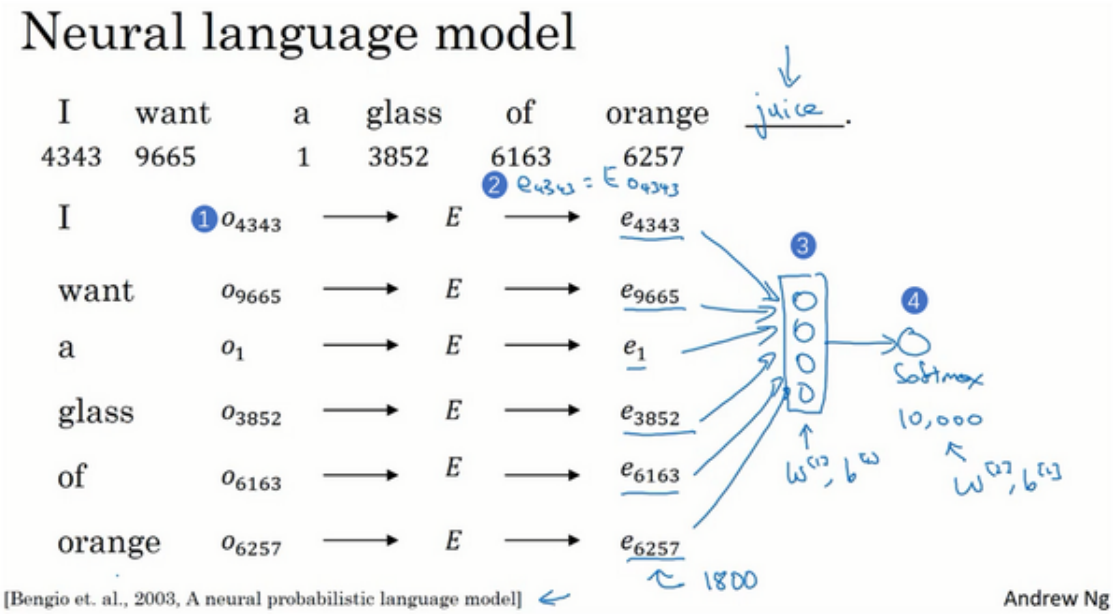
## 2.5 学习词嵌入（Learning Word Embeddings）

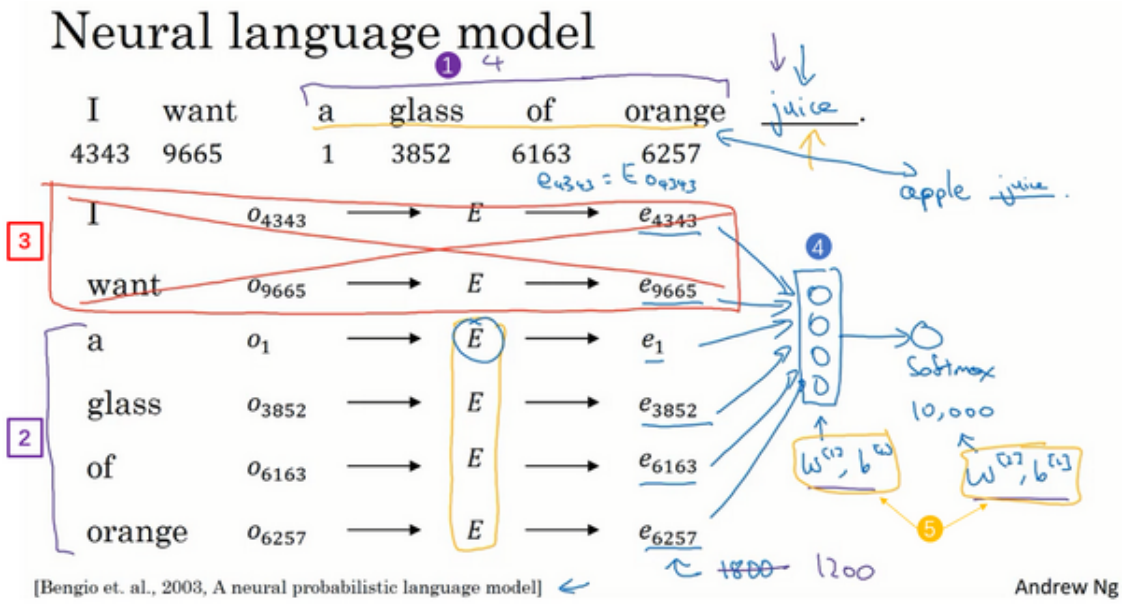
假如你在构建一个语言模型，并且用神经网络来实现这个模型。于是在训练过程中，你可能想要你的神经网络能够做到比如输入：“**I want a glass of orange ___.**”，然后预测这句话的下一个词。在每个单词下面，我都写上了这些单词对应词汇表中的索引。实践证明，建立一个语言模型是学习词嵌入的好方法，我提出的这些想法是源于 **Yoshua Bengio**，**Rejean Ducharme**，**Pascal Vincent**，**Rejean Ducharme**，**Pascal Vincent** 还有 **Christian Jauvin**。




