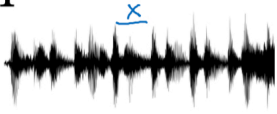
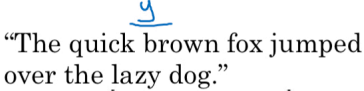


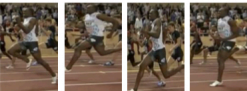


1. 为什么选择序列模型

RNN的应用场景：

- 语音识别
输入和输出都是序列数据
- 音乐生成
输入是一个数字或空，代表一种应用风格，输出的乐谱是序列数据
- 情感分类
输入是一段话，是一个序列，输出对应的类别
- DNA序列分析
找到输入的DNA序列的蛋白质表达的子序列
- 机器翻译
- 视频行为识别
识别输入的视频帧序列中的人物行为
- 命名实体识别
从输入的句子中识别实体的名字

Examples of sequence data

Speech recognition		→	
Music generation	\emptyset	→	
Sentiment classification	"There is nothing to like in this movie."	→	
DNA sequence analysis	AGCCCCTGTGAGGAAGTAG	→	AGCCCCTGTGAGGAAGTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger .

2. 数学符号

x: Harry Potter and Hermione Granger invented a new spell.

$x^{(1)}$ $x^{(2)}$ \dots $x^{(t)}$ \dots $x^{(9)}$

$T_x = 9$ 输入序列长度

y: 1 1 0 1 1 0 0 0 0
 $y^{(1)}$ $y^{(2)}$ \dots $y^{(t)}$ \dots $y^{(9)}$

$T_y = 9$ 输出序列长度

$x^{(i)(t)}$ 第i个序列的第t个元素

$y^{(i)(t)}$ 第i个输出序列的第t个元素

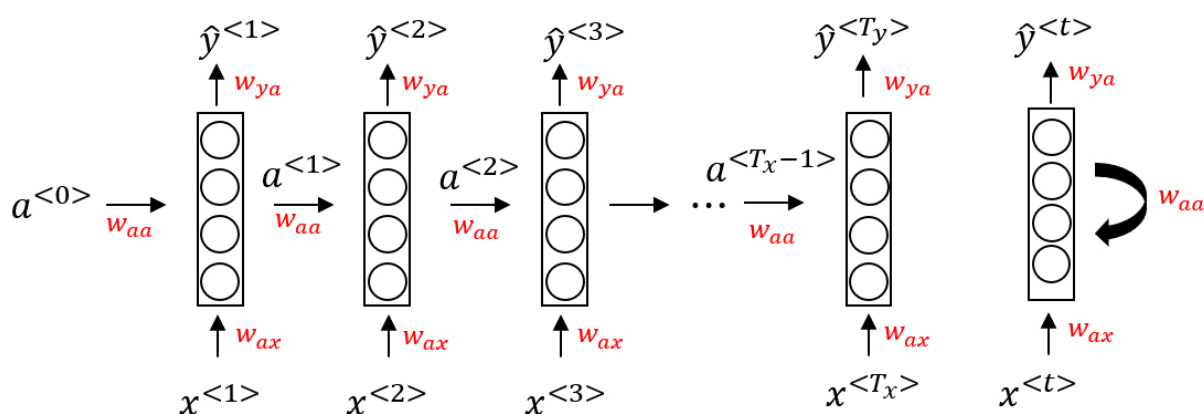
对于单词，用one-hot向量表示，未在词典中的单词统一为<UNK>

3. RNN Model

对于序列数据，使用标准的前馈神经网络主要存在如下问题：

- 对于不同的样本，输入和输出的长度可能不同，因此输入层和输出层的神经元个数无法固定；
- 从输入文本的不同位置学得特征无法共享；
- 模型参数太多，计算量太大（比如用one-hot表示词向量，输入神经元个数是非常大）

