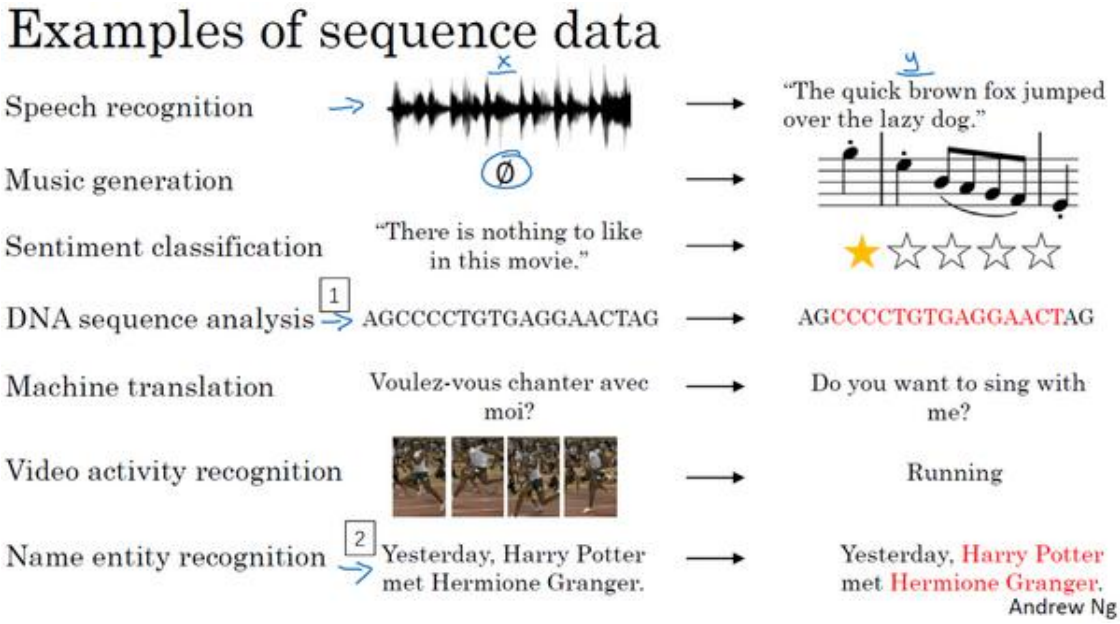
第一周 循环序列模型

1.1 为什么选择序列模型？

循环神经网络（RNN）之类的模型在语音识别、自然语言处理和其他领域中引起变革。我们先看一些例子，这些例子都有效使用了序列模型。



语音识别：输入音频片段 x，并要求输出对应的文字记录 y。输入和输出数据都是序列模型，因为 x 是一个按时播放的音频片段，输出 y 是一系列单词。

音乐生成：输出数据 y 是序列，而输入数据可以是空集，也可以是个单一的整数，这个数可能指代你想要生成的音乐风格，也可能是你想要生成的那首曲子的头几个音符。

DNA 序列分析：DNA 可以用 A、C、G、T 四个字母来表示。于是输入为给定一段 DNA 序列，需要标记出哪部分是匹配某种蛋白质。

情感分类：输入数据x是序列，你会得到类似这样的输入：**“There is nothing to like in this movie.”**，输出 y 是x所分的类别。

机器翻译：输入序列为某种语言的连成的句子，如**“Voulez-vou chante avecmoi?”**（法语：要和我一起唱么？），然后要求输出另一种语言的翻译结果。

视频行为识别：输入为一系列视频帧，然后要求识别其中的行为。

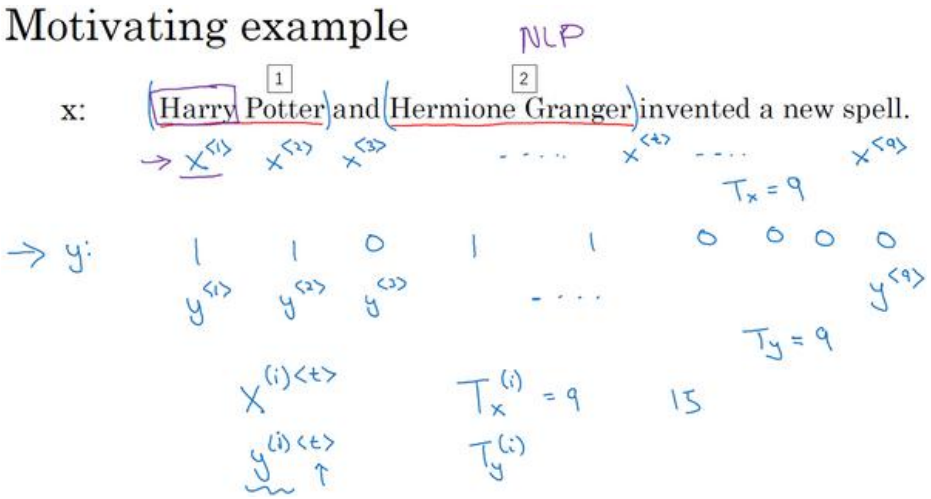
命名体识别：给定一个句子要你识别出句中的人名/地名等。

所有这些问题都可以被称作使用标签数据 (x,y)作为训练集的监督学习。但从这一系列例子中你可以看出序列问题有很多不同类型。有些问题里，输入数据 x 和输出数据 y 都是序列，但就算在那种情况下，x 和 y 有时也会不一样长。或者像上图编号 1 所示和上图编号 2 的 x 和 y 有相同的数据长度。在另一些问题里，只有 x 或者只有 y 是序列。

1.2 数学符号

本节介绍一些定义序列问题要用到的符号。

以下面这个命名体识别问题为例，它的输入语句是这样的：**“Harry Potter and Herminoe Granger invented a new spell.”**，需要建立一个自动识别句中人名位置的序列模型。



输入、输出形式

给定这样的输入数据x，想要一个序列模型输出y，使得输入的每个单词都对应一个输出值，同时这个y能够表明输入的单词是否是人名的一部分。

输入数据：9 个单词组成的序列，所以最终我们会有 9 个特征集和来表示这 9 个单词，并按序列中的位置进行索引，$x^{<1>}$、$x^{<2>}$、$x^{<3>}$等等一直到$x^{<9>}$来索引不同的位置，我将用$x^{<t>}$来索引这个序列的中间位置。t 意味着它们是时序序列，但不论是否是时序序列，我们都将用 t 来索引序列中的位置。

输出数据：我们用$y^{<1>}$、$y^{<2>}$、$y^{<3>}$等等一直到$y^{<9>}$来表示输出数据。

T_x来表示输入序列的长度，T_y表示输出序列的长度。

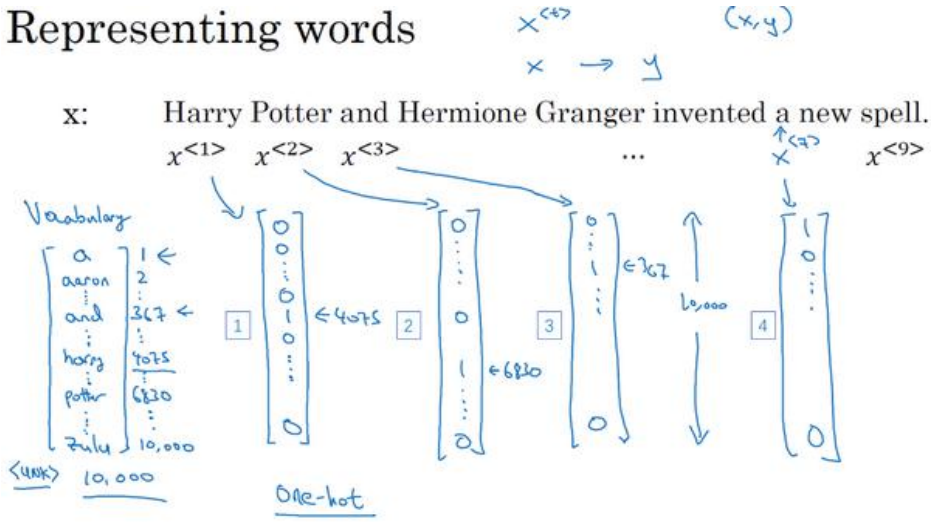
$x^{(i)}$来表示第 i 个训练样本，$x^{(i)<t>}$表示训练样本 i 的序列中第 t 个元素。

$T_x^{(i)}$就代表第 i 个训练样本的输入序列长度，$T_y^{(i)}$就代表第 i 个训练样本的输出序列长度。

词的表示：

此节中介绍 one-hot 向量表示法：

先构建一张词表，即列出你的表示方法中用到的单词。如下图左所示。



词典大小：对于一般规模的商业应用 30,000 到 50,000 词大小的词典比较常见，但是 100,000 词也有，有些大型互联网公司会用百万词，甚至更大的词典。

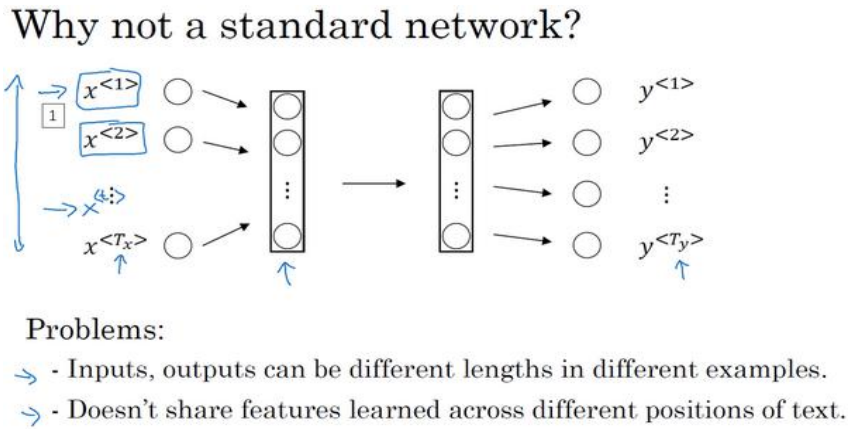
比如 **Harry** 在词表中 4075 这个位置，对应向量$x^{<1>}$表示 **Harry** 这个单词，它就是一个第 4075 行是 1，其余值都是 0 的向量（上图编号 1 所示）。这就是 one-hot 向量表示。

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

1.3 循环神经网络模型

本节中介绍如何建立一个神经网络来学习 X 到 Y 的映射。

标准神经网络

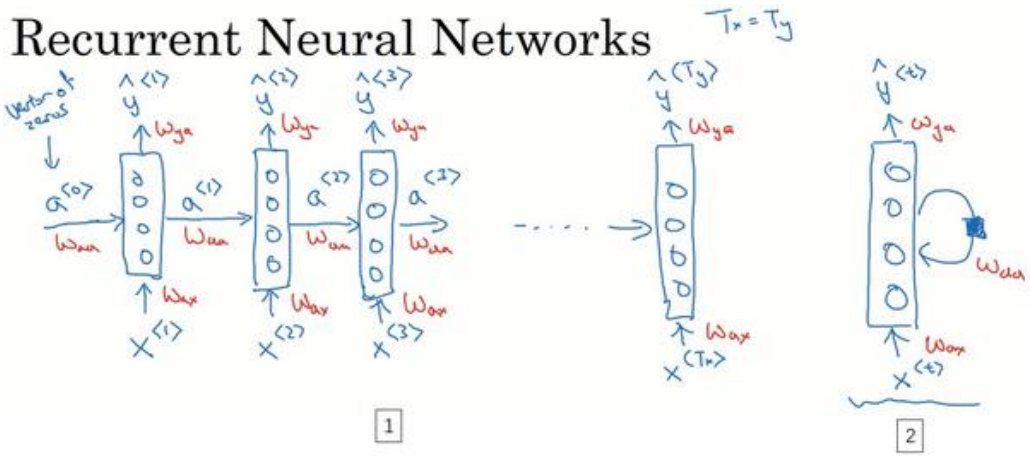


主要有两个问题：

①输入和输出数据在不同例子中可以有不同的长度，不是所有的例子都有着同样输入长度T_x或是同样输出长度的T_y。即使可以 pad，但仍然不是一个好的表达方式。

②神经网络不共享从文本的不同位置上学到的特征。比如神经网络学习到了位置 1 出现的 Harry 可能是人名的一部分，我们希望 **Harry** 出现在其他位置，比如$x^{<t>}$时，它也能够自动识别其为人名的一部分。

循环神经网络



我们将第一个单词$x^{<1>}$输入一个神经网络层，可以让神经网络尝试预测输出，判断这是否是人名的一部分。循环神经网络做的是，当它读到句中的第二个单词时，假设是$x^{<2>}$，它不是仅用$x^{<2>}$就预测出$\hat{y}^{<2>}$，他也会输入一些来自时间步 1

的信息。具体而言，时间步 1 的激活值就会传递到时间步 2。然后，在下一个时间步，循环神经网络输入了单词$x^{<3>}$，然后它尝试预测输出了预测结果$\hat{y}^{<3>}$，等等，一直到最后一个时间步，输入了$x^{<T_x>}$，然后输出了$\hat{y}^{<T_y>}$。在每一个时间步中，循环神经网络传递一个激活值到下一个时间步中用于计算。

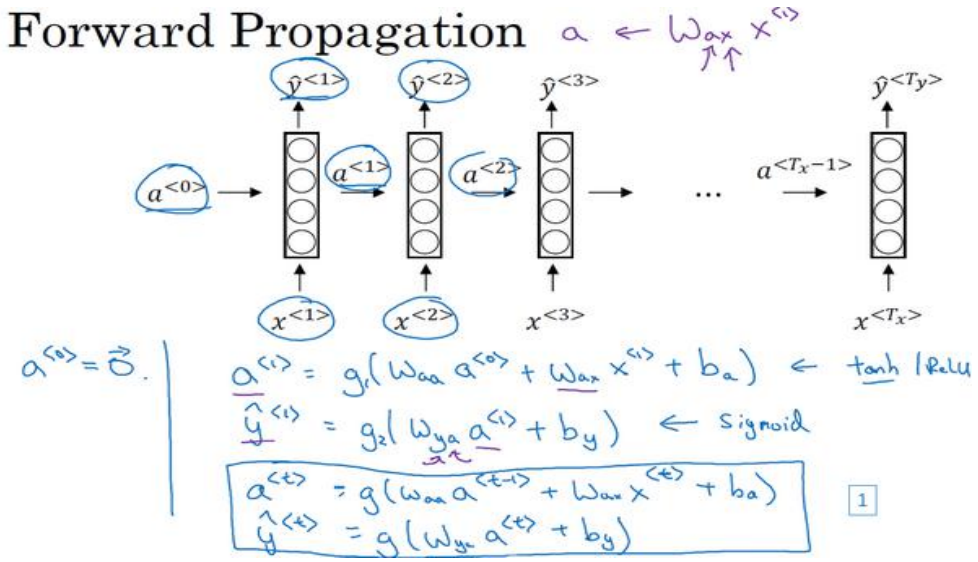
上图所示例子中$T_x = T_y$，如果T_x和T_y不相同，上述结构会需要作出一些改变。

零时刻需要构造一个激活值$a^{<0>}$，通常是零向量。

循环神经网络是从左向右扫描数据，每个时间步的参数也是共享的。

上图循环神经网络的缺点：只使用了这个序列中之前的信息来做出预测。比如“**Teddy Roosevelt was a great President.**”，为了判断 **Teddy** 是否是人名的一部分，仅仅知道句中前两个词是完全不够的，还需要知道句中后部分的信息，这也是十分有用的，因为句子也可能是这样的，“**Teddy bears are on sale!**”。所以后续的解决方法可以使用双向循环神经网络（BRNN）。

RNN 前向传播示意图：



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

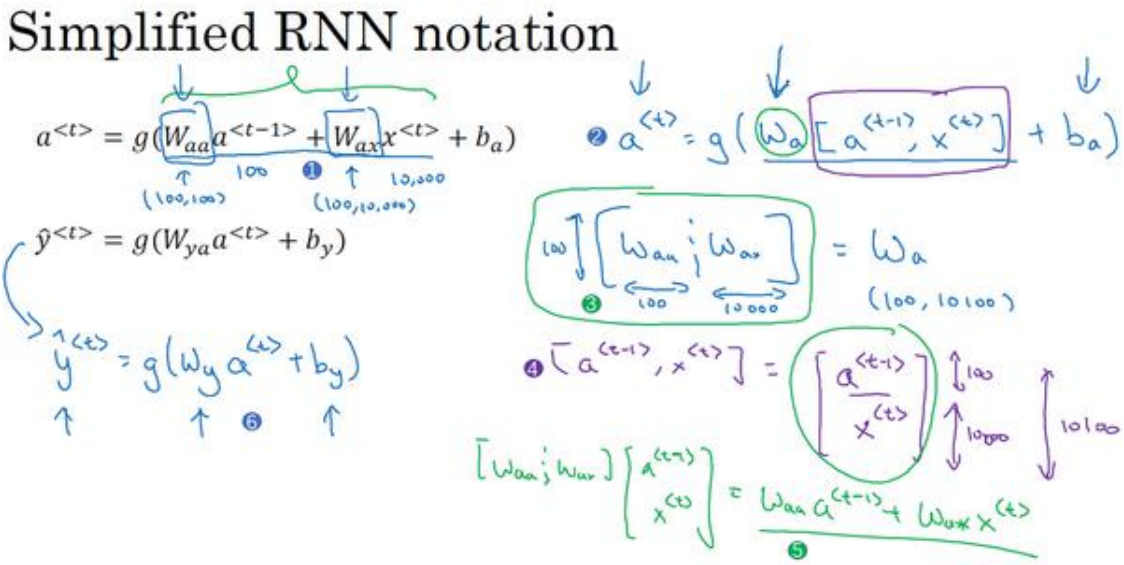
$$\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

循环神经网络用的激活函数经常是 tanh，不过有时候也会用 ReLU，但是 tanh 是更通常的选择。选用哪个激活函数连接输出是取决于你的输出y，如果它是一个二分问题，可以用 **sigmoid** 函数作为激活函数，如果是k类别分类问题的话，那么可以选用 **softmax** 作为激活函数。

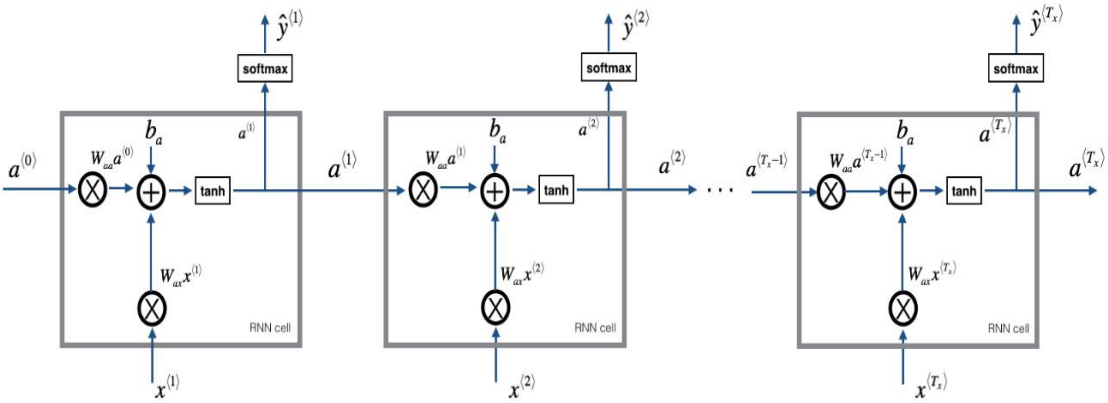
你可以从零向量$a^{<0>}$开始，然后用$a^{<0>}$和$x^{<1>}$来计算出$a^{<1>}$和$\hat{y}^{<1>}$，然后用$x^{<2>}$和$a^{<1>}$一起算出$a^{<2>}$和$\hat{y}^{<2>}$等等，像图中这样，从左到右完成前向传播。

为了简化符号，将（$W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$）写做$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$，我们定义$W_a$的方式是将矩阵$W_{aa}$和矩阵$W_{ax}$水平并列放置，$[W_{aa} : W_{ax}] = W_a$，用这个符号（$[a^{<t-1>}, x^{<t>}]$）表示这两个向量堆在一起，即$\begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$。而对于（$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$），

用更简单的方式重写，$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$。$W_y$就表明它是计算 y 类型的量的权重矩阵，而上面的$W_a$和$b_a$则表示这些参数是用来计算$a$类型或者说是激活值的。



RNN 前向传播示意图：



1.4 通过时间进行反向传播

首先分析一下前向传播的计算，现在你有一个输入序列，$x^{<1>}$，$x^{<2>}$，$x^{<3>}$一直到$x^{<T_x>}$，然后用$x^{<1>}$还有$a^{<0>}$计算出时间步 1 的激活项，再用$x^{<2>}$和$a^{<1>}$计算出$a^{<2>}$，然后计算$a^{<3>}$等等，一直到$a^{<T_x>}$，由激活值又可以计算出$\hat{y}$。