下面我将介绍如何建立神经网络来预测序列中的下一个单词，让我为这些词列一个表格，“**I want a glass of orange**”，我们从第一个词 **I** 开始，建立一个 **one-hot** 向量表示这个单词 **I**。这是一个 **one-hot** 向量（上图编号 1 所示），在第 4343 个位置是 1，它是一个 10,000 维的向量。然后要做的就是**生成一个参数矩阵$E$，然后用$E$乘以$O_{4343}$，得到嵌入向量$e_{4343}$，**这一步意味着$e_{4343}$是由矩阵$E$乘以 **one-hot** 向量得到的（上图编号 2 所示）。然后我们对其他的词也做相同的操作，单词 **want** 在第 9665 个，我们将$E$与这个 **one-hot** 向量（$O_{9665}$）相乘得到嵌入向量$e_{9665}$。对其他单词也是一样，**a** 是字典中的第一个词，因为 **a** 是第一个字母，由$O_1$得到$e_1$。同样地，其他单词也这样操作。

于是现在你有许多 300 维的嵌入向量。我们能做的就是把它们全部放进神经网络中（上图编号 3 所示），经过神经网络以后再通过 **softmax** 层（上图编号 4 所示），这个 **softmax** 也有自己的参数，然后这个 **softmax** 分类器会在 10,000 个可能的输出中预测结尾这个单词。假如说在训练集中有 **juice** 这个词，训练过程中 **softmax** 的目标就是预测出单词 **juice**，就是

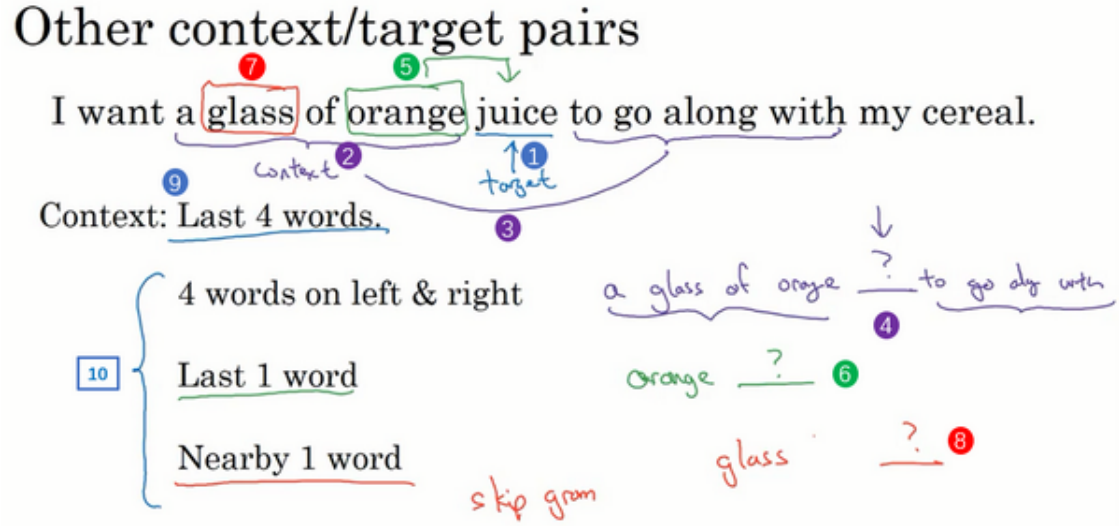
结尾的这个单词。这个隐藏层（上图编号 3 所示）有自己的参数，我这里用$W^{[1]}$和$b^{[1]}$来表示，这个 **softmax** 层（上图编号 4 所示）也有自己的参数$W^{[2]}$和$b^{[2]}$。如果它们用的是 300 维大小的嵌入向量，而这里有 6 个词，所以用 6×300，所以这个输入会是一个 1800 维的向量，这是通过将这 6 个嵌入向量堆在一起得到的。




