

# 目录


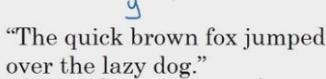




Week1 循环序列模型 .....	1
1.1 为什么要选择序列模型.....	1
1.2 数学符号 .....	2
1.3 RNN 模型.....	3
1.4 通过时间的反向传播 BPTT.....	5
1.5 不同类型的 RNN.....	6
1.6 语言模型和序列生成 .....	7
1.7 对新序列采样 .....	10
1.8 梯度消失.....	11
1.9 门控循环单元 GRU.....	12
1.10 长短期记忆 Long Short Term Memory.....	14
1.11 双向神经网络 Bidirectional RNN.....	15
1.12 深层循环神经网络 Deep RNNs .....	16

## Week1 循环序列模型

### 1.1 为什么要选择序列模型

在实际生活中，我们经常要求模型的输入或输出是序列化的数据。语音、文字、视频等都是序列化数据。序列化数据中的每个数据是有一定关联的，如果采用传统的模型每次把数据作为独立的输入，效果并不好。所以，这里专门采用 RNN 模型来处理序列化数据。

以下例子都属于序列化数据：

Examples of sequence data		
Speech recognition		→ 
Music generation		→ 
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, <b>Harry Potter</b> met <b>Hermione Granger</b> .

例 1：语音识别。将输入的一段音频转换成对应的文本内容。

例 2：音乐生成。生成一段音乐（音乐也是序列化的数据）。

例 3：情感分类。根据输入的一段文字，猜测其感情色彩。

例 4：DNA 分析。人的 DNA 是一种由 ACTG 四种含氮碱基组成的序列。识别其中某部分序列对应的蛋白质。

例 5：机器翻译。输入一段文本，将其翻译成另一种语言的文本。

例 6：视频行为识别。输入一段视频（可视为由一系列视频帧组成的序列），识别视频中物体的动作/行为。

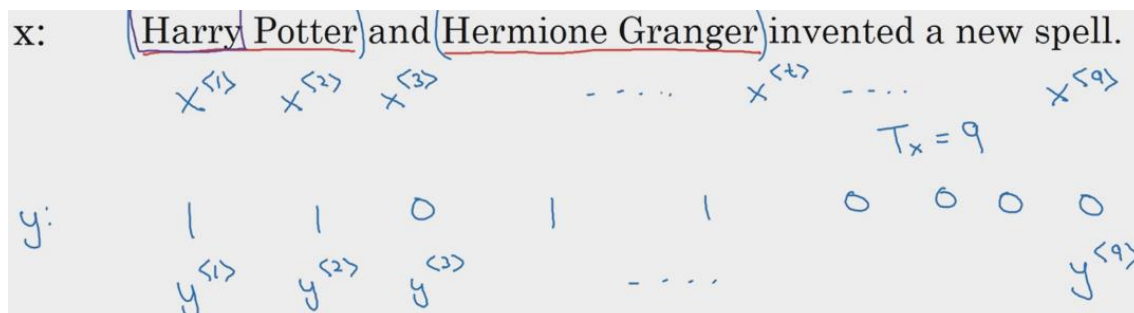
例 7：命名实体识别。给定一个句子，识别其中的人名等。

由上，序列模型输入  $x$  和输出  $y$  可以都是序列，也可以只有其中一个是序列。

## 1.2 数学符号

现在以 Name entity recognition 为例，设置数学符号表示，逐步构建模型：

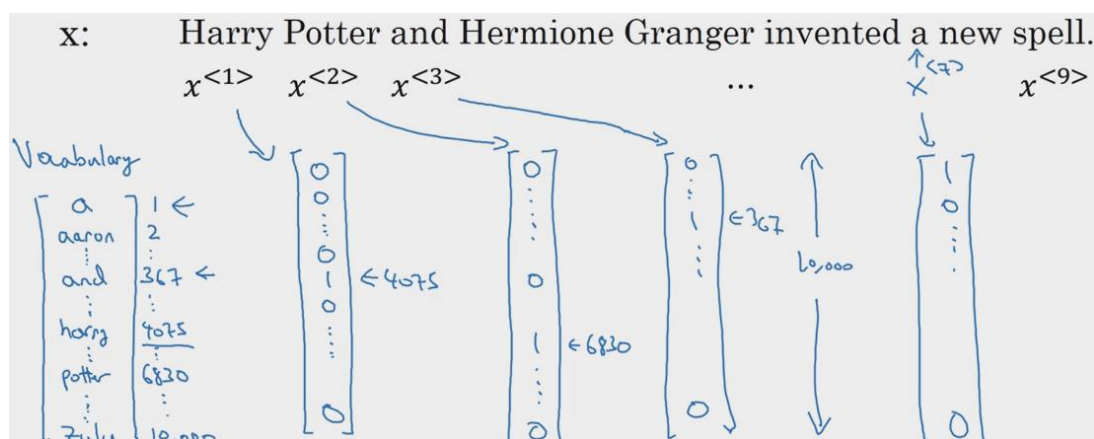
设输入为  $x$ ，其中  $x^{<t>}$  代表该输入文本序列第  $t$  个单词， $T_x$  代表输入  $x$  的序列长度；输出  $y$  也是相同长度的 01 序列（0 代表非名字，1 代表是人名），同理  $y^{<t>}$  代表序列第  $t$  个位置的识别结果，且  $T_y = T_x$ ：



对比以前， $x^{(i)}$  代表数据集（许多文本）的第  $i$  个样本，那么此时  $x^{(i)<t>}$  则代表第  $i$  个文本样本的第  $t$  个单词； $y^{(i)<t>}$  代表第  $i$  个样本的第  $t$  个单词的预测结果。改写得  $T_x^{(i)} = T_y^{(i)}$ 。由于每个样本的文本长度可能不同，所以不同样本间  $T_x^{(i)} \neq T_x^{(j)}$ 。

将输入文本转换为序列：

接下来介绍怎样表示输入文本中的每个单词：先确定一个适当大小的词汇表（Vocabulary/Dictionary，类似 ML 做 naive bayes 时的例子），词汇表选择英语中最常用的 10K 个单词；其次用 **one-hot** 方法，对照词汇表可把每个单词抽象成一个 one-hot 向量（仅一个元素为 1，其余全部为 0）：



