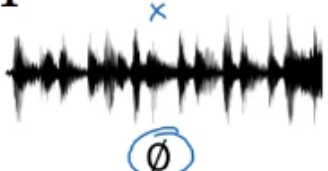
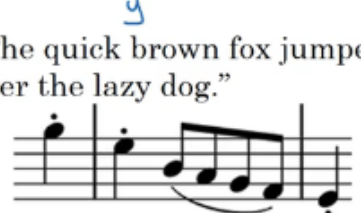

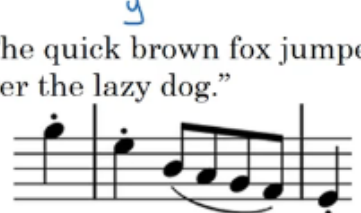




1. 序列模型

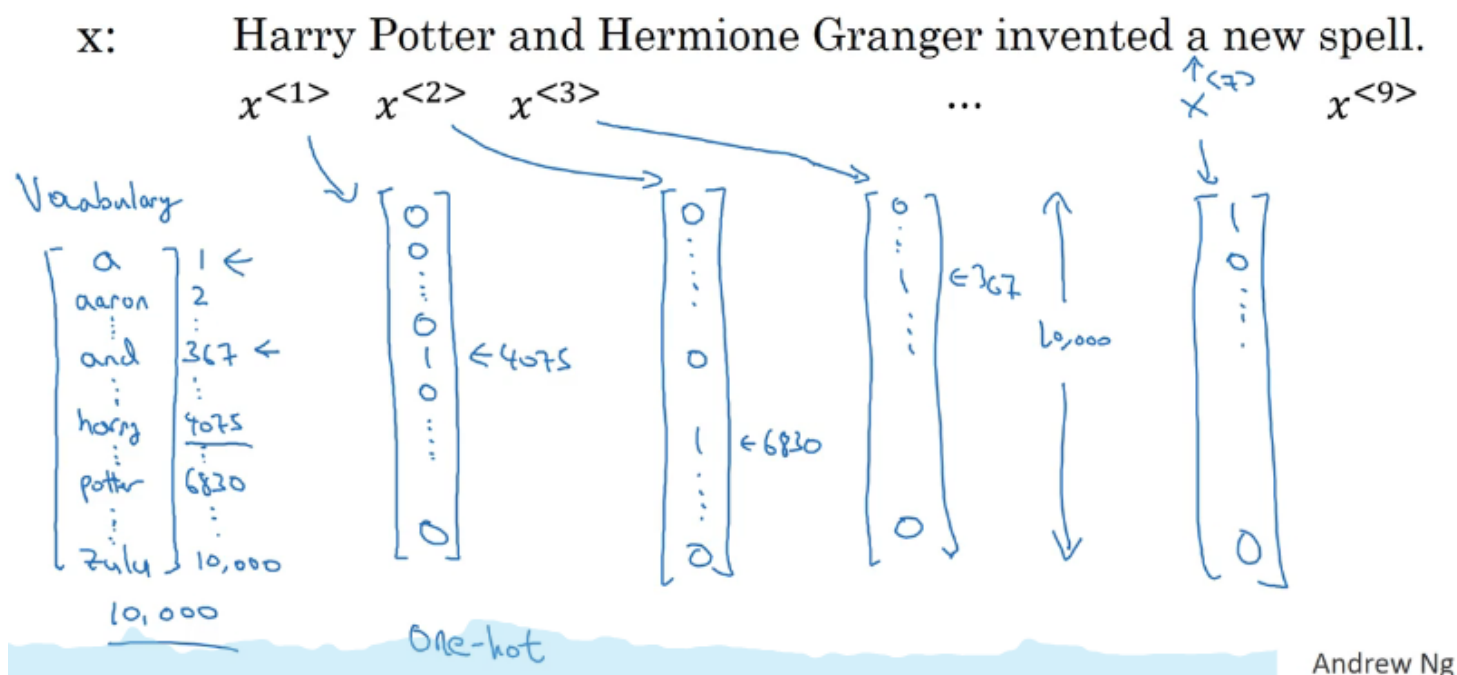
- 语音识别：输入的语音信号转化为文本信息，语音信号和文本信息均是序列数据。
- 音乐生成：生成音乐乐谱。输入可以是空集或者数字，输出的乐谱属于序列数据。
- 情感分类：输入的评论句子转化为对应的等级/评分。输入的评论属于序列模型。
- DNA序列分析：找到输入的DNA序列所匹配的蛋白质。
- 机器翻译：不同语言之间的转换。输入输出均为序列数据。
- 视频动作识别：输入的视频帧序列转化为相应的动作。
- 命名实体识别：从输入的句子中识别实体的名字。

Examples of sequence data

Speech recognition		→	
Music generation		→	
Sentiment classification	"There is nothing to like in this movie."	→	
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
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

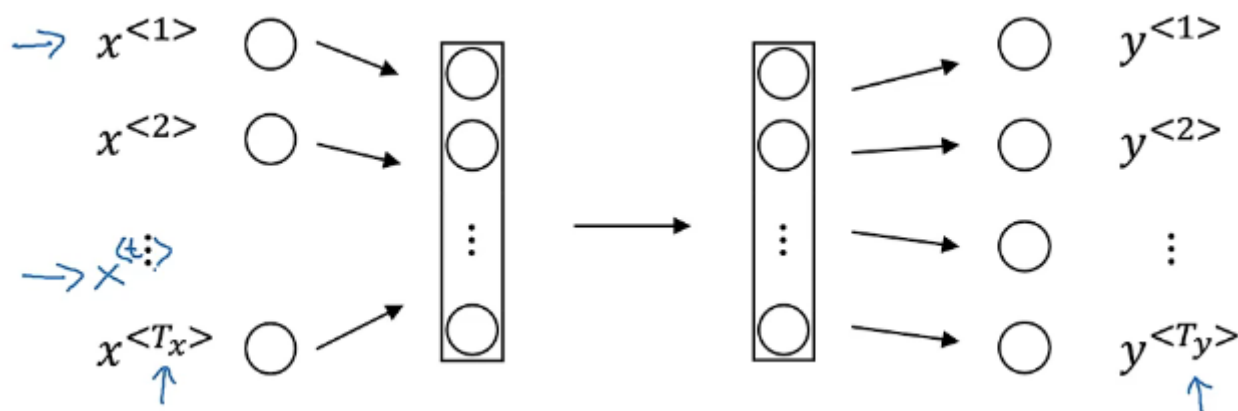
- Input x : $x^{<t>}$ 表示输入 x 中的第 t 个符号, T_x 表示输入 x 的长度, $x^{(i)<t>}$ 表示第 i 个输入样本的第 t 个符号。
- Output y : $y^{<t>}$ 表示输出 y 中的第 t 个符号, T_y 表示输出 y 的长度。
- 利用字典来表示每一个输入的符号, 比如one-hot向量。