实际上更常见的是有一个固定的历史窗口，举个例子，你总是想预测给定四个单词（上图编号 1 所示）后的下一个单词，注意这里的 4 是算法的超参数。这就是如何适应很长或者很短的句子，方法就是总是只看前 4 个单词，所以说我只用这 4 个单词（上图编号 2 所示）而不去看这几个词（上图编号 3 所示）。如果你一直使用一个 4 个词的历史窗口，这就意味着你的神经网络会输入一个 1200 维的特征变量到这个层中（上图编号 4 所示），然后再通过 **softmax** 来预测输出，选择有很多种，用一个固定的历史窗口就意味着你可以处理任意长度的句子，因为输入的维度总是固定的。所**以这个模型的参数就是矩阵$E$，对所有的单词用的都是同一个矩阵$E$，而不是对应不同的位置上的不同单词用不同的矩阵。然后这些权重（上图编号 5 所示）也都是算法的参数，你可以用反向传播来进行梯度下降来最大化训练集似然，通过序列中给定的 4 个单词去重复地预测出语料库中下一个单词什么。**

事实上通过这个算法能很好地学习词嵌入，原因是，如果你还记得我们的 **orange jucie**，**apple juice** 的例子，在这个算法的激励下，**apple** 和 **orange** 会学到很相似的嵌入，这样做能够让算法更好地拟合训练集，因为它有时看到的是 **orange juice**，有时看到的是 **apple juice**。如果你只用一个 300 维的特征向量来表示所有这些词，算法会发现要想最好地拟合训练集，就要使 **apple**（**苹果**）、**orange**（**橘子**）、**grape**（**葡萄**）和 **pear**（**梨**）等等，还有像 **durian**（**榴莲**）这种很稀有的水果都拥有相似的特征向量。

这就是早期最成功的学习词嵌入，学习这个矩阵$E$的算法之一。现在我们先概括一下这个算法，看看我们该怎样来推导出更加简单的算法。现在我想用一个更复杂的句子作为例子来解释这些算法，假设在你的训练集中有这样一个更长的句子：**“I want a glass of orange juice to go along with my cereal.”**。我们在上个幻灯片看到的是算法预测出了某个单词 **juice**，我们把它叫做目标词（下图编号 1 所示），它是通过一些上下文，在本例中也就是这前 4 个词（下图编号 2 所示）推导出来的。如果你的目标是学习一个嵌入向量，研究人员已经尝试过很多不同类型的上下文。如果你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。但如果你的目标不是学习语言模型本身的话，那么你可以选择其他的上下文。




比如说，你可以提出这样一个学习问题，它的上下文是左边和右边的四个词，你可以把**目标词左右各 4 个词作为上下文**（上图编号 3 所示）。这就意味着我们提出了一个这样的问题，算法获得左边 4 个词，也就是 **a glass of orange**，还有右边四个词 **to go along with**，然后要求预测出中间这个词（上图编号 4 所示）。提出这样一个问题，这个问题需要将左边的还有右边这 4 个词的嵌入向量提供给神经网络，就像我们之前做的那样来预测中间的单词是什么，来预测中间的目标词，这也可以用来学习词嵌入。