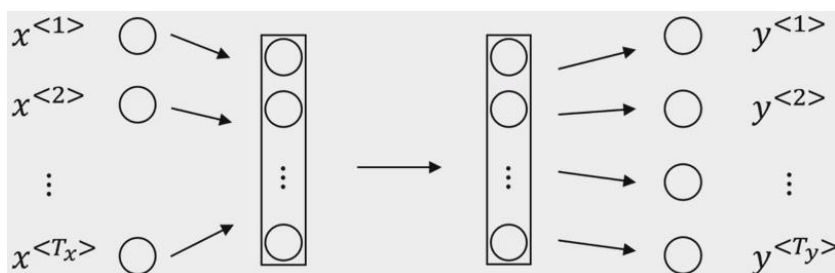
这样，输入的文本就可以看出一个个的向量序列了。此处我们把它视为**监督学习**，每个句子已经有了现成的标签  $y^{(i)}$ 。后面的事情就是学习输入的序列数据  $x^{<1>}, x^{<2>}, \dots, x^{<t>}$  与标签  $y^{<1>}, y^{<2>}, \dots, y^{<t>}$  之间的映射了。

## 1.3 RNN 模型

### 使用传统神经网络的缺陷：

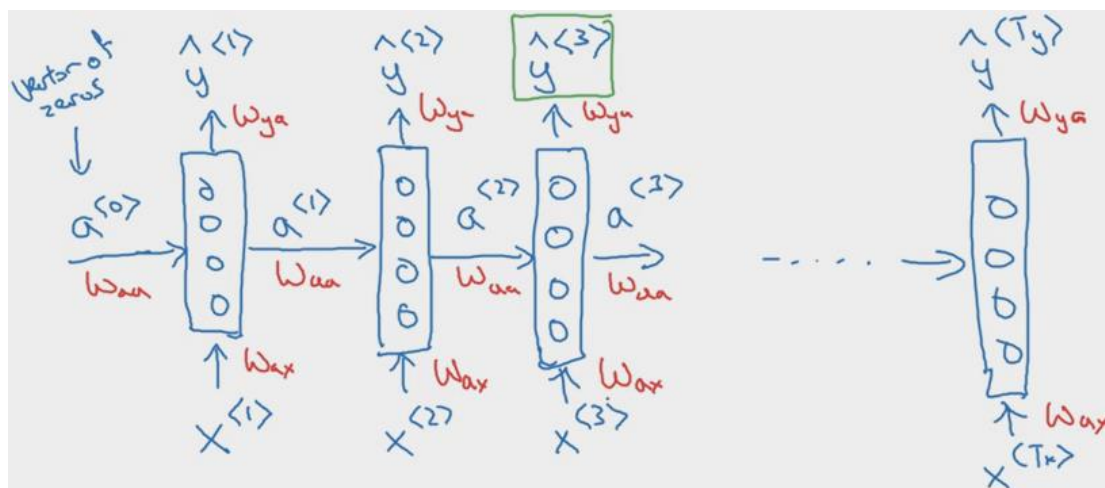
1. 由于每个样本文本的长度可能不同，所以导致固定的网络模型无法适配不同的数据。尽管有时候可以使用零扩展使得每个样本长度都变为最大长度，但这样做的效果也并不好；

2. 不能共享参数。在 CNN 中，我们能将模型学到的图片某一部分内容快速推广到图片的其他部分。同理，在 RNN 中，我们希望把学到的部分人名(如 Larry)，在下次学到的时候迅速识别为人名的一部分。参数共享还能减少模型参数的数量。



### RNN 模型：

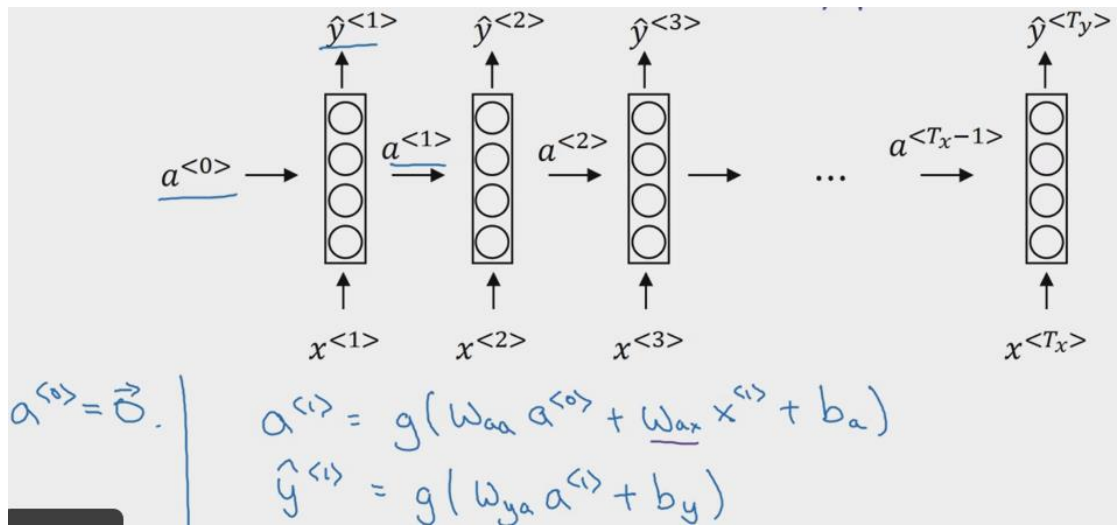
在 RNN 中，每个单词的预测值不仅与当前输入的单词  $x^{<t>}$  有关，还与上一个单词的激活值有关。这更符合文本的处理，因为相邻单词间由于语法的限制存在一定关系：



由该 RNN 结构可知，每次预测当前值时，会综合之前所有出现过的历史信息。但是它的缺点是没有结合未来的信息。

在实际语法中，后面的单词肯定也与当前单词有关。由此专门引入了双向循环神经网络 BRNN。

RNN 的前向传播:



简化写法:

将两个参数矩阵  $W_{aa}, W_{ax}$  横着合并成  $W_a$ , 并将矩阵  $a^{<t-1>}$  和矩阵  $x^{<t>}$  竖着合并为一个矩阵。则新的 FP 公式如下:

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

Dimensions:  $(100, 100) \times 100 + (100, 10,000) \times 10,000$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

Dimensions:  $(100) \times (100, 10,000)$

Matrix concatenation:  $\begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix} = W_a$  (Dimensions:  $(100, 10,100)$ )

证明过程如下:

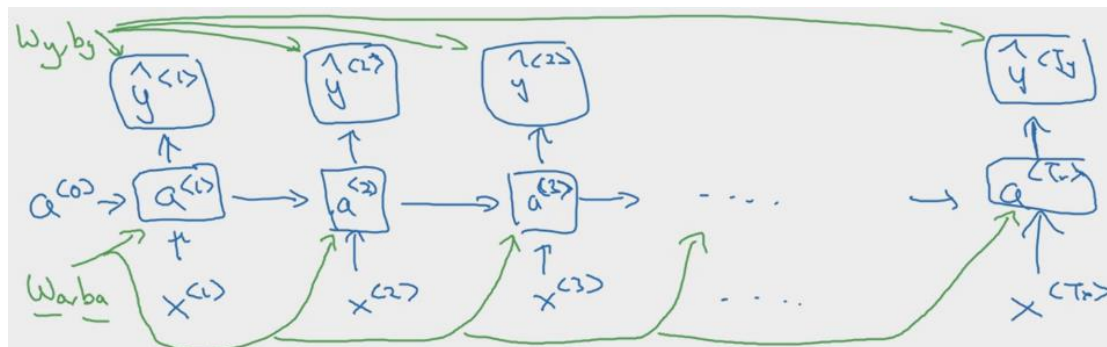
$$A \cdot X + B \cdot Y = [A \ B] \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$AX + BY = \begin{bmatrix} \sum_k A_{1k} X_k + \sum_j B_{1j} Y_j & \dots & \dots \\ \vdots & & \\ \dots & \dots & \sum_k A_{mk} X_k + \sum_j B_{mj} Y_j \end{bmatrix} = [A \ B] \begin{bmatrix} X \\ Y \end{bmatrix}$$



## 1.4 通过时间的反向传播 BPTT

回顾前向传播，由于参数共享，所以每个时间步计算激活项  $a$  或者预测值  $y$  时，用到的参数  $w_a, b_a, w_y, b_y$  等都是相同的。如下图：



定义误差函数：

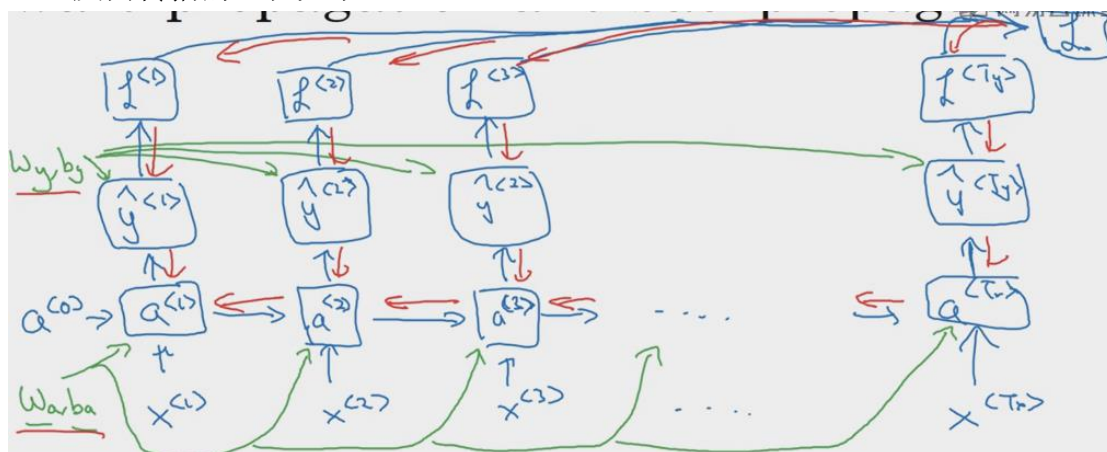
现在为了计算误差，我们先定义误差函数  $L^{(t)}$  代表当前样本序列中第  $t$  个单词的误差。则当前样本的总误差  $L$  为所有单词误差之和：