这其中用到的参数有W_a和b_a，W_y和b_y。

我们定义一个元素的损失函数：

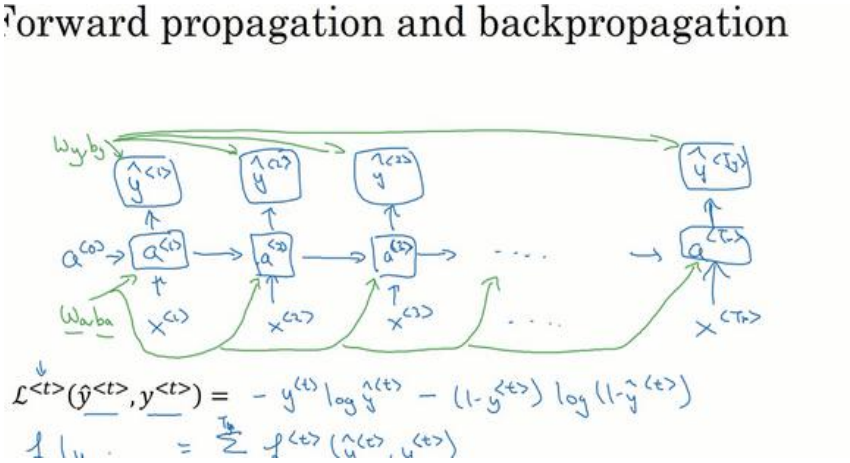
$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>}\log\hat{y}^{<t>} - (1-\hat{y}^{<t>})log(1-\hat{y}^{<t>})$$

对应的是序列中一个具体的词，如果它是某个人的名字，那么$y^{<t>}$的值就是 1，然后神经网络将输出这个词是名字的概率值，比如 0.1。我们的损失函数定义为标准逻辑回归损失函数，即交叉熵损失函数。

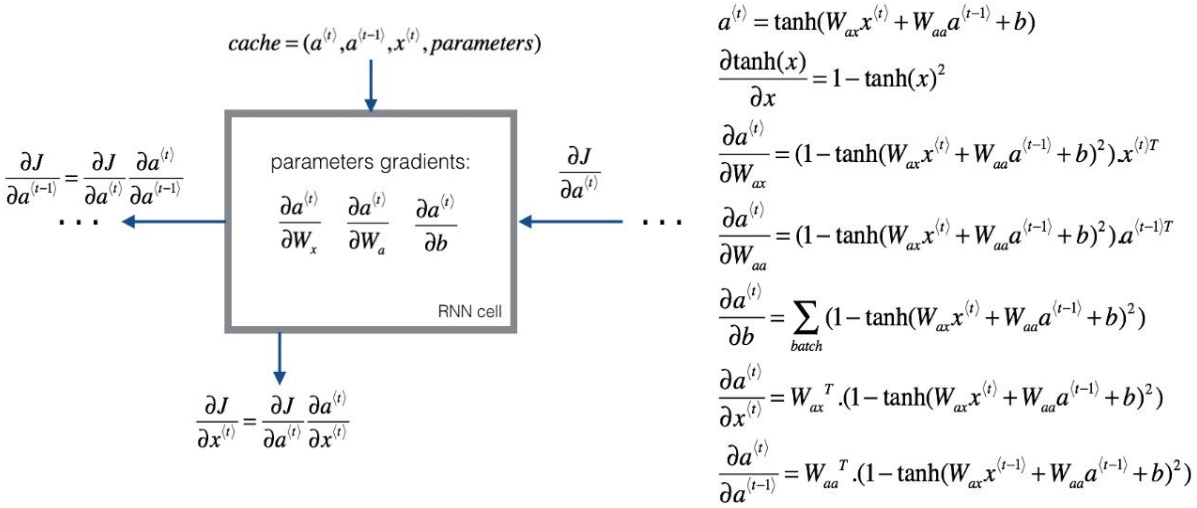
定义整个序列的损失函数：

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

每个时间步上的损失函数相加。

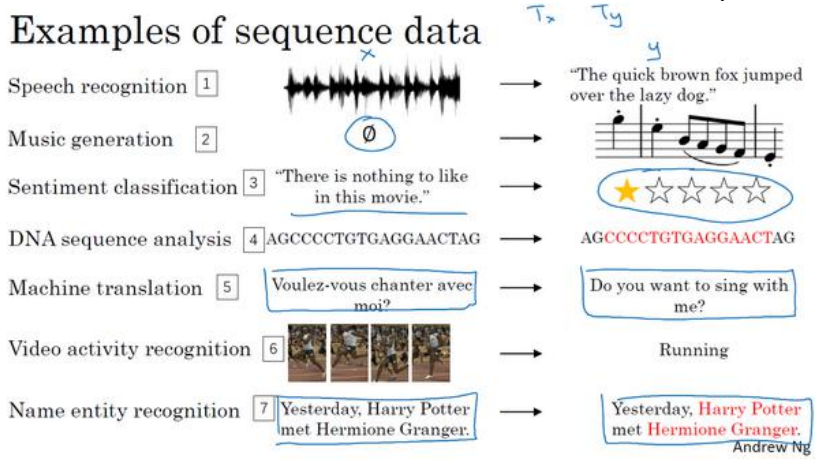


RNN 反向传播示意图：

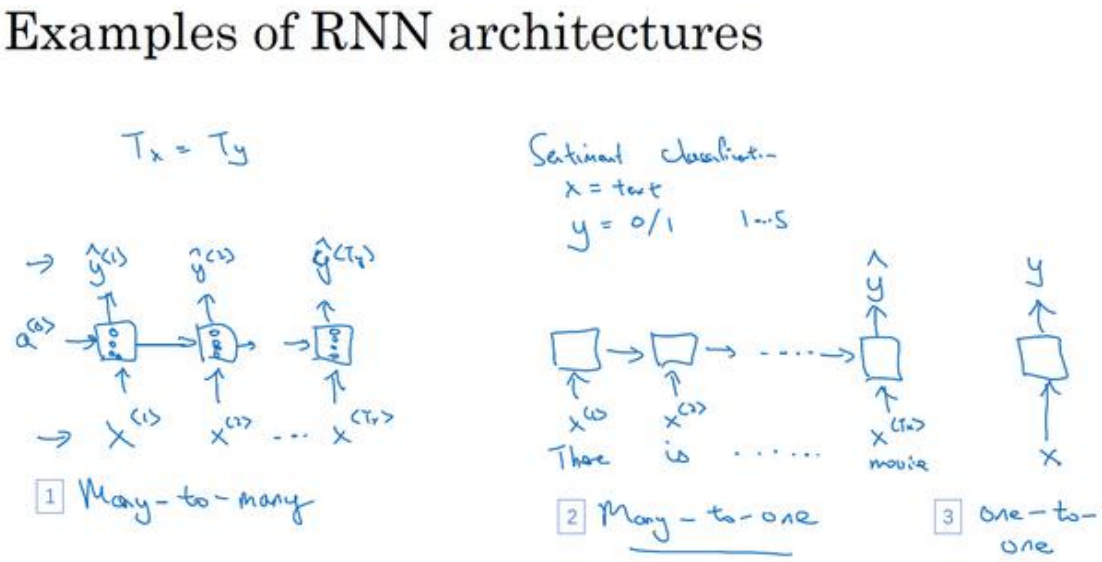


1.5 不同类型的循环神经网络

第一节课中介绍的例子中，并不是所有的情况都满足$T_x = T_y$。



比如音乐生成这个例子，T_x可以是长度为 1 甚至为空集。再比如电影情感分类，输出y可以是 1 到 5 的整数，而输入是一个序列。还有一些情况，输入长度和输出长度不同，他们都是序列但长度不同，比如机器翻译，一个法语句子和一个英语句子不同数量的单词却能表达同一个意思。

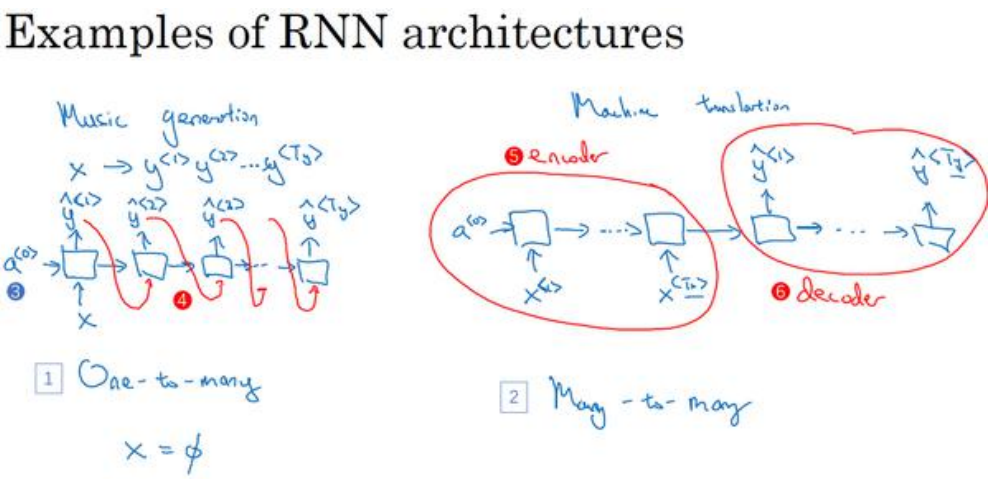


几种 RNN 的结构

多对多：输入序列$x^{<1>}$，$x^{<2>}$，一直到$x^{<T_x>}$，我们的循环神经网络这样工作，输入$x^{<1>}$来计算$\hat{y}^{<1>}$，$\hat{y}^{<2>}$等等一直到$\hat{y}^{<T_y>}$。

多对一：比如情感分类问题。

一对一：输入 x 然后得到输出 y，一个小型的标准的神经网络。



一对多：如音乐生成。输入 x 可以是一个整数，表示你想要的音乐类型或者是你想要的音乐的第一个音符，并且如果你什么都不想输入，x 可以是空的输入，可设为 0 向量。

多对多的特殊情形（序列到序列）：输入和输出长度不同的情况，比如机器翻译问题，需要一个编码器，一个解码器。

1.6 语言模型和序列生成

语言模型，通俗而言，用于计算出一句话的可能性。

比如一个语音识别系统，听到一个句子，“**the apple and pear（pair） salad was**

delicious.”，需要判断是 “**the apple and pair salad**”，还是“**the apple and pear salad**”？人类听起来大概率会认为听到的是后一句话，因为后一句话在人类语言中出现更为合理，语言模型的作用，则是模仿人类这一预测的过程，计算出这两句话各自的可能性。

一个语音识别模型可能算出第一句话的概率是:

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

而第二句话的概率是 P(The apple and pear salad) $= 5.7 \times 10^{-10}$,

比较这两个概率值，听到的语音更像是第二句，因为第二句话的概率比第一句高出 1000 倍以上，这就是为什么语音识别系统能够在这两句话中作出选择。

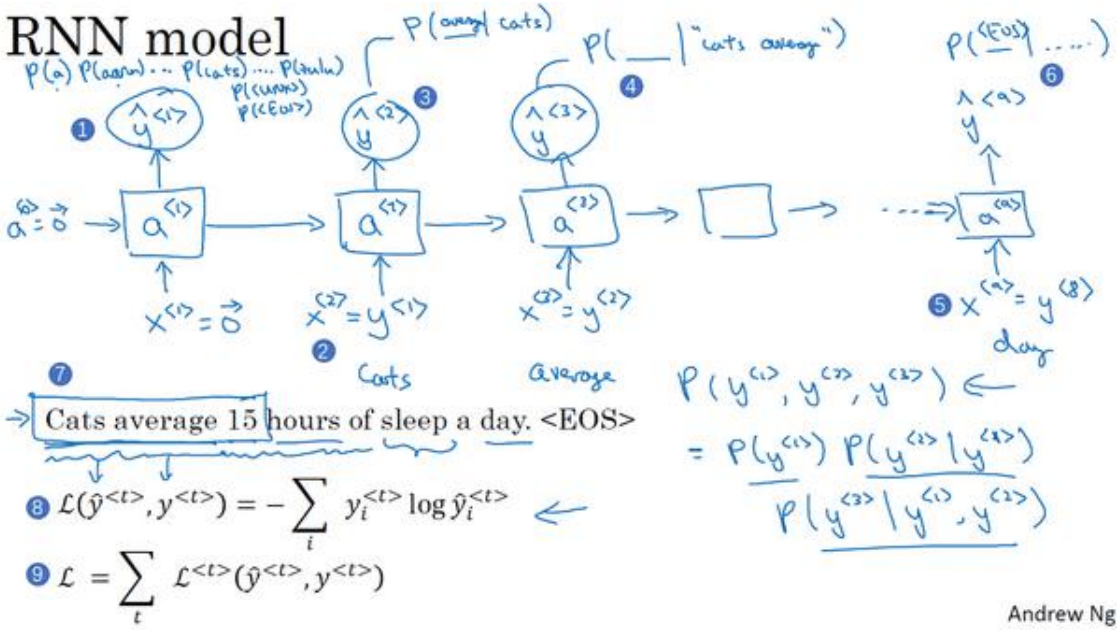
语言模型做的最基本工作就是输入一个句子，准确地说是一个文本序列，$y^{<1>}$，$y^{<2>}$一直到$y^{<T_y>}$。对于语言模型来说，用 y 来表示这些序列比用 x 来表示要更好，然后语言模型会估计某个句子序列中各个单词出现的可能性。

如何建立语言模型？

首先需要一个训练集，包含一个很大的英文文本语料库（corpus）或者其它的语言，你想用于构建模型的语言的语料库。

加入训练集中得到这么一句话，“**Cats average 15 hours of sleep a day.**”，首先需要将这个句子标记化（建立词典，单词转换为 one-hot 向量，EOS 标记结尾）。

完成标识化的过程后，这意味着输入的句子都映射到了各个标志上，或者说字典中的各个词上。然后，构造一个 RNN 来构建这些序列的概率模型。



这个 RNN 中，用$y^{<t-1>}$作为$x^{<t>}$，即前一个预测结果作为下一时间步的输入。

在第 0 个时间步，需要计算激活项$a^{<1>}$，它是以$x^{<1>}$作为输入的函数，而$x^{<1>}$会被设为全为 0 的集合，也就是 0 向量。在之前的$a^{<0>}$按照惯例也设为 0 向量，于是$a^{<1>}$要做的就是它会通过 **softmax** 进行一些预测来计算出第一个词可能会是什么，其结果就是$\hat{y}^{<1>}$（上图编号 1 所示），这一步其实就是通过一个 **softmax** 层来预测字典中的任意单词会是第一个词的概率，比如说第一个词是a的概率有多少，第一个词是 **Aaron** 的概率有多少，第一个词是 **cats** 的概率又有多少，就

这样一直到 **Zulu** 是第一个词的概率是多少，还有第一个词是 **UNK**（未知词）的概率有多少，还有第一个词是句子结尾标志的概率有多少，表示不必阅读。所以$\hat{y}^{<1>}$的输出是 **softmax** 的计算结果，它只是预测第一个词的概率，而不去管结果是什么。在我们的例子中，最终会得到单词 **Cats**。所以 **softmax** 层输出 10,000 种结果，因为字典中有 10,000 个词，或者会有 10,002 个结果，因为可能加上了未知词，还有句子结尾这两个额外的标志。

然后 **RNN** 进入下个时间步，在下一时间步中，仍然使用激活项$a^{<1>}$，在这步要做的是计算出第二个词会是什么。现在我们依然传给它正确的第一个词，我们会告诉它第一个词就是 **Cats**，也就是$\hat{y}^{<1>}$，告诉它第一个词就是 **Cats**，这就是为什么$y^{<1>} = x^{<2>}$（上图编号 2 所示）。然后在第二个时间步中，输出结果同样经过 **softmax** 层进行预测，**RNN** 的职责就是预测这些词的概率。

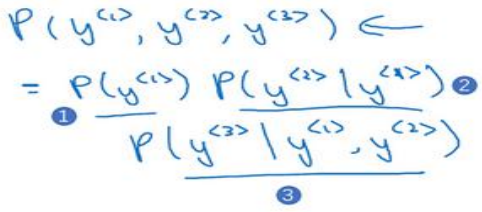
一直预测到最后，对于这一句话而言，停留在第九个时间步。我们希望此时能预测出 EOS 句子结尾标志的概率会很高。

为了训练这个网络，我们需要定义代价函数，

于是，在某个时间步t，如果真正的词是$y^{<t>}$，而神经网络的 **softmax** 层预测结果值是$\hat{y}^{<t>}$，那么损失函数为$L\left(\hat{y}^{<t>}, y^{<t>}>\right) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$。

总体损失函数为$L = \sum_t L^{<t>}\left(\hat{y}^{<t>}, y^{<t>}\right)$。

句子概率的计算：

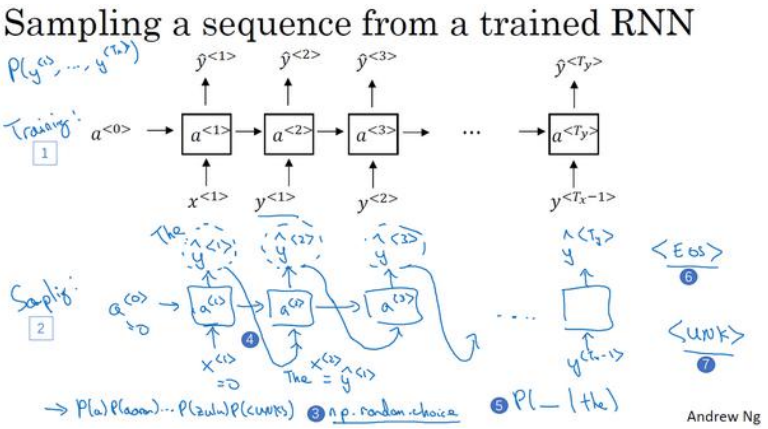


$y^{<1>}$的概率，考虑$y^{<1>}$的情况下$y^{<2>}$的概率，考虑$y^{<1>}$和$y^{<2>}$的情况下$y^{<3>}$的概率；这三者相乘，得到这个含 3 个词（顺序）的整个句子的概率。

1.7 对新序列采样

当训练完一个序列模型之后，想了解到这个模型学到了什么，一种非正式的方法就是进行一次新序列采样。

一个序列模型模拟了任意特定单词序列的概率，我们要做的就是对这些概率分布进行采样来生成一个新的单词序列。



①对第一个词进行采样，输入$x^{<1>}=0$，$a^{<0>}=0$，于是第一个时间步得到的是所有可能的输出是经过 softmax 层后得到的概率，然后根据这个 softmax 的分布进行随机采样。（`np.random.choice`）

②第二个时间步需要$\hat{y}^{<1>}$作为输入，然后 softmax 层预测$\hat{y}^{<2>}$是什么。

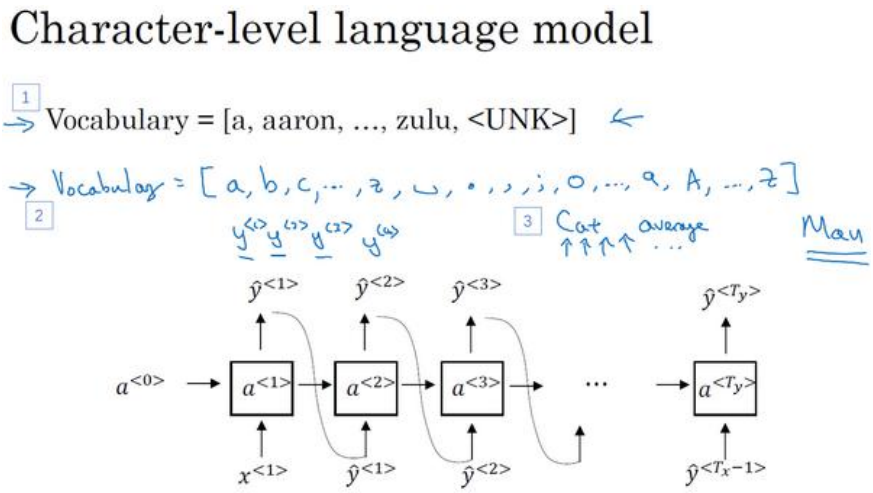
③之后的时间步重复②步的操作，将预测值传递给下一个时间步，对下一个词进行采样，一直到最后一个时间步。

停止采样的标志：

①一直进行采样直到得到 EOS 标识（代表句子结尾的标识）

②设定停止采样的时间步

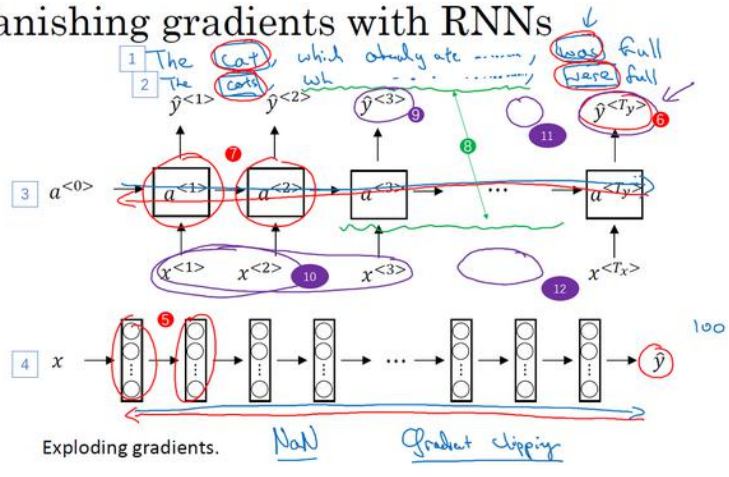
注意：如果采样过程中出现未知的标识（<UNK>），则继续在剩下的词中进行采样，直到得到一个不是未知标识的词。