或者你想用一个更简单的上下文，也许只提供目标词的前一个词，比如只给出 **orange** 这个词来预测 **orange** 后面是什么（上图编号 5 所示），这将会是不同的学习问题。只给出一个词 **orange** 来预测下一个词是什么（上图编号 6 所示），你可以构建一个神经网络，只把目标词的前一个词或者说前一个词的嵌入向量输入神经网络来预测该词的下一个词。

还有一个效果非常好的做法就是上下文是附近一个单词，它可能会告诉你单词 **glass**（上图编号 7 所示）是一个邻近的单词。或者说我看见了单词 **glass**，然后附近有一个词和 **glass** 位置相近，那么这个词会是什么（上图编号 8 所示）？这就是用附近的一个单词作为上下

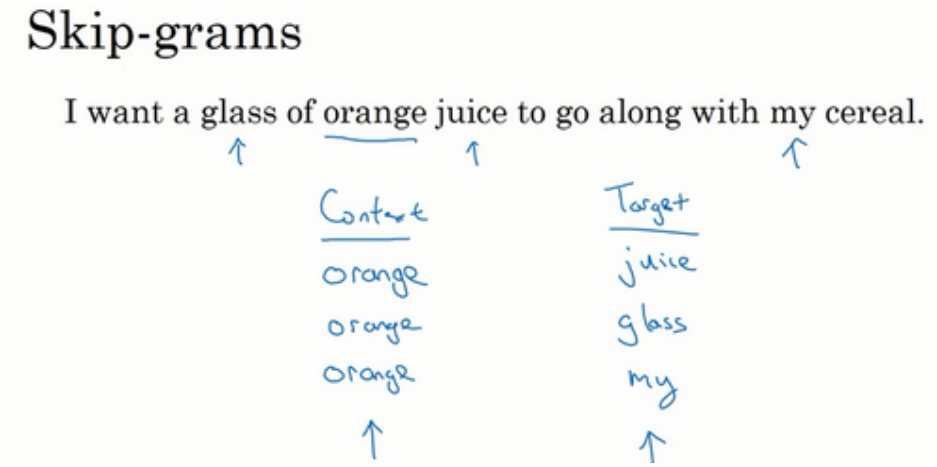
文。我们将在下节视频中把它公式化，这用的是一种 **Skip-Gram** 模型的思想。这是一个简单算法的例子，因为上下文相当的简单，比起之前 4 个词，现在只有 1 个，但是这种算法依然能工作得很好。

研究者发现，如果你真想建立一个语言模型，用目标词的前几个单词作为上下文是常见做法（上图编号 9 所示）。但如果你的目标是学习词嵌入，那么你就可以用这些其他类型的上下文（上图编号 10 所示），它们也能得到很好的词嵌入。我会在下节视频详细介绍这些，我们会谈到 **Word2Vec** 模型。

总结一下，在本节视频中你学习了语言模型问题，模型提出了一个机器学习问题，即输入一些上下文，例如目标词的前 4 个词然后预测出目标词，学习了提出这些问题是怎样帮助学习词嵌入的。在下节视频，你将看到如何用更简单的上下文和更简单的算法来建立从上下文到目标词的映射，这将让你能够更好地学习词嵌入，一起进入下节视频学习 **Word2Vec** 模型。

## 2.6 Word2Vec

在上个视频中你已经见到了如何学习一个神经语言模型来得到更好的词嵌入，在本视频中你会见到 **Word2Vec** 算法，这是一种简单而且计算时更加高效的方式来学习这种类型的嵌入，让我们来看看。