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

$$L(\hat{y}, y) = \sum_{t=1}^T L^{(t)}(\hat{y}^{(t)}, y^{(t)}) \leftarrow$$

同理，之后我们通过反向传播算法，可以得到总误差关于每个参数的梯度；再调用梯度下降算法，就可以很方便得到最优化的模型了。

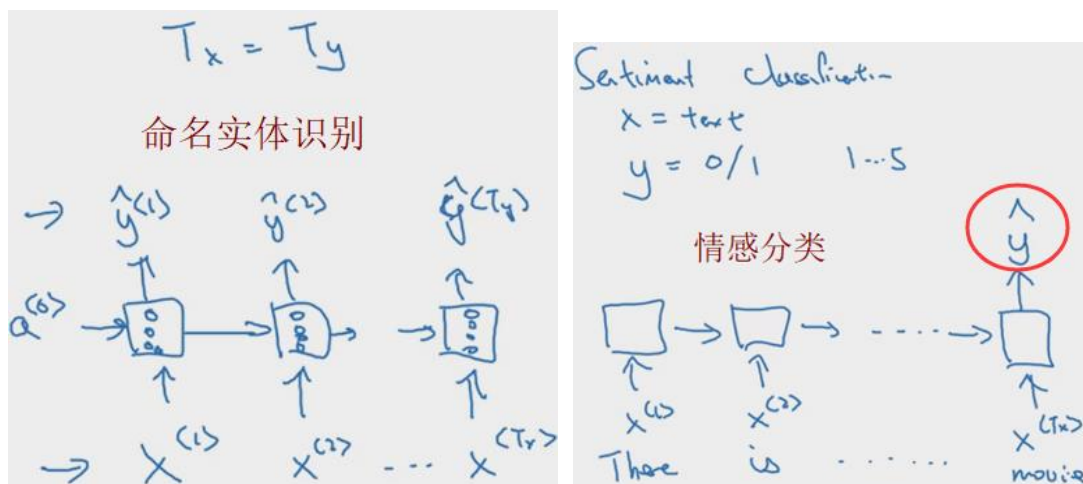
反向传播的过程如下：



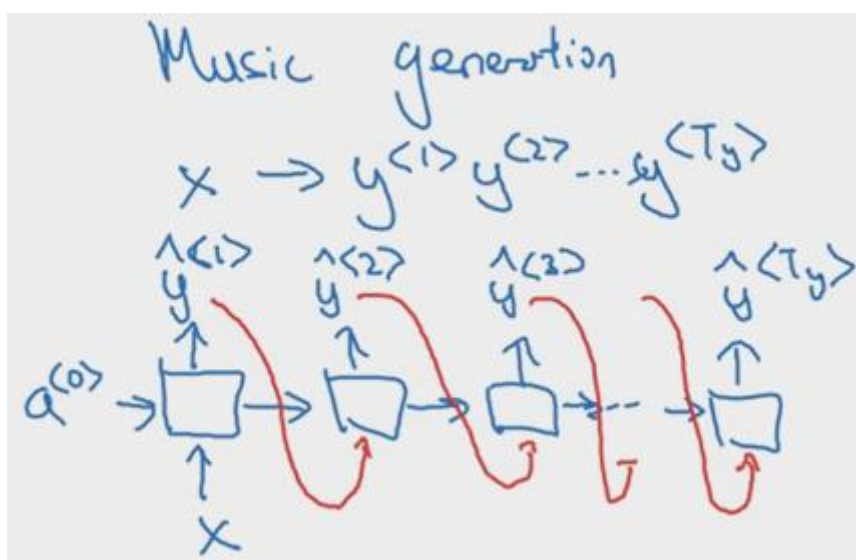
## 1.5 不同类型的 RNN

根据 RNN 中输入  $x$  和输出  $y$  的长度大小关系，我们可以将 RNN 分为多对多、多对一等结构。

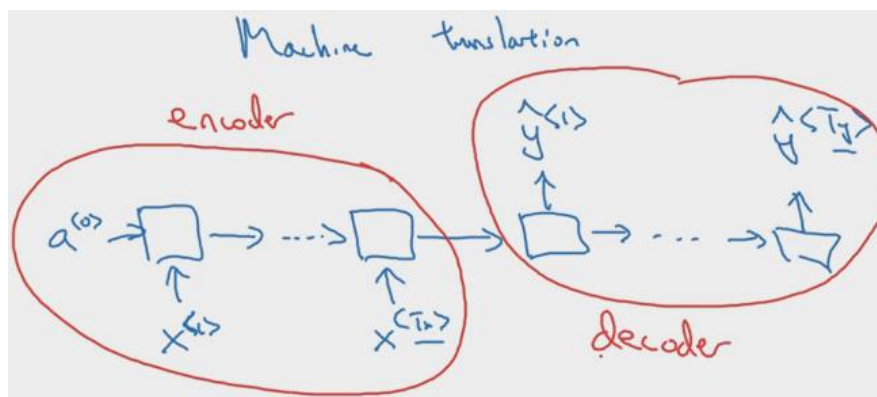
上文中的例子为命名实体识别，输出一个与  $x$  等长的向量  $\hat{y}$ ，所以它属于多对多结构；而在情感分类模型中，输入一段序列，我们只在模型的最后一步才有输出（评分/喜爱程度），所以它是多对一模型：



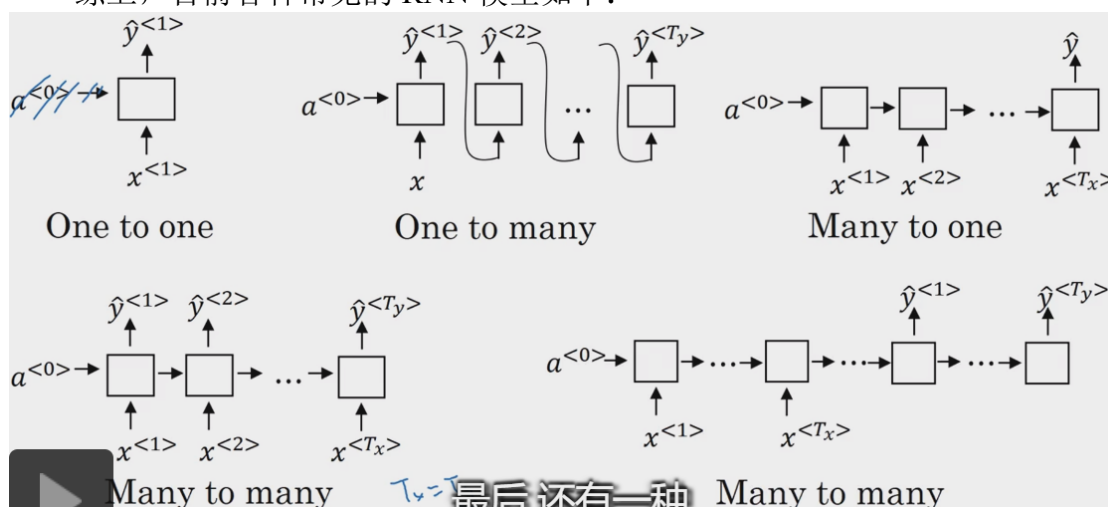
对于音乐生成模型。输出是音乐（音频序列），它的输入可以是空，所以它属于一对多模型：



而对于多对多模型，输入和输出序列的长度是可以不同的。比如机器翻译应用中，英文的文本和对应的中文文本长度肯定不相同。一般会使用 Encoder-Decoder 结构，模型如下：



综上，目前各种常见的 RNN 模型如下：



## 1.6 语言模型和序列生成

语言模型(Language Model)是 NLP 中最基础和最重要的任务之一，它可以通过 RNN 很好地实现。在 Speech recognition 情景中，语言模型的功能是告诉你某个句子出现的概率是多少。同理，机器翻译中，也会得知每个目的句子正确的概率，并选择最大正确率的输出：

$$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$$

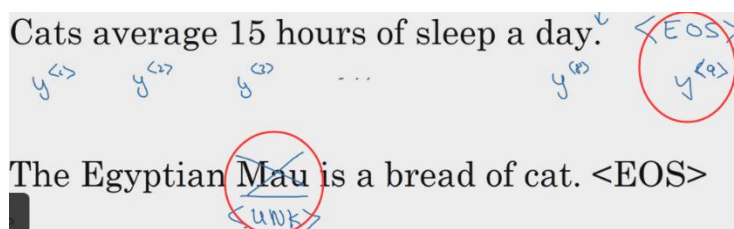
$$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$$

$$P(\text{sentence}) = ?$$

用 RNN 建立语言模型：

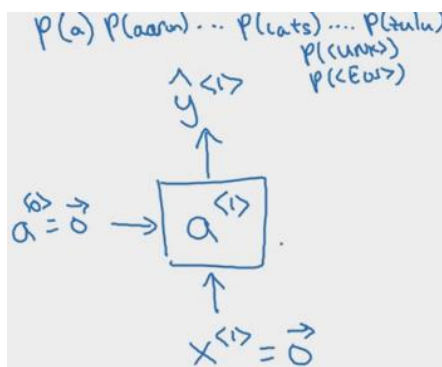
首先进行标志化 Tokenize：在每个序列的末尾加上结束符 EOS(End Of Sentence)；其次根据词汇表（由训练集生成 Top1000 常出现单词），将输入序列中未出现过的单词设置为 UNK (Unknown Word)：



