

序列模型 (Sequence Model)

作者 : Leyuan Yu, Ji Yang

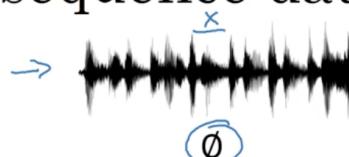
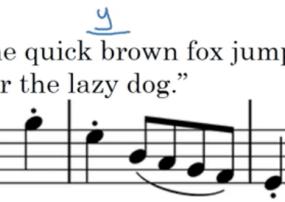
为什么要学习序列模型呢？序列模型，普遍称为RNN(递归神经网络 - Recurrent Neural Network)，做为深度学习中非常重要的一环，有着比普通神经网络更广的宽度与更多的可能性，其应用领域包括但不限于“语音识别”，“自然语言处理”，“DNA序列分析”，“机器翻译”，“视频动作分析”，等等等等... 有这样一种说法，也许并不严谨，但是有助于我们理解RNN，大意是这样的：

普通神经网络处理的是一维的数据，CNN处理的是二维的数据，RNN处理的是三维的数据。

最直观的理解是在CNN对图片的分析基础上，RNN可以对视频进行分析，这里也就引入了第三维“时间”的概念。

另外这篇笔记意在做一个概述和科普，如果想看更多的细节和推导过程请参考我们的[英文版笔记](#)

Examples of sequence data

Speech recognition	→		→	"The quick brown fox jumped over the lazy dog."
Music generation	→		→	
Sentiment classification	→	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	→	AGCCCTGTGAGGAAC TAG	→	AGCCCCTGTGAGGAAC TAG
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. Andrew No

第一周

准备工作

这一小节通过一个小例子为我们打开序列模型的大门，例子如下：

给出这样一个句子 "Harry Potter and Herminone Granger invented a new spell."(哈利波特与赫敏格兰杰发明了一个新的咒语。)，我们的任务是在这个句子中准确的定位到人名 Harry Potter 和 Herminone Granger. 用深度学习的语言来描述如下图 - 每一个单词对应一个输出0或者1，1代表着是人名，0代表不是。

Motivating example

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

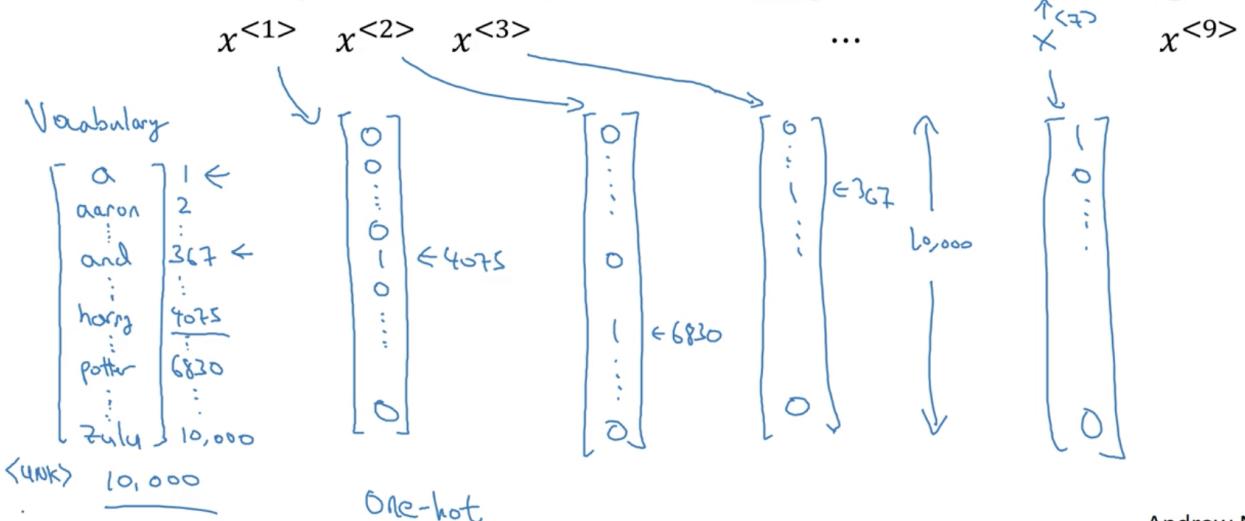
y: | | 0 | | 0 0 0 0

接下来我们要解决的一个问题是如何才能代表一个单词，比如我们例子中的“Harry”，如果每个单词都直接使用的话一是英文总体单词量太大，二是对机器来说输入的每个单词长度应该是一致的。所以这里我们介绍一种新的编码方式，就是用另一种方式来代表每一个单词 - **独热编码 (One-Hot Encoding)**。具体流程是这样，假设我们有10000个常用词，为其构建一个 10000×1 的矩阵(column matrix)，假如第一个词是苹果(apple), 那么对应的第一个位置为1，其他都为0，所以称之为独热。这样每个单词都有对应的矩阵进行表示，如果这个词没有出现在我们的字典中，那么我们可以给一个特殊的符号代替，常用的是 `<UNK>` (unknown)

Representing words

$$x^{<\leftrightarrow>} \quad (x, y) \\ x \rightarrow y$$

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



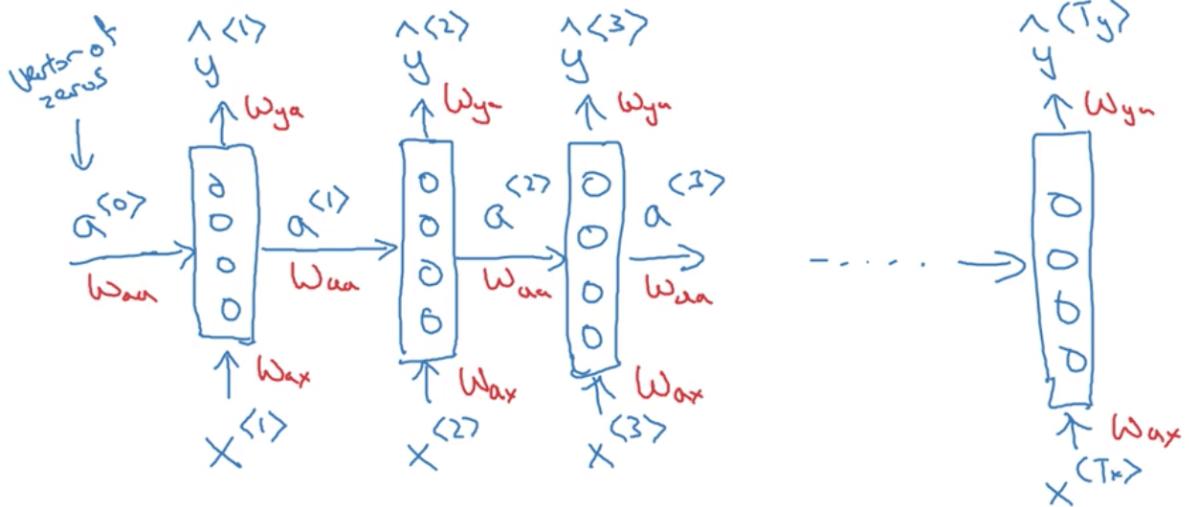
Andrew Ng

序列模型概览

RNN模型可以用下图表示

Recurrent Neural Networks

$$T_x = T_y$$



T_x, T_y 是时间单位，这里统称为“时刻”，在这例子中对应不同时刻是输入的第几个单词， x 是“输入值”，例子中是当前时刻的单词（以独热编码的形式）， y 是“输出值”0或者1， a 称为激活值用于将前一个单元的输出结果传递到下一个单元， W_{ax}, W_{ay}, W_{aa} 是不同的“权重矩阵”也就是我们神经网络update的值。每一个单元有两个输入， $a^{<Tx-1>}$ 和 x ，有两个输出 $a^{<Tx>}$ 和 y 。图中没有出现的g是“激活函数”。

符号	名字
x	输入值
a	激活值
T_x, T_y	x, y 时刻
W_{ax}, W_{ay}, W_{aa}	权重矩阵

回到上述的例子，我们为什么不用普通的神经网络（neural network）解决呢？因为普通神经网络主要存在两个问题：

1. 不同的语句中，输入输出的结果长度不一致
2. 单词和单词之间的特性无法被捕捉到

此外这节课还介绍了rnn正向传播的基本公式，在正向传播的过程中可以看到 a 的值随着时间的推移被传播了出去，也就一定程度上保存了单词之间的特性：

$$a^{<0>} = \vec{0}$$

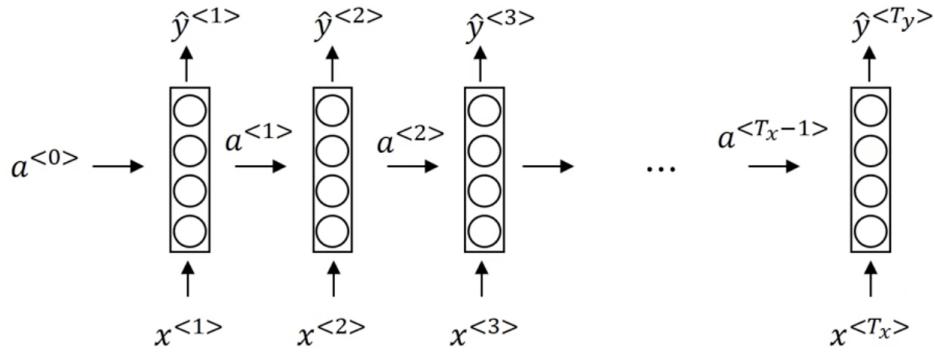
$$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$$

$$\hat{y}^{<1>} = g_2(W_{ya}a^{<1>} + b_y)$$

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

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

Forward Propagation



反向传播(Backpropagation)