1.8 循环神经网络的梯度消失

基本的 RNN 算法存在一个梯度消失的问题，本节会讨论这个问题，下几节中会讨论一下解决方法。

假如看到这个句子，“**The cat, which already ate ……, was full.**”，前后应该保持一致，因为 **cat** 是单数，所以应该用 **was**。“**The cats, which ate ……, were full.**”，**cats** 是复数，所以用 **were**。这个例子中的句子有长期的依赖，最前面的单词对句子后面的单词有影响。但是基本的 RNN 模型，不擅长捕获这种长期依赖效应。下面解释一下原因。



基本的 **RNN** 模型会有很多局部影响，意味着这个输出$\hat{y}^{<3>}$主要受$\hat{y}^{<3>}$附近的值的影响，这个区域都很难反

向传播到序列的前面部分，也因此网络很难调整序列前面的计算。这是基本的 **RNN** 算法的一个缺点。

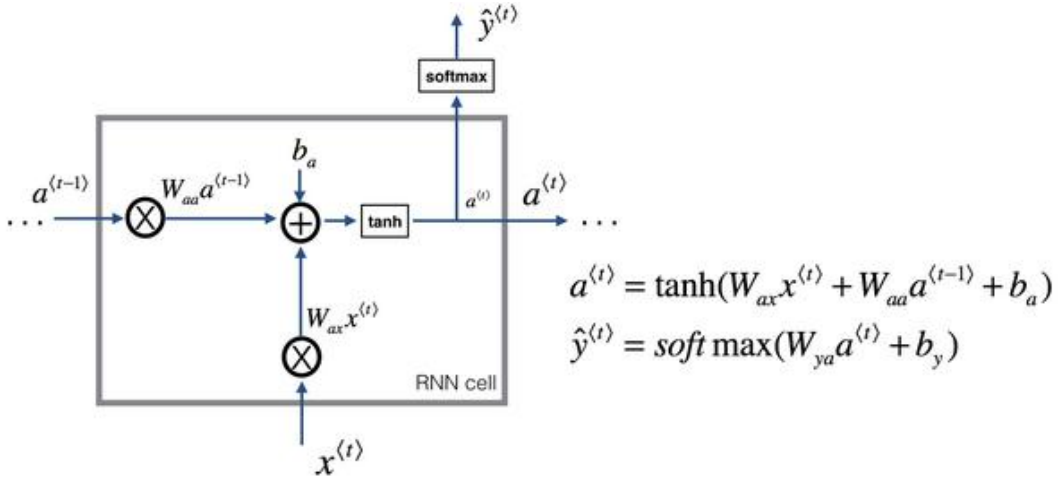
很深的神经网络不仅会导致梯度消失问题，也可能导致梯度爆炸，但是事实上梯度消失在训练 **RNN** 时是首要的问题。由于梯度爆炸可以用梯度修剪的方法解决，而梯度消失更难解决。

一个 **RNN** 处理 1,000 个时间序列的数据集或者 10,000 个时间序列的数据集，这就是一个 1,000 层或者 10,000 层的神经网络，很容易出现梯度消失和梯度爆炸问题，后者可以用梯度修剪解决，下节会介绍 **GRU**，门控循环单元网络，这个网络可以有效地解决梯度消失的问题，并且能够使神经网络捕获更长期的依赖。

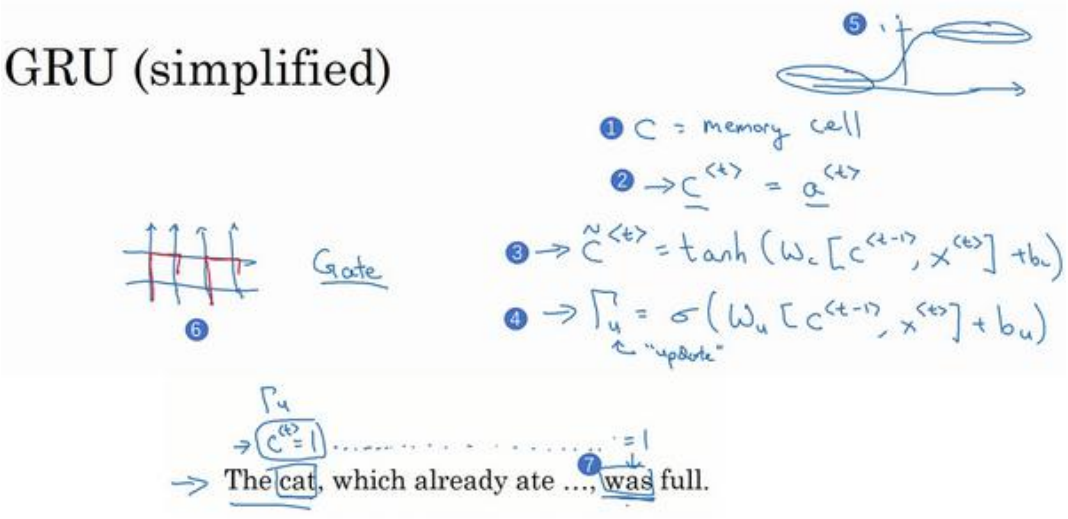
1.9 GRU 单元

本节主要介绍门控循环单元（GRU），它改变了 RNN 的隐藏层，使其可以更好地捕获深层连接，并改善梯度消失问题。

RNN 的隐藏层示意图：



GRU 的想法是引入一个记忆细胞 c，在时间t处，有记忆细胞$c^{<t>}$，$c^{<t>} = a^{<t>}$（LSTM 时这两个值不一样）。



在每个时间步，我们将用一个候选值重写记忆细胞，即$\tilde{c}^{<t>}$的值，所以它就是个候选值，替代了$c^{<t>}$的值。然后我们用 **tanh** 激活函数来计算：

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

GRU 中的重要思想，是有一个更新门，把这个门叫做Γ_u，这个一直在 0 到 1 之间的门值，实际上这个值是把这个式子带入 **sigmoid** 函数得到的：

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

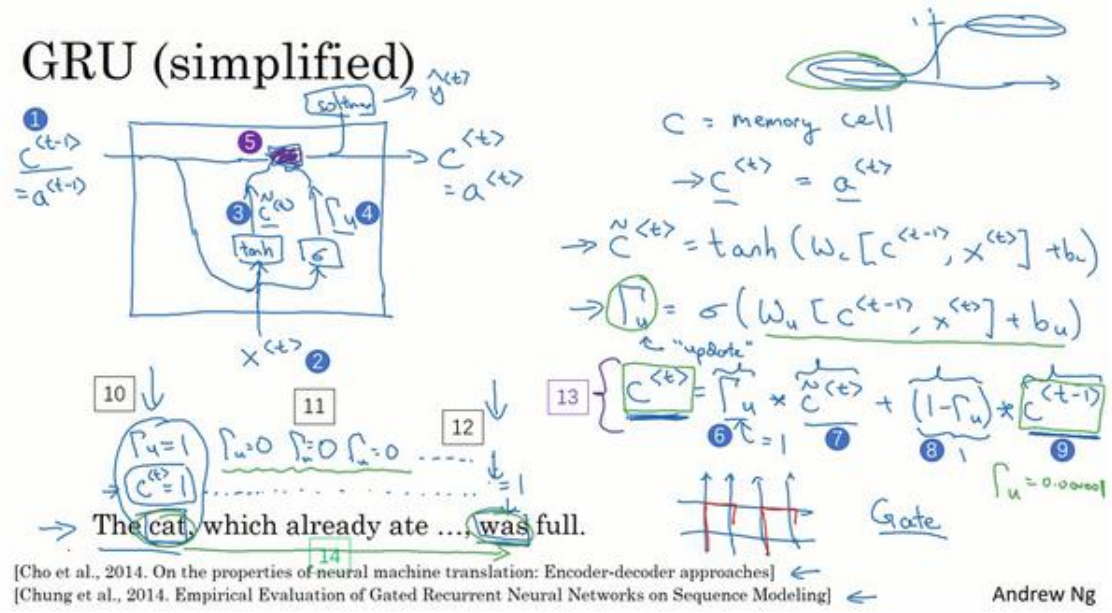
对于大多数可能的输入，**sigmoid** 函数的输出总是非常接近 0 或者非常接近 1。在这样的直觉下，可以想到Γ_u在大多数的情况下非常接近 0 或 1。

GRU 的关键部分在于要不要用$\tilde{c}$更新c。可以这么理解，记忆细胞$c^{<t>}$将被设定为 0 或者 1，这取决于你考虑的单词在句子中是单数还是复数，因为这里是单数情况，所以我们先假定它被设为了 1，或者如果是复数的情况我们就把它设为 0。然后 GRU 单元将会一直记住$c^{<t>}$的值，直到上图编号 7 所示的位置，$c^{<t>}$的值还是 1，这就告诉它，噢，这是单数，所以我们用 **was**。于是门Γ_u的作用就是决定什么时候你会更新这个值，特别是看到词组 **the cat**，即句子的主语猫，这就是一个好时机去更新这个值。然后使用完它的时候，**“The cat, which already ate……, was full.”**，我们不需要记住它了，我们可以忘记它了。

我们用下式得到下一时间步的记忆细胞值：

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

由上式知，如果门值设为 1，即$\Gamma_u = 1$，也就是说把这个新值$c^{<t>}$设为候选值$\tilde{c}^{<t>}$。如果门值为 0，则不更新它，使用旧的值。



当你从左到右扫描这个句子，当门值为 0 的时候，不更新记忆细胞值，就用旧的值，同时记住这个值，一直处理句子到上图编号 4（was）所示，$c^{<t>}$应该会一直等于$c^{<t-1>}$，于是它仍然记得猫是单数的。

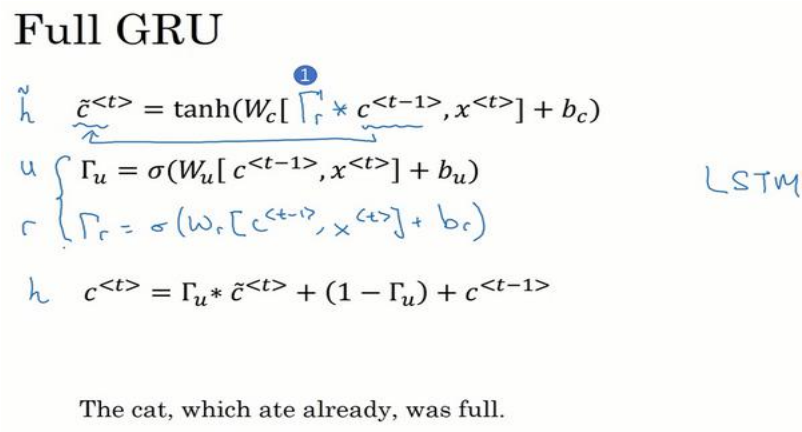
GRU 的优势：

通过门决定是否更新记忆细胞。因为 **sigmoid** 的值，现在因为门很容易取到 0 值，只要这个值是一个很大的负数，再由于数值上的四舍五入，上面这些门大体上就是 0，或者说非常非常非常接近 0。

所以在这样的情况下，这个更新式子（上图编号 13 所示的等式）就会变成$c^{<t>} = c^{<t-1>}$，这非常有利于维持细胞的值，同时也会减缓梯度消失问题（可以参照之前反向传播的课程，涉及到相邻时间步的导数）。因为Γ_u很接近 0，这就是说$c^{<t>}$几乎就等于$c^{<t-1>}$，而且$c^{<t>}$的值也很好地被维持了，即使经过了很多

很多的时间步。这就是缓和梯度消失的关键。因此允许神经网络运行在非常庞大的依赖词上，比如说 cat 和 was 单词即使被中间的很多单词分割开。

完整的 GRU 单元



给记忆细胞的候选值添加一项，添加一个门Γ_r，可以认为 r 代表相关性（relevance）。这个Γ_r门告诉你计算出的下一个$c^{<t>}$的候选值$\tilde{c}^{<t>}$跟$c^{<t-1>}$有多大的相关性。

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

1.10 长短时记忆网络（LSTM）

上一节课中的 GRU：

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

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

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

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

$$a^{<t>} = c^{<t>}$$

对于 **GRU** 我们有$a^{<t>} = c^{<t>}$

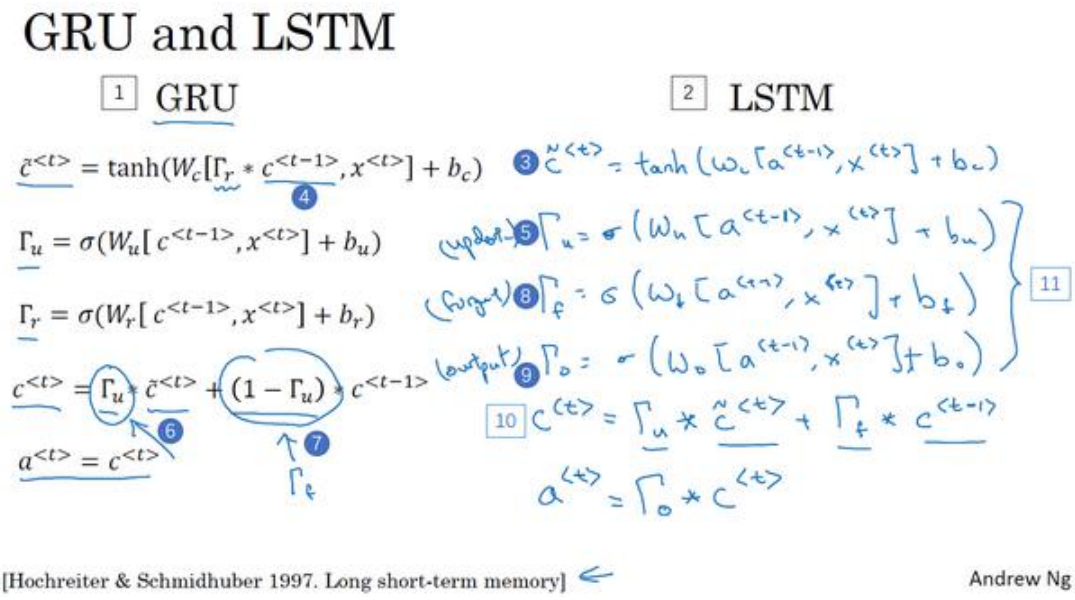
还有两个门:

更新门Γ_u（**the update gate**）

相关门Γ_r（**the relevance gate**）

$\tilde{c}^{<t>}$，这是代替记忆细胞的候选值，然后我们使用更新门Γ_u来决定是否要用$\tilde{c}^{<t>}$ 更新$c^{<t>}$。

LSTM 是一个比 GRU 更加强大和通用的版本。



我们用$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c$来更新候选值$\tilde{c}^{<t>}$，注意，在 **LSTM** 中我们不再有$a^{<t>} = c^{<t>}$的情况，我们专门使用$a^{<t>}$或者$a^{<t-1>}$，而不是用$c^{<t-1>}$，

我们不用Γ_r，即相关门。

保留了更新门，$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$。

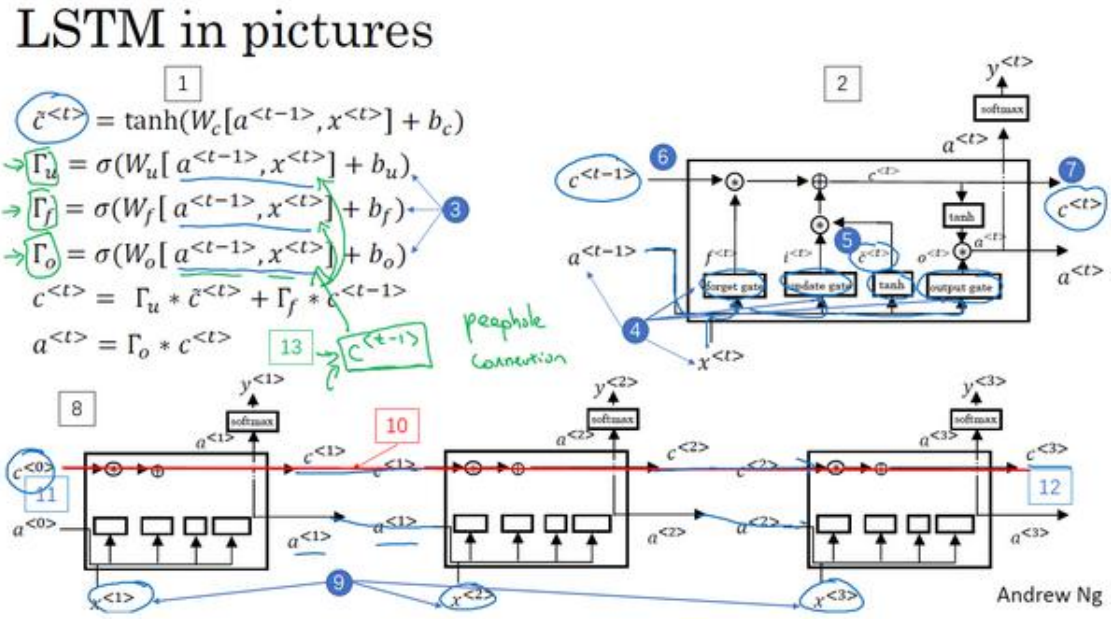
将用不同的项来代替Γ_u和 $1 - \Gamma_u$，分别是Γ_u和Γ_f。

我们设置了遗忘门Γ_f，$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$

LSTM 中有新的输出门，$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] +> b_o)$

最后$a^{<t>} = c^{<t>}$的式子会变成$a^{<t>} = \Gamma_o * c^{<t>}$

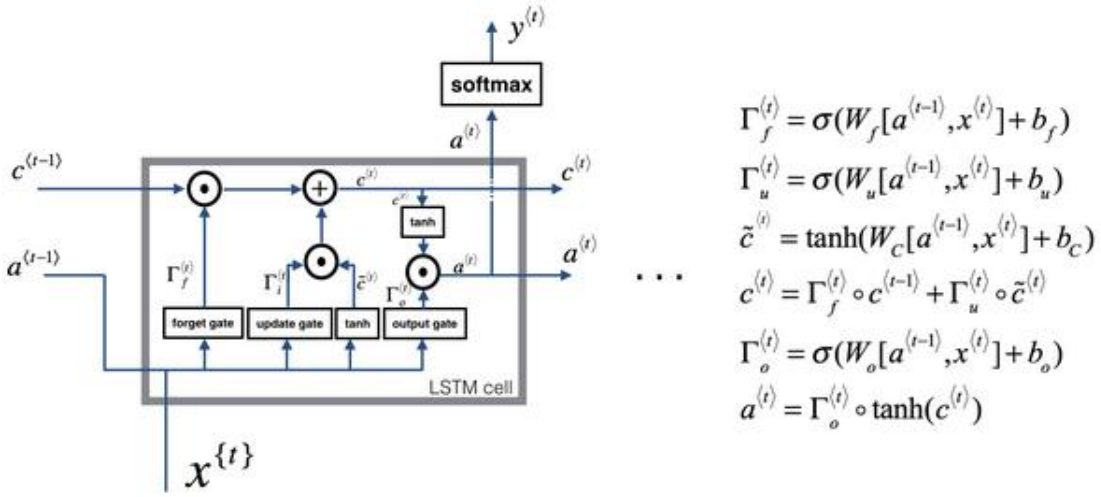
主要区别在于有两个门来控制记忆细胞的更新，以及需要另外用一个门来更新激活值。



注意到上面这里有条线（上图编号 10 所示的线），这条线显示了只要你正确地设置了遗忘门和更新门，**LSTM** 是相当容易把$c^{<0>}$的值（上图编号 11 所示）一直往下传递到右边，比如$c^{<3>} = c^{<0>}$（上图编号 12 所示）。这就是为什么 **LSTM**

和 **GRU** 非常擅长于长时间记忆某个值，对于存在记忆细胞中的某个值，即使经过很长很长的时间步。

LSTM 前向传播图：



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

GRU 与 LSTM 的选择

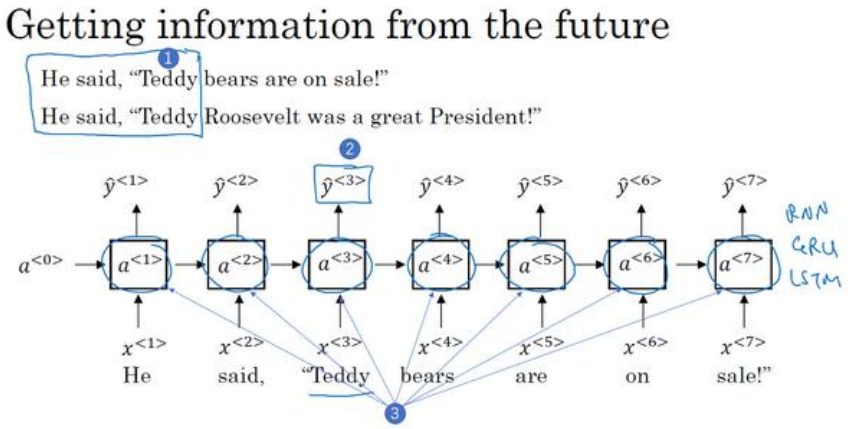
什么时候应该用 **GRU**？什么时候用 **LSTM**？这里没有统一的准则。在深度学习的历史上，**LSTM** 也是更早出现的，而 **GRU** 是最近才发明出来的，它可能源于 **Pavia** 在更加复杂的 **LSTM** 模型中做出的简化。