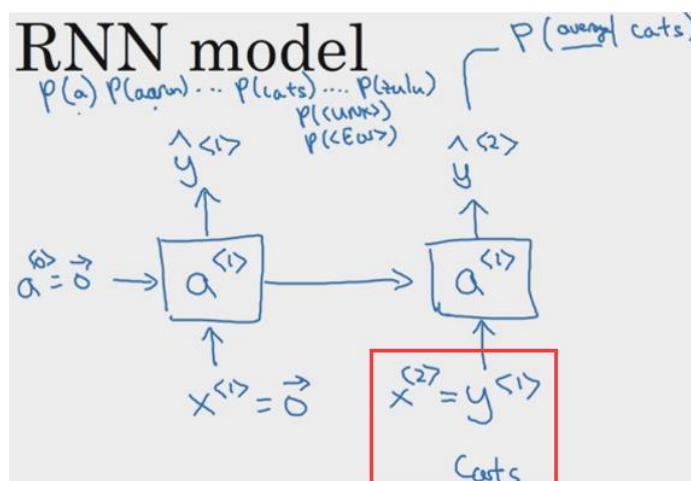


### 序列生成:

首先, 令  $a^{(0)}=0$ ,  $x^{(1)}=0$ , 把它们作为输入计算得  $a^{(1)}$ 。把  $a^{(1)}$  作为 softmax 的输入将得到第一个单词是 ??? 的概率, 即  $y^{(1)}=[p(a), p(Aron), p(cat), \dots, p(Zulu), p(EOS), p(UNK)]$ , 它是一个向量, 每个元素代表输出句子第一个单词为词汇表中第  $i$  个单词的概率。注意: 由于它是第一个单词, 所以不是条件概率!



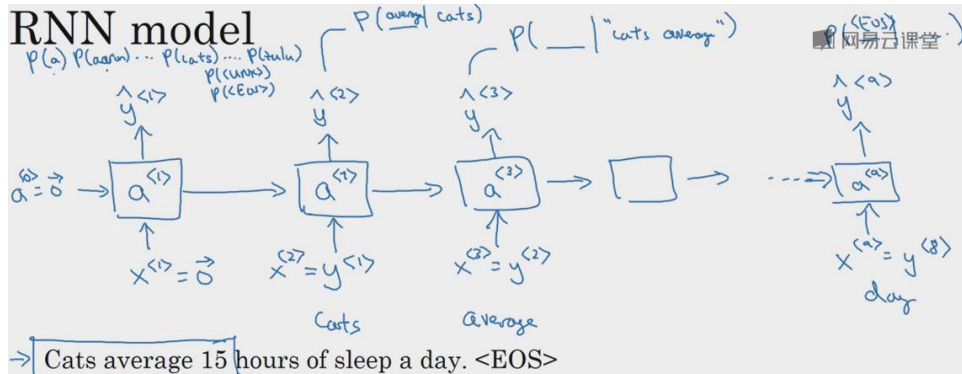
其次, 进入下个时间步。  $y^{(1)}$  将预测第二个单词是字母表各个单词的概率。但是此时我们会把第一个正确单词的 label 值  $y^{(1)}$  作为输入  $x^{(2)}$ , 所以此时得到的预测值  $y^{(2)}$  是一个条件概率值, 它是基于给定正确值  $x^{(2)}=$  "cats" 的条件下的概率值。即  $y^{(2)}=[p(\text{average}|\text{cats}), p(a|\text{cats}), \dots, p(UNK|\text{cats})]$ 。



再进入下一时间步, 此时我们将预测第三个单词。我们会把第二个单词的 label 值  $y^{(2)}$  作为输入  $x^{(3)}$ 。也就相当于我们现在要在已知前两个单词 "cats average" 的情况下, 预测第三个单词的值。故  $y^{(3)}=[p(a|\text{"cats average"}), p(b|\text{"cats$

average”)...p(UNK|”cats average”)】。

由此直到序列最后一个单词， $x^{<9>}=y^{<8>}=$ ”day”，得到  $y^{<9>}$ （以上各  $y^{<t>}$  的维度都相同！）。当然，我们希望从  $y^{<9>}$  中得到正确的选择  $p(\text{EOS}|\text{”cats average 15 hours of sleep a day”})$ 。良好的模型将在这个概率上有较大的值，即有较高的准确率：



以上过程类似前向传播：把单词序列作为输入，依次得到各步的预测值。又由于已知各处真实值（one-hot 形式），于是可由代价函数来训练该 RNN。

**代价函数：**

这里，为了训练 RNN，我们需要设置代价函数  $L$ 。若第  $t$  个单词的真实值（label）为  $y^{<t>}$ ，预测值为  $\hat{y}^{<t>}$ 。则 RNN 在该样本的第  $t$  个单词的误差为  $L(y^{<t>}, \hat{y}^{<t>})$ ：

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

注意，这里由于 RNN 每层的输出  $\hat{y}^{<t>}$  都是一个向量，向量的每个值  $\hat{y}_i^{<t>}$  代表当前单词为词汇表第  $i$  个单词的概率。所以此处需要对该单词每种预测的误差求和。其次， $y^{<t>}$  作为 label 此处已不是单词，而是一个形如  $[1 \ 0 \cdots 0]$  的特征向量。

综上，整个 RNN 所有层的总误差为每层误差之和，即  $L$ ：

$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

分析知：由以下公式可以求得 RNN 模型输出任意某个可能的句子的概率。

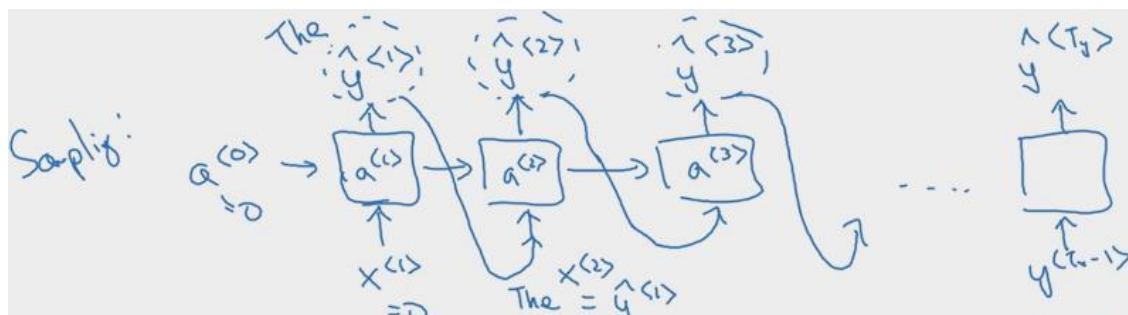
$$\begin{aligned} & P(y^{<1>}, y^{<2>}, y^{<3>}) \\ &= \frac{P(y^{<1>}) P(y^{<2>} | y^{<1>})}{P(y^{<3>} | y^{<1>}, y^{<2>})} \end{aligned}$$