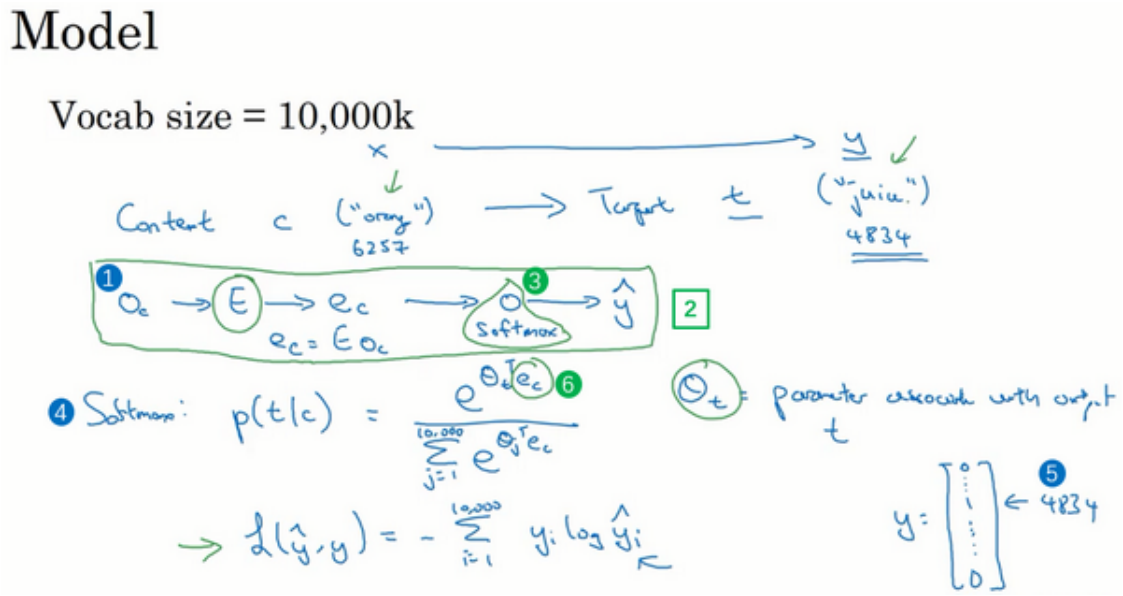

本视频中的大多数的想法来源于 **Tomas Mikolov**，**Kai Chen**，**Greg Corrado** 和 **Jeff Dean**。（**Mikolov T, Chen K, Corrado G, et al. Efficient Estimation of Word Representations in Vector Space[J]. Computer Science, 2013.**）

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.”**，在 **Skip-Gram** 模型中，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，比如选 **orange** 这个词，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。可能你正好选到了 **juice** 作为目标词，正好是下一个词（表示 **orange** 的下一个词），也有可能你选到了前面第二个词，所以另一种配对目标词可以是 **glass**，还可能正好选到了单词 **my** 作为目标词。

于是我们将构造一个监督学习问题，它给定上下文词，要求你预测在这个词正负 10 个词距或者正负 5 个词距内随机选择的某个目标词。显然，这不是个非常简单的学习问题，因为在单词 **orange** 的正负 10 个词距之间，可能会有很多不同的单词。但是构造这个监督学习问题的目标并不是想要解决这个监督学习问题本身，而是想要使用这个学习问题来学到一个好的词嵌入模型。

接下来说说模型的细节，我们继续假设使用一个 10,000 词的词汇表，有时训练使用的

词汇表会超过一百万词。但我们要解决的基本的监督学习问题是学习一种映射关系，从上下文 **c**，比如单词 **orange**，到某个目标词，记为 **t**，可能是单词 **juice** 或者单词 **glass** 或者单词 **my**。延续上一张幻灯片的例子，在我们的词汇表中，**orange** 是第 6257 个单词，**juice** 是 10,000 个单词中的第 4834 个，这就是你想要的映射到输出$y$的输入$x$。




为了表示输入，比如单词 **orange**，你可以先从 **one-hot** 向量开始，我们将其写作$O_c$，这就是上下文词的 **one-hot** 向量（上图编号 1 所示）。然后和你在上节视频中看到的类似，你可以拿嵌入矩阵$E$乘以向量$O_c$，然后得到了输入的上下文词的嵌入向量，于是这里$e_c = EO_c$。在这个神经网络中（上图编号 2 所示），我们将把向量$e_c$喂入一个 **softmax** 单元。我通常把 **softmax** 单元画成神经网络中的一个节点（上图编号 3 所示），这不是字母 **O**，而是 **softmax** 单元，**softmax** 单元要做的就是输出$\hat{y}$。然后我们再写出模型的细节，这是 **softmax** 模型（上图编号 4 所示），预测不同目标词的概率：

$$\boldsymbol{Softmax: p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}}$$