RNN的反向传播通常都由类似Tensorflow, Torch之类的库或者框架帮你完成，不过感官上和普通神经网络类似，算梯度值然后更新权重矩阵。更多信息请参考原版笔记。

RNN的不同应用领域

序列模型对输入与输出的长度没有要求，在常见的例子中，机器翻译就是多个输入与多个输出，简称“多对多”，语音识别可视为“单对多”，它的反例是音乐生成-“单对多”。课程中介绍了多种可能的RNN模式，我们用下面一张图概括：

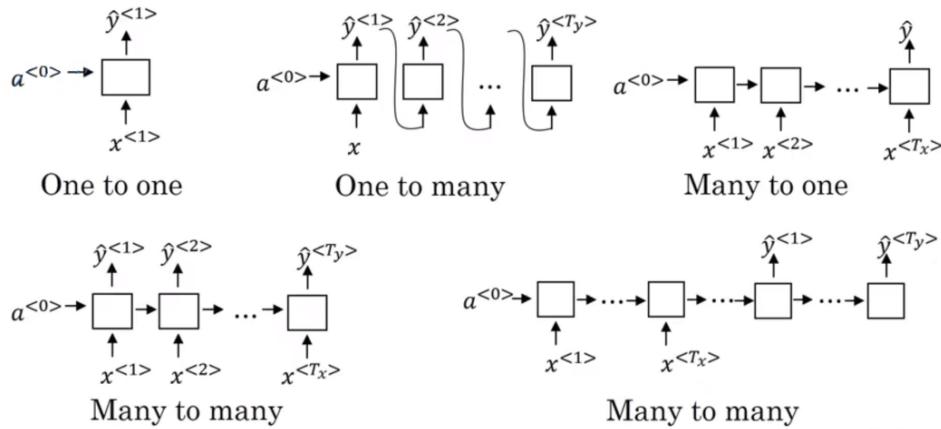


Figure 2: RNN architectures, adopted from Andrew Ng's Deep Learning course

不同的结构给了我们更多的可能性。

语言模型 & 序列生成

- 什么是语言模型 (Language Modelling)

在语音识别中，这两句话听起来几乎一模一样。但如果是人为判断这一句话，瞬间就可以判定应该翻译成第二句。

Speech recognition

The apple and pair salad.

→ The apple and pear salad.

那么机器该如何识别呢？可以通过构建语言模型我们可以预测每句话出现的概率是多少来进行甄别

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

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

- 如何用RNN训练一个语言模型

首先我们需要一个大的文本库，然后将每个单词字符化（ tokenize ）依然可以用独热编码代替其中的常用词，用 `<EOS>` (end of sentence) 代替句号等符号表示句子的结束，不常见或者没见过的词仍然用（ unknown ）代替。

开始训练语言模型时，将文本以句子的形式输入，比如这句话：

“猫平均每天睡15个小时”（视频中用的英文“Cat average 15 hours sleep a day”）

在第一个单元，初始化 $a^{<0>}$ 和 $x^{<1>}$ 的值为 0， y^1 将会对第一个字可能出现的每一个可能进行概率的判断。当然在最开始的时候没有任何的依据，可能得到的是完全不相干的字，因为只是根据初始的值和激活函数做出的取样。但在将值传入下一步的过程中永远传入实际文本的字，也就是不管我们第一步预测的是什么都将传入真正的第一个字“猫”。这个比较好理解，如果我们一直传错误的值，将永远也无法得到字与字之间的关系。

随着训练的次数的增多，或者常用词出现的频率的增多，语言模型便慢慢的会开始掌握简单的词语比如“平均”，“每天”，“小时”。一个完善的语言模型看到类似“10个小时”的时候，应该就能准确的判定下一个字是“时”。（当然也许实际情况是“10个小朋友”，所以通常会有更多的判断因素，这里只是一个例子）

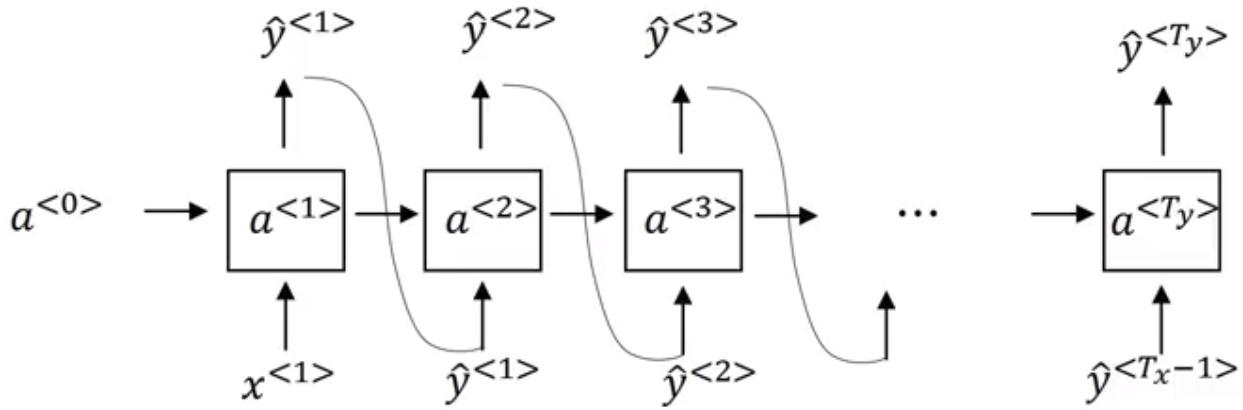
语言模型的损失函数：

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

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

语言模型的取样(Sampling)

在训练好模型之后，可以通过取样的方式来对模型做一个简单的评测，方法如下：



在每一步输出 y 时，通常使用 softmax 作为激活函数，然后根据输出的分布，随机选择一个值，也就是对应的一个字或者英文单词。然后将这个值作为下一个单元的 x 输入进去直到我们输出了终结符，或者输出长度超过了提前的预设值 n .

RNN的梯度消失、爆炸问题

梯度值在RNN中也可能因为反向传播的层次太多导致过小或者过大。当梯度值过小的时候，神经网络将无法有效地调整自己的权重矩阵导致训练效果不佳，称之为“梯度消失问题”(gradient vanishing problem)；过大时可能直接影响到程序的运作因为程序已经无法存储那么大的值，直接返回 NaN，称之为“梯度爆炸问题”(gradient exploding problem)。

当梯度值过大的时候有一个比较简便的解决方法，每次将返回的梯度值进行检查，如果超出了预定的范围，则手动设置为范围的边界值。

```
if (gradient > max) {
    gradient = max
}
```

但梯度值过小的解决方案要稍微复杂一点，比如这句话

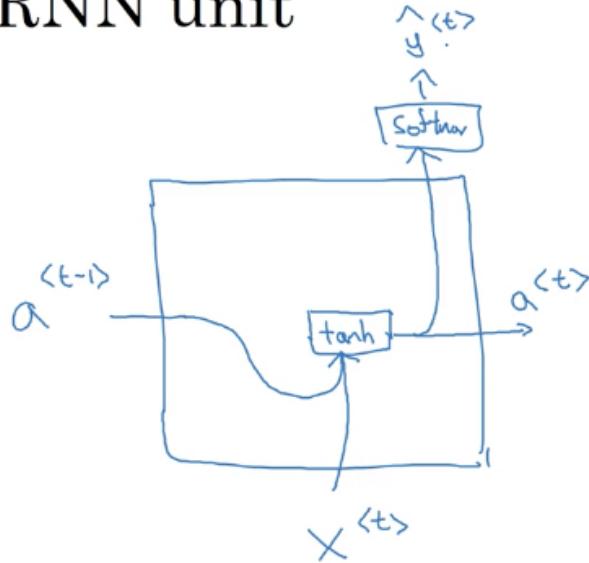
“The cat , which already ate apple , yogurt , banana , ... , was full.”