## 1.7 对新序列采样

当训练好一个序列模型之后，如果想看看这个模型究竟学习了什么，一种方法就是进行一次新序列采样。也就是看它会自动生成什么句子。

假设我们已经训练好如下 RNN 模型，采样时最初我们输入  $a^{<0>} = x^{<1>} = 0$ ，我们将得到 softmax 后的  $y^{<1>}$ ，它代表第一个单词为每个 xxx 的概率。其次用 `np.random.choice` 来根据  $y^{<1>}$  向量中的概率分布进行采样。

其次对第二步进行采样：值得注意的是，此处的输入  $x^{<2>}$  不再是训练时的用的 label 即  $y^{<1>}$  了，而是刚刚采样得到的  $y^{<1>}$ 。以此重复，直到最后一步。也就是每次把当前预测值作为下一步的输入，不施加外界矫正，任由其发展，看看最终模型得到的是什么句子。



结束方法：若 EOS 在词汇表中，则只用一直采样，当  $y^{<t>}$  出现 EOS 停止即可。若 EOS 不在词汇表，则可以设置一个时间步长度，当到达这个界限之后停止即可。

当然，有时候很可能某一步的输出是 UNK，显然我们希望的输出中不应该含有 UNK。此时只需拒绝采样 UNK，再在剩下的单词中继续采样即可。

### 字符模型 Character-level language model:

根据实际需要，模型可以是单词模型也可以是字符模型。当为单词模型时，我们输入的序列是 [cats average]；当为字符模型时，输入的序列应该是 [c a t s - a v e r a g e]，此时词汇表中的元素也是一个一个的字母。字符模型的优点是不用考虑 UNK 的情况，因为即使把所有的字符囊括进词汇表中，词汇表也不大。字符模型的缺点是输出将是一个很长的字符串，毕竟一个单词一般占五六个字符。

→ Vocabulary = [a, aaron, ..., zulu, <UNK>] ←

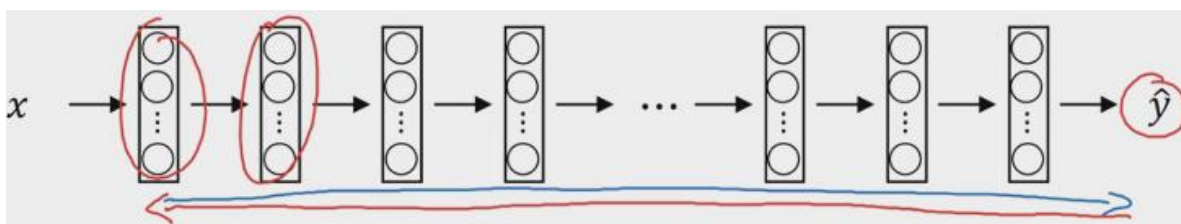
→ Vocabulary = [a, b, c, ..., z, ., ,, ;, 0, ..., 9, A, ..., Z]

目前大部分场合用的仍是单词模型，但是随着计算机性能越来越强，在一些特定场合也会使用词汇模型。

## 1.8 梯度消失

目前我们已经知道 RNN 前向传播、反向传播如何工作，如何实现语言模型、实体命名识别等等。但是基本的 RNN 还有一个非常大的毛病就是梯度消失（Gradients Vanishing）。

如下例子中，两个句子中间都有一个很长的定语从句，而句子末尾究竟是 was 还是 were，取决于句子最前面的单词是 cat 还是 cats。由此可见，前面的单词可能对很后面的单词产生影响。而实验发现，基本的 RNN 不擅长捕获长期依赖效应，它当前的单词只受附近单词的影响。



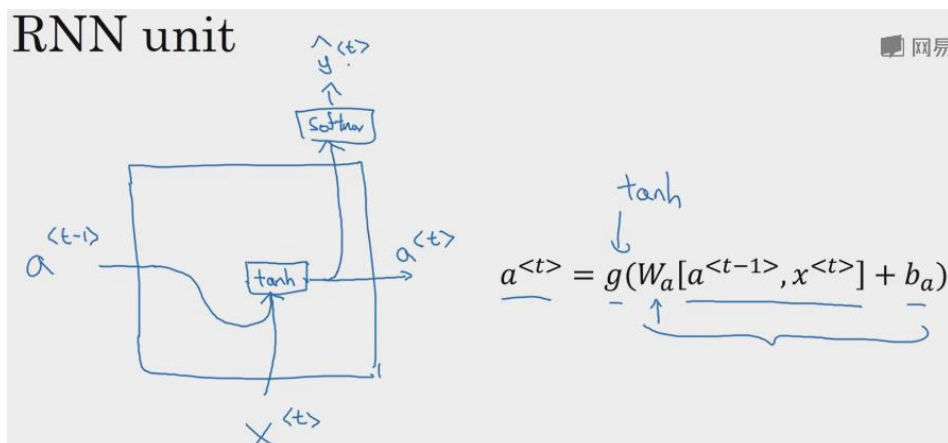
因为对于很深的网络来说，我们很难通过反向传播将梯度从最后传送到前面，梯度会逐渐消失，于是前面参数的改动将变得微乎其微，所以训练时很后面的单词 was/were 将难以对前面的 cat/cats 产生影响，也就是不能捕捉到这种长期（long-term）的依赖。显然，在英语语法中，中间的定语从句可以无限长，于是传统的 RNN 将不适应这一规则。

当然，不仅是梯度消失，梯度爆炸（Gradient exploding）的问题在训练 RNN 时也可能出现。但是由于梯度爆炸较为明显，它能使我们的参数变得特别大，以至于最终网络崩溃，我们可用梯度修建（Gradient clipping）解决。所以训练网络首要问题是解决梯度消失。

## 1.9 门控循环单元 GRU

门控循环单元（Gated Recurrent Unit）改变了 RNN 的隐藏层，使 RNN 能更好地捕捉深层连接，并改善了梯度消失的问题。

普通 RNN 的隐藏层单元：



简化 GRU：

GRU 中将有一个新的变量称为  $c$ ，即记忆细胞（cell）。它提供了记忆能力，如猫是单数还是复数，所以最终决定动词是单数还是复数。

在 GRU 中  $c^{<t>} = a^{<t>}$  二者相等。同时，在每个时间步中，我们将用一个候选值（Candidate）重写记忆细胞，候选值为：

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

GRU 中最重要的思想是：设置一个门  $\Gamma_u$ （Gate，下标  $u$  代表 update），它是一个 0 到 1 间的值，且  $\Gamma$  大部分时候都十分接近 0 或者 1，就像一个开关。因为  $\Gamma_u$  由 sigmoid 函数而来：