GRU 的优点是这是个更加简单的模型，所以更容易创建一个更大的网络，而且它只有两个门，在计算性上也运行得更快，然后它可以扩大模型的规模。

但是 **LSTM** 更加强大和灵活，因为它有三个门而不是两个。如果你想选一个使用，**LSTM** 在历史进程上是个更优先的选择，大部分的人还是会把 LSTM 作为默认的选择来尝试。虽然最近几年 GRU 获得了很多支持，而且越来越多的团队也正在使用 GRU，因为它更加简单，而且还效果还不错，它更容易适应规模更加大的问题。

1.11 双向循环神经网络（BRNN）

双向 RNN 的动机

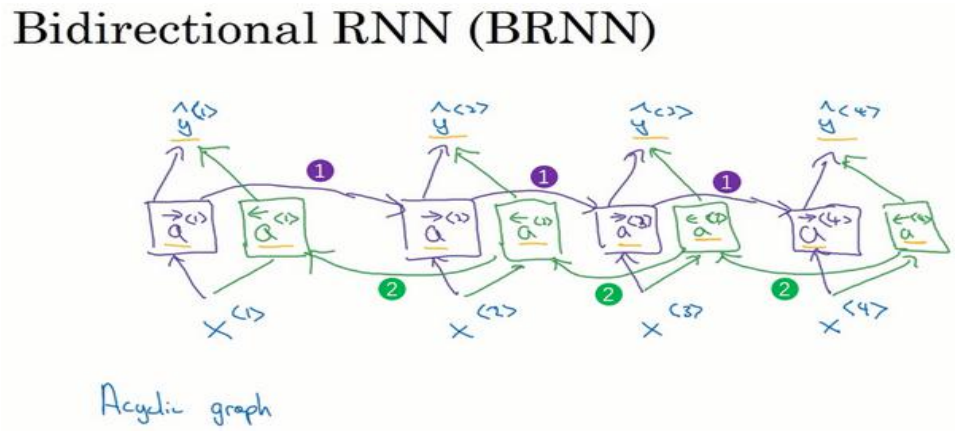


这是一个命名体识别问题，在判断第三个词 **Teddy** 是不是人名的一部分时，

光看句子前面部分是不够的，因为根据前 3 个单词无法判断他们说的是 **Teddy 熊**，还是前美国总统 **Teddy Roosevelt**，所以这是一个非双向的或者说只有前向的 **RNN**。

BRNN 的结构示意

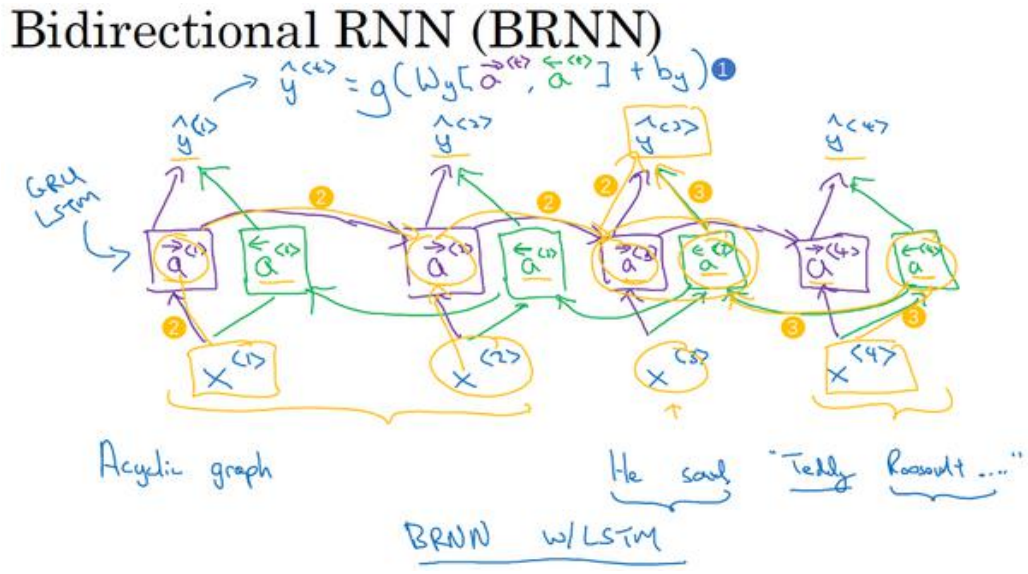
增加一个反向循环层，左箭头代表反向连接，$\overleftarrow{a}^{<1>}$、$\overleftarrow{a}^{<2>}$、$\overleftarrow{a}^{<3>}$、$\overleftarrow{a}^{<4>}$反向连接。



同时，将a的反向连接依次反向向前连接（上图编号 2 所示）。这样，这个网络就构成了一个无环图。给定一个输入序列$x^{<1>}$到$x^{<4>}$，这个序列首先计算前向的$\overrightarrow{a}^{<1>}$，然后计算前向的$\overrightarrow{a}^{<2>}$，接着$\overrightarrow{a}^{<3>}$，$\overrightarrow{a}^{<4>}$。而反向序列从计算$\overleftarrow{a}^{<4>}$开始，反向进行，计算反向的$\overleftarrow{a}^{<3>}$。

计算的是网络激活值，这不是反向而是前向的传播，而图中这个前向传播一部分计算是从左到右，一部分计算是从右到左。把所有这些激活值都计算完了就可以计算预测结果了。

预测结果：



$$\hat{y}^{<t>} = g(W_g[\overrightarrow{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$$

比如你要观察时间 3 这里的预测结果，信息从$x^{<1>}$过来，流经这里，前向的$\overrightarrow{a}^{<1>}$到前向的$\overrightarrow{a}^{<2>}$，这些函数里都有表达，到前向的$\overrightarrow{a}^{<3>}$再到$\hat{y}^{<3>}$（上图②所示路径），所以从$x^{<1>}$，$x^{<2>}$，$x^{<3>}$来的信息都会考虑在内，而从$x^{<4>}$来的信息

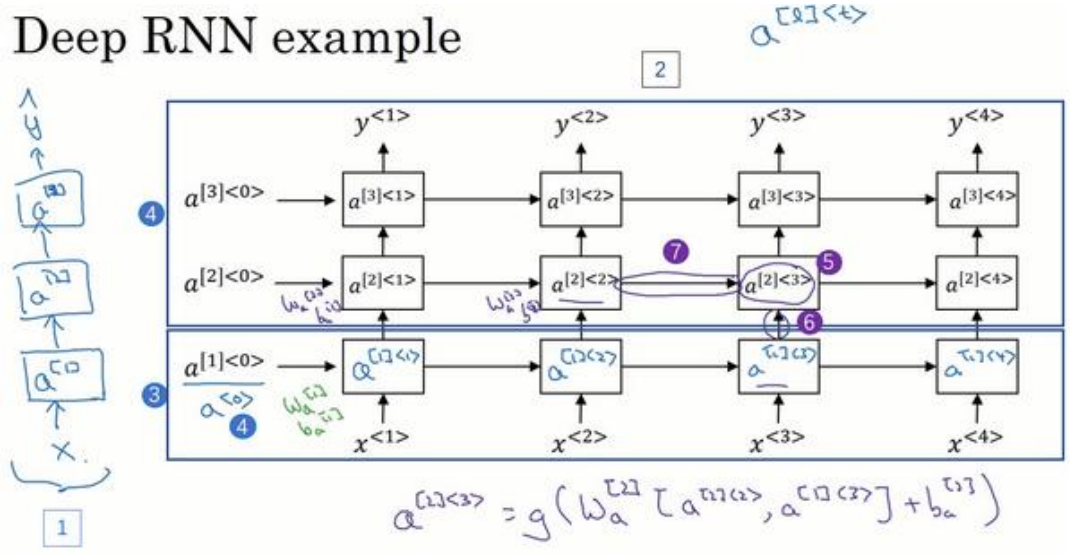
会流过反向的$\overleftarrow{a}^{<4>}$，到反向的$\overleftarrow{a}^{<3>}$再到$\hat{y}^{<3>}$（上图③所示路径）。

双向 RNN 网络模型的缺点就是你需要完整的数据的序列，才能预测任意位置。对于很多自然语言处理的应用，如果总是可以获取整个句子，这个标准的双向 RNN 算法实际上很高效。

1.12 深层循环神经网络（Deep RNNs）

通常我们会把 RNN 的多个层堆叠在一起构建更深的模型。本节中介绍如何构建这些更深的 RNN。

下图所示三层 RNN 堆叠的网络。

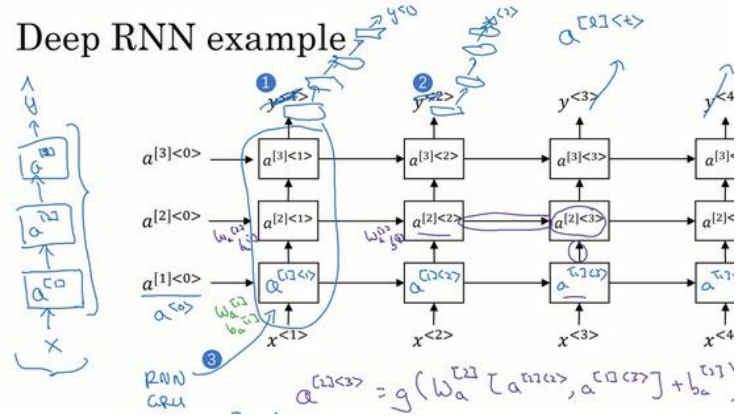


我们现在用$a^{[l]<t>}$来表示第 l 层的激活值，这个<t>表示第t个时间步。

我们下面看一下激活值$a^{[2]<3>}$是怎么计算的，$a^{[2]<3>}$有两个输入，一个是下一层对应时间步的激活值，一个是从左边过来的激活值作为输入。

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

每一层都有自己的参数$W_a^{[l]}$和$b_a^{[l]}$



很深的网络，甚至有 100 层深，而对于 **RNN** 来说，有三层就已经不少了。

由于时间的维度，**RNN** 网络会变得相当大，即使只有很少的几层，很少会看到这种网络堆叠到 100 层。但有一种组合的网络可能很常见，就是在每一个上面堆叠循环层，把这里的输出去掉（上图编号 1 所示），然后换成一些深的层，这些层并不水平连接，只是一个深层的网络，然后用来预测$y^{<1>}$。

这些单元也不一定是标准的 RNN，也可以是 GRU 单元或者 LSTM 单元，也可以构建深层双向 RNN 网络。

由于深层 RNN 训练需要很多计算资源，所以深层的循环层不多，不像卷积神经网络一样有大量的隐含层。

第二周 自然语言处理与词嵌入

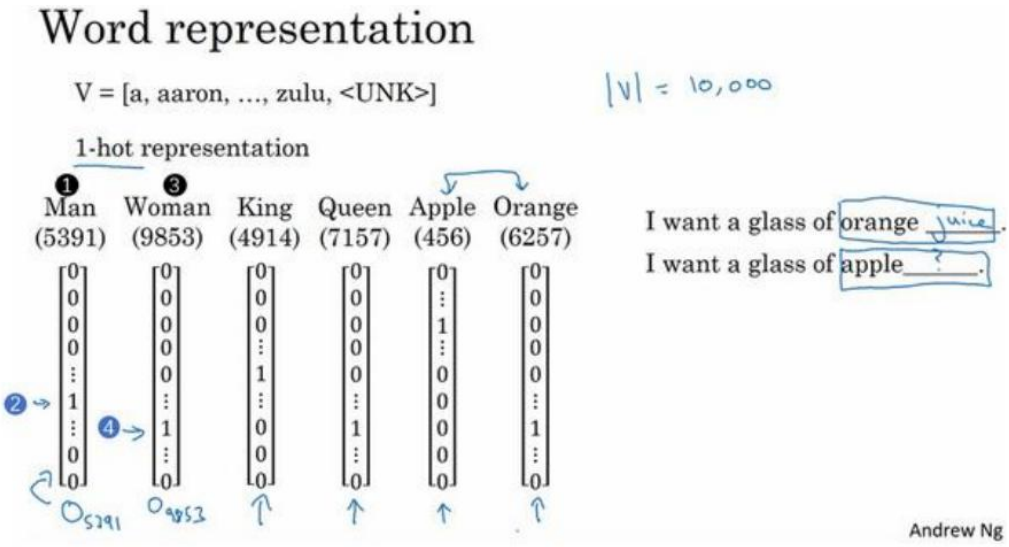
2.1 词汇表征

本节主要引入自然语言处理中的一个重要概念——词嵌入，阐述了为什么要学习或者使用词嵌入的原因。

• One-hot 向量表示：存在局限性

把每个词孤立起来，使得算法对相关词的泛化能力不强。

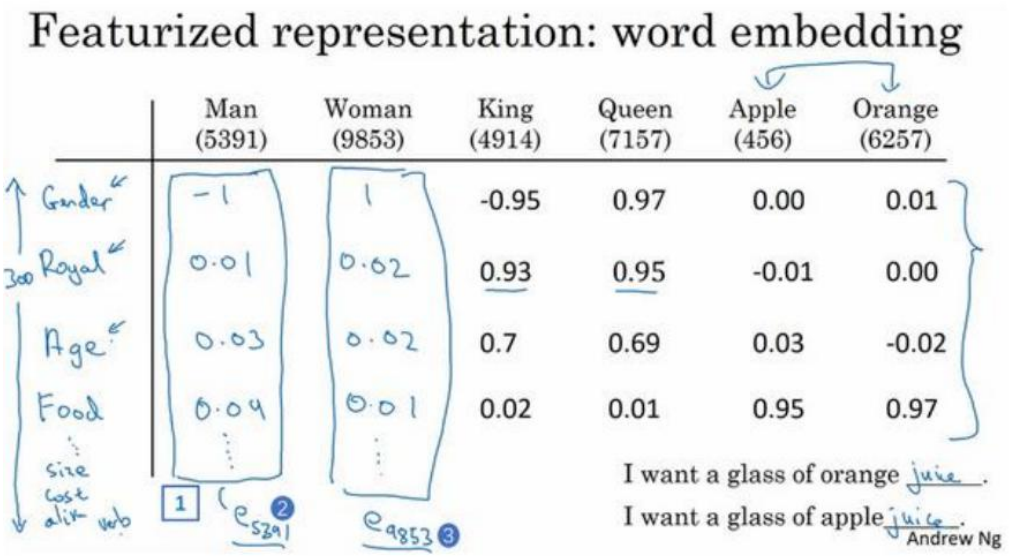
比如算法不能从 orange juice 学习到 apple juice，词向量之间正交，无法体现相似性，所以泛化能力不强。



• 词嵌入：特征化的表示

用不同维度去刻画词的不同特征，体现词与词之间的相似性和差异性。

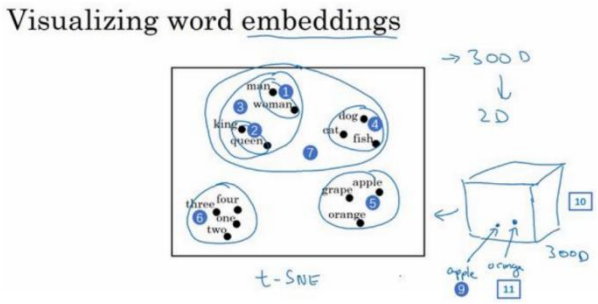
比如 Apple 和 Orange 在 Food 等多个维度上具有很大的相似性，同时又在少部分颜色或者大小维度上有所区别，既可以区分两者，又有利于算法更好的泛化。



• 可视化词嵌入表示

如果我们能够学习到一个 300 维的特征向量（或者说 300 维的词嵌入），我们可以把这 300 维的数据嵌入到一个二维空间里，从而实现可视化。

常用的可视化算法是 t-SNE 算法。通过可视化的观察，可以发现 man 和 woman 这些词聚集在一块，king 和 queen 聚集在一块，这四个词都表示人，也都聚集在一起。动物都聚集在一起，水果也都聚集在一，像 1、2、3、4 这些数字也聚集在一起。如果把这些生物看成一个整体，他们也聚集在一起。



• 词嵌入概念的解释

比如 300 维上的词嵌入，即 300 维空间里的特征表示，取一个单词比如 orange，它对应一个 300 维的特征向量，所以这个词就被嵌在这个 300 维空间里的一个点上了，而 apple 这个词就被嵌在这个 300 维空间的另一个点上了，所以这是词嵌入概念的解释。为了可视化，t-SNE 算法把这个空间映射到低维空间，从而可以观察到词之间的关联性。

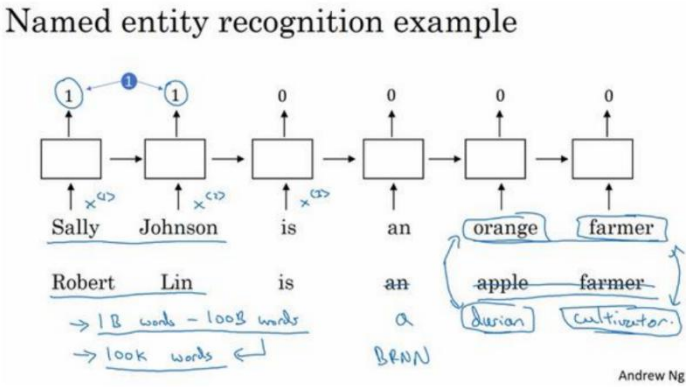
2.2 使用词嵌入

上节讲述了不同单词的特征化表示，本节主要内容是如何把词嵌入表示应用到 NLP 任务中。

以命名体识别为例：

Sally Johnson is an orange farmer.这句话中， 根据 orange farmer 可以确定 Sally Johnson 是一个人名而不是一个公司名。

用词嵌入作为输入训练好的模型，如果你看到一个新的输入：“Robert Lin is an apple farmer.”因为学习到 orange 和 apple 很相近，那么算法很容易就知道 Robert Lin 也是一个人。



要是看到：“Robert Lin is a durian cultivator.”（Robert Lin 是一个榴莲培育家），由于 durian（榴莲）是水果，就像 orange（橙子）一样，并且 cultivator（培育家），做培育工作的人其实跟 farmer（农民）差不多，容易归纳出“a durian cultivator”（榴莲培育家）也是一个人。

总结一下，用词嵌入做迁移学习的步骤：

第一步，先从大量的文本集中学习词嵌入。一个非常大的文本集，或者可以下载网上预训练好的词嵌入模型。（1-100B words）

第二步，把词嵌入模型迁移到新的只有少量标注训练集的任务中。运用词嵌入可以将稀疏的向量变得更加密集，且增加相关性。（100k words）

第三步，选择是否进行进一步的微调，用新的数据调整词嵌入。

实际中，只有第二步中有很大的数据集才会继续微调，如果第二步中标记的数据集不是很大，通常不会在微调词嵌入上费力气。

当任务的训练集相对较小时，词嵌入的作用最明显。词嵌入广泛用于 NLP 领域，如命名实体识别，用在文本摘要，用在文本解析、指代消解。词嵌入在语言模型、机器翻译领域用的少一些，由于这些任务通常有大量的数据。

• 迁移学习的特点：

从某一任务 A 迁移到某个任务 B，只有 A 中有大量数据，而 B 中数据少时，迁移的过程才有用。

• 词嵌入与人脸编码的区别：

人脸识别：人脸识别算法涉及到海量的人脸照片，训练一个网络，任给一个人脸照片，甚至是没有见过的照片，神经网络都会计算出相应的一个编码结果。

词嵌入：则是有一个固定的词汇表，比如 10000 个单词，我们学习向量e_1到e_{10000}，学习一个固定的编码，每一个词汇表的单词的固定嵌入。

2.3 词嵌入的特性

上节介绍了词嵌入是如何应用在自然语言处理中，本节主要介绍词嵌入可以实现类比推理的特性。

Analogies

	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

用四维向量e_{man}、e_{woman}分别来表示 man 和 woman 这两个单词，对 king 和 queen 采用一样的表示方法。由于本例中假设的词向量维度较少，所以可以比较直观地发现词向量作差之后的关联特性，如下所示。

用$e_{man} - e_{woman}$得到：

$$e_{\text{man}} - e_{\text{woman}} = \begin{bmatrix} -1 \\ 0.01 \\ 0.03 \\ 0.09 \end{bmatrix} - \begin{bmatrix} 1 \\ 0.02 \\ 0.02 \\ 0.01 \end{bmatrix} = \begin{bmatrix} -2 \\ -0.01 \\ 0.01 \\ 0.08 \end{bmatrix} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