这里$\theta_t$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。我省略了 **softmax** 中的偏差项，想要加上的话也可以加上。

最终 **softmax** 的损失函数就会像之前一样，我们用$y$表示目标词，我们这里用的$y$和$\hat{y}$都是用 **one-hot** 表示的，于是损失函数就会是：

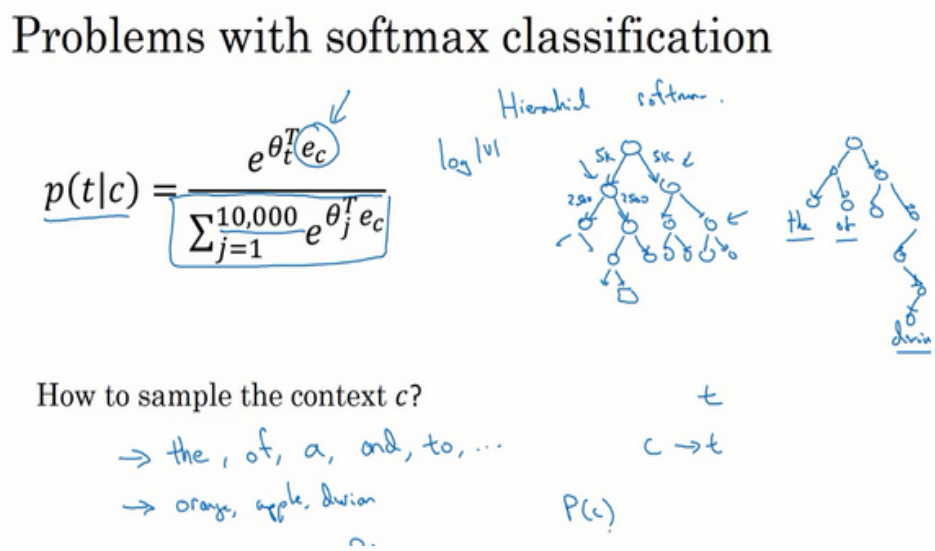
$$\boldsymbol{L(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i}$$

这是常用的 **softmax** 损失函数，$y$ 就是只有一个 1 其他都是 0 的 **one-hot** 向量，如果目

标词是 **juice**，那么第 4834 个元素就是 1，其余是 0（上图编号 5 所示）。类似的$\hat{y}$是一个从 **softmax** 单元输出的 10,000 维的向量，这个向量是所有可能目标词的概率。

总结一下，这大体上就是一个可以找到词嵌入的简化模型和神经网络（上图编号 2 所示），其实就是个 **softmax** 单元。矩阵$E$将会有很多参数，所以矩阵$E$有对应所有嵌入向量$e_c$的参数（上图编号 6 所示），**softmax** 单元也有$\theta_t$的参数（上图编号 3 所示）。如果优化这个关于所有这些参数的损失函数，你就会得到一个较好的嵌入向量集，这个就叫做 **Skip-Gram 模型**。它把一个像 **orange** 这样的词作为输入，并预测这个输入词，从左数或从右数的某个词，预测上下文词的前面一些或者后面一些是什么词。