Andrew Ng

3. 循环神经网络

3.1 朴素神经网络



对于学习 X 和 Y 的映射，一种很直接的方法就是使用传统的神经网络。可以将序列 X 进行字典编码后，输入到网络中得到对应的输出 Y 。

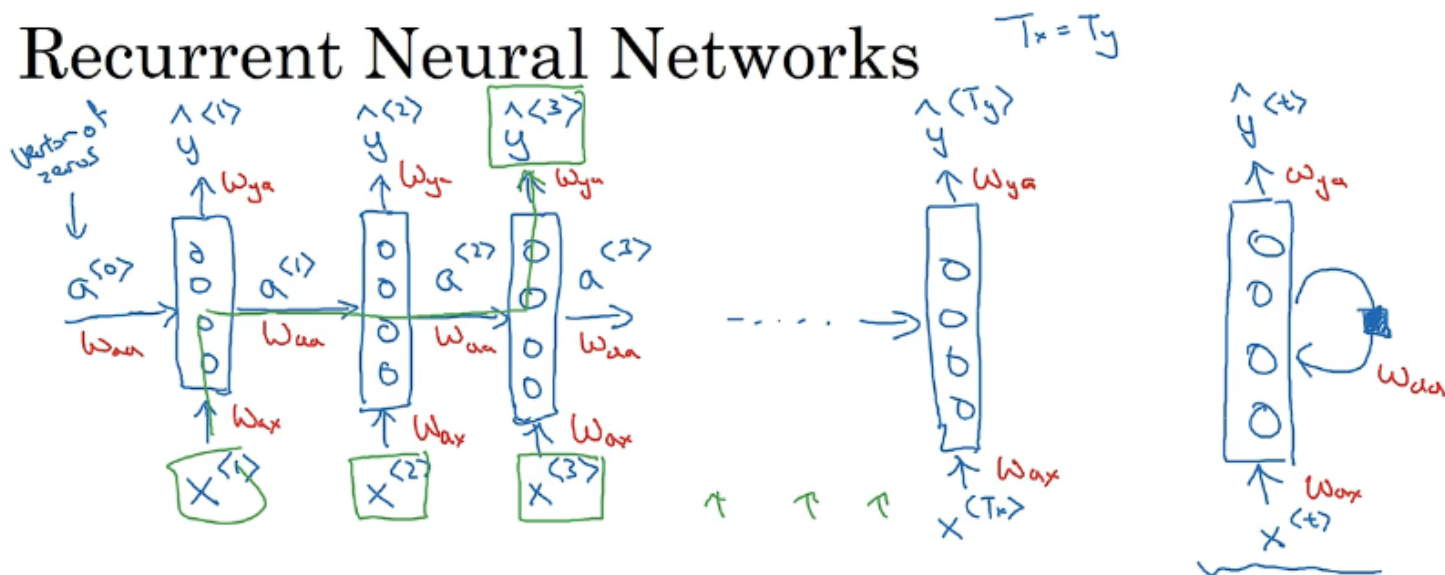
但是这种方法存在两个问题：

- 输入和输出数据在不同的例子中可以有不同的长度。
- 这种朴素神经网络不能共享从文本不同位置学习到的特征。

3.2 循环神经网络的结构

RNN会传递一个激活值到下一个时间步中，用于下一个时间步的计算。

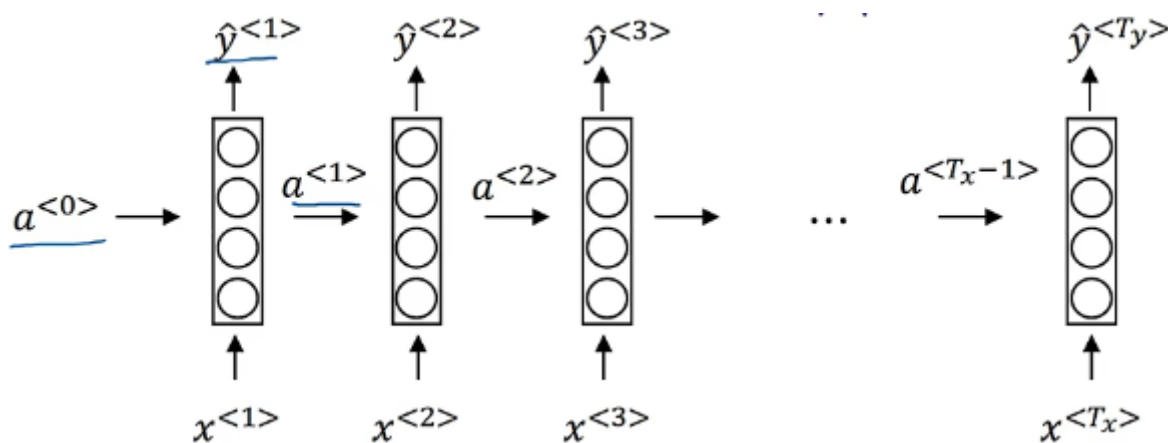
Recurrent Neural Networks



- W_{ax} 是输入 $x^{<t>}$ 到隐藏层的连接，每个时间步使用相同的 W_{ax} 。
- W_{aa} 是激活值 $a^{<t>}$ 到隐藏层的连接。
- W_{ya} 是隐藏层到输出 $y^{<t>}$ 的连接。

RNN的缺点是每个输出预测 $y^{<t>}$ 仅使用了前面的输入信息，使用双向RNN可以解决这一问题。

3.3 循环神经网络的前向传播



- 输出激活向量: $a^{<0>} = \vec{0}$ 。
- $a^{<1>} = g(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$ ，通常使用tanh为激活函数。
- $\hat{y}^{<1>} = g(W_{ya}a^{<1>} + b_y)$ ，激活函数的使用取决于具体的问题。
 - 二分类问题: sigmoid
 - 多分类问题: softmax
- 在某一时刻 t :
 - $a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$
 - $\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$
- 可以对上式进行简化:

- $a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$ 。其中 $W_a = [W_{aa}|W_{ax}]$, $[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$ 。如果 $a^{<t-1>}$ 是100维, $x^{<t>}$ 是10000维, 那么 W_{aa} 的维度是 100×100 , W_{ax} 的维度是 100×10000 , W_a 的维度是 100×10100 。同理, $[a^{<t-1>}, x^{<t>}]$ 是一个10100维的矩阵。
- $\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$

Simplified RNN notation

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

\uparrow (100,100) 100 \uparrow (100,10,000) 10,000

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

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

\uparrow (100) $\begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix}$ \uparrow (100,10100) $= W_a$

$\left[\begin{array}{c} \xleftarrow{100} \quad \xleftarrow{10000} \end{array} \right]$

$$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$$

\uparrow 100 \uparrow 10000 \uparrow 10100

$$\begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix} \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$$

4. 反向传播

定义一个loss函数:

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

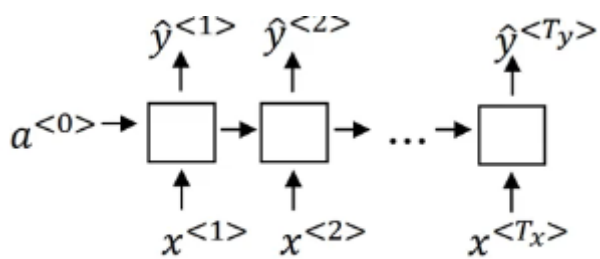
$$L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

将每个时间步的损失相加就是总体的损失。再利用导数和梯度下降对参数进行更新。

5. 不同类型的RNN

不同的应用 T_x 和 T_y 并不一定相同。

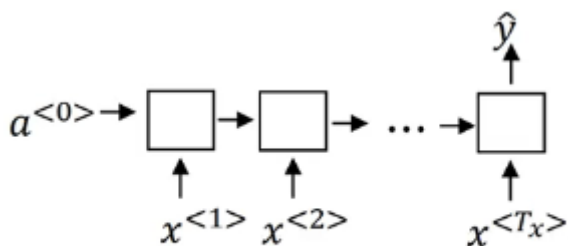
5.1 many-to-many ($T_x = T_y$)



Many to many

5.2 many-to-one

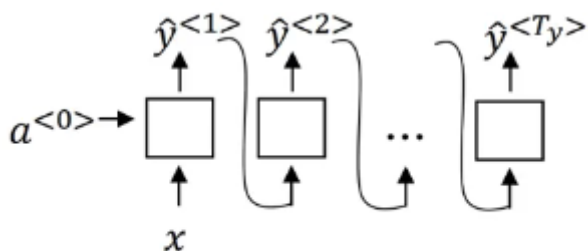
比如给电影打分的系统。



Many to one

5.3 one-to-many

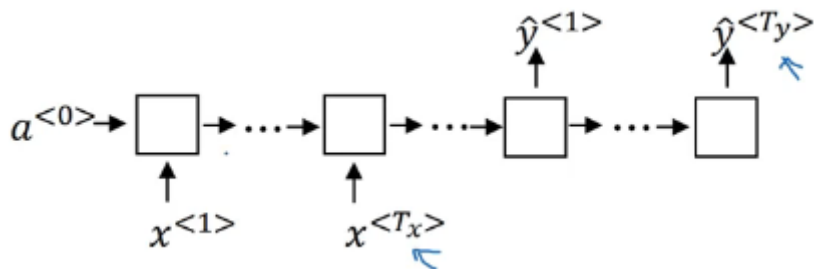
比如音乐生成系统，输入音乐的类型，生成一段音乐序列。



One to many

5.4 many-to-many ($T_x \neq T_y$)

比如机器翻译，跨语种进行文本翻译。



6. 语言模型和序列生成

6.1 什么是语言模型

两句话有相似的发音，但想表达的意义不同，如何让我们构建的语音系统能够输出正确的句子。我们可以评估各个句子中单词出现的可能性，进而给出整个句子出现的可能性。

Speech recognition

The apple and pair salad.

→ The apple and pear salad.