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

在例子“The cat, which already ate..., was full”中，单词 cat 处的  $c^{<t>}$  为 0 还是 1 可能取决于它是单数还是复数。假设这里因为 cat 是单数而设置  $c^{<t>} = 1$ 。然后 GRU 将一直记住  $c^{<t>}$  的值，直到单词 was 处， $c^{<t>}$  的值还是 1。通过这种方法就将一个久远的单数信息逐步传递到后面，并告诉单词 was/were 原本主语是单数，于是选择 was。

而门控  $\Gamma$  的作用就是决定什么时候更新  $c$  的值。如最初遇到“The cat”，我们知道遇上了一个新的物体，所以记住它；而当最后到达“was full”处时，我们知道关于 cat 的描述已经结束了，于是或许可以忘记（更新） $c$  的值。所以， $c^{<t>}$  的值如下：



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

分析上式，根据开关  $\Gamma_u$  的 01 值，将决定当前  $c^{<t>}$  采用新计算而来的候选值  $\hat{c}^{<t>}$ ，还是继续延续旧的值  $c^{<t-1>}$ 。

·当时时间步处于从句“which already ate...”中时，门控  $\Gamma_u$  应该约等于 0，这样  $c^{<t>} \approx c^{<t-1>}$ ，才能使久远的 cat 单数主语信息往后保留；

·当时时间步处于“was full”之后，代表当前对 cat 的描述已经结束了，故应该忘记关于 cat 的记忆，所以此时门控  $\Gamma_u$  应该约等于 1，使得  $c^{<t>} \approx \hat{c}^{<t>}$ ，即赋值为当前通过重新计算获取到的信息：

$$\begin{aligned} & \Gamma_u = 1 \quad \Gamma_u = 0 \quad \Gamma_u = 0 \quad \Gamma_u = 0 \\ & \Rightarrow \boxed{c^{<t>} = 1} \dots \dots \dots = 1 \\ & \text{The cat, which already ate ..., was full.} \end{aligned}$$

值得注意的是，尽管实际过程中， $\Gamma_u$  的值并不恰好为 0 或 1，理论上是一个加权平均。但是由于非常接近 0/1，所以上式  $c^{<t>}$  可视为在  $\hat{c}^{<t>}$  和  $c^{<t-1>}$  中二选一。

综上，简化 GRU 的公式和图解如下：

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



思考：

实际过程中，因为  $a^{<t>}$  是一个向量，所以  $c^{<t>}$ ， $\hat{c}^{<t>}$ ， $\Gamma_u$  也是向量，且它们的维度都相同。此时公式 3 的乘法是对应元素相乘即  $\odot$ 。此时  $c^{<t>}$  保存了不同分支的历史信息， $\hat{c}^{<t>}$  保存了刚计算得的各分支的候选值， $\Gamma_u$  是许多开关，决定每个分支的开与闭。

此时门控  $\Gamma_u$  的每个元素决定：在当前 step 选择不同的单词的分支下，对应的记忆  $c^{<t>}$  是否继续保留。即此时  $c^{<t>}$  中每个 bit 保留了不同的记忆，如 bit1 代表 cat 主语是单数还是复数，bit2 决定当前讨论的是是什么食物（因为句子中有 ate 单词）。

且每次更新可以只更新  $c^{<t>}$  中的某几个 bit。例如时间到“was”后， $c^{<t>}$  中代表 cat 主语单/复数的位就可以清除了，而其他位不变。

## 完整版 GRU:

多年来学者们研究了多种不同的单元来设计 GRU，去尝试在网络中产生更深层的影响，并解决梯度消失的问题，于是得到这个目前最常用的 GRU 版本：变化是**新增**一个门  $\Gamma_r$ （其中  $r$  代表相关性），它代表用  $c^{<t-1>}$  计算出  $c^{<t>}$  有多大的相关性。完整公式如下：

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

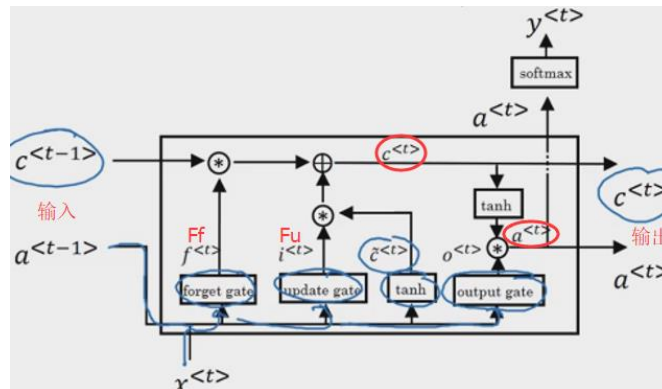
## 1.10 长短期记忆 Long Short Term Memory

相比 GRU，LSTM 甚至是一个更强大、更通用的版本。在 LSTM 中，不再有  $c^{<t>} = a^{<t>}$  成立，故原 GRU 公式中的  $c^{<t-1>}$  此时应替换成  $a^{<t-1>}$ 。其次，LSTM 中有三个专门的门控，分别是更新门  $\Gamma_u$ 、遗忘门  $\Gamma_f$  和输出门  $\Gamma_o$ 。

### GRU 和 LSTM 的公式对比：

$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$	$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$
$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$	$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$
$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$	$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$
$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$	$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$
$a^{<t>} = c^{<t>}$	$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$
GRU	LSTM

### LSTM 的原理图：



容易观察，不管是 GRU 还是 LSTM 的结构，都可以使  $c^{<t>}$  很方便地流通，这也是它们有长效记忆的原因。

实际使用中，LSTM 还有许多变体，比如在求输出门  $\Gamma_o$  时，我们不仅希望与  $a^{<t-1>}$  和  $x^{<t>}$  有关，还希望与  $c^{<t-1>}$  也有关。即  $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}, c^{<t-1>}] + b_o)$ 。这就是 Peephole connection。

