 3\rangle}$ 为例，有 $a^{[2]\langle 3\rangle} = g(W_a^{[2]}[a^{[2]\langle 2\rangle}, a^{[1]\langle 3\rangle}] + b_a^{[2]})$