用$e_{king} - e_{queen}$得到：

$$e_{\text{king}} - e_{\text{queen}} = \begin{bmatrix} -0.95 \\ 0.93 \\ 0.70 \\ 0.02 \end{bmatrix} - \begin{bmatrix} 0.97 \\ 0.95 \\ 0.69 \\ 0.01 \end{bmatrix} = \begin{bmatrix} -1.92 \\ -0.02 \\ 0.01 \\ 0.01 \end{bmatrix} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

这个结果表示，man 和 woman 主要的差异是 gender（性别）上的差异，而 king 和 queen 之间的主要差异，根据向量的表示，也是 gender（性别）上的差异。这也就是为什么$e_{man} - e_{woman}$与$e_{king} - e_{queen}$得到的结果是相同的。

• 所以得出这种类比推理结论的方法是：

当算法被问及 man 对 woman 相当于 king 对什么时，算法所做的就是计算$e_{man} - e_{woman}$，然后找出一个向量，使得$e_{man} - e_{woman} \approx e_{king} - e_{?}$，当这个新词是 queen 时，式子左右两边会近似相等。

• 上述类比推理的数学推导：

找到单词 w 来使得，$e_{man} - e_{woman} \approx e_{king} - e_{?}$成立，则需要找到单词 w 来最大化$e_{king} - e_{man} + e_{woman}$的相似度，即

$$\text{Find word w: argmax Sim}(e_w, e_{king} - e_{man} + e_{woman})$$

• 我们需要一些用于测算e_w和$e_{king} - e_{man} + e_{woman}$之间的相似度的函数：

①余弦相似度：

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

用向量夹角的余弦值衡量相似性。

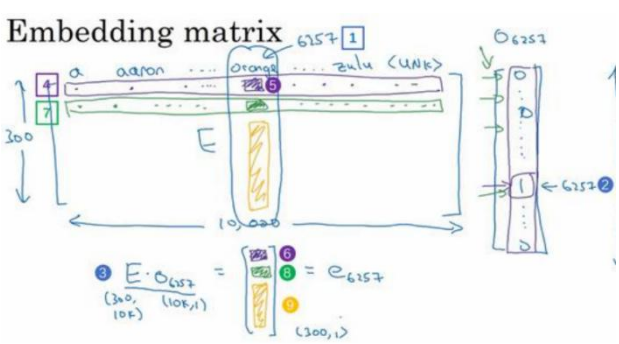
②欧式距离表示相似度：$\|u - v\|^2$

需要对此式取负，才能满足向量越相似，该式的值越大。

2.4 嵌入矩阵

本节介绍如何将学习词嵌入这一问题具体化。当我们运用算法来学习词嵌入时，实际上是学习一个嵌入矩阵。

假设词汇表里有 10000 个单词，词汇表里有 a，aaron，orange，zulu 等，可能还有一个未知词标记<UNK>我们要做的就是学习一个嵌入矩阵，它将是一个 300×10000（$|E| \times |V|$）的矩阵，将词表中单词的 one-hot 向量进行降维特征表示。



假设 orange 的单词编号是 6257，代表词汇表中第 6257 个单

词，我们用符号O_{6257}来表示这个 one-hot 向量，假设这个嵌入矩阵叫做矩阵 E

$$E \cdot O_{6257} = e_{6257}$$

e_{6257}为表示 orange 的嵌入向量。

更一般地，$E \cdot O_j = e_j$，e_j表示的是字典中单词 j 的嵌入向量。

词嵌入操作在具体实现时，由于大量矩阵和向量相乘来计算它，同时 one-hot 为矩阵维度很高的稀疏矩阵，乘法运算效率很低下；于是实践中会使用一个专门的函数来单独查找矩阵E的某列，而不是用通常的矩阵乘法来实现。

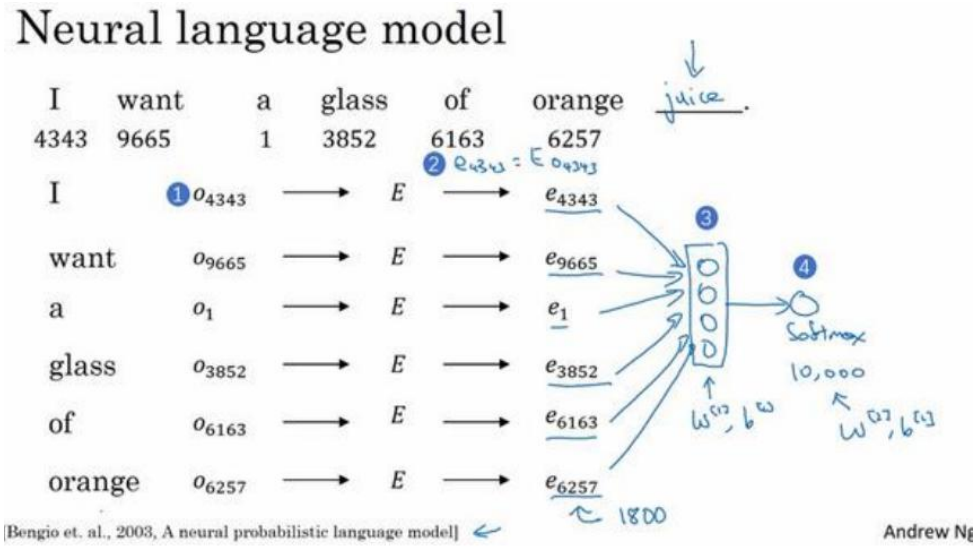
我们的算法的目标是学习一个嵌入矩阵 E，下一节中会说明这一步骤，首先随机初始化矩阵 E，然后用梯度下降法来学习这个矩阵中的各个参数。

2.5 学习词嵌入

本节中主要讲述一些具体的学习词嵌入的算法。

假设我们在构建一个语言模型，并且用神经网络来实现这个模型。在训练中，我们希望神经网络能通过输入“I want a glass of orange ___.”预测出句子中的下一个词。实践证明，建立一个语言模型是学习词嵌入的好方法。

下面将介绍如何建立神经网络来预测序列中的下一个单词：



①对每个单词用 one-hot 向量表示

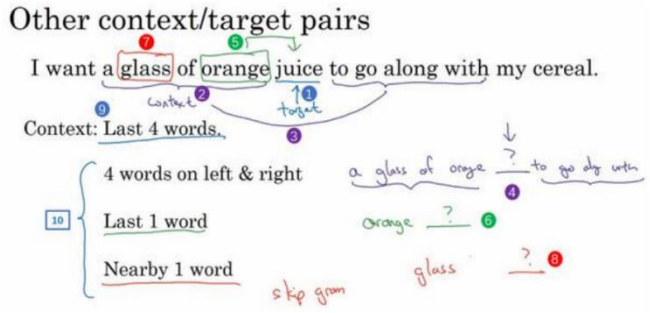
②生成参数矩阵 E，利用$E \cdot O_j = e_j$得到每个词的嵌入向量表示。

③将嵌入向量作为神经网络的输入，经过神经网络进行处理

④经过 Softmax 层，预测各个单词可能出现的概率。

更常见的算法是利用一个固定的历史窗口，比如预测给定四个单词后的下一个单词，这里的 4 是算法的超参数。如果算法的目标是学习一个嵌入矩阵，研究人员已经尝试过很多不同类型的上下文。如果要建立一个语言模型，那么一般选取目标词之前的几个词作为上下文。但如果目标不是学习语言模型本身的话，那么可以选择其他的上下文。

可以选择的上下文有：
①前/后 n 个单词
②目标词前一个词
（其实为①中的特例）
③目标词邻近一个词



2.6 Word2Vec

上节内容介绍了如何用学习一个神经语言模型来得到较好的词嵌入矩阵。本节主要内容是 Word2Vec 算法，一种简单且计算更加高效的方式来学习词嵌入。

• Skip-Gram 模型：

假设在训练集中给定了一个这样的句子：“I want a glass of orange juice to go along with my cereal.”我们要做的是抽取上下文和目标词配对，来构造一个监督学习问题。

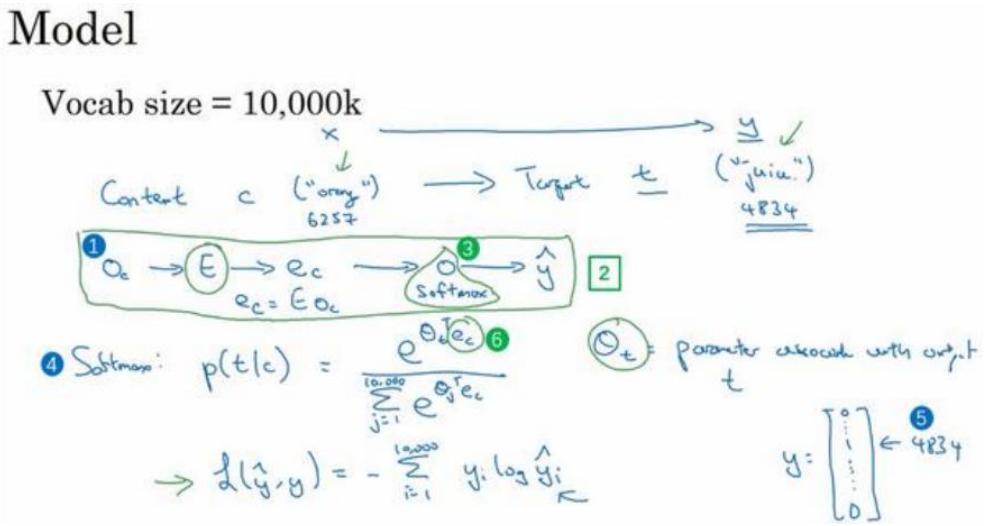
上下文和目标词的选择：

上下文不一定总是目标单词之前离得最近的四个单词，或最近的n个单词。

我们要的做的是随机选一个词作为上下文词，然后我们要做的是随机在一定词距内选另一个词，比如在上下文词前后 5 个词内或者前后 10 个词内，我们就在这个范围内选择目标词。

比如选 orange 作为上下文词，在 orange 前后一定词距范围内选择目标词，比如选择 juice 或 glass 或 my 等作为目标词。

模型要解决的基本的监督学习问题是学习一种映射关系。比如选择上下文词为 orange，记为 c，目标词为 juice，记为 t。模型需要学习的从输入 c 到输出 t 的映射。



①one-hot 向量表示输入：用 one-hot 向量表示上下文词 orange，记作O_c。

②得到输入上下文词的嵌入向量：$e_c = E \cdot O_c$

③把e_c喂给 softmax 层，预测不同目标词的概率。

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

这里θ_t是一个与输出 t 有关的参数，即某个词t和标签相符的概率是多少。上式中省略了 softmax 中的偏差项，可以添加上。

我们用 y 表示目标词，$\hat{y}$为预测词，于是损失函数为：

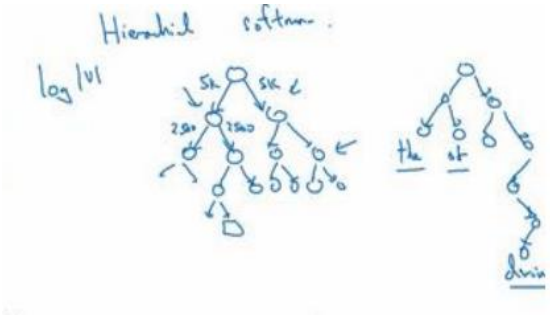
$$L(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i$$

• 算法存在的问题：计算速度很慢。Softmax 层中需要对词表中所有词做求和计算，分母求和操作相当慢。

• 解决方案：①分级的 softmax 分类器 ②负采样（下一节介绍）

• 分级的 softmax 分类器：

思想：不是一开始就确定预测词属于 10,000 类中的哪一类，第一个分类器告诉我们目标词是在词汇表的前 5000 个中还是在词汇表的后 5000 个词中；然后第二个分类器会告诉我们这个词在词汇表的前 2500 个词中，或者在词汇表的第二组 2500 个词中，以此类推，直到最终你找到一个词准确所在的分类器，那么就是这棵树的一个叶子节点。



这样是一个树形的分类器，意味着树上内部的每一个节点都可以是一个二分类器，比如逻辑回归分类器，所以你不需要再为单次分类，对词汇表中所有的 10,000 个词求和了。使用这种分类树，计算成本与词汇表大小的对数成正比。

实际上，分级的 softmax 分类器会被构造成常用词在树的根部附近，然而不常用的词像 durian 会在树的更深处（有点像哈夫曼树的思想）。

• 如何对上下文 c 采样：

一旦对上下文 c 进行采样，那么目标词 t 就会在上下文 c 的正负若干个词距内进行采样。但是如何选择上下文 c？

其中一种选择是对语料库均匀且随机地采样，但这么做的缺点是有一些词，像 the、of、a、and、to 诸如此类是出现得相当频繁的，但是其他词，像 orange、apple 或 durian 就不会那么频繁地出现了，于是采样得到的训练集都是出现得很频繁的词，因为这会导致我们花大部分的力气来更新这些频繁出现的单词的e_c，但我们想要的是花时间更新像 durian 这些更少出现的词的嵌入。

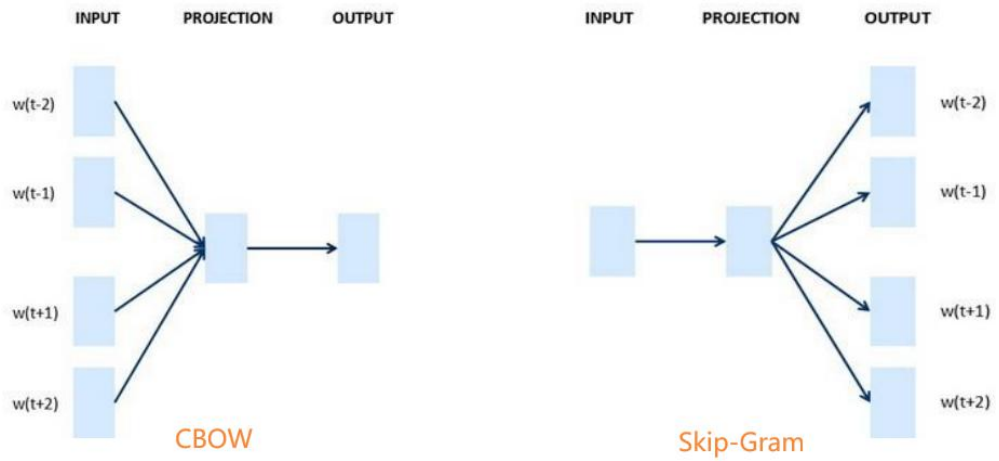
实际上词$p(c)$的分布并不是单纯的在训练集语料库上均匀且随机的采样得到的，而是采用了不同的分级来平衡更常见的词和不那么常见的词。

以上介绍的是 Word2Vec 的 Skip-Gram 模型，还有一种版本的 Word2Vec 模型。

• CBOW 模型：

连续词袋模型（Continuous Bag-Of-Words Model），与 Skip-Gram 的输入和输出相反，CBOW 获得中心词两边的的上下文，然后用周围的词去预测中间的词。

CBOW 是从原始语句推测目标字词，Skip-Gram 正好相反，是从目标字词推测出原始语句。

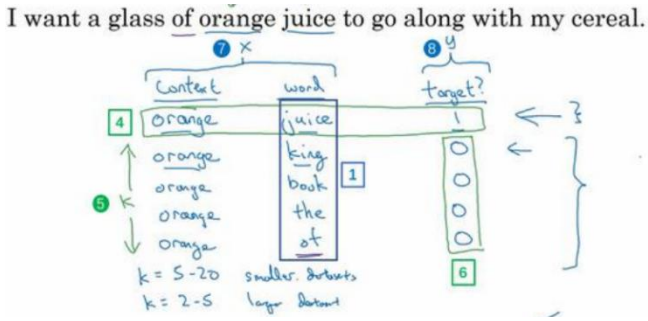


CBOW 对小型数据库比较合适，而 Skip-Gram 在大型语料中表现更好。通常情况下，Skip-Gram 模型用到更多一些。

2.7 负采样

上节中学习了 Skip-Gram 模型如何完成一个监督学习任务，把上下文映射到了目标词上，并学习到一个实用的词嵌入。但是缺点在于 softmax 计算很慢。本节中会学习到一个解决方案：负采样。

这个算法中要做的是构造一个监督学习问题，给定一对单词，比如 orange 和 juice，我们要去预测这是否是一对上下文词-目标词。这个例子中，orange 和 juice 是一个正样本，我们用 1 标记；orange 和 king 就是一个负样本，我们用 0 标记。



• 正负样本的选取：

正样本的选择：先抽取一个上下文词，在一定词距内比如说正负 10 个词距内选一个目标词，并给定标签为 1。

负样本的选择：用相同的上下文词，从字典中选取随机的词，king、book、the、of 等，并给定标签为 0。（负样本选取 K 次）

以上是训练集的生成方法，这个算法就是要分辨这两种不同的采样方式。

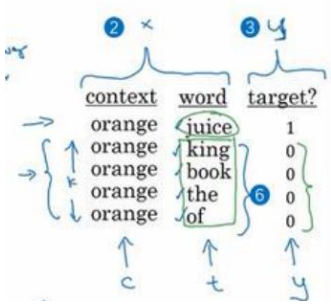
• K 的选取：

小数据集：K 取 5 到 20 比较好

大数据集：K 选取的小一些，2 到 5 比较好

数据集越小，K就越大

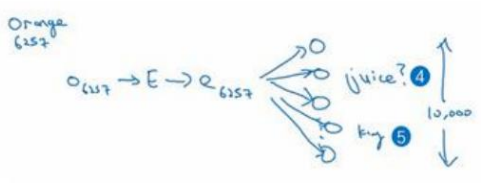
• 模型说明：



为了定义模型，我们将使用记号c表示上下文词，记号t表示可能的目标词，用y表示 0 和 1，表示是否是一对上下文-目标词。我们要做的就是定义一个逻辑回归模型，给定输入的c，t对的条件下，y=1 的概率，即：

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

对每一个可能的目标词有一个参数向量θ_t和另一个参数向量e_c，即每一个可能上下文词的的嵌入向量，我们将用这个公式估计y =1 的概率。如果有K+1 个样本，可以把这个看作 1:K 的正负样本比例，即每一个正样本你都有K个对应的负样本来训练一个类似逻辑回归的模型。



将上述过程从神经网络的角度理解，如果果输入词是 orange，即O_{6257}，要做的就是输入 one-hot 向量，再传递给E，通过两者相乘获得嵌入向量e_{6257}，于是得到了 10000 个可能的二分类问题，但是每次迭代只训练其中 5 个分类器（K=4 时），比如其中一个分类器将用来预测 juice 是否是目标词，一个分类器将用来预测 king 是否是目标词，等等。

这也是为什么这个算法计算成本更低，因为只需更新K+1 个逻辑单元，K+1 个二分类问题，相对而言每次迭代的成本比更新 10,000 维的 softmax 分类器成本低。

• 如何选取负样本（更具体的说明）：即在选取了上下文词 orange 之后，如何对这些词进行采样生成负样本？

一个办法是根据候选目标词在语料中的经验频率进行采样，就是通过词出现的频率对其进行采样。但问题是这会导致你在 like、the、of、and 诸如此类的词上有很高的频率。另一个极端就是用 1 除以词汇表总词数，即$\frac{1}{|v|}$，均匀且随机地抽取负样本，这对于英文单词的分布是非常没有代表性的。

所以 Mikolov 等人根据经验，发现这个经验值的效果最好，它位于这两个极端的采样方法之间，既不用经验频率，也就是实际观察到的英文文本的分布，也不用均匀分布，如下式所示：

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

其中 $f(w_i)$是观测到的在语料库中的某个英文词的词频。

事实上经验频率为指数=1 的情形，均匀分布为指数=0 的情形，这里指数取$\frac{3}{4}$位于这两种极端情形之间，课程中没有论证其数学上的理论成立性，但是在实际过程中该方法效果不错。

2.8 GloVe 词向量

前几节中已经介绍了几个计算词嵌入的算法，另一个在 NLP 领域中有一定势头的算法是 GloVe 算法，这个算法并不如 Word2Vec 或是 Skip-Gram 模型用的多，但是也有人热衷于它，可能是因为它简便。

GloVe 算法（global vectors for word representation），即用词表示的全局变量。在之前的算法中，通过挑选语料库中位置相近的两个词，列举出词对，即上下文和目标词，来进行监督学习。而 GloVe 算法做的就是使其关系开始明确化，假定X_{ij}是单词 i 在单词 j 上下文中出现的次数，可以认为X_{ij}等同于X_{ct}；可

以遍历训练集得到单词 i 在不同单词 j，如果将上下文和目标词的范围定义为出现于左右各 10 词以内的话，那么就会有一种对称关系$X_{ij} = X_{ji}$。对于 GloVe 算法，我们可以定义上下文和目标词为任意两个位置相近的单词，假设是左右各 10 词的距离，那么X_{ij}就是一个能够获取单词i和单词j出现位置相近时或是彼此接近的频率的计数器。