一个标准的RNN结构如下图所示：



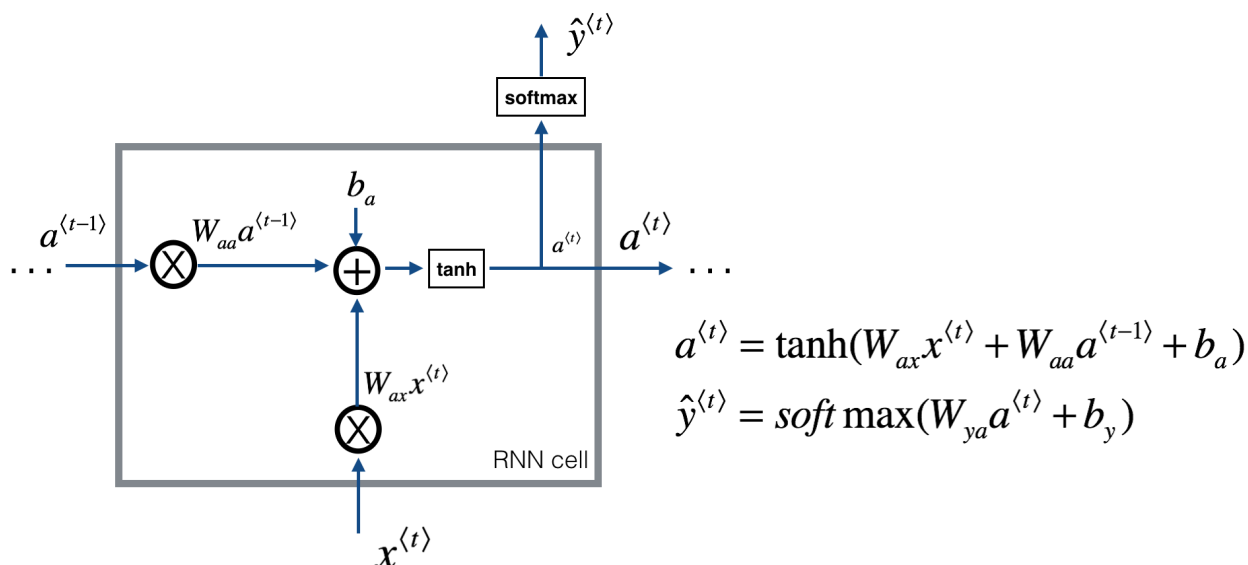
在 $t = 0$ 时刻，需要构造一个初始激活值 $a^{<0>}$ ，通常输入一个零向量。

循环神经网络从左向右扫描数据，每个时间步的参数是共享的：

- w_{aa} ：激活值到隐藏层的权重
- w_{ax} ：输入 $x^{<t>}$ 到隐藏层的权重
- w_{ya} ：隐藏层到输出的权重

4. 前向传播

一个神经元的结构如下图：



前向传播公式如下：

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

激活函数 g_1 通常选用 \tanh 函数，有时也用Relu； g_2 如果是二分类选用 sigmoid 函数，如果是多分类，选用 softmax 函数。

可以对上面的一般表达式进行简化：

$$\begin{aligned} a^{<0>} &= \vec{0} \\ a^{<t>} &= g_1(W_a[x^{<t>}; a^{<t-1>}] + b_a) \\ \hat{y}^{<t>} &= g_2(W_{ya}a^{<t>} + b_y) \end{aligned}$$

其中， $W_a = [W_{ax}, W_{aa}]$

5. 反向传播

在单个位置（或单个时间步）上某个单词预测的损失函数使用交叉熵损失函数：

$$\begin{aligned} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) &= -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log(1 - \hat{y}^{<t>}) \\ \mathcal{L}(\hat{y}, y) &= \sum_{i=1}^m \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) \end{aligned}$$

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：

$cache = (a^{(t)}, a^{(t-1)}, x^{(t)}, parameters)$

parameters gradients:

$\frac{\partial a^{(t)}}{\partial W_x}$	$\frac{\partial a^{(t)}}{\partial W_a}$	$\frac{\partial a^{(t)}}{\partial b}$
---	---	---------------------------------------

RNN cell

$$\frac{\partial J}{\partial a^{(t-1)}} = \frac{\partial J}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial a^{(t-1)}}$$

...

$$\frac{\partial J}{\partial x^{(t)}} = \frac{\partial J}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial x^{(t)}}$$

$$\frac{\partial J}{\partial a^{(t)}} \dots$$

$$a^{(t)} = \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b)$$

$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh(x)^2$$

$$\frac{\partial a^{(t)}}{\partial W_{ax}} = (1 - \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b)^2) x^{(t)T}$$