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

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

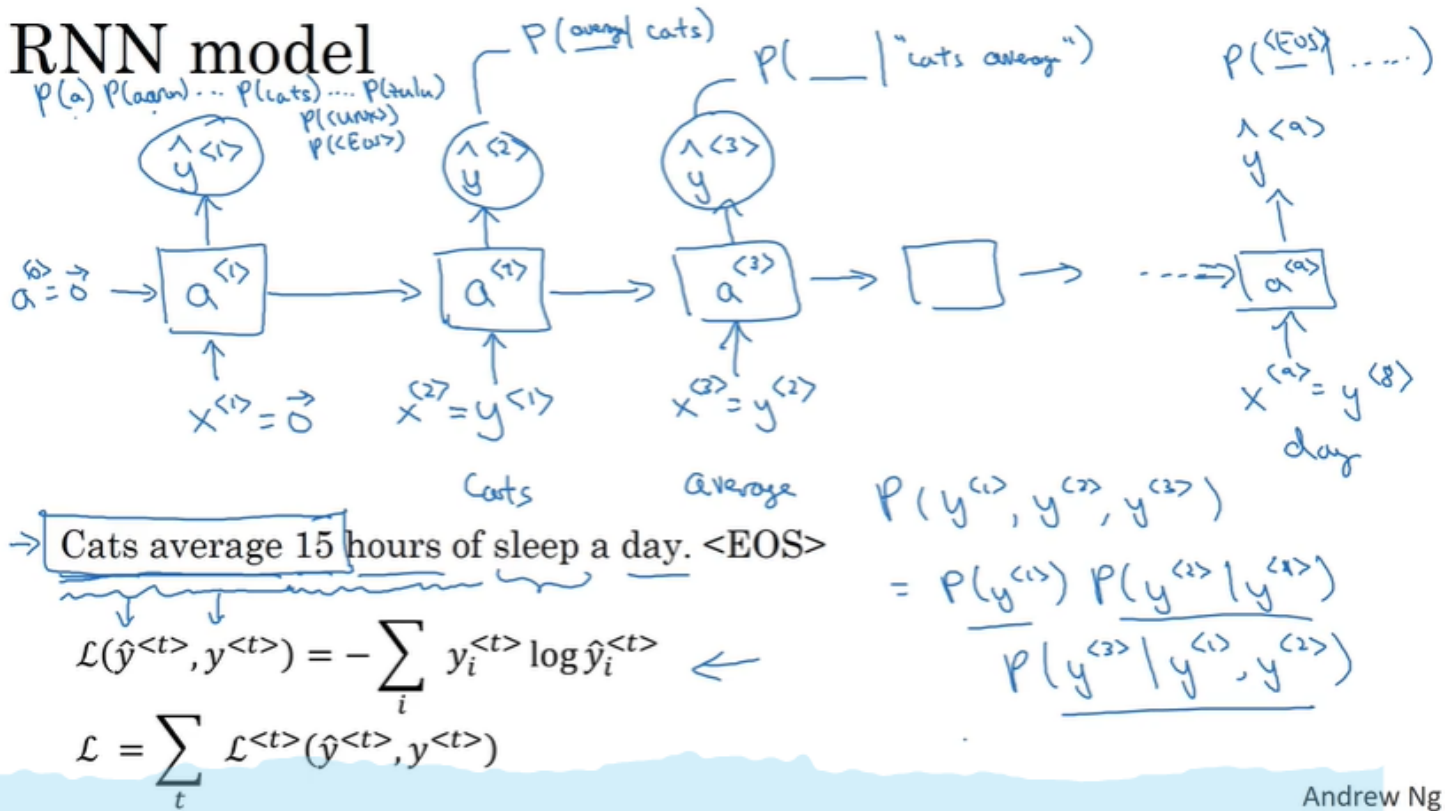
$$P(\text{Sentence}) = ?$$

$$P(y^{(1)}, y^{(2)}, \dots, y^{(T)})$$

6.2 使用RNN构建语言模型

- Training set: 一个很大的语言文本语料库。
- Tokenize: 利用字典标记化句子。
 - 在句子结尾加上 $\langle EOS \rangle$ 。
 - 未出现在字典库中的词使用"UNK"表示。
- 建立RNN:
 - 使用零向量对输出进行预测，预测第一个单词是某个单词的可能性。
 - 通过前面的输入，预测后一个单词出现的概率。
 - 使用softmax计算损失，进行参数更新。

RNN model



Andrew Ng

7. 新序列采样

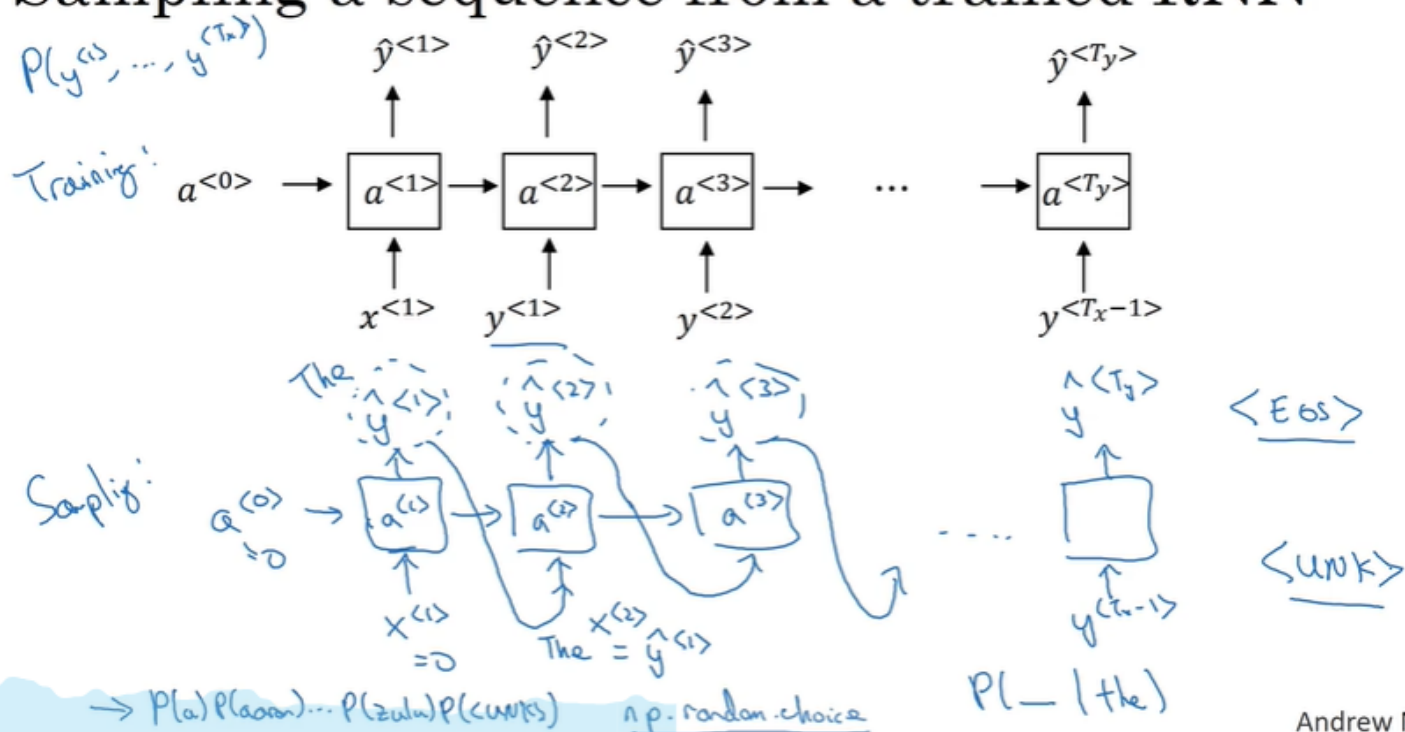
完成语言模型的训练后，如果想要了解该模型学到了什么，一种非正式的方法就是进行一次新序列采样 (sample novel sequences)。一个序列模型，它模拟了任意特定单词序列的概率