GloVe 模型做的就是进行优化，将训练词向量间的相似度与真实词对频率之间的差距进行最小化处理：

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

对于上式的三点说明：

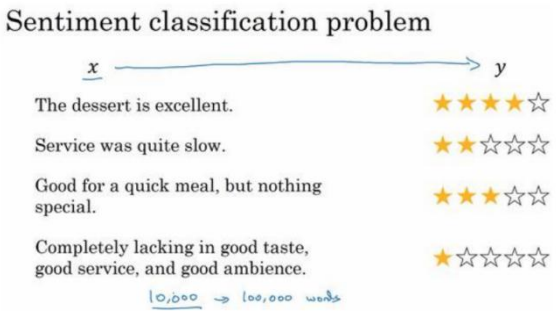
• 如果$X_{ij} = 0$，那么$log0$ 就是未定义的，是负无穷大的，所以我们想要对X_{ij}为 0 时进行求和，因此需要添加一个额外的加权项$f(Xij)$。如果Xij等于 0 的话，同时我们会用一个约定，即 $0log0 = 0$，这个的意思是如果$Xij = 0$，先不要进行求和，所以这个$log0$ 项就是不相关项。

• $f(Xij)$另一个作用是作为加权函数，不给像 this，is，of，a 这样在英语里出现更频繁的词过分的权重，也不给不常用词（durion）太小的权值。

• θ_t和e_j是完全对称的，因此一种训练算法的方法是一致地初始化θ和e，然后使用梯度下降来最小化输出，当每个词都处理完之后取平均值。所以，给定一个词w，就会得到$e_w^{(final)} = \frac{e_w + \theta_w}{2}$。

2.9 情感分类

情感分类任务就是看一段文本，然后分辨这个人是否喜欢他们在讨论的这个东西，这是 NLP 中最重要的模块之一。情感分类一个最大的挑战就是标记的训练集没有那么多，但是有了词嵌入，即使只有中等大小的标记的训练集，也能构建一个不错的情感分类器。



对于情感分类任务来说，训练集大小从 10,000 到 100,000 个单词都很常见，甚至有时会小于 10,000 个单词，采用了词嵌入能够带来更好的效果，尤其是只有很小的训练集时。

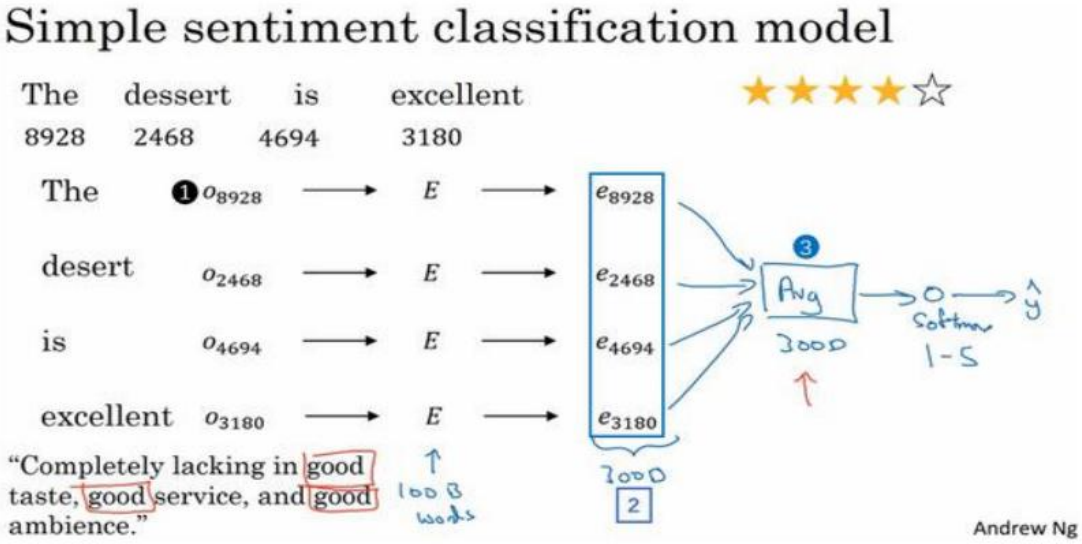
• 简单的情感分类的模型

假设有一个句子"The dessert is excellent"，然后在词典里找这些词，我们通常用 10,000 个词的词汇表。

①取出这四个词，构造对应的 one-hot 向量；

②乘以嵌入矩阵 E，E 可以从一个很大的文本集中学习到，比如它可以从一亿个词或者一百亿个词里学习嵌入，然后得到嵌入向量 e。

③取这些嵌入向量，求和或者取平均，得到一个 300 维的特征向量。
④把这个特征向量送到 softmax 分类器，输出 5 个可能的预测值。



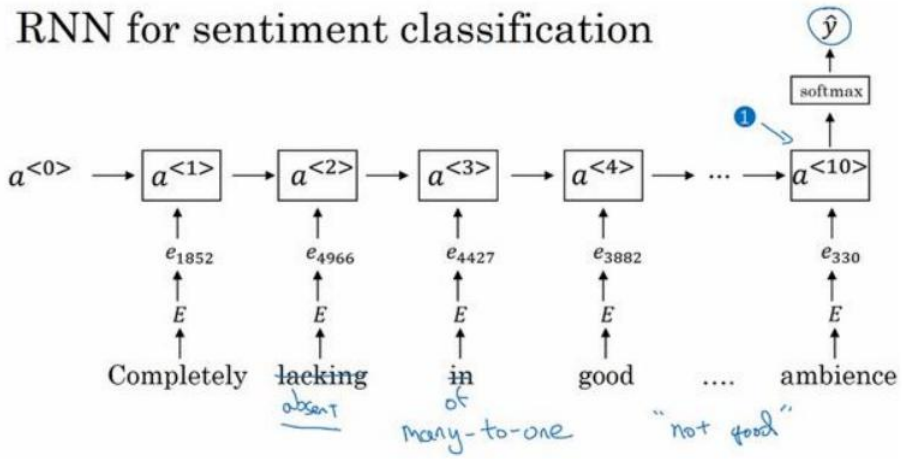
这里用的平均值运算单元，这个算法适用于任何长短的评论，因为即使评论是 100 个词长，你也可以对这一百个词的特征向量求和或者平均它们，然后得到一个表示一个 300 维的特征向量表示，然后把它送进 softmax 分类器。它实际上会把输入中所有单词的意思给平均起来，或者把所有单词的意思加起来。

算法存在的问题：没考虑词序。

比如这样一个负面的评价，"Completely lacking in good taste, good service, and good ambiance."但是 good 这个词出现了很多次，有 3 个 good，如果算法忽略词序，仅仅把所有单词的词嵌入加起来或者平均下来，最后的特征向量会有很多 good 的表示，分类器很可能认为这是一个好的评论，尽管事实上这是一个差评，只有一星的评价。

• RNN 来做情感分类

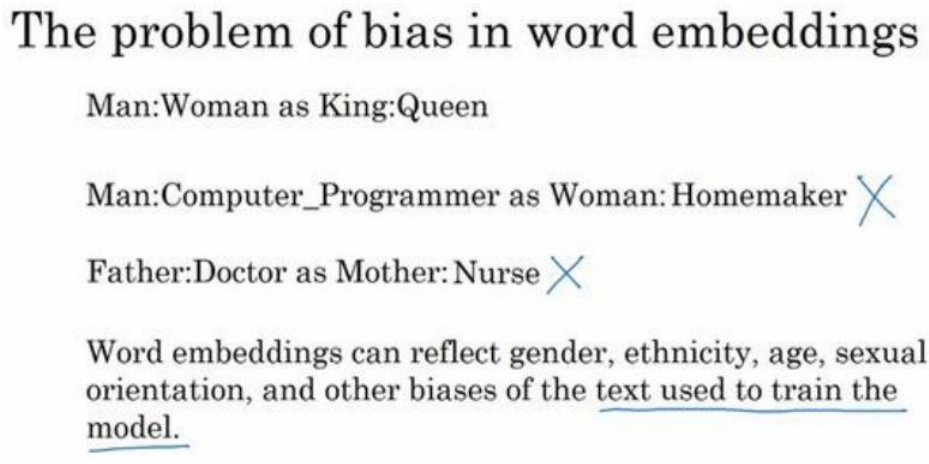
①找出输入评论中每个词的 one-hot 向量。
②用每个 one-hot 向量乘以嵌入矩阵 E，得到词嵌入 e。
③将词嵌入按序送进 RNN 中，RNN 最后一步对结果进行预测。



用 RNN 做情感分类考虑了词的顺序。

2.10 词嵌入除偏

本节主要展示词嵌入中一些相关方法，来减少或消除非预期形式偏见影响，比如说性别歧视、种族歧视。



• 词嵌入中存在的问题：

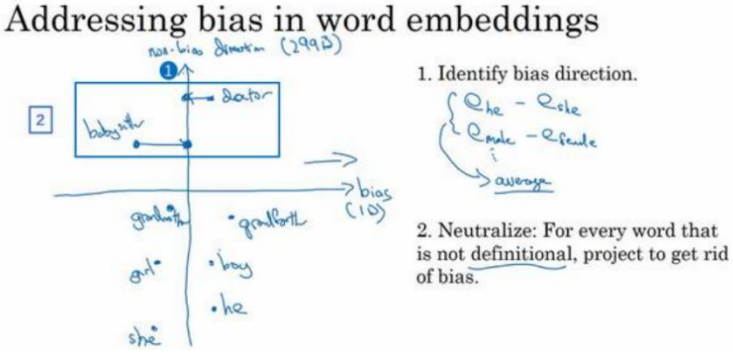
一个已经完成学习的词嵌入，如果你这样问，如果 Man 对应 Computer Programmer，那么 Woman 会对应什么呢？一些研究人员发现了一个十分可怕的结果，已经完成学习的词嵌入可能会通过 Man：Computer Programmer，输出 Woman：Homemaker，这体现了一个十分不良性的性别歧视。

因此根据训练模型所使用的文本，词嵌入能够反映出性别、种族、年龄、性取向等其他方面的偏见，这些偏见都和社会经济状态相关。但因为机器学习算法正用来制定十分重要的决策，它也影响着世间万物，从大学录取到人们找工作的途径，到贷款申请，再到刑事司法系统，甚至是判决标准，所以我们尽量修改学习算法来尽可能减少或是理想化消除这些非预期类型的偏见是十分重要的。

• 减少词嵌入中偏见的方法

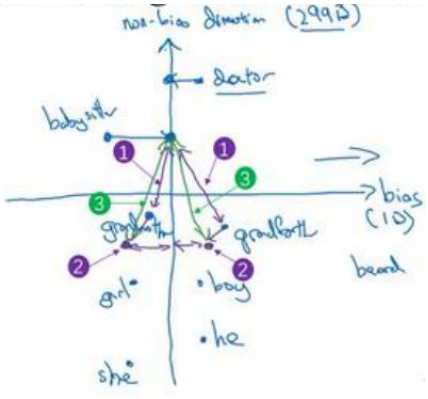
以性别为例：

①找到含有性别趋势的词对，首先作差，如$e_{he}-e_{she}$，$e_{male}-e_{female}$等，然后将这些值取平均，得到的向量代表偏见趋势，与这个轴正交的轴代表无偏见趋势。为了简化，我们将偏见趋势看做 1D 子空间，所以无偏见趋势将会是 299D 空间。



②中和步骤。对于那些性别中立的词可以进行移动处理，避免偏见。像 grandmother、grandfather、girl、boy、she、he，他们的定义中本就含有性别的内容这一步中不做处理；而像 doctor、babusitter 我们想使之在性别方面是中立的，我们将这些词移动至在无偏见趋势轴（上图标号 1）上进行处理，或者说减少在与无偏见趋势轴的距离。

③均衡步骤。对于含有性别含义的词对，如 grandmother 和 grandfather，或者 girl 和 boy，对于这些词嵌入，你只希望性别是其区别。在图中，即使经过中和步，babysitter 和 grandmother 之间的距离或者说是相似度实际上是小于 babysitter 和 grandfather 之间的，因此这可能会加重不良状态，或者可能是非预期的偏见。所以此步骤通过线性代数的步骤，主要做的是将 grandmother 和 grandfather（图中编号对②所示）移至与中间轴线等距的一对点上，现在性别歧视的影响也就是这两个词与 babysitter 的距离就完全相同了。



• 该方法中的细节：如何确定哪些词是中立的？

对于这个例子来说 doctor 看起来像是一个应该对其中立的单词来使之性别不确定或是种族不确定。相反地，grandmother 和 grandfather 就不应是性别不确定的词。也会有一些像是 beard 词，一个统计学上的事实是男性相比于比女性更有可能拥有胡子，因此也许 beard 应该比 female 更靠近 male 一些。

我们可以训练一个分类器来尝试解决哪些词是有明确定义的，哪些词是性别确定的，哪些词不是。结果表明英语里大部分词在性别方面上是没有明确定义的，只有一小部分词像是 grandmother-grandfather，girl-boy，sorority-fraternity 等等，不是性别中立的。因此需要平衡的词对的数实际上是很小的，至少对于性别歧视这个例子来说，可以人工列举来需要平衡的大部分词对。完整的算法会比课程中展示的更复杂一些。

第三周 序列模型和注意力机制

3.1 序列结构的各种序列

本周主要学习 seq2seq 模型，在机器翻译、语音识别中都能起到很大作用。以及学习集束搜索（Beam search）和注意力模型（Attention Model），最后学习到音频模型。

seq2seq 模型

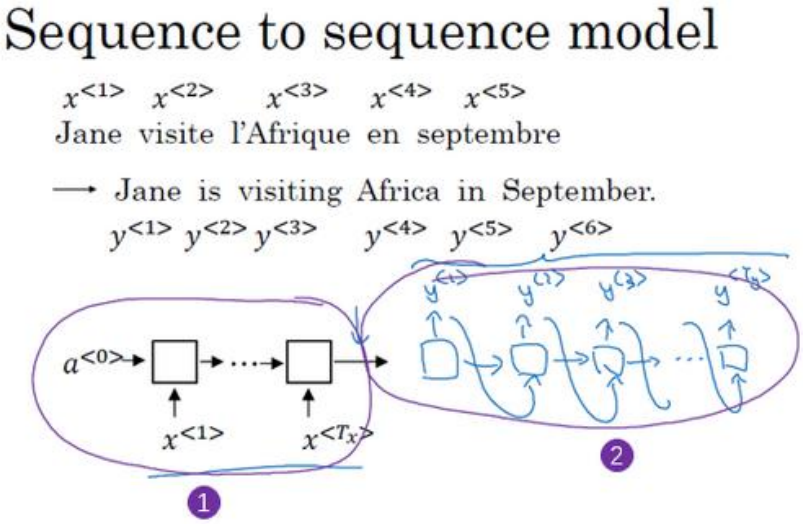
• 翻译问题：

“Jane visite l'Afrique en septembre.”（法）→ **“Jane is visiting Africa in September.”**（英）

• 目标：训练一个网络来输入序列 x 和输出序列 y

• 编码网络：如下图编号①所示，是一个 RNN 结构。RNN 单元可以是 GRU 或者 LSTM。每次只向该网络中输入一个法语单词，将输入序列接收完毕后，这个 RNN 网络会输出一个向量来代表这个输入序列。

• 解码网络：如下图编号②所示，它以编码网络的输出作为输入，之后它可以被训练为每次输出一个翻译后的单词，一直到它输出序列的结尾或者句子结尾标记，这个解码网络的工作就结束了。



[Sutskever et al., 2014. Sequence to sequence learning with neural networks]

[Cho et al., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation]

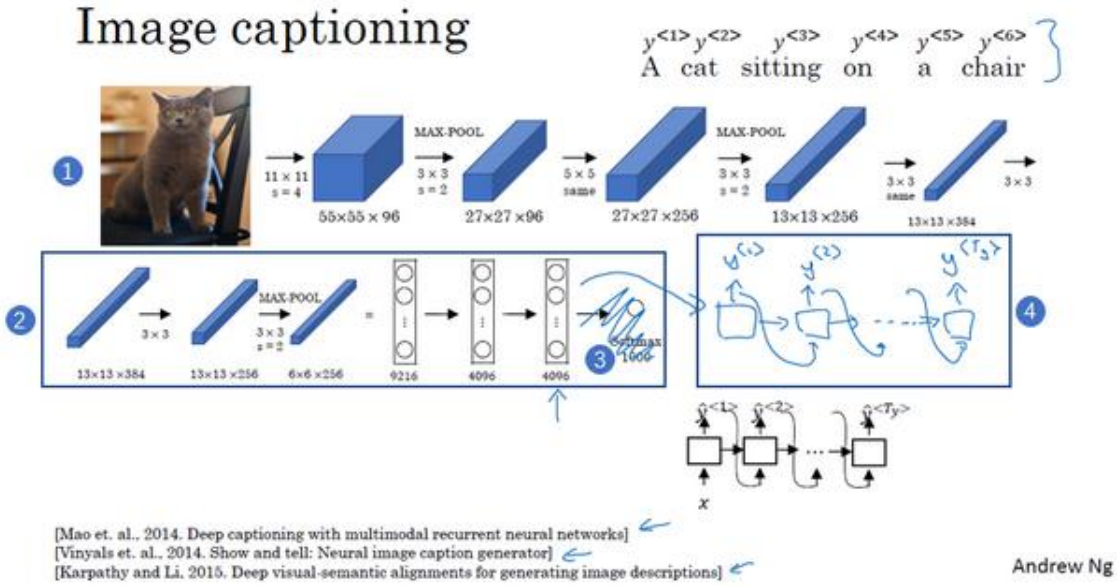
image to sequence 模型

• 图像描述问题：

给出一张图片，比如这张猫的图片（下图编号 1 所示），它能自动地输出该图片的描述。

• 编码网络：将图片输入到卷积神经网络中，比如一个预训练的 **AlexNet** 结构（下图编号 2 方框所示），然后让其学习图片的编码，去掉 **AlexNet** 结构最后的 **softmax** 单元，这个预训练的 **AlexNet** 结构会给你一个 4096 维的特征向量，向量表示的就是这只猫的图片。

• 解码网络：RNN 结构（下图编号 4 方框所示），输入为编码网络的输出的特征向量，然后让网络生成一个输出序列，或者说一个一个地输出单词序列。



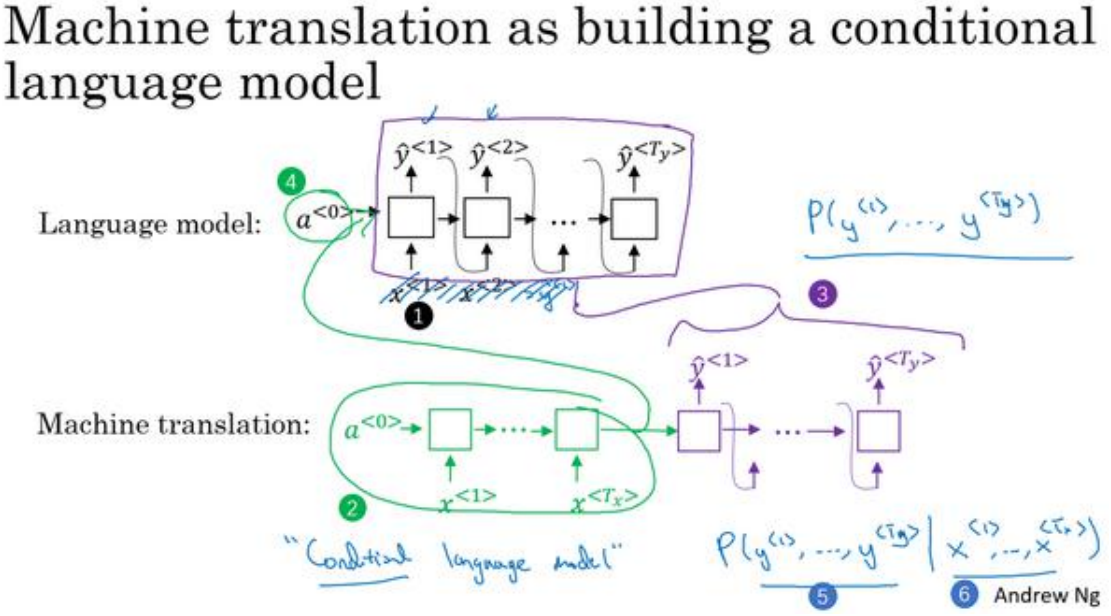
3.2 选择最可能的句子

本节主要介绍 seq2seq 机器翻译模型如何选取最贴切的描述输出。

可以把机器翻译想成建立一个条件语言模型。

语言模型：能够估计句子的可能性，也可以用于生成一个新的句子，只需要将前一个单元的输出变成下一个单元的输入。

机器翻译模型：下图中的绿色部分代表 encoder 网络，紫色代表 decoder 网络，decoder 网络和语言模型非常相似，不同点在于语言模型总以零向量作为开始，decoder 是从 encoder 经过计算后输出的输入句子的表示向量开始的，而不是从零向量开始。相比语言模型，机器翻译 decoder 的输出句子的概率同时取决于 encoder 部分的输出，所以它是一个条件语言模型。



条件语言模型的机制：

法语句子**"Jane visite l'Afrique en septembre."**经过 encoder 表示为向量x，我们需要考察不同的英语翻译所对应的概率，使得条件概率最大化，如下式：