将在接下来的两个章节介绍两种方法来解决梯度过小问题，目标是当一些重要的单词离得很远的时候，比如例子中的“cat”和“was”，能让语言模型准确的输出单数人称过去时的“was”，而不是“is”或者“were”。两个方法都将引入“记忆”的概念，也就是为RNN赋予一个记忆的功能

GRU

GRU (Gated Recurrent Unit) 是一种用来解决梯度过小的方法，首先来看下在一个时刻下的RNN单元，激活函数为 tanh

RNN unit



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

\tanh

The equation shows the computation of the hidden state $a^{<t>}$ as a function g of the previous hidden state $a^{<t-1>}$ and the current input $x^{<t>}$, plus a bias b_a . The function g is labeled as \tanh .

这里引入两个新的符号c和Gamma(Γ) :

c (memory Cell) - 记忆单元 , 用它来记住“cat”是单数而不是复数 , GRU中 $c^{<t>} = a^{<t>}$

在每一时刻 (RNN单元内) , 计算候选c值

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

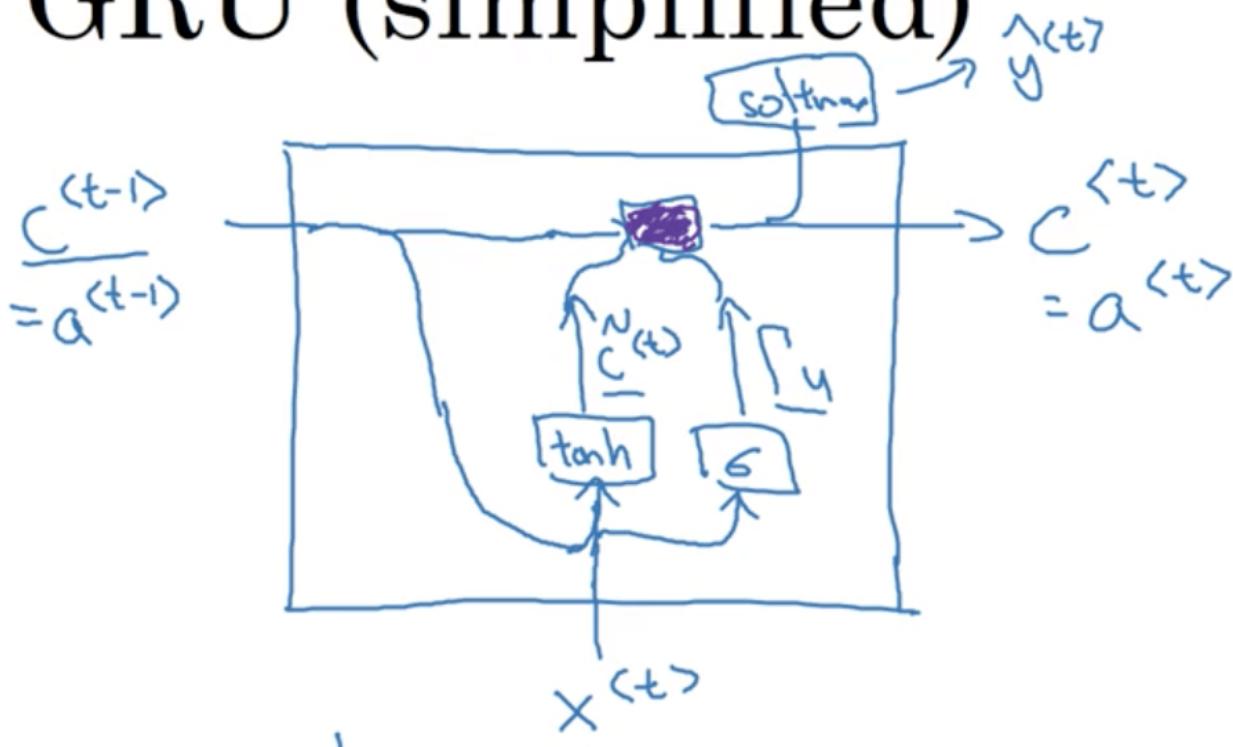
$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^t] + bu)$$

这里因为使用的激活函数是sigmoid , 所以 Γ_u 的值可以看做非1即0. 接下来是一个重要的公式 , 决定了什么时候更新c值 , 也就是用 \tilde{c} 替换 c

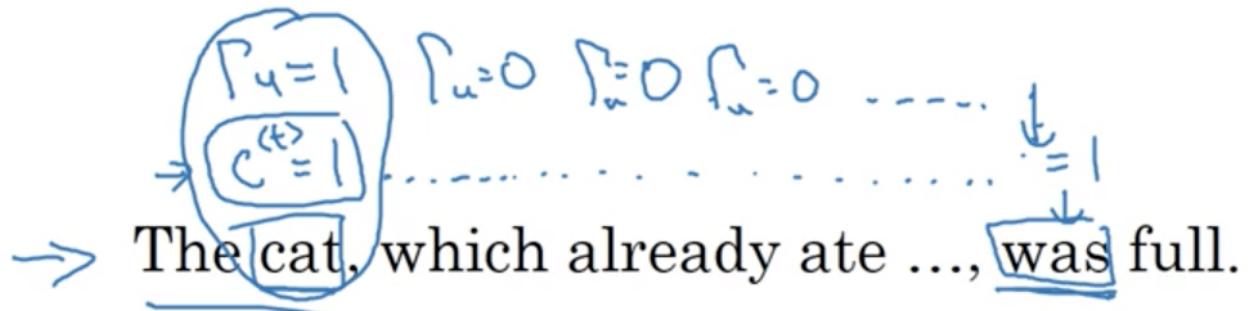
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

加入了这两个概念以后 , 单元就变成了这个样子

GRU (simplified)



希望达到的效果，在读到“cat”的时候 $\Gamma_u = 1$ ，其他时候一直为0，知道要输出“was”的时刻我们仍然知道“cat”的存在，也就知道它为单数了。



上述是简化了的GRU，在完整版中还存在另一个符号 Γ_r ，这个符号的意义是控制 c^{t-1} 和 \tilde{c}^{t-1} 之间的联系强弱，完整版公式如下：

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

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

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

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

LSTM(Long Short Term Memory)

有了GRU的基础，非常便于我们理解LSTM。相比于GRU，LSTM要更强大，通过设置不同的Gamma值，来更全面的控制了记忆部分，公式如下：

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

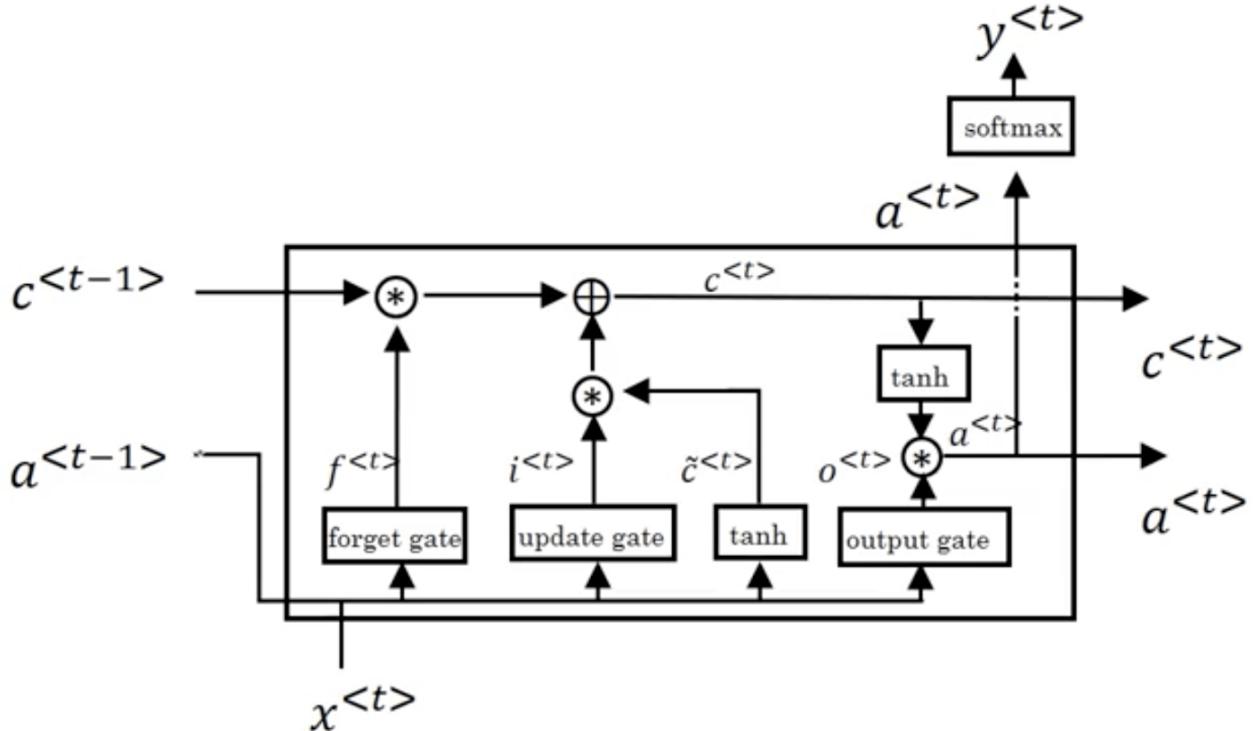
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

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

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

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

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



业界普遍认为LSTM要比GRU的效果更好，但是实际上这两种方法都有很广泛的应用。目前为止已经将搭建RNN的基础部分了解的差不多了，接下来是稍微详细的讲解了两种RNN的延伸版本，双向RNN 和 深度RNN.