$P(y^{(1)}, y^{(2)}, \dots, y^{(T_y)})$ ，我们对这个概率分布进行采样，生成一个新的单词序列。

对于一个已经训练好的RNN模型：

- 第一个时间步，输入 $x^{(1)} = 1$ 和 $a^{(0)} = 0$ ，得到经过softmax层的概率输出。根据softmax的概率分布，进行随机采样，获得第一个单词 $\hat{y}^{(1)}$ 。
- 在下一个时间步，我们以上一个采样得到的 $\hat{y}^{(1)}$ 作为该时间步的输入，进而预测输出 $\hat{y}^{(t)}$ 。以此类推。
- 如果字典中存在结束标志，那么输出该符号时便表示结束；如果没有这种标志，则可以设置结束的时间步。

Sampling a sequence from a trained RNN

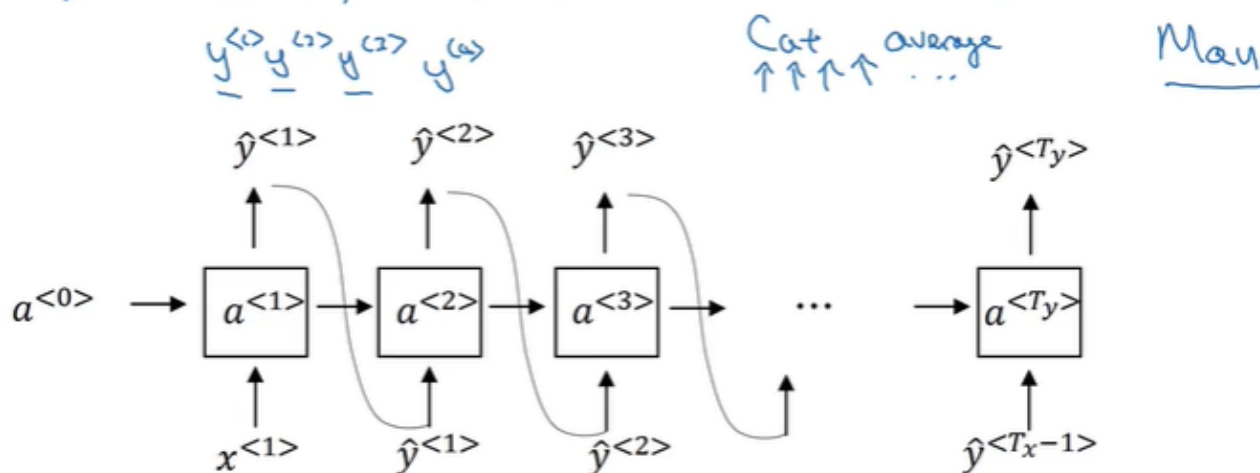


上面的模型是基于词汇的语言模型，根据具体的应用，我们还能构建基于字符的RNN模型。这种模型的缺点就是会得到太长的输出序列，影响我们捕捉句子前后词语的依赖关系。而且基于字符的语言模型的训练代价较高，因此目前使用最广泛的是基于词汇的语言模型。但随着计算机运行能力的增强，在一些情况下也会开始使用基于字符的语言模型。

Character-level language model

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

Handwritten: Vocabulary = [a, b, c, ..., z, \, ., , , ;, 0, ..., 9, A, ..., Z]



8. RNN的梯度消失

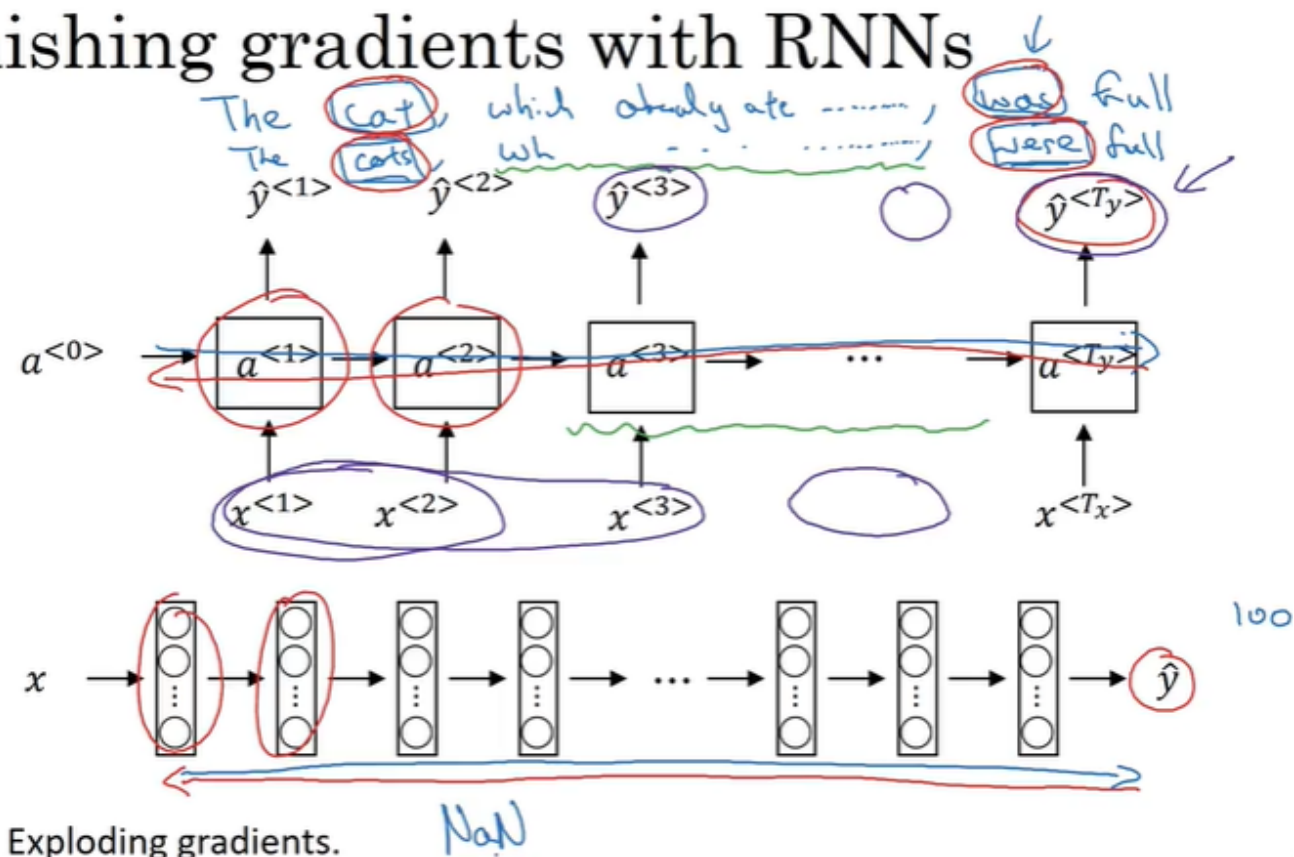
下面的两个例句：

- The cat, which already ate, was full.
- The cats, which already ate, were full.