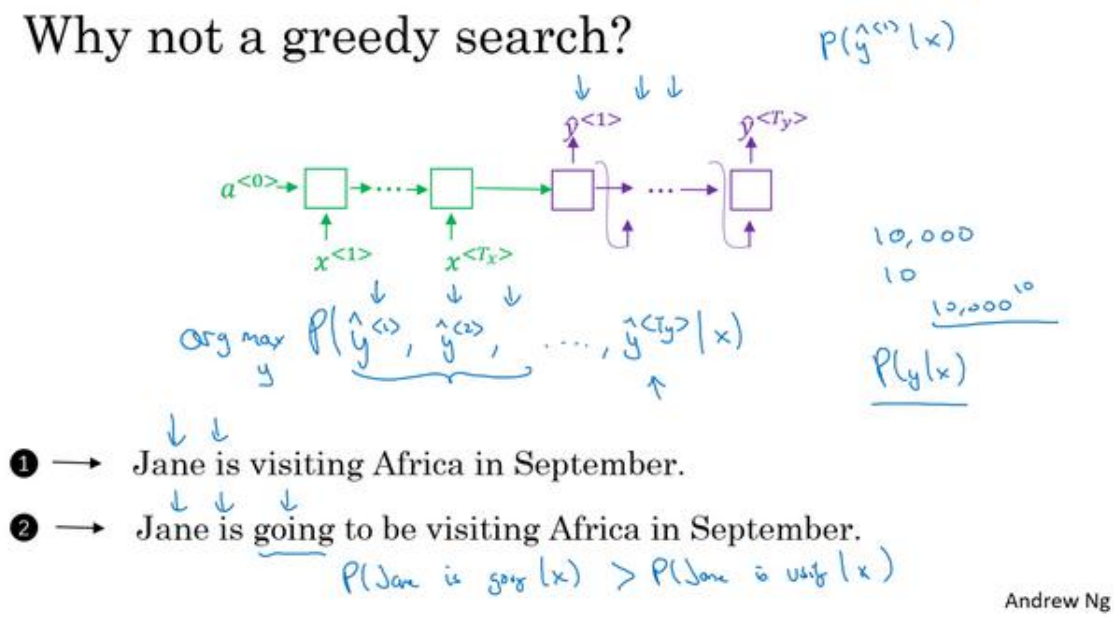
$$\arg \max_{y^{<1>},\ldots,y^{<T_y>}} P(y^{<1>},\ldots,y^{<T_y>}|x)$$

解决这个问题的通用方法为**束搜索**（Beam Search）。

为什么不用**贪心搜索**（Greedy Search）？

贪心搜索是一种来自计算机科学的算法，生成第一个词的分布以后，它将会根据你的条件语言模型挑选出最有可能的第一个词进入你的机器翻译模型中，在挑选出第一个词之后它将会继续挑选出最有可能的第二个词，然后继续挑选第三个最有可能的词。

但我们真正需要的是一次性挑选出整个单词序列，从$y^{<1>}$、$y^{<2>}$到$y^{<T_y>}$来使得整体的概率最大化。如果贪心地去找，一旦找错一个单词，后面就会出现多米诺骨牌效应，全部出错。



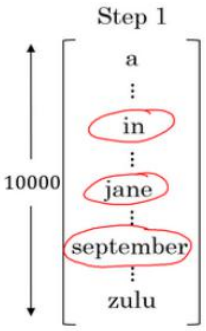
比如上面这个例子，①翻译要比②翻译好，但如果贪心算法挑选出了"Jane is"作为前两个词，因为在英语中 going 更加常见，于是对于法语句子来说"Jane is going"相比"Jane is visiting"会有更高的概率作为法语的翻译，所以很有可能如果仅仅根据前两个词来估计第三个词的可能性，得到的就是 going，最终会得到一个欠佳的句子。

同时枚举搜索的方法也不可取，假设字典里有 10000 个词，翻译的词长为 10，那么可能的翻译组合就有10000^{10}这么多，不可能去计算每种组合的可能性，这时常常用一个近似的搜索方法，可以尽量使得挑选出的句子使得条件概率最大化。下节将阐述这种搜索方法——集束搜索。

3.3 集束搜索（Beam Search）

本节主要介绍集束搜索算法，尽量使得挑选出的句子使得条件概率最大化。

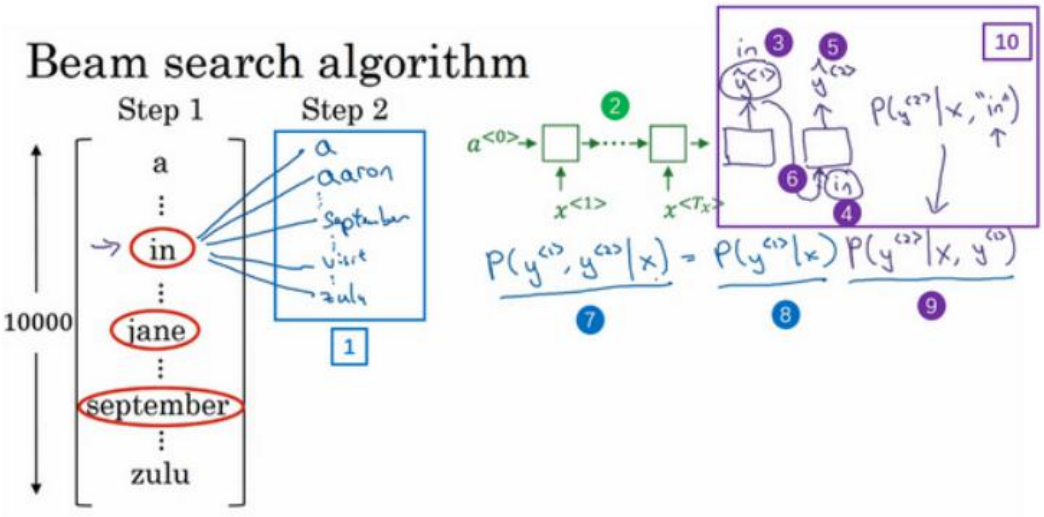
“Jane visite l'Afrique en Septembre.”（法语句子），我们希望翻译成英语，"Jane is visiting Africa in September".（英语句子）。集束搜索算法会有一个参数 B，叫做集束宽（beam width）。在这个例子中我把这个集束宽设成 3，这样就意味着集束搜索不会只考虑一个可能结果，而是一次会考虑 3 个。



①集束搜索算法首先做的就是挑选要输出的英语翻译中的第一个单词。来评估第一个单词的概率值，给定输入序列 x，即法语作为输入，第一个输出 y 的概率值是多少，即 $P(y^{<1>} | x)$。取前三大的概率值对应单词存起来。

②假设第一步已经选出了 in、jane、september 作为第一个单词三个最可能的选择，先保存这个概率值 $P(y^{<1>}|x)$，然后再乘以第二个概率值 $P(y^{<2>}| x, y^{<1>})$，得到第一个和第二个单词对的概率 $P(y^{<1>}, y^{<2>}|x)$。由于对第一步选出的三个单词均进行计算单词对概率的操作，所以会得到 30000 个单词对的概率值，从中挑选出前三大的概率值对应单词，并保存单词及单词对序列的概率值。

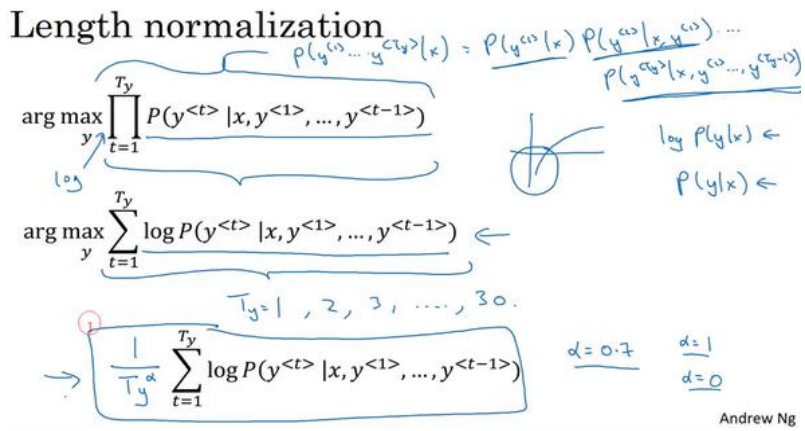
③重复第二步，集束搜索最终会找到“Jane visits africa in september”这个句子，终止在句尾符号，也可能有别的终止条件，看具体的设定。



3.4 改进集束搜索

本节中会学习到一些技巧，能够使集束搜索算法运行得更好。

长度归一化



前一节中介绍到束搜索需要最大化下面这个概率：

$$P(y^{<1>}...y^{<T_y>}|X)$$

即：$P(y^{<1>}|X)\ P(y^{<2>}|X,y^{<1>})\ P(y^{<3>}|X,y^{<1>},y^{<2>})\cdots P(y^{<T_y>}|X,y^{<1>},y^{<2>}...y^{<T_y-1>})$

如果计算这些，其实这些概率值都是小于 1 的，通常远小于 1。很多小于 1 的数乘起来，会得到很小很小的数字，会造成数值下溢。因此在实践中,我们不会最大化这个乘积，而是取 log 值。log 函数是严格单调递增的函数，所以最大化 logP(y|x)和最大化 P(y|x)结果一样。

所以实际工作中，我们总是记录**概率的对数和**，而不是概率的乘积。

如果参照原来的目标函数，则算法可能偏向于短句的输出，由于句子越长，概率值会越小。所以我们需要把它归一化，通过除以翻译单词的数量，也就是取平均，减少对输出长结果的惩罚。

归一化时，相比于直接除T_y，也就是输出句子的单词总数，我们有时会用一个更柔和的方法**加上指数a**，a 可以等于 0.7。（如果 a 等于 1，就相当于完全用长度来归一化，即取平均；如果 a 等于 0，相当于没有归一化）。

如何选择束宽 B

• 大束宽：会考虑很多的可能，会得到一个更好的结果；算法会运行的慢一些，内存占用也会增大。

• 小束宽：算法运行中，保存的选择更少，结果会没有那么好；算法运行的更快，内存占用也小。

在产品中，经常可以看到把束宽设到 10，束宽为 100 对于产品系统来说有点大了，这也取决于不同应用。但是对科研而言，人们想压榨出全部性能，这样有个最好的结果用来发论文，也经常看到大家用束宽为 1000 或者 3000，这也是取决于特定的应用和特定的领域。

束宽由 1（贪心算法）增加到 3、到 10，可以看到一个很大的改善；但当束宽从 1000 增加到 3000，效果就没有那么明显了。

• 与其它搜索算法的关系：

广度优先搜索（BFS）、深度优先搜索（DFS）这些是精确搜索的算法，不同于这些算法，束搜索运行的更快，但是不能保证一定能找到 argmax 的准确的最大值。

3.5 集束搜索的误差分析

由于束搜索是一种近似搜索的算法，也被称作启发式搜索算法，它不总是输出可能性最大的句子；所以本节将介绍误差分析和束搜索算法是如何相互起作用的。如何利用误差分析发现是束搜索算法还是 RNN 模型出现了问题。

以一个例子来说明：“**Jane visite l'Afrique en septembre**”（法）翻译为英语，人工是这样翻译的: **Jane visits Africa in September**，将这个标记为y^*；机器翻译结果为：**Jane visited Africa last September**，我们将它标记为$\hat{y}$。这是一个十分糟糕的翻译，它实际上改变了句子的原意，因此这不是个好翻译。

模型主要有两个部分：RNN 模型（编码器和解码器）；束搜索算法（以某个束宽 B 运行）。我们需要分析找到是造成这个错误的原因出在哪个部分上。

误差分析方法：

用 RNN 模型计算$P(y^*|x)$和$P(\hat{y}|x)$，然后比较这两个值的大小。

①$P(y^*|x) > P(\hat{y}|x)$

束搜索算法的目的是寻找使$P(y|x)$最大的y，但是这种情况下，相比于$\hat{y}$，y^*能使得$P(y|x)$更大，因此你能够得出束搜索算法实际上不能够给你一个能使$P(y|x)$最大化的y值，所以是束搜索算法出错了。

②$P(y^*|x) < P(\hat{y}|x)$

在我们的例子中，y^*是比$\hat{y}$更好的翻译结果，不过根据 RNN 模型的结果，$P(y^*)$是小于$P(\hat{y})$的，也就是说，相比于$\hat{y}$，y^*成为输出的可能更小。所以是 RNN 模型出了问题。

Error analysis process

Human	Algorithm	$P(y^*\|x)$	$P(\hat{y}\|x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	2×10^{-10}	1×10^{-10}	B
...	...	—	—	R
...	...	—	—	B R R ⋮

Figures out what faction of errors are "due to" beam search vs. RNN model

我们遍历开发集，用上述误差分析方法找出每个错误例子的错误原因，找出模型的哪一部分造成了较多的错误。如果是束搜索算法造成大部分错误，则需要增大束宽；如果是 RNN 模型出了更多错，则需要进行更深层次的分析，来决定是需要正则化还是获取更多的训练数据，或者是更换网络结构。

3.6 BLEU 得分

机器翻译会遇到一个问题：一个法语句子可以有多种英文翻译而且都同样好，所以当有多个同样好的答案时，怎样评估一个机器翻译系统？本节介绍的 BLUE 得分，便用于衡量机器翻译的准确性。

BLUE 代表 **bilingual evaluation understudy** (双语评估替补)，给定一个机器生成的翻译，它能够自动地计算一个分数来衡量机器翻译的好坏。

我们逐个对 MT 中的单词出现在人工翻译集中的次数进行计数，在达到某个文本中的出现上限时进行**截断计数**。

French: Le chat est sur le tapis.

→ Reference 1: The cat is on the mat.

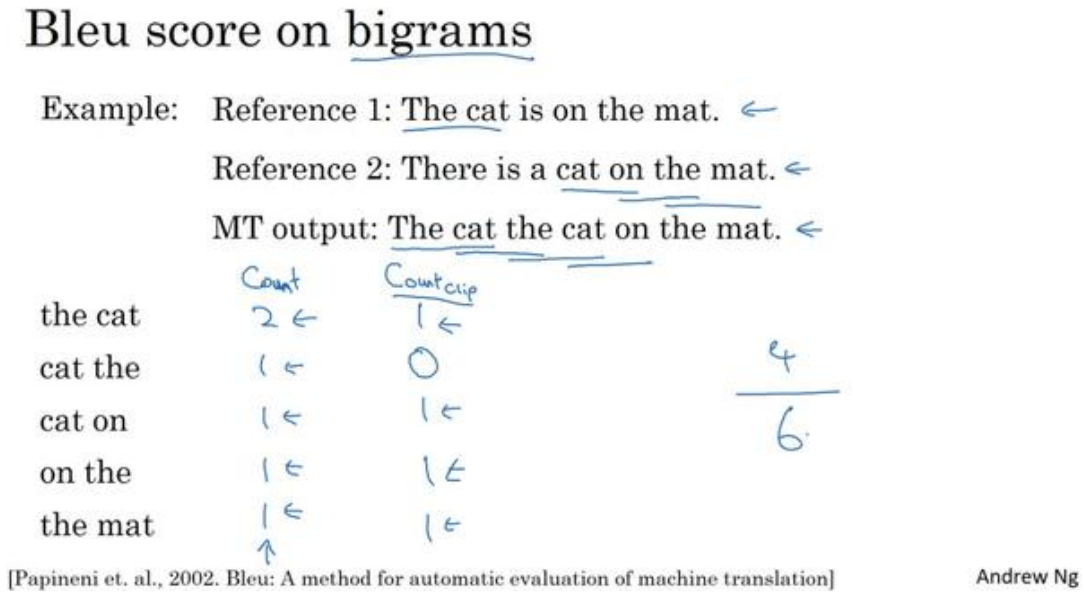
→ Reference 2: There is a cat on the mat.

→ MT output: the the the the the the the.

Precision: $\frac{7}{7}$ Modified precision: $\frac{2}{7}$

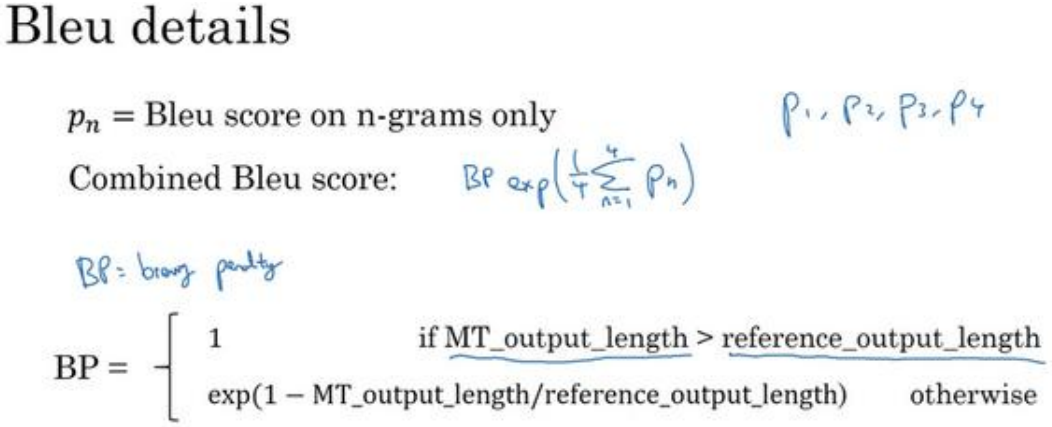
比如上面这个例子，逐个对 MT 中的单词计数，看是否出现在了 Reference 中，由于 the 出现在了 Reference 中，如果不采用截断计数，计算的 BLUE 分数为 1，这显然不合理；而进行截断计数后，由于一个文本中最多出现 2 个 the，所以对单词 the 的计数不能超过 2，所以得出的 BLUE 分数为$\frac{2}{7}$。

除了考虑单个单词 unigram，我们还可以对二元词组 bigram 进行计数，计数方式也是采取截断计数，如下所示。



对于 n-gram 的截断计数（P_n为 n 元词组精确度）为：

$$P_n = \frac{\sum_{n-grams\in\hat{y}} Count_{clip}(n-gram)}{\sum_{n-grams\in\hat{y}} Count(n-gram)}$$



将这些综合一下，得到最终的 BLEU 得分：

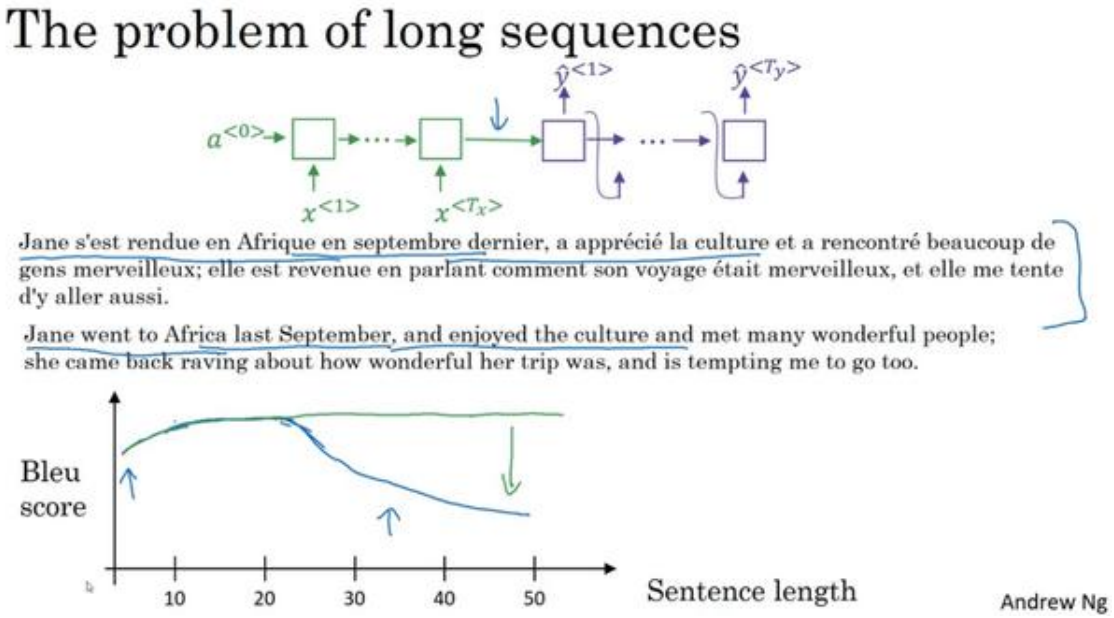
$$\mathrm{BP}\cdot\exp\left(\frac{1}{4}\sum_{n=1}^{4}\mathrm{P_n}\right)$$

惩罚因子 BP：由于输出一个很短的翻译，不过我们并不想要特别短的翻译结果容易得到一个高精确度，不过我们并不想要特别短的翻译结果，所以需要惩罚因子来惩罚输出太短翻译结果的翻译系统。（**注意：**上图中计算 BP 的第二种情况下