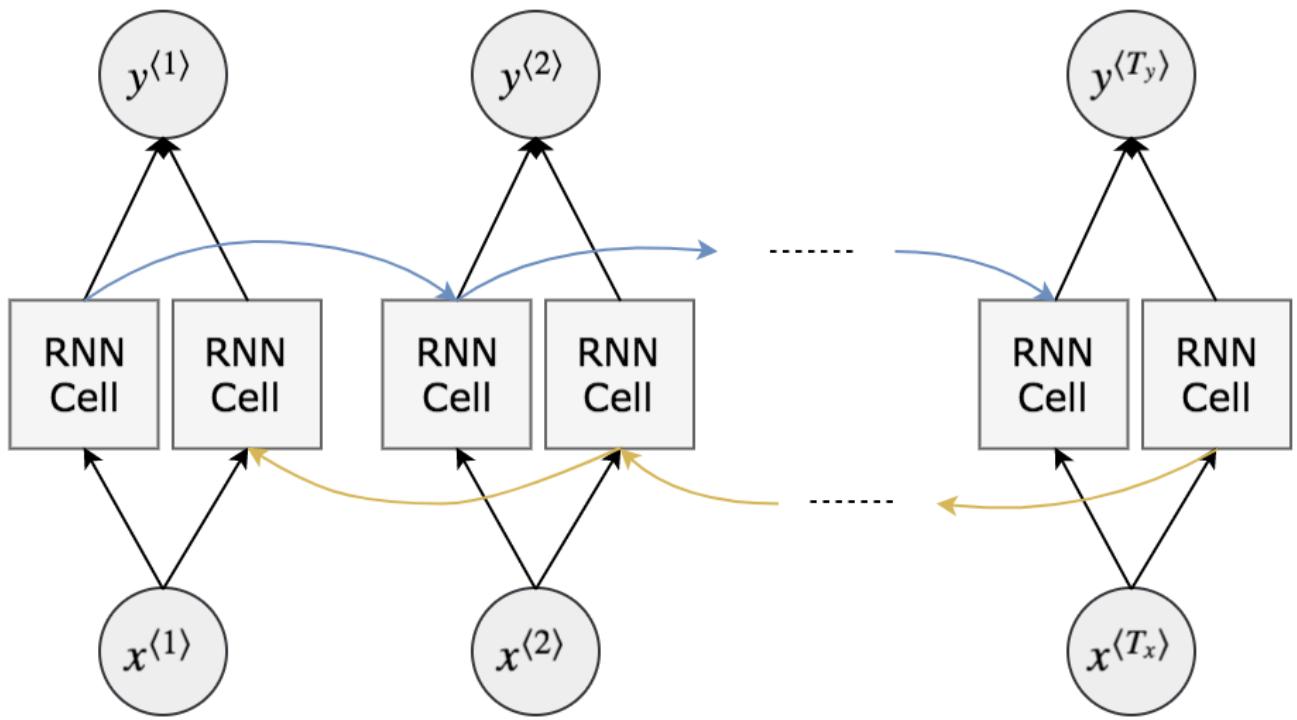
双向RNN

比如下面两句话

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

He said, "Teddy bears are on sale".

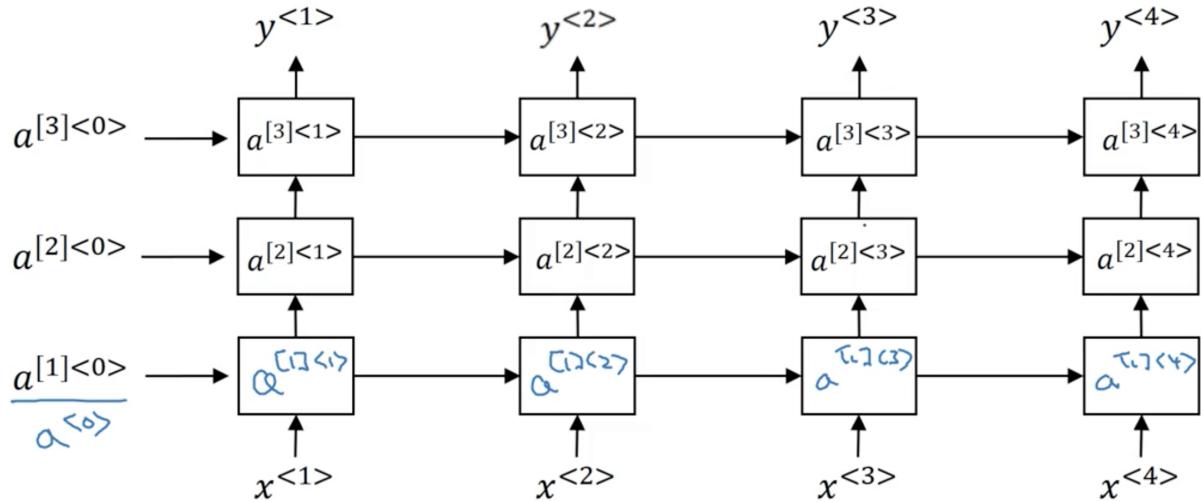
第一句中的Teddy Roosevelt是人名，但第二句中的Teddy bears是泰迪熊，同样都是单词Teddy对应的输出在第一句中应该是1，第二句中应该是0。像这样的例子如果想让我们的序列模型明白就需要借助不同的结构比如 - 双向递归神经网络(Bidirectional RNN)。神经网络首先从正面理解一遍这句话，再从反方向理解一遍。



这样的好处就是更好的理解了我们所说的“语境”，以便区分开两个不同Teddy的含义。

深层RNN(Deep RNN)

有的时候为了得到更好的结果，需要叠加很多个RNN在一起，这种网络我们称之为深层RNN



比如我们要计算 $a^{[2]<3>}$ ，则

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

第二周

上周的学习中，学习了如何用独热编码来代表一个词，这一节我们来探究一下词和词之间的联系。比如有下面这句话：

“I want a glass of orange _____”

假如我们的RNN的模型通过训练已经学会了短语“orange juice”，并准确的预测了这句话的空格部分，那么如果遇到了另一句话时，比如：

“I want a glass of apple _____”

是否需要从头学习短语“apple juice”呢？能否通过构建“apple”与“orange”的联系让它不需要重学就能进行判断呢？这就是第二周课程要讲的学习内容！

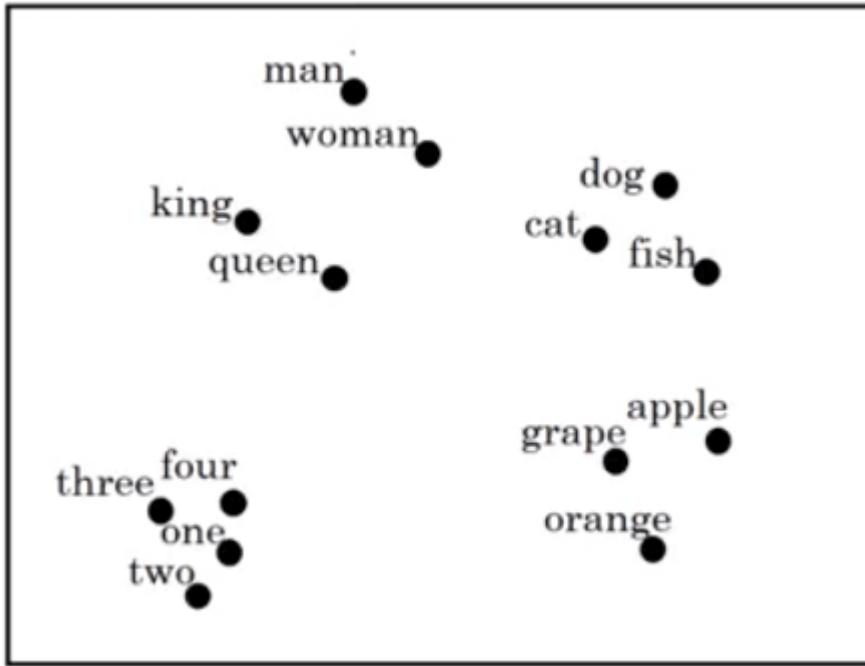
词汇的特性

单词与单词之间是有很多共性的，或在某一特性上相近，比如“苹果”和“橙子”都是水果；或者在某一特性上相反，比如“父亲”在性别上是男性，“母亲”在性别上是女性，通过构建他们其中的联系可以将在一个单词学习到的内容应用到其他的单词上来提高模型的学习的效率，这里用一个简化的表格说明：