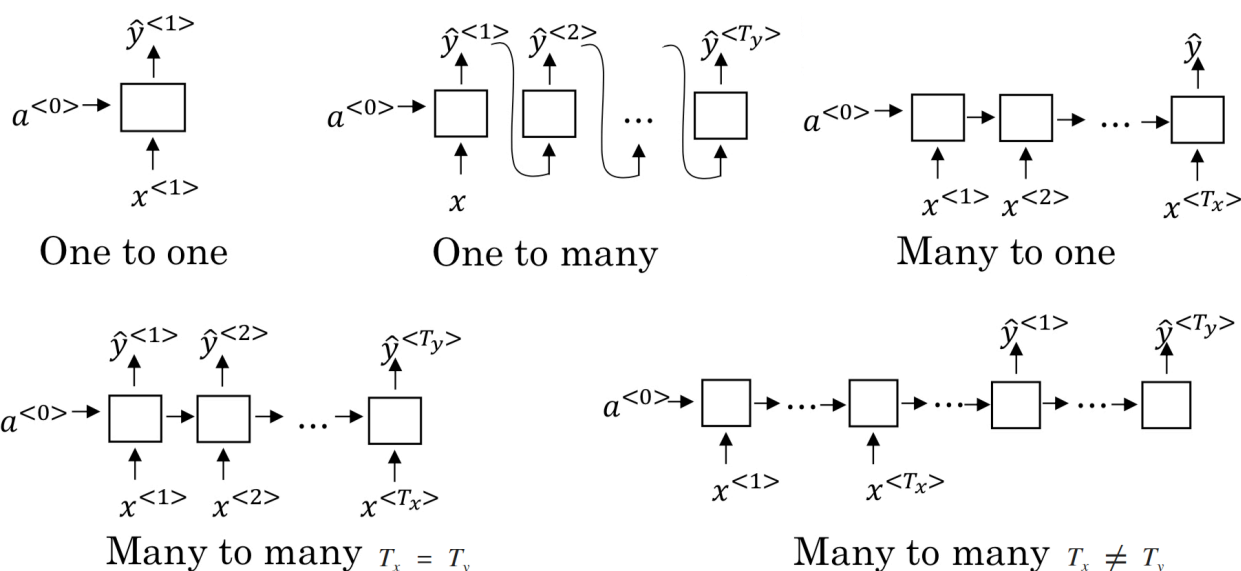
$$\frac{\partial a^{(t)}}{\partial W_{aa}} = (1 - \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b)^2) a^{(t-1)T}$$

$$\frac{\partial a^{(t)}}{\partial b} = \sum_{batch} (1 - \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b)^2)$$

$$\frac{\partial a^{(t)}}{\partial x^{(t)}} = W_{ax}^T \cdot (1 - \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b)^2)$$

$$\frac{\partial a^{(t-1)}}{\partial a^{(t-1)}} = W_{aa}^T \cdot (1 - \tanh(W_{ax}x^{(t-1)} + W_{aa}a^{(t-2)} + b)^2)$$