cat对应was，cats对应were，中间存在大量的其他单词。而cat-was和cat-were又是直接相关的，这种关系被称为长期依赖 (long-term dependencies)。但是目前的RNN模型不擅长捕捉这种依赖关系。

在很深的神经网络中，输出y得到的梯度很难通过反向传播对前几层的权重产生影响。在RNN中，这表现为后面的单词很难记住前面的单词，前面的单词很难对后面的输出产生影响。

Vanishing gradients with RNNs



梯度爆炸的问题在RNN中也是非常常见的。因为指数级的梯度会使网络参数变得很大，得到很多NaN或者非数字输出，因此梯度爆炸是很容易发现的。我们可以使用梯度的方法，如果梯度向量大于某个阈值，就对其进行缩放，保证它不会太大。而梯度消失这个问题比较难解决。

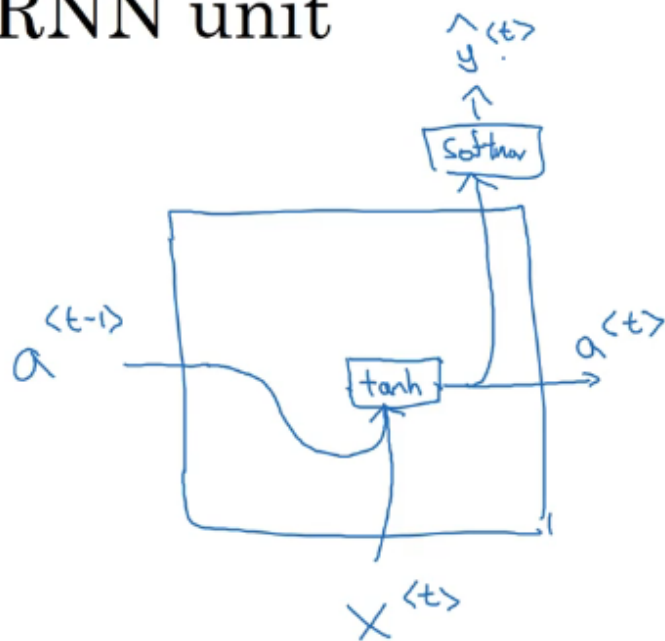
9. GRU单元

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

9.1 RNN单元

对于一个RNN单元，计算 $a^{<t>}$ 的公式如下所示：

RNN unit



$$a^{<t>} = g(\underbrace{W_a[a^{<t-1>}, x^{<t>}]}_{\uparrow} + b_a)$$

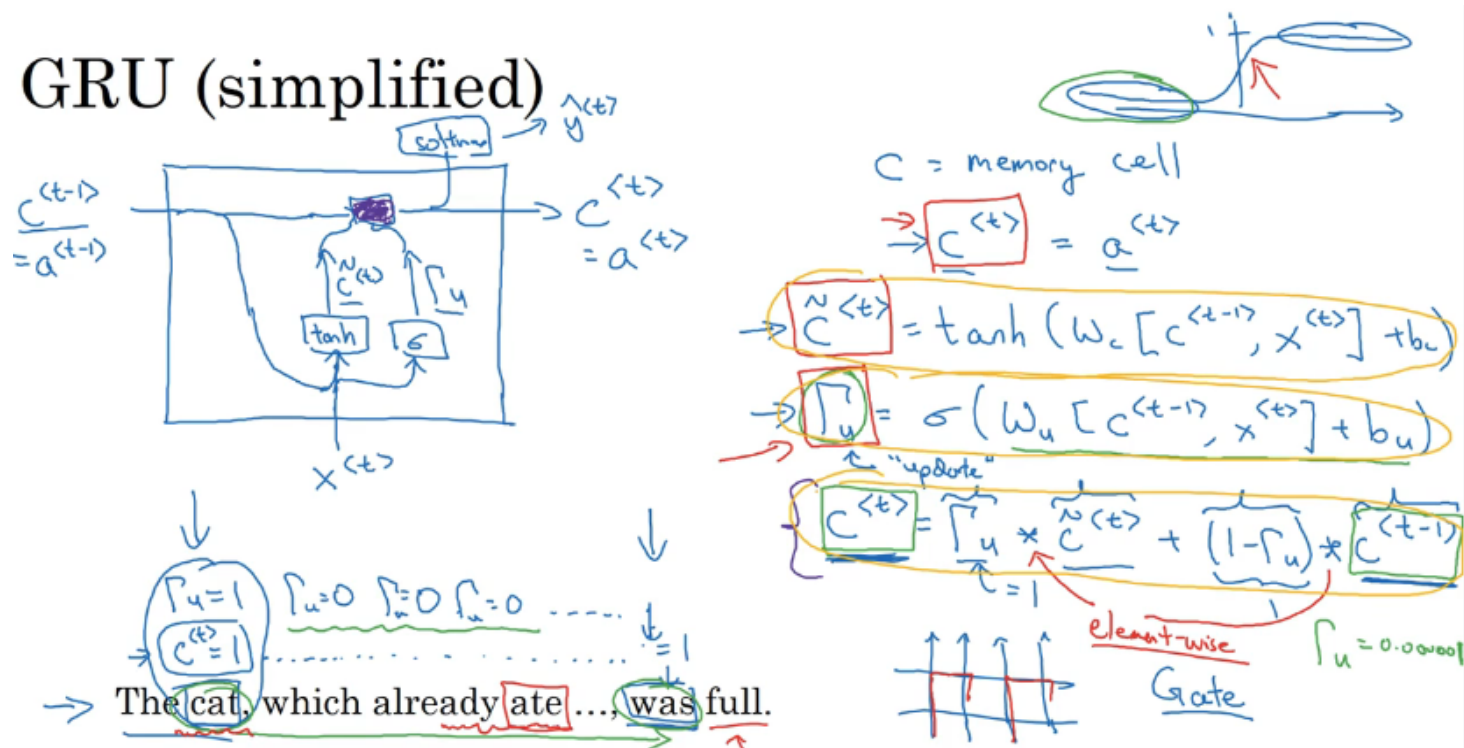
$\downarrow \tanh$

9.2 简化的GRU单元

设定一个新的变量，称作 c ，作为记忆细胞。

- $c^{<t>} = a^{<t>}$ ，记忆细胞输出的是在 t 时间步上的激活值 a 。
- $\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$ ，使用一个新值 $\tilde{c}^{<t>}$ 以代替原本的记忆细胞 $c^{<t>}$ 。
- $\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$ ，用以决定是否对当前的记忆细胞进行更新。
- $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$ ，记忆细胞的更新规则，能够很好的解决梯度消失的问题。
- $c^{<t>} | \tilde{c}^{<t>} | \Gamma_u$ 具有相同的维度。

GRU (simplified)



9.3 完整的GRU单元