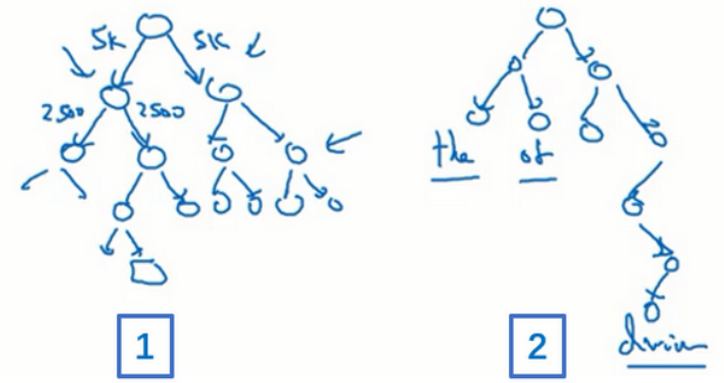
实际上使用这个算法会遇到一些问题，首要的问题就是计算速度。尤其是在 **softmax** 模型中，每次你想要计算这个概率，你需要对你词汇表中的所有 10,000 个词做求和计算，可能 10,000 个词的情况还不算太差。如果你用了一个大小为 100,000 或 1,000,000 的词汇表，那么这个分母的求和操作是相当慢的，实际上 10,000 已经是相当慢的了，所以扩大词汇表就更加困难了。



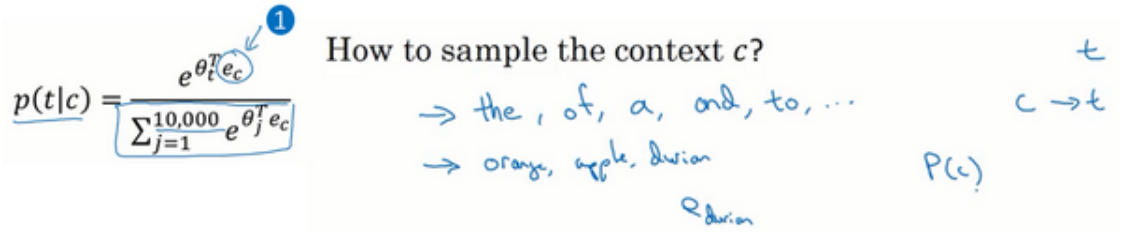

这里有一些解决方案，如**分级（hierarchical）的 softmax 分类器和负采样（Negative Sampling）**。

在文献中你会看到的方法是使用一个分级（**hierarchical**）的 **softmax** 分类器，意思就是说不是一下子就确定到底是属于 10,000 类中的哪一类。想象如果你有一个分类器（上图编号 1 所示），它告诉你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，假如这个二分类器告诉你这个词在前 5000 个词中（上图编号 2 所示），然后第二个分类器会告诉你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，诸如此类，

直到最终你找到一个词准确所在的分类器（上图编号 3 所示），那么就是这棵树的一个叶子节点。像这样有一个树形的分类器，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。实际上用这样的分类树，计算成本与词汇表大小的对数成正比（上图编号 4 所示），而不是词汇表大小的线性函数，这个就叫做分级 **softmax** 分类器。




我要提一下，在实践中分级 **softmax** 分类器不会使用一棵完美平衡的分类树或者说一棵左边和右边分支的词数相同的对称树（上图编号 1 所示的分类树）。实际上，分级的 **softmax** 分类器会被构造成常用词在顶部，然而不常用的词像 **durian** 会在树的更深处（上图编号 2 所示的分类树），因为你想更常见的词会更频繁，所以你可能只需要少量检索就可以获得常用单词像 **the** 和 **of**。然而你更少见到的词比如 **durian** 就更合适在树的较深处，因为你一般不需要到那样的深处，所以有不同的经验法则可以帮助构造分类树形成分级 **softmax** 分类器。所以这是你能在文献中见到的一个加速 **softmax** 分类的方法，但是我不会再花太多时间在这上面了，你可以从我在第一张幻灯片中提到的 **Tomas Mikolov** 等人的论文中参阅更多的细节，所以我不会再花更多时间讲这个了。因为在下个视频中，我们会讲到另一个方法叫做负采样，我感觉这个会更简单一点，对于加速 **softmax** 和解决需要在分母中对整个词汇表求和的问题也很有作用，下个视频中你会看到更多的细节。