6. 不同类型的RNN



7. 语言模型和序列生成

7.1 什么是语言模型

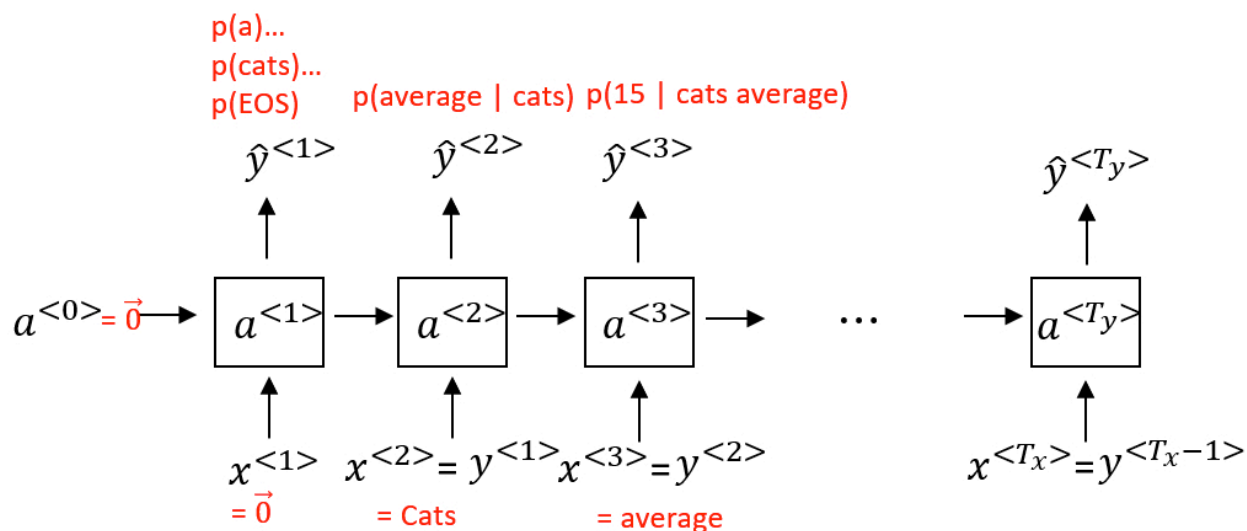
语言模型 (Language Model) 是根据语言客观事实而进行的语言抽象数学建模，能够估计某个序列中各元素出现的可能性。例如，在一个语音识别系统中，语言模型能够计算两个读音相近的句子为正确结果的概率，以此为依据作出准确判断。

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。

7.2 使用RNN构建语言模型

- 准备语料库 (Corpus)
- Tokenize: 分词构建字典
- 将语料库中的每个词用对应的 one-hot 向量表示;
- 需要增加一个额外的标记 <EOS> (End of Sentence)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;
- 对于为登录词OOV (Out of Vocabulary), 在字典中加入一个 <UNK> (Unknown Words)标记来表示。



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

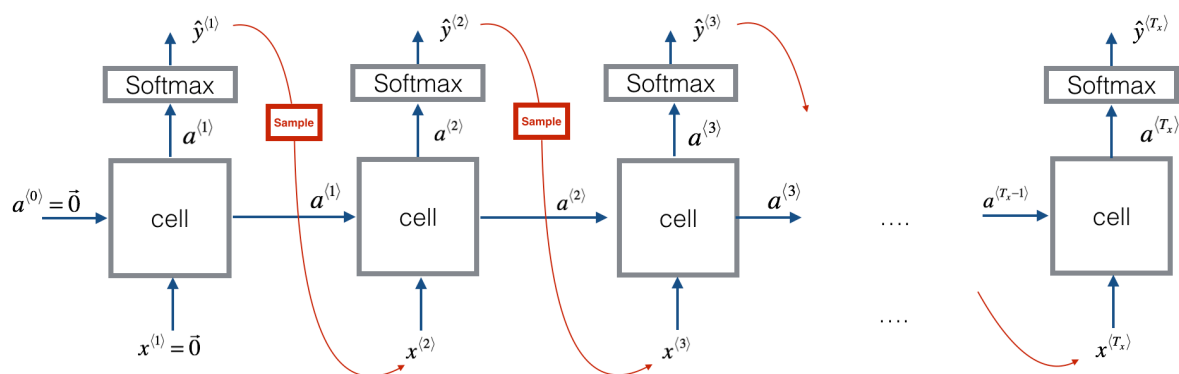
$$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = - \sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$
$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

8. 新序列采样

在完成一个序列模型的训练之后, 如果我们想要了解这个模型学到了什么, 其中一种非正式的方法就是进行一次新序列采样 (sample novel sequences) 。

对于一个序列模型, 其模拟了任意特定单词序列的概率, 如 $P(y^{<1>}, \dots, y^{<T_y>})$, 而我们要做的就是对这个概率分布进行采样, 来生成一个新的单词序列。

采样过程如下图所示:



采样步骤如下：

1. 第一个时间步的输入 $a^{<0>} = \vec{0}$, $x^{<1>} = \vec{0}$, 输出为预测字典中每个词作为第一个词的概率, 根据 softmax 后的分布, 进行随机采样 (`np.random.choice`), 获得第一个采样词 $\hat{y}^{<1>}$;
2. 将采样得到的 $\hat{y}^{<1>}$ 传入第二时间步的 cell, 进入 softmax 层预测出下一个输出 $\hat{y}^{<2>}$, 然后继续采样;
3. 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。

9. RNN的梯度消失

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

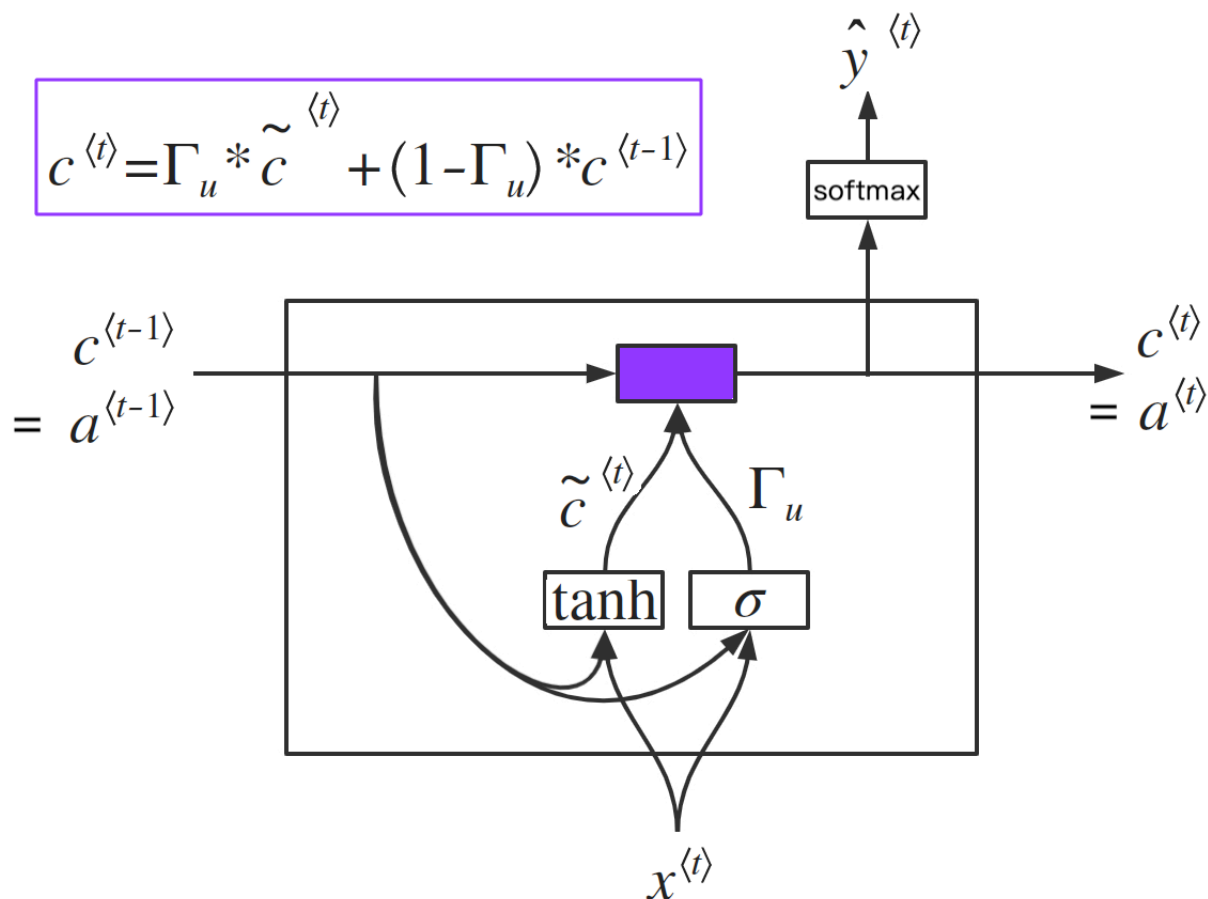
The cats, which already ate a bunch of food, were full.

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, 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。

10. GRU

门控循环单元 (Gated Recurrent Unit, GRU) 改善了隐藏层单元, 使其能够更好捕捉更深层次连接, 并改善了梯度消失问题!