增加一个门 Γ_r ，表示前后两个记忆细胞的相关性，公式如下：

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

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

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

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

10. LSTM

长短期记忆 (Long short-term memory, LSTM) 对捕捉序列中更深层次的联系要比GRU更加有效。LSTM 使用了单独的更新门 Γ_u 和遗忘门 Γ_f 以及一个输出门 Γ_o ，公式如下：

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

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

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

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

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

LSTM的结构如下所示：

LSTM in pictures

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

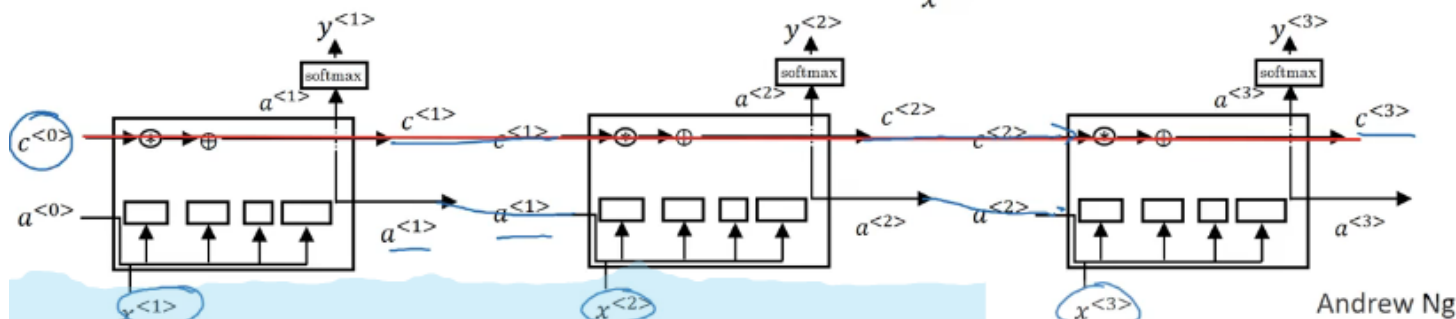
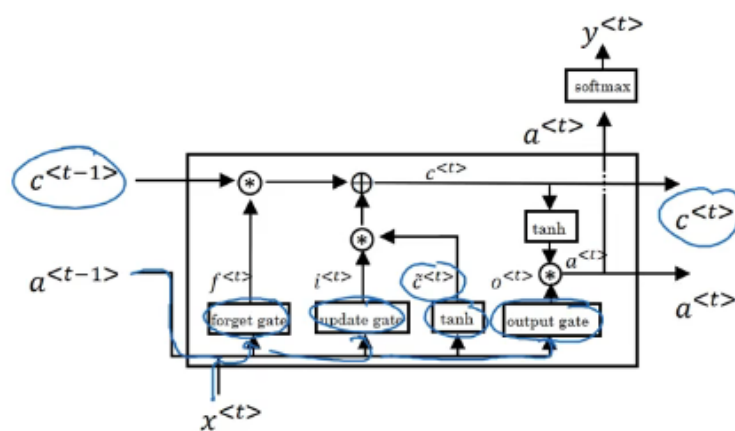
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

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

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$



11. 双向RNN

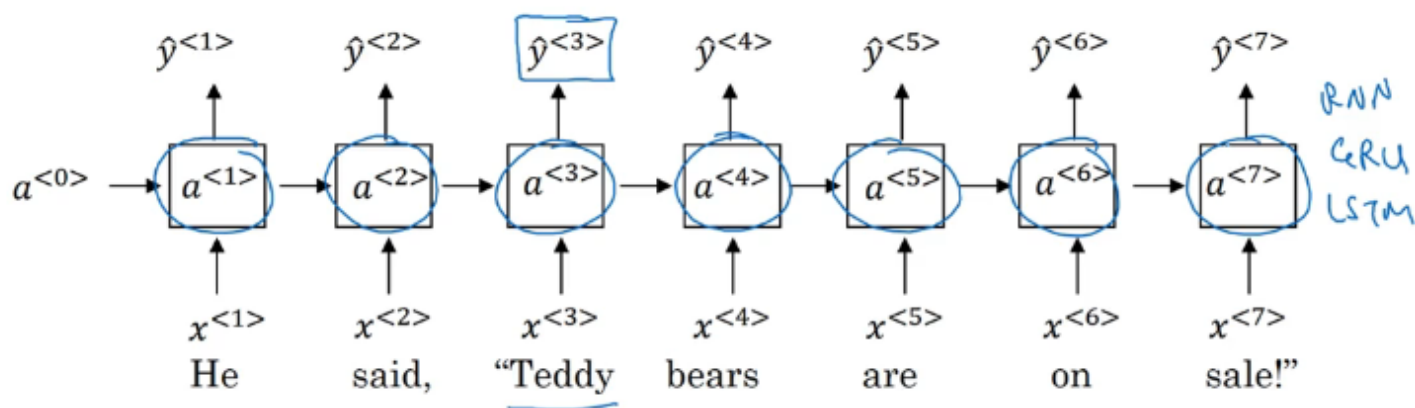
双向RNN (Bidirectional RNNs)能够是序列的某处，不仅可以获得之前的信息，也可以获取未来的信息。