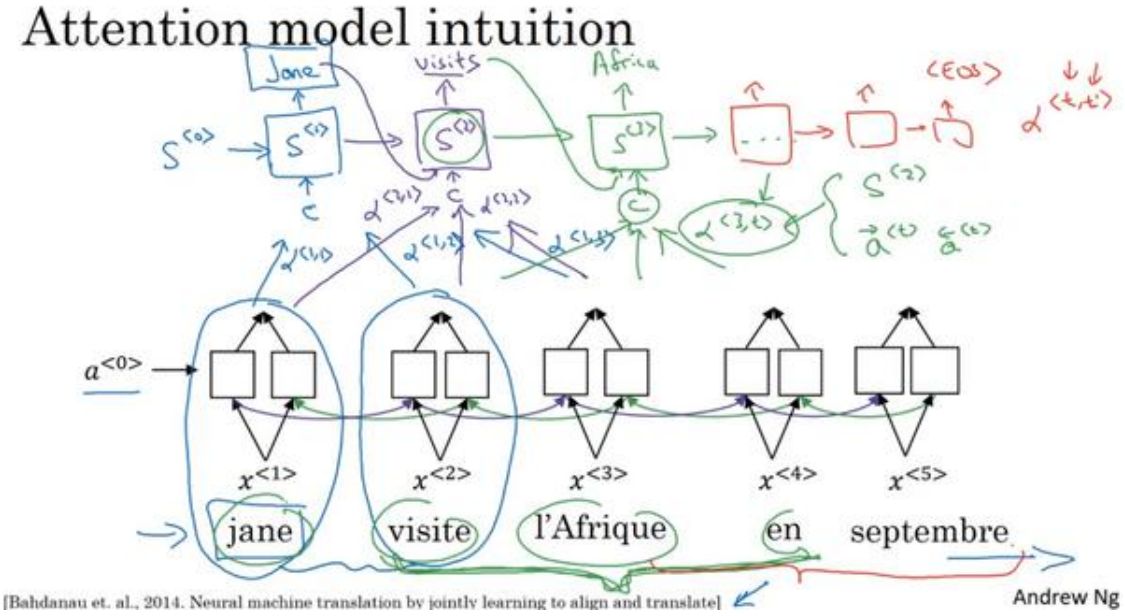
除式的分子分母写反了）

3.7 注意力模型直观理解

之前的几节中主要使用编码解码的结构来完成机器翻译，我们现在要对其做一些改变，得到注意力模型，可以使它工作得更好。



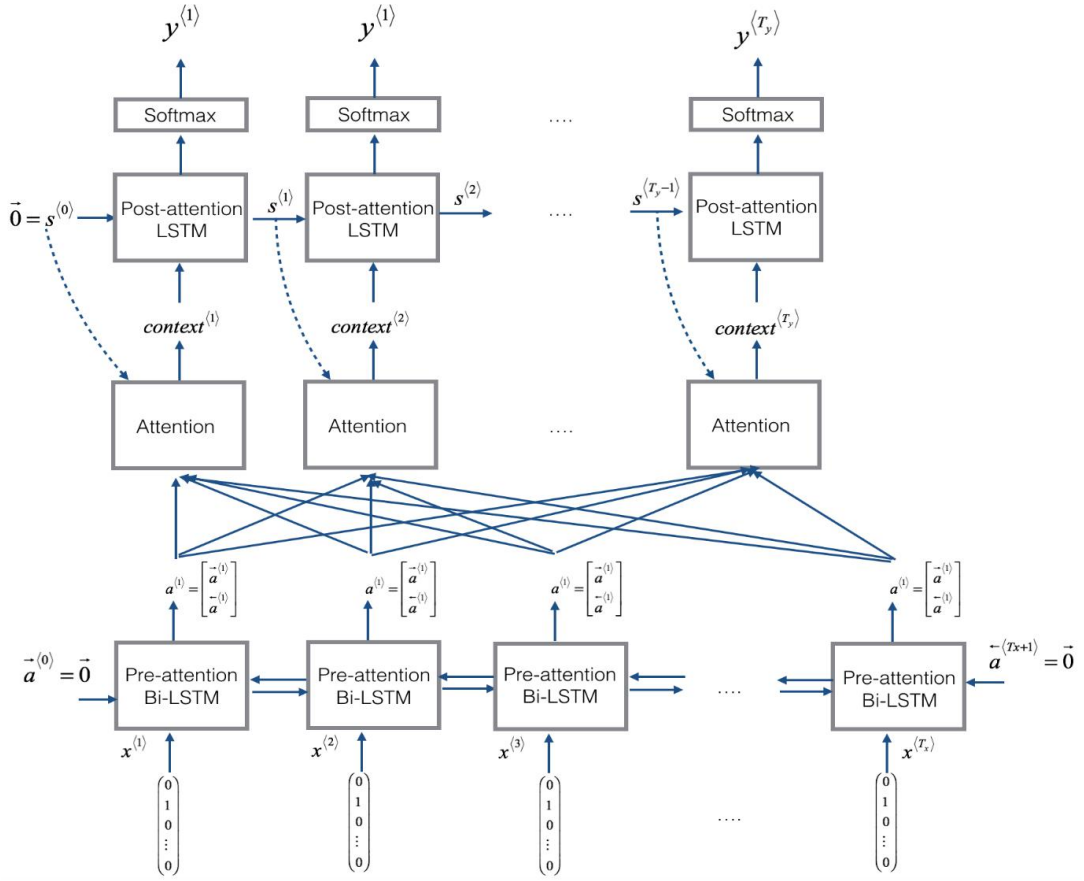
编码解码这种结构，对短句子效果很好，会得到较高的 BLUE 得分，但是对于长句子而言，它的表现就会变差，这是由于神经网络对记忆长句子比较困难。而注意力模型翻译时更像人类，一次翻译句子的一部分，所以不会看到有一个巨大的下倾（如绿色线条所示）。



比如一个句子(法语)Jane visite l'Afrique en Septembre，使用一个双向的 RNN，当生成第一个词时，注意力模型计算注意力权重，我们用$a^{<1,1>}$来表示当你生成第一个词时你应该放多少注意力在这个第一块信息处，然后计算第二个，$a^{<1,2>}$告诉我们当你尝试去计算第一个词 Jane 时，我们应该花多少注意力在输入的第二个词上面。依次类推，得出每个词的注意力分数后，我们可以计算出隐藏层$S^{<1>}$

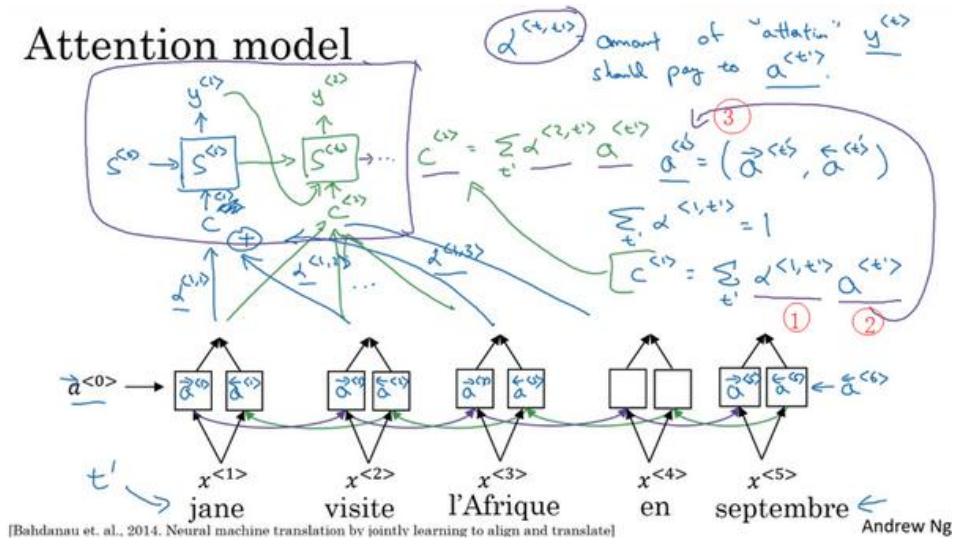
的值，从而输出第一个值，之后每一步重复计算注意力的这一过程。基本思想为：当生成一个特定的英文词时，这允许它在每个时间步去看周围词距内的法语词要花多少注意力。

更为具体的图描述：



3.8 注意力模型

上一节中我们了解了注意力模型的主要的思想，对它有了直观的理解，下面将阐述一些计算上的细节。

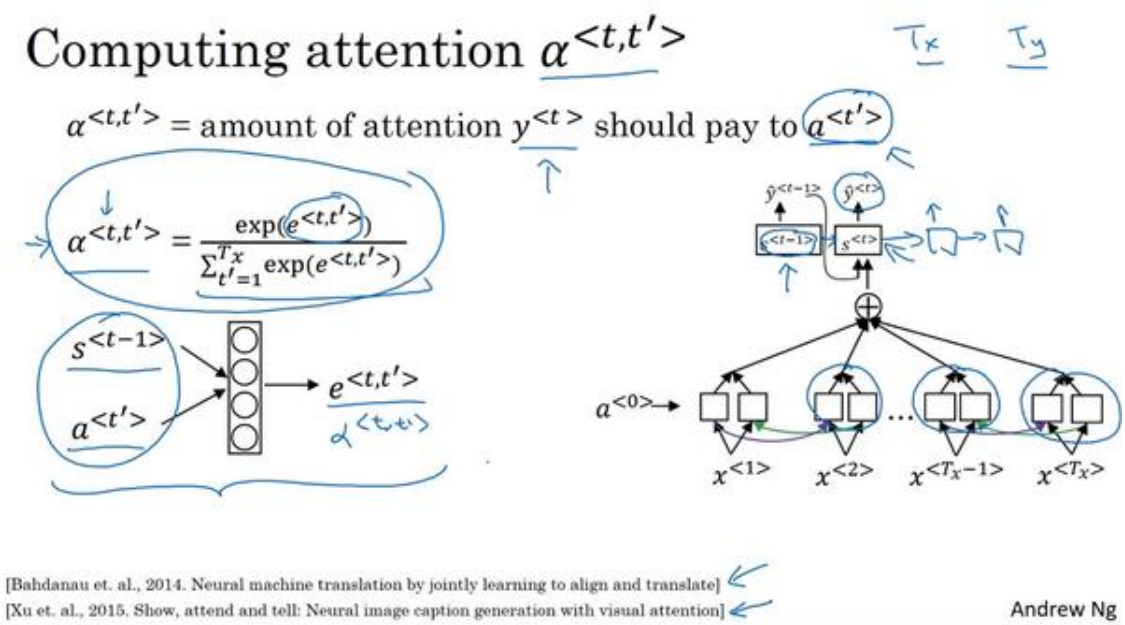


我们将用 t'来索引法语句子里面的词，由于双向 RNN 在时间步 t'会有两个特

征向量（前向和后向），我们用$a^{<t'>}$表示($\overrightarrow{a^{<t'>}}$，$\overleftarrow{a^{<t'>}}$)连接起来的向量。上下文$c^{<t>}$的计算取决于连接特征向量$a^{<t'>}$和注意力权重$\alpha^{<t,t'>}$，实际上是两者的加权和。

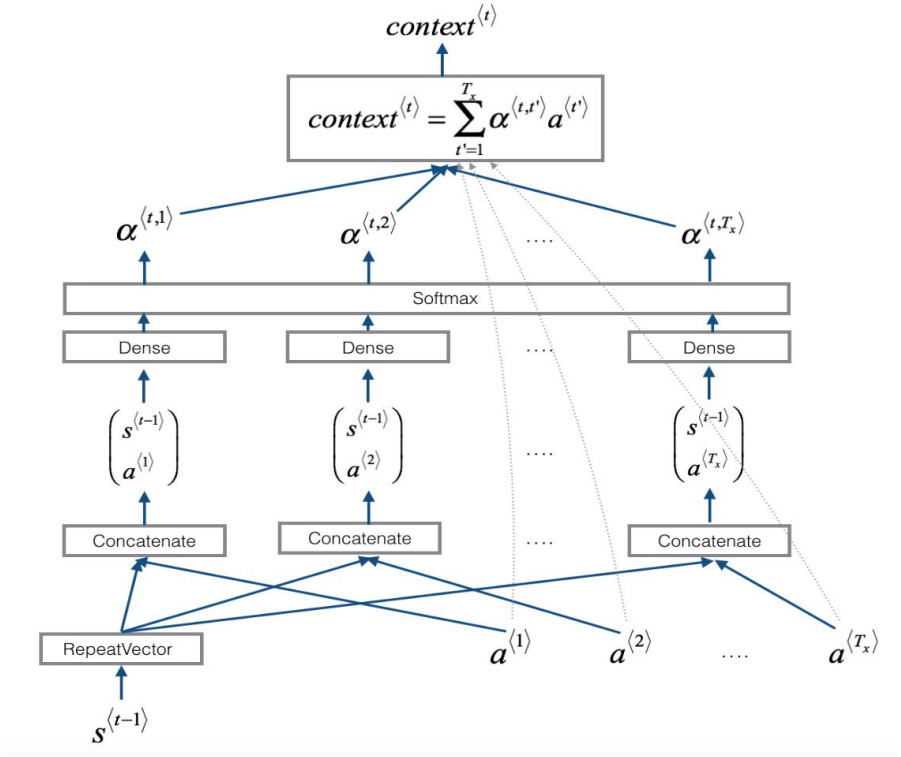
$$c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>}$$

下面我们学习如何计算注意力权重$\alpha^{<t,t'>}$，即$y^{<t>}$应该在 t'时花在 a 上注意力的数量。



$s^{<t-1>}$和$a^{<t'>}$经过一个小的神经网络，得到$e^{<t,t'>}$，然后经过 softmax 层，可以得到注意力权重$\alpha^{<t,t'>}$。

$e^{<t,t'>}$的计算过程：



除了是一个线性层（点乘注意力）计算注意力$e^{<t,t'>}$，我们还可以考虑如下方式（摘自 cs224n）：

有几种方法可以从 $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ 计算 $\boldsymbol{e} \in \mathbb{R}^N$ 和 $\boldsymbol{s} \in \mathbb{R}^{d_2}$

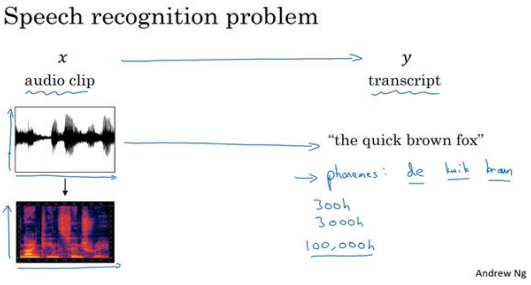
- 基本的点乘注意力 $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - 注意：这里假设 $d_1 = d_2$
  - 这是我们之前看到的版本
- 乘法注意力 $e_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ 是权重矩阵
- 加法注意力 $e_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - 其中 $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ 是权重矩阵，$\boldsymbol{v} \in \mathbb{R}^{d_3}$ 是权重向量
  - d_3(注意力维度)是一个超参数

3.9 语音识别

seq2seq 模型在语音识别方面准确性有了很大的提升。本节将介绍 seq2seq 模型是如何应用于音频数据的（audio data），比如语音。

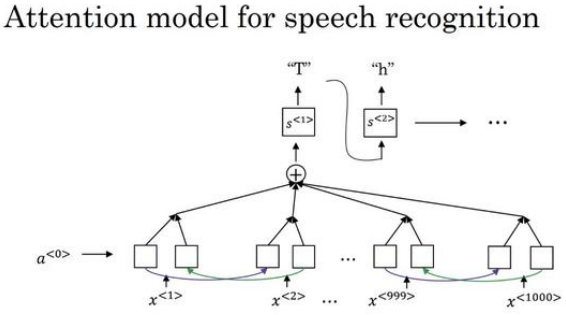
什么是语音视频问题呢？现在你有一个音频片段 x（**an audio clip,x**），你的任务是自动地生成文本 y。

假如一个音频片段的内容是："the quick brown fox"(敏捷的棕色狐狸)，这时我们希望一个语音识别算法，通过输入这段音频，然后输出音频的文本内容。考虑到人的耳朵并不会处理声音的原始波形，而是通过一种特殊的物理结构来测量这些，不同频率和强度的声波。音频数据的常见预处理步骤，就是运行这个原始的音频片段，然后生成一个声谱图（spectrogram），就像这样。同样地，横轴是时间，纵轴是声音的频率，而图中不同的颜色，显示了声波能量的大小，也就是在不同的时间和频率上这些声音有多大。



通过向系统中输入音频片段，然后直接输出音频的文本，在文本音频数据集中同时包含 x 和 y，通过深度学习算法大大推进了语音识别的进程。那么，如何建立一个语音识别系统呢？

在输入音频的不同时间帧上，可以用一个注意力模型，来输出文本描述，如"the quick brown fox"，或者其他语音内容。

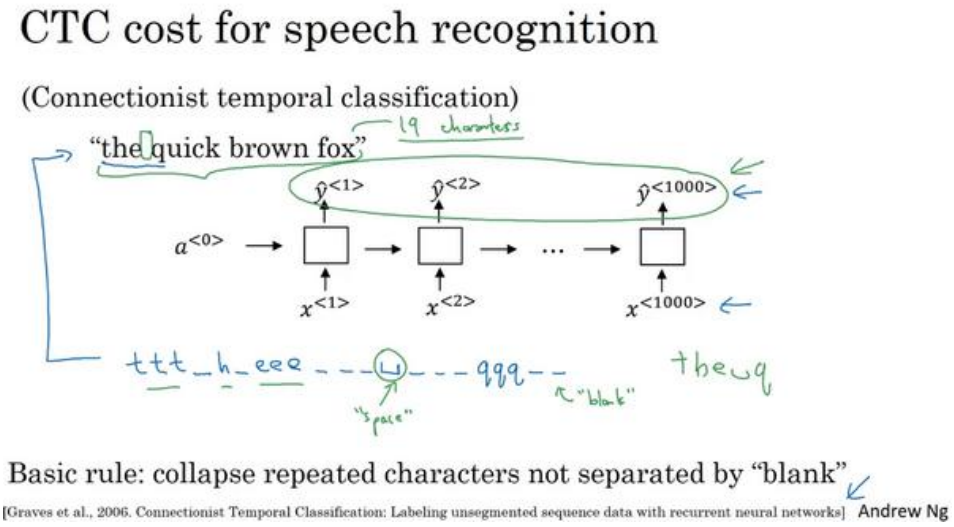


CTC 损失函数

算法思想：

我们使用的网络输入 x 和输出 y 的数量都是一样的，而在语音识别中，通常输入的时间步数量要比输出的时间步的数量多出很多。比如一段 10 秒的音频，并且特征是 100 赫兹的，即每秒有 100 个样本，于是这段 10 秒的音频片段就会

有 1000 个输入，但可能输出就没有 1000 个字母了。**CTC** 损失函数允许 **RNN** 生成这样的输出：**ttt**，这是一个特殊的字符，叫做空白符，我们这里用下划线表示，这句话开头的音可表示为 **h_eee_ _ _**，然后这里可能有个空格，我们用这个来表示空格，之后是**_ _ _qqq__**，这样的输出也被看做是正确的输出。下面这段输出对应的是**"the q"**。CTC 损失函数的一个**基本规则**是将空白符之间的重复的字符折叠起来。

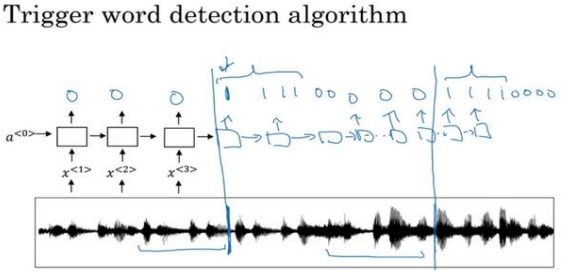


3.10 触发字检测

随着语音识别的发展，越来越多的设备可以通过你的声音来唤醒，这有时被叫做触发字检测系统（rigger word detection systems）。本节介绍如何建立一个触发字系统。

触发字系统的例子包括 Amazon echo，它通过单词 Alexa 唤醒；还有百度 DuerOS 设备，通过"小度你好"来唤醒；苹果的 Siri 用 Hey Siri 来唤醒；Google Home 使用 Okay Google 来唤醒，这就是触发字检测系统。假如你在卧室中，有一台 Amazon echo，你可以在卧室中简单说一句：Alexa, 现在几点了?就能唤醒这个设备。它将会被单词"Alexa"唤醒，并回答你的询问。如果你能建立一个触发字检测系统，也许你就能让你的电脑通过你的声音来执行某些事。下面介绍，如何构建一个触发字检测系统。

我们要做的就是把一个音频片段计算出它的声谱图特征得到特征向量 $x^{<1>}$, $x^{<2>}$, $x^{<3>}$..，然后把它放到 RNN 中，最后要做的，就是定义我们的目标标签 y。并在标标签时，在输出变回 0 之前，多次输出 1，或说在固定的一段时间内输出多个 1。这样的话，可以提高 1 与 0 的比例，使训练集更加平衡。



3.11 结论与致谢

Specialization outline

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring Machine Learning Projects
4. Convolutional Neural Networks
5. Sequence Models

在之前的课程中，我们学习了神经网络和深度学习，如何改进深度神经网络，如何结构化机器学习项目，和卷积神经网络以及循环神经网络。通过深度学习算法，你可以让计算机拥有"视觉"，可以让计算机自己合成小说，或者合成音乐，可以让计算机将一种语言翻译成另一种，或者对放射影像进行定位然后进行医疗诊断，或者构建自动驾驶系统，总得来说，It is amazing and has infinite possibilities.

——Notes Taken By：FangYe