GRU Cell具体公式为：

$$\begin{aligned}
 c^{<t>} &= a^{<t>} \\
 \tilde{c}^{<t>} &= \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c) \\
 \Gamma_u &= \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u) \\
 c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}
 \end{aligned}$$

c ：代表记忆细胞， t 时刻， $c^{<t>} = a^{<t>}$

$\tilde{c}^{<t>}$ ：下一个 c 的候选值

Γ_u ：更新门，用于决定什么时候更新记忆细胞的值

更新门使用sigmoid函数作为激活函数，则 Γ_u 在大多数时间都非常接近0或1。当 $\Gamma_u = 1$ 时， $c^{<t>}$ 更新为 $\tilde{c}^{<t>}$ ，否则保持上一时间步的激活值 $c^{<t-1>}$ 。因为 Γ_u 可以很接近0，因此 $c^{<t>}$ 几乎就等于 $c^{<t-1>}$ 。在经过很长的时间序列后， c 的值依然被维持，从而实现“记忆”功能。

向量中某一维=1

assume $\Gamma_u=1$ $\Gamma_u=0$ $\Gamma_u=0$ $\Gamma_u=0$ $\Gamma_u=1$

$c^{<t>}=1$... $c^{<t>}=1$... $c^{<t>}=other$

The cat, which ate already, was full.

上述是简化版的GRU，完整的GRU还包含了相关门（**Relevance Gate**），用于表示 $c^{<t>}$ 和 $\tilde{c}^{<t>}$ 的相关性：

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

注：上面所有式子中的*表示element wise product.

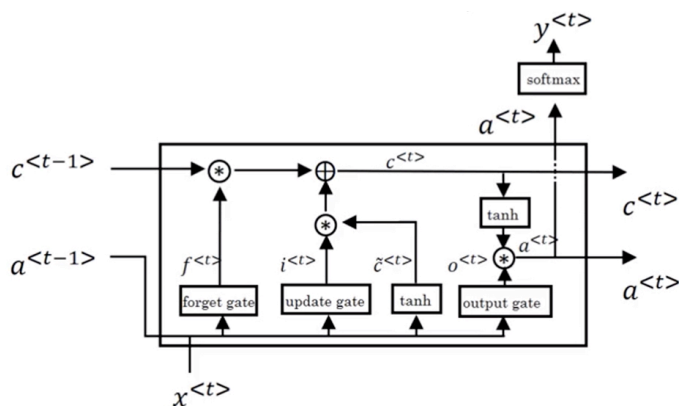
11. LSTM

GRU能够让我们在序列中学习到更深的联系，长短期记忆（**long short-term memory, LSTM**）对捕捉序列中更深层的联系要比GRU更加有效。

$$\begin{aligned}\tilde{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\ \Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \\ \Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \\ \Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \\ c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\ a^{<t>} &= \Gamma_o * \tanh c^{<t>}\end{aligned}$$

$c^{<0>}$ 常被初始化为零向量。

$$\begin{aligned}\tilde{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\ \Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \\ \Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \\ \Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \\ c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\ a^{<t>} &= \Gamma_o * \tanh c^{<t>}\end{aligned}$$