	Man (5391)	Woman (9853)	Apple (456)	Orange (6257)
性别	-1	1	0	0.1
年龄	0.01	0.02	-0.01	0.00
食物	0.04	0.01	0.95	0.97
颜色	0.03	0.01	0.70	0.30

在表格中可以看到不同的词语对应着不同的特性有不同的系数值，代表着这个词语与当前特性的关系。括号里的数字代表这个单词在独热编码中的位置，可以用这个数字代表这个单词比如 $Man = O_{5391}$ ，Man的特性用 e_{5391} ，也就是那一纵列。在实际的应用中，特性的数量远不止4种，可能有几百种，甚至更多。对于单词“orange”和“apple”来说他们会共享很多的特性，比如都是水果，都是圆形，都可以吃，也有些不同的特性比如颜色不同，味道不同，但因为这些特性让RNN模型理解了他们的关系，也就增加了通过学习一个单词去预测另一个的可能性。

这里还介绍了一个 t-SNE 算法，因为词性表本身是一个很高维度的空间，通过这个算法压缩到二维的可视化平面上，每一个单词 嵌入 属于自己的一个位置，相似的单词离的近，没有共性的单词离得远，这个“嵌入”的概念就是下一节的内容 词嵌(word embeddings)



使用词嵌

先下一个非正规定义 “词嵌 - 描述了词性特征的总量，也是在高维词性空间中嵌入的位置，拥有越多共性的词，词嵌离得越近，反之则越远”。值得注意的是，表达这个“位置”，需要使用所有设定的词性特征，假如有300个特征（性别，颜色，...），那么词嵌的空间维度就是300.

使用词嵌主要通过以下三步：

1. 获得词嵌：获得的方式可以通过训练大的文本集或者下载很多开源的词嵌库
2. 应用词嵌：将获得的词嵌应用在我们的训练任务中
3. 可选：通过我们的训练任务更新词嵌库（如果训练量很小就不要更新了）

词嵌的特征

假设有如下的问题：

`"Man" -> "Woman" 那么 "King" -> ?`

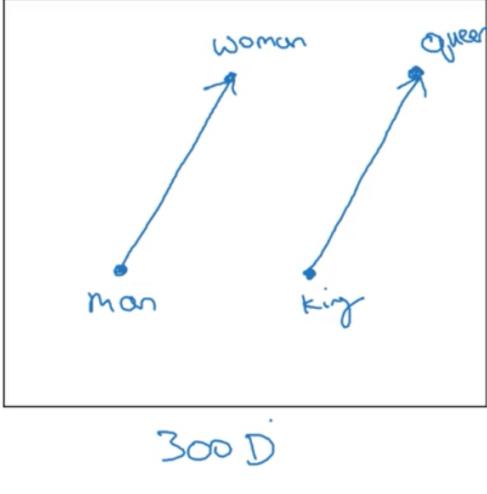
这个问题被称作词汇的类比问题，通过研究词嵌的特征可以解决这样的问题。

Analogy

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

$e_{man} - e_{woman} \approx e_{king} - e_w$
 e_w 是什么呢？在高纬度空间中 (300D)
 $\text{argmax}_w sim(e_w, e_{king} - e_{man} + e_{woman})$
 这个公式相当于在算两个向量(vector)的cos相似度

$$sim(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$


 $e_{man} - e_{woman} \approx e_{king} - e_{?}$

词嵌矩阵

这一节中主要讲了词嵌矩阵的shape，如果词嵌(词性特征的总量)是300，独热编码的长度是10000，那么词嵌矩阵的的shape就是 $300 * 10000$ 。所以就有了下面的式子：

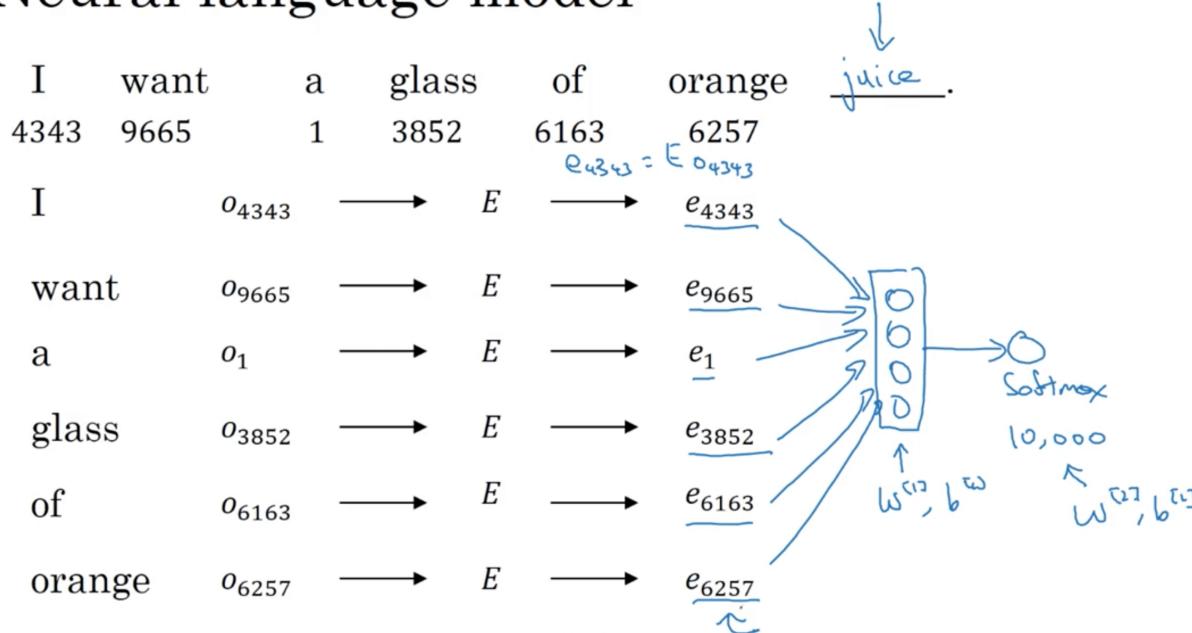
$$\text{词嵌矩阵} * \text{单词的独热编码} = \text{单词的词嵌}$$

$$(300, 10000) * (10000, 1) = (300, 1)$$

构建词嵌表

可以通过训练神经网络的方式构建词嵌表 E ，这次换个方式，先放图

Neural language model



[Bengio et. al., 2003, A neural probabilistic language model]

Andrew Ng

在这个训练模式中，是通过全部的单词去预测最后一个单词然后反向传播更新词嵌表 E

假设要预测的单词为 W ，词嵌表仍然为 E ，需要注意的是训练词嵌表和预测 W 是两个不同的任务。

如果任务是预测 W ，最佳方案是使用 W 前面 n 个单词构建语境。

如果任务是训练 E ，除了使用 W 前全部单词还可以通过：前后各 4 个单词、前面单独的一个词、前面语境中随机的一个词（这个方式也叫做 Skip Gram 算法），这些方法都能提供很好的结果。

Word2Vec

视频中一直没有给 Word2Vec 下一个明确的定义，我们再次下一个非正式定义便于理解 “word2vec 是指将词语 word 变成向量 vector 的过程，这一过程通常通过浅层的神经网络完成例如CBOW或者skip gram，这一过程同样可以视为构建词嵌表 E 的过程”。

这里着重介绍了 skip gram model，上一节介绍过这是一个用一个随机词预测其他词的方法。比如下面这句话中 “I want a glass of orange juice.”

我们可以选 orange 作为随机词 c (Context)，通过设置窗口值例如前后 5 个单词以监督学习的方式去预测其中的词 t (Target) 例如“juice, glass, a, of” 但需要注意的是，这个过程仍然是为了搭建(更新)词嵌表 E 而不是为了真正的去预测，所以如果预测效果不好并不用担心，表达式：

$$O_c \rightarrow E \rightarrow e_c \rightarrow \underset{\text{softmax}}{g} \rightarrow \hat{y}$$

softmax:

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$$

$$L(\hat{y}, y) = - \sum_{i=1}^{10000} y_i \log \hat{y}_i$$

在skip gram中有一个不足是softmax作为激活函数需要的运算量太大，在上限为10000个单词的词库中就已经比较慢了。一种补救的办法是用一个它的变种“Hierachical Softmax”，通过类似二叉树的方法提高训练的效率。

负取样(Negative Sampling)

对于skip gram model而言，还要解决的一个问题是如何取样（选择）有效的随机词 c 和目标词 t 呢？如果真的按照自然随机分布的方式去选择，可能会大量重复的选择到出现次数频率很高的单词比如说“the, of, a, it, I, ...”重复的训练这样的单词没有特别大的意义。

如何有效的去训练选定的词如 orange 呢？在设置训练集时可以通过“负取样”的方法，下表中第一行是通过和上面一样的窗口法得到的“正”（1）结果，其他三行是从字典中随机得到的词语，结果为“负”（0）。通过这样的负取样法可以更有效地去训练skip gram model.