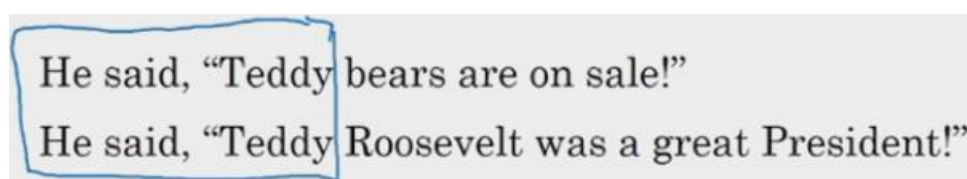
### GRU 和 LSTM 的对比：

GRU 在结构上更简单，门也更少，运算更快，所以更容易创建一个庞大的网络。但 LSTM 更强大更灵活，门也更多。如果非要二选一，大部分人还是会选择 LSTM。

## 1.11 双向神经网络 Bidirectional RNN

BRNN 模型可以让你在序列的某处不仅获取之前的历史信息，还可以获取到未来的信息。

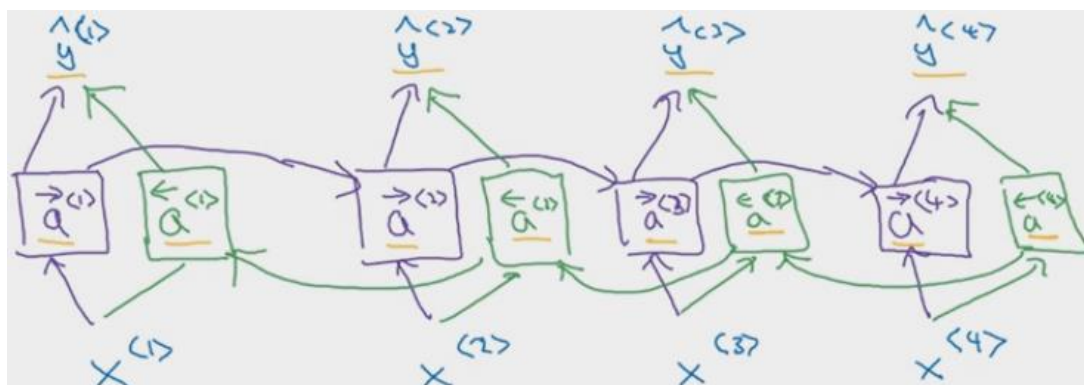
eg. 如图，在命名实体识别例子中，“Teddy”可以是人名（句子 2）也可以是非人名（句子 1）。当仅读取前三个单词“He said, “Teddy””时，要判断 Teddy 是信息不足的，我们还需要从后面的单词获取信息才能做出判断。



### BRNN 的工作原理：

BRNN 中，每层都有两个神经元，它们的方向不同。每个神经元都同普通 RNN 的单元相同：接收当前输入  $x^{<t>}$  以及上一次的激活值  $a^{<t-1>}$ 。

对于一个输入序列  $x^{<1>}, x^{<2>}, x^{<3>}, x^{<4>}$ ，需要进行两次前向传播：首先按照  $x^{<1>}, x^{<2>}, x^{<3>}, x^{<4>}$  的顺序从左到右传播一遍，再按照  $x^{<4>}, x^{<3>}, x^{<2>}, x^{<1>}$  的顺序从右往左传输一遍。当两次前向传播都完成之后，就可以得出预测结果了。



例如，当识别到“Teddy”时，网络在该处的输出将预测该单词是人名的概率。观察公式可知： $a^{<1>}$ 包含了之前的历史信息“He said”，而  $a^{<1>}$  由于反向计算包含了未来的信息“bears are on sale!”。所以，BRNN 是综合了历史和未来，对当前做出预测！

Tip: 图中的神经元既可以是标准 RNN 单元，也可以是 GRU 单元，也可以是 LSTM 单元。当我们有大量的 NLP 文本时，通常会选择 LSTM+BRNN 的高效组合。

### BRNN 的缺点：

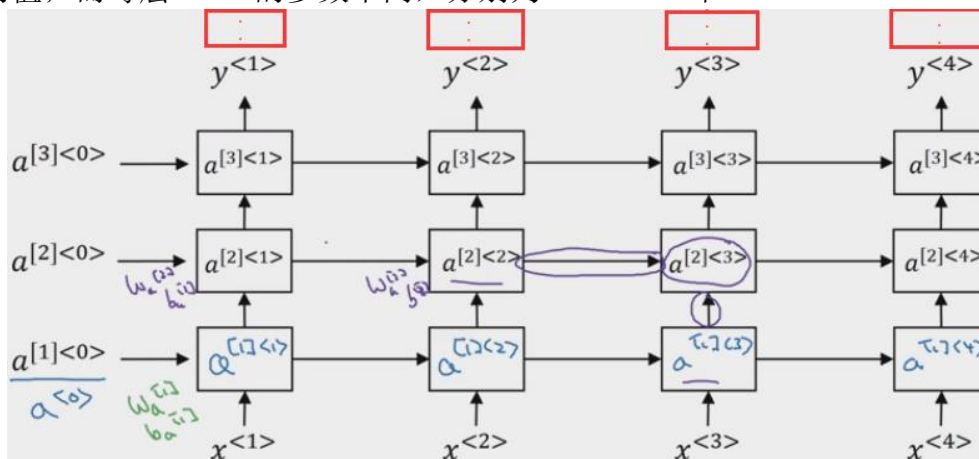
BRNN 要求你必须最开始就知道完整的输入序列，才能保证两次正向传播顺利完成，才能保证你在任意位置做出预测。

eg. 当我们要实现语音识别时，这就要求系统必须先获取到这人说的所有话，才能做出翻译，所以它不能满足实时翻译的需要。需要对其改进才能正常使用。

## 1.12 深层循环神经网络 Deep RNNs

前文介绍的各种 RNN 变体已经能满足正常需要，但当需要学习非常复杂的函数时，我们需要把各种 RNN 堆叠在一起，构建更深的模型。

设第  $L$  层 RNN 的第  $t$  个时间步处神经元的激活项为  $a^{[L]<t>}$ 。由于参数共享，同一层 RNN 的参数是相同的，记为  $W^{[L]}$  和  $b^{[L]}$ ，它将用于计算第  $L$  层每个神经元的值；而每层 RNN 的参数不同，分别为  $W^{[1]} \dots W^{[L]}$  和  $b^{[1]} \dots b^{[L]}$ 。



如图，通常 Deep RNNs 得到的输出  $y^{<1>}$ ,  $y^{<2>}$ ,  $y^{<3>}$ ,  $y^{<4>}$  不会直接输出，而是作为其他复杂网络（如神经网络）的输入，最后再得到结果。

此处，若计算  $a^{[2]<3>}$  的值，需要上一层 RNN 的  $a^{[1]<3>}$  和当前层 RNN 的  $a^{[2]<2>}$  同时作为输入计算而得：

$$a^{[2]<3>} = g(W_a^{[2]} [a^{[1]<3>}, a^{[2]<2>}] + b_a^{[2]})$$