下图的单向RNN中，使用RNN或者GRU或者LSTM均很难判断"Teddy"是否为人名。仅仅使用前两个单词是不够的，需要使用后面的信息来判断。

Getting information from the future

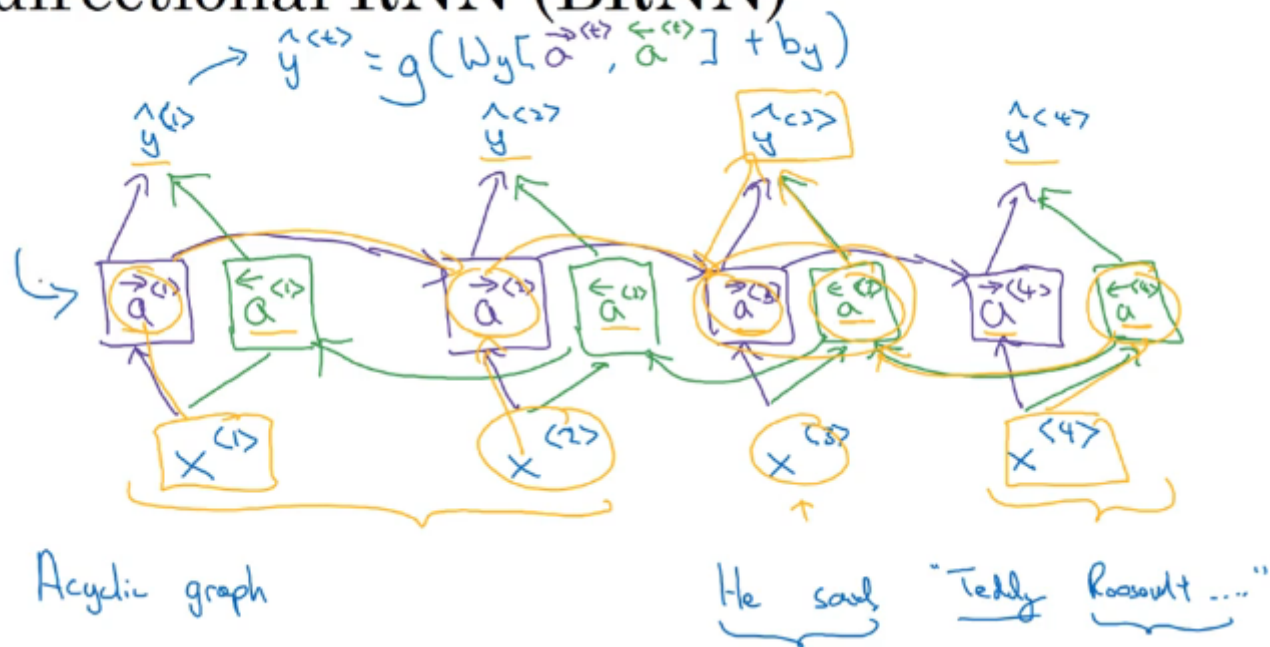
He said, "Teddy bears are on sale!"

He said, "Teddy Roosevelt was a great President!"



BRNN不仅有从左向右的连接层，还存在从右向左的反向连接层。

Bidirectional RNN (BRNN)

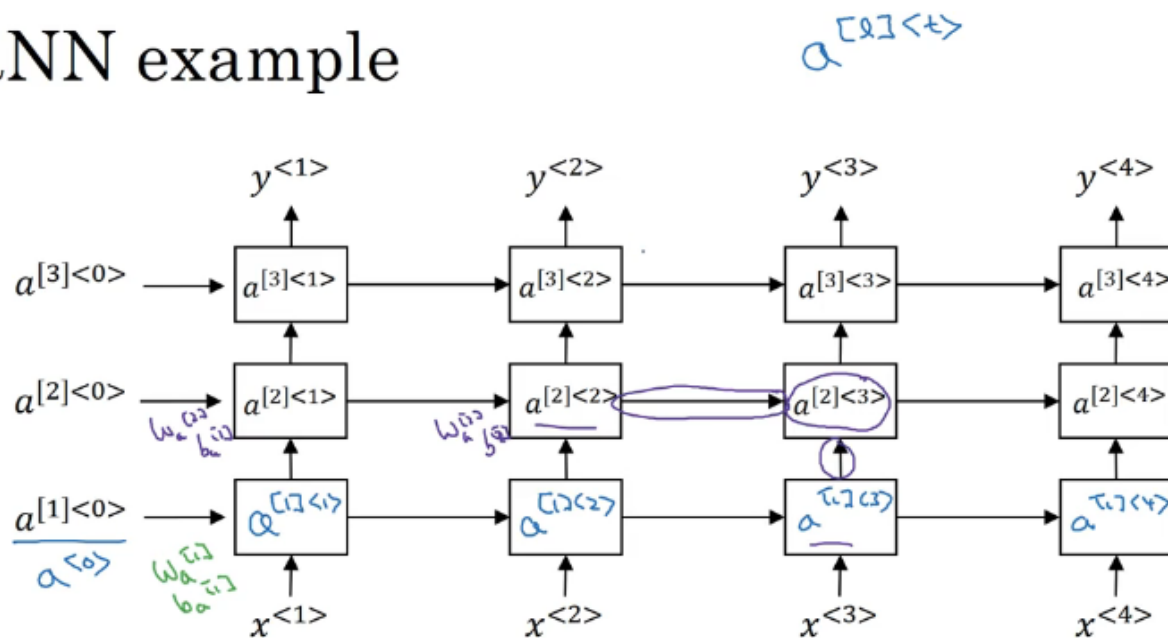
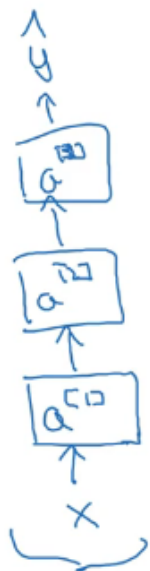


预测输出的值 $\hat{y}^{<t>} = g(W_y[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$ 。在NLP问题中，最常使用的就是BRNN的LSTM。

12. 深层RNN

传统的神经网络可能有上百层，但是对于RNN来说，三层的网络深度就已经很多了。因为RNN存在时间维度，RNN的网络会变得相当大。

Deep RNN example



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