context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0

负取样的个数 k 由数据量的大小而定，上述例子中为3. 实际中数据量大则 $k = 2 \sim 5$ ，数据量小则可以相对大一些 $k = 5 \sim 20$

通过负取样，我们的神经网络训练从softmax预测每个词出现的频率变成了经典binary logistic regression问题，概率公式用 sigmoid 代替 softmax从而大大提高了速度。

$$x_1 = (orange, juice) \rightarrow y_1 = 1$$

$$x_2 = (orange, king) \rightarrow y_2 = 0$$

...

$$P(y = 1|c, t) = \sigma(\theta_t^T e_c)$$

最后我们通过一个并没有被理论验证但是实际效果很好的方式来确定每个被负选样选中的概率为：

$$P(w_i) = \frac{f(w_i^{\frac{3}{4}})}{\sum_{j=1}^{10000} f(w_j)^{\frac{3}{4}}}$$

GloVe

GloVe(global vectors for word representation) 作为自然语言处理中的算法，没有像work2vec中的模型那么流行，但它仍然有自己独特的优点 - 简单。

X_{ij} = 词汇 i 在语境 j 中出现的次数，i 相当于之前的 t，j 相当于之前的 c，这个模型的意图是最小化下面这个公式：

$$\sum_{i=1}^{10000} \sum_{j=1}^{10000} f(x_{ij})(\theta_i^T e_j + b_i + b_j' - \log x_{ij})^2$$

$f(x_{ij})$ 存在是为了防止当 $x_{ij} = 0$ 时上面的式子等于0而不是无穷大。

在接下来的两节中将会介绍用到了这些技术的应用。

情绪分类

情绪分类是指将一段文字要表达的情绪通过RNN识别出来。比如说我们开了一家餐厅，在微博上有很多人给我们留言，现在的任务是将这些留言以分数的形式量化(1 ~ 5分)，以便做更好的改良，在RNN的帮助下可以做到这一点



The dessert is excellent.



Service was quite slow.



Good for a quick meal, but nothing special.



Completely lacking in good taste, good service, and good ambience.



这是一个“多对单”的应用，输入一句话，输出一个分数。

词嵌的去偏见化

因为RNN通常是通过大量的网络数据文本集进行训练得到的，所以很多时候文本集中的偏见会反映在词嵌以及最终的结果中，例如

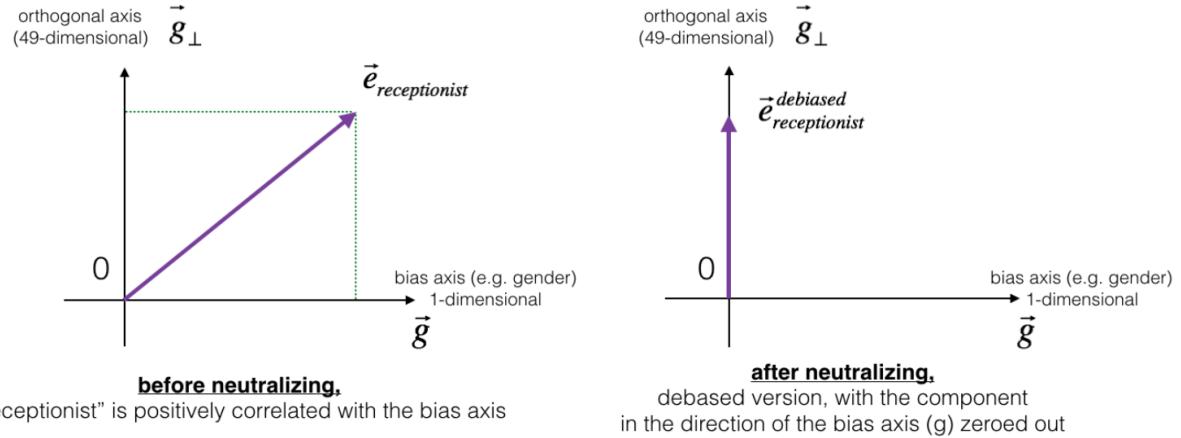
如果：“男人” 对应 “医生”，那么“女人” 对应 什么？ RNN: “女人” 对应 “护士”。

这种带有偏见的结果是应该尽力避免的，这类偏见大量存在于网络数据文本中，包括 性别偏见，种族偏见，年龄偏见，等等...

给词嵌去偏见主要分三步（在词嵌的高维空间中完成）：

1. 找到偏见的方向（确定偏见的x, y轴）
2. 将非定义化的词平移到x=0（父亲，母亲这类词就是定义化的词，本身就带有了性别的暗示）
3. 使定义化的词据离移动的词距离相等

上述的描述有些抽象，在作业中会有专门针对这一部分的练习。



第三周（最后一周啦！）

序列对序列

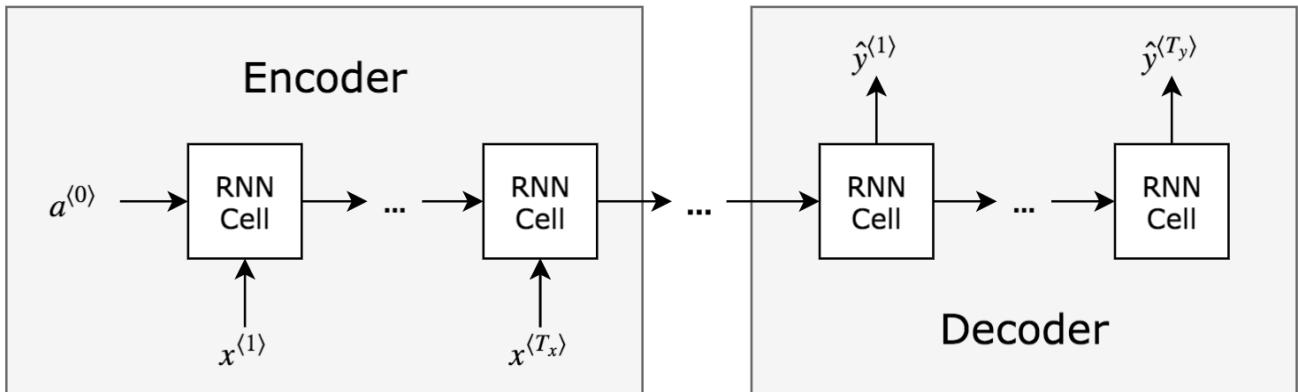
假设需要翻译下面这句话

"简将要在9月访问中国"

我们希望得到的结果是

"Jane is visiting China in September"

在这个例子中输入的数量是10个中文汉字，输出为6个单词， T_x 与 T_y 数量不一致，就需要用到序列对序列的RNN模型



类似的例子还有用机器为下面这张图片生成描述

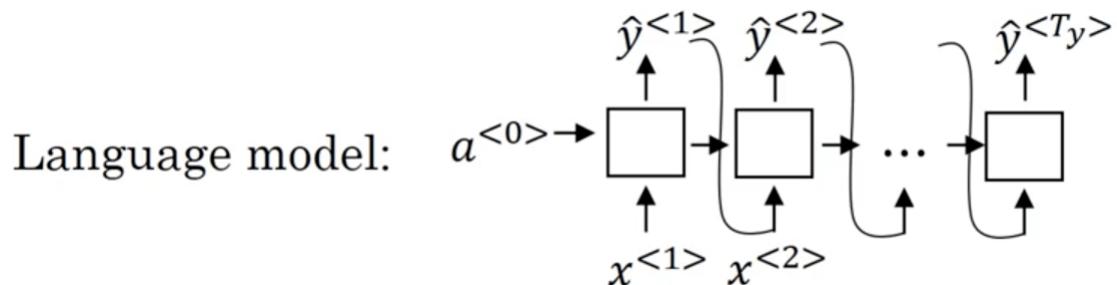


只需要将encoder部分用一个CNN模型替换就可以了，比如AlexNet，就可以得到“一只（可爱的）猫躺在楼梯上”

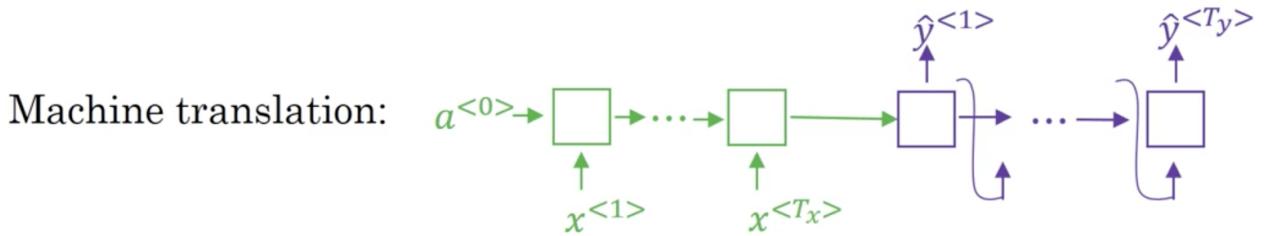
选择最优句子

下面将之前学习的语言模型和机器翻译模型做一个对比，P为概率

语言模型：



机器翻译模型：



可以看到，机器翻译模型的后半部分其实就是语言模型，Andrew将其称之为“条件语言模型”，在语言模型之前有一个条件也就是被翻译的句子：

$$P(y^{<1>} , \dots , y^{<T_y>} | x^{<1>} , \dots , x^{<T_x>})$$

注意，与语言模型不同的是，机器模型在输出部分不再使用softmax随机分布的形式进行取样，因为很容易得到一个不准确的翻译，取而代之的是使用Beam Search做最优化的选择。