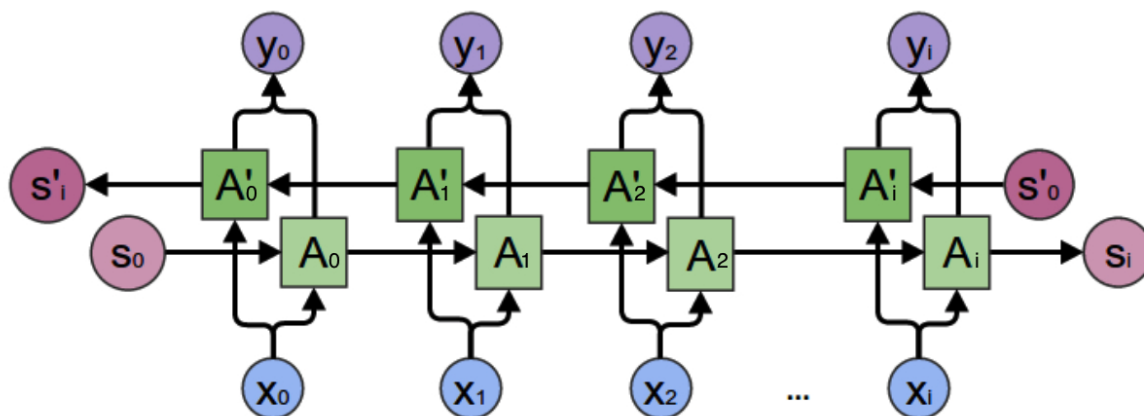


其中，在实际使用时，几个门值不仅仅取决于 $a^{<t-1>}$ 和 $x^{<t>}$ ，还可能会取决于上一个记忆细胞的值 $c^{<t-1>}$ ，这也叫做偷窥孔连接。

12. 双向RNN

单向的循环神经网络在某一时刻的预测结果只能使用之前输入的序列信息。双向循环神经网络

（**Bidirectional RNN, BRNN**）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，结构如下图所：



$$\hat{y}^{<t>} = g\left(W_y[\vec{a}^{<t>}, \overleftarrow{a}^{<t>} + b_y]\right)$$

这个改进的方法不仅能用于基本的 RNN，也可以用于 GRU 或 LSTM。

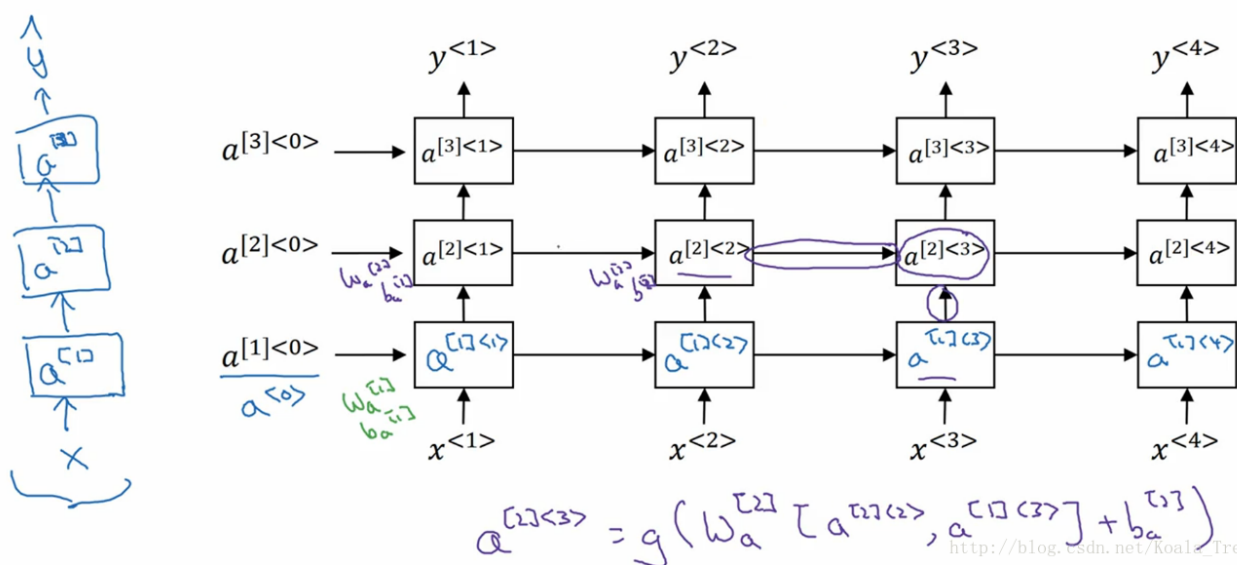
缺点：

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。

13. 深层RNN

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：

Deep RNN example



14. 疑问

1. 用RNN训练语言模型，如此定义损失函数，是否要求每条预料都是等长的？（或扩充到等长？）

$$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = - \sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$
$$\mathcal{L} = \sum_t \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

2. 采样时，经softmax层后每个单词的概率是不等的，使用 `np.random.choice` 时是如何保证依概率采样？还是等可能采样？

15. 参考

1. [吴恩达《深度学习》系列课程笔记](#)