Beam Search

Beam Search是greedy search的加强版本，首先要预设一个值 beam width，这里等于3（如果等于1就是greedy search）。然后在每一步保存最佳的3个结果进行下一步的选择，以此直到遇到句子的终结符

仍然是翻译这句话“简将要在9月访问中国”，前三步的示例：

- 第一步：

经过encoder以后，decoder给出最有可能的三个开头词依次为“in”，“jane”，“september” - $P(y^{<1>} | x)$

- 第二步：

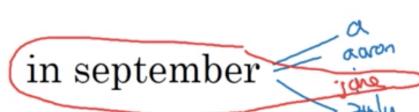
经过将第一步得到的值输入到第二步中，最有可能的三个个翻译为“in september”，“jane is”，“jane visits” - $P(y^{<2>} | x, y^{<1>})$

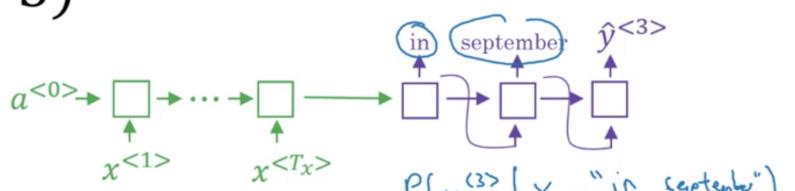
(这里，september开头的句子由于概率没有其他的可能性大，已经失去了作为开头词资格)

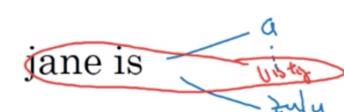
- 第三步：

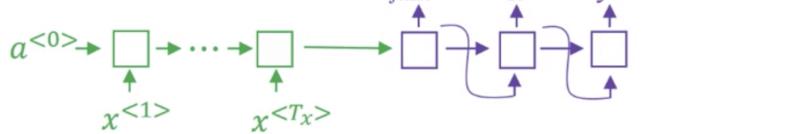
继续这个过程... $P(y^{<3>} | x, y^{<1>} , y^{<2>})$

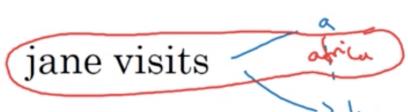
Beam search ($B = 3$)

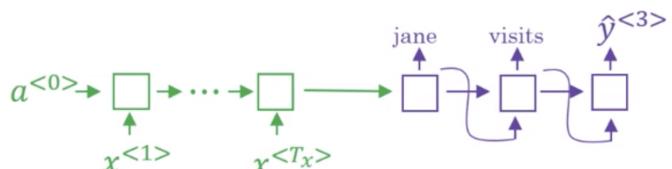












$$P(y^{<1>} , y^{<2>} | x)$$

jane visits africa in september. <EOS>

在实际应用和科研中beam width可能从10~3000不等

调试 Beam Search

由于在每一步计算的都是一个概率，那么最后得到一个翻译出来句子的概率就是：

$$\underset{y}{\operatorname{argmax}} \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

因为一直在算乘积的关系，每多一步就会使这个乘积越小，就会导致两个问题：

1. 乘积过小导致无法正确显示和读取
2. 翻译结果偏向于翻译短句因为句子越短乘积越大

为了解决这两个问题，用下面的式子替换掉原来的式子

$$\frac{1}{T_y^\alpha} \underset{y}{\operatorname{argmax}} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

其中 $\frac{1}{T_y^\alpha}$ 是为了归一化 (normalization) 长度问题， α 取值 0~1 之间，用 log 概率的和代替了概率乘积是为了防止越乘越小。

Beam Search的错误分析

现在有两个部分决定了翻译的结果RNN模型，和Beam Search，如果翻译的结果不是很准确，需要通过分析来判定是哪一部分出了问题。

设准确的翻译结果分析为 y^* ，模型翻译结果为 y^{\wedge} ，有且只有两种结果

$$1. P(y^* | x) > P(y^{\wedge} | x)$$

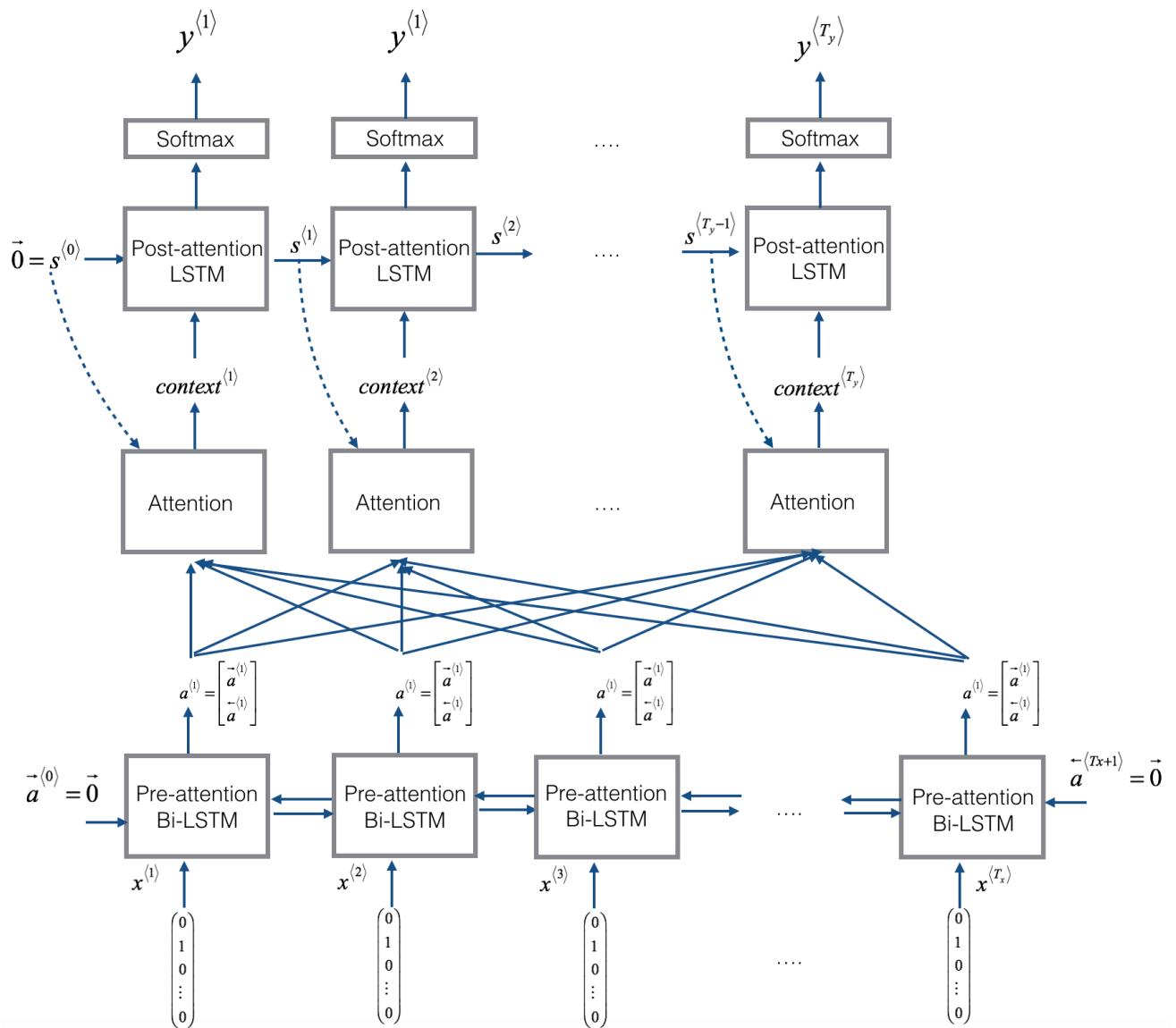
也就是说RNN输出结果为 y^* ，但beam选择了 y^{\wedge} ，是beam的部分出现了问题

$$2. P(y^{\wedge} | x) \leq P(y^* | x)$$

如果是这种结果说明RNN给出的就是 y^{\wedge} (不准确的结果)，所以是RNN部分出现了问题

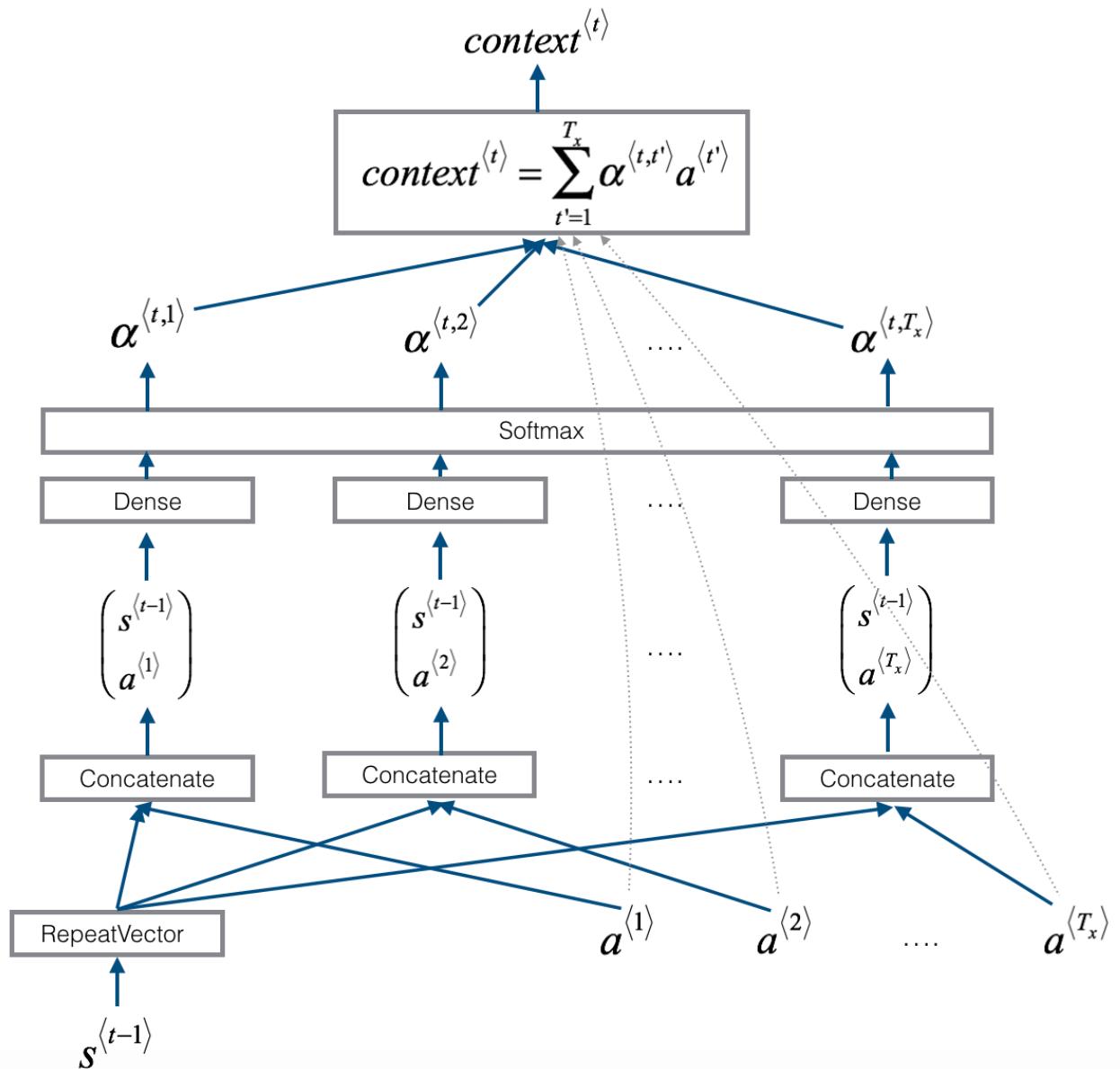
Attention Model(二合一)

通过之前的学习可以看到机器翻译是将所有要翻译的内容统一输入然后再开始生成结果，但这样有一个弊端就是在句子特别长的时候后面的内容有的时候无法翻译的特别的准确。通过搭建attention model可以解决这个问题：



如图所示，这是一个双向的rnn，并且在普通rnn的基础上增加attention层，将阶段性的输入部分转化为输出，这样得方式也更符合人类的翻译过程。

让我们拿出细节部分仔细的理解一下，首先是attention层，也就是下图中 $context^{<t>}$ ，每一个attention单元接受三个单词的输入所以也称作语境单元（context）， α 是每单个输入词在语境单元中占得权重。对每一个语境单元 t 来说，因为 α 是通过softmax决定的，所以 $\sum_{i=1}^{T_x} \alpha^{t,i} = 1$ 。这里决定最终每一个单词占得语境权重仍然是通过一个小型的神经网络来进行计算并且最后得到的。



输出的 $context^{(t)}$ 进入到下一层Post LSTM 这一步就和之前学习过的那样子，将前一步的输出与这一步经过重重分析的输入综合到一起产生这一步的输出。

让我们评估一下attention model：由于结构的复杂，计算量与时间比普通的语言模型要多和慢许多。不过对于机器翻译来说，由于每一句话并不会特别特比特的长，所以有的时候稍微慢一点也不是完全无法接受:p

一个很重要attention model的应用就是语音识别，人通过麦克风输入一句话让机器来翻译输入的内容，让我们来看一下是如何实现的

语音识别

当人对着麦克风录入一句话，麦克风记录下来的是空气细微的震动的强度，以及频率。人耳在听到一句话的时候其实做的是类似的处理。在深度学习没有特别流行之前，比较流行的是用音节做语音识别，但现在因为有了强大的attention model，得到的结果比音节的效果更好。

- CTC(connectionist temporal classification)

因为输入的数据是空气的震动与频率，所以 T_x 也就是x的长度有的时候要远远的多于输出的长度。比如1000长度的 x 可能产生出4个单词，所以我们用CTC来进行输出方面的编码，可以让 y 的长度更接近与 x 的长度，例如：

```
a a a a _ _ <space> c c c _ a a _ t t t t t
```

上面的这个序列其实代表的是“a cat”，注意中间的 `<space>` 对应了翻译中的空格，重复的字母和“_”都是可以被压缩的。

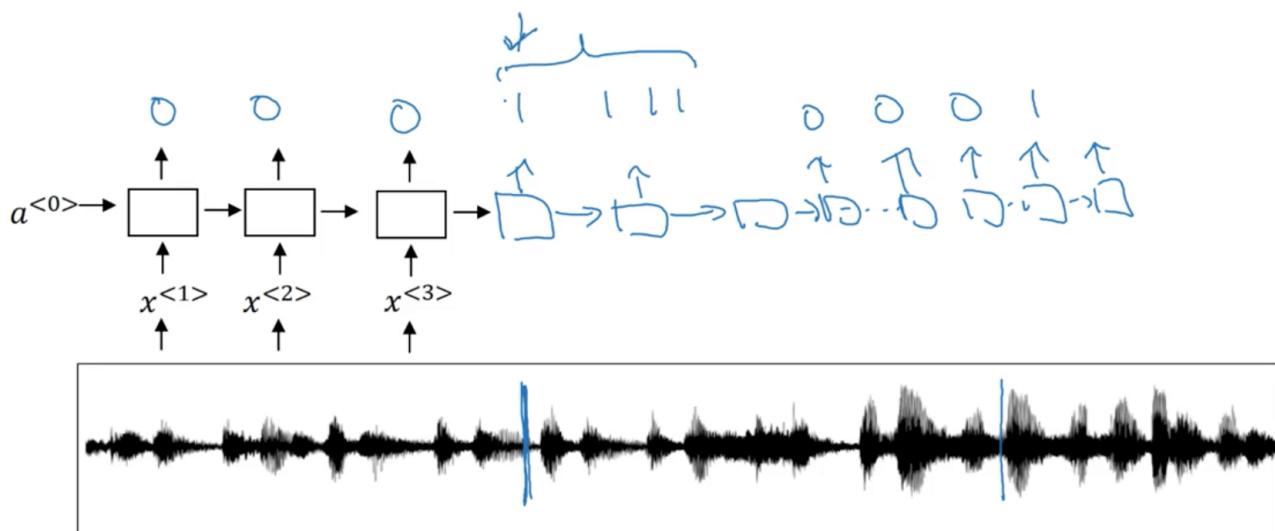
唤醒词检测

例如苹果的siri，谷歌的google home都有自己独特的唤醒词

"Hey Siri" 或者 "Okay Google"

最后让我们看下如何训练rnn可以检测到这些唤醒词，假设下图式训练集中的一段音频，其中包含了两次唤醒词

Trigger word detection algorithm



搭建一个attention model，在听到唤醒词之前一直输出的是0，在听到唤醒词以后输出1，但因为一个唤醒词会持续半秒左右所以我们也不仅仅只输出一次1，而是将输出的1持续一段时间，通过这样的方式训练出的rnn就可以很有效的检测到唤醒词了。

总结

感谢吴恩达和他的团队给我们带来这么好的教程，希望这篇笔记对你有所帮助 :)