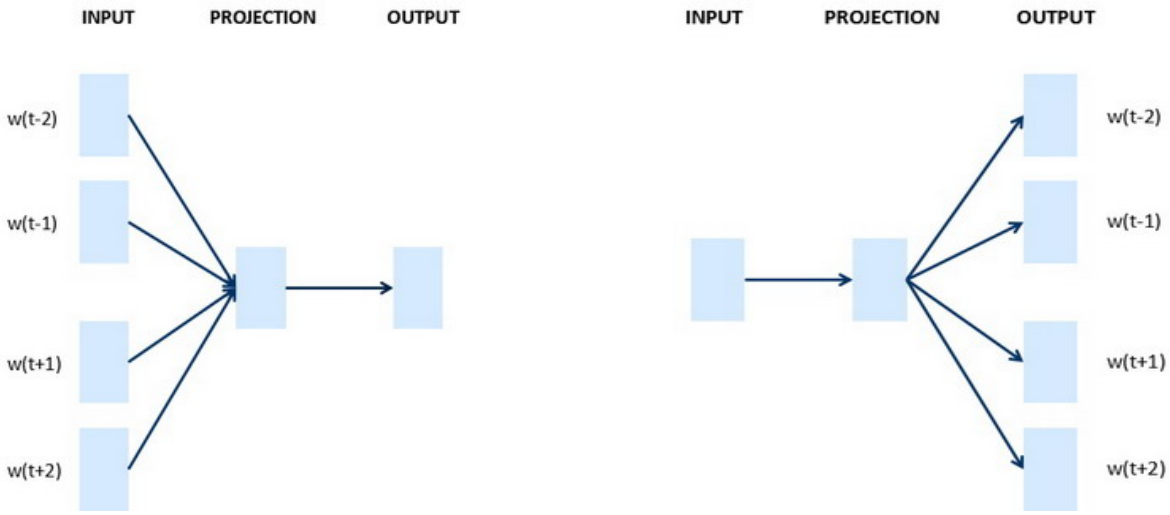



但是在进入下个视频前，我想要你理解一个东西，那就是怎么对上下文 **c** 进行采样，一旦你对上下文 **c** 进行采样，那么目标词 **t** 就会在上下文 **c** 的正负 10 个词距内进行采样。但是你要如何选择上下文 **c**？一种选择是你可以就对语料库均匀且随机地采样，如果你那么做，

你会发现有一些词，像 **the**、**of**、**a**、**and**、**to** 诸如此类是出现得相当频繁的，于是你那么做的话，你会发现你的上下文到目标词的映射会相当频繁地得到这些种类的词，但是其他词，像 **orange**、**apple** 或 **durian** 就不会那么频繁地出现了。你可能不会想要你的训练集都是这些出现得很频繁的词，因为这会导致你花大部分的力气来更新这些频繁出现的单词的$e_c$（上图编号 1 所示），但你想要的是花时间来更新像 **durian** 这些更少出现的词的嵌入，即$e_{\text{durian}}$。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。

这就是 **Word2Vec** 的 **Skip-Gram** 模型，如果你读过我之前提到的论文原文，你会发现那篇论文实际上有两个不同版本的 **Word2Vec** 模型，**Skip-Gram** 只是其中的一个，另一个叫做 **CBOW**，即连续词袋模型（**Continuous Bag-Of-Words Model**），它获得中间词两边的的上下文，然后用周围的词去预测中间的词，这个模型也很有效，也有一些优点和缺点。

（下图左边为**CBOW**，右边为**Skip-Gram**）




**CBOW对小型数据库比较合适，而Skip-Gram在大型语料中表现更好。**

总结下：**CBOW** 是从原始语句推测目标字词；而 **Skip-Gram** 正好相反，是从目标字词推测出原始语句。而刚才讲的 **Skip-Gram** 模型，关键问题在于 **softmax** 这个步骤的计算成本非常昂贵，因为它需要在分母里对词汇表中所有词求和。通常情况下，**Skip-Gram** 模型用到更多点。在下个视频中，我会展示给你一个算法，它修改了训练目标使其可以运行得更有效，因此它可以让你应用在一个更大的训练集上面，也可以学到更好的词嵌入。

## 2.7 负采样（Negative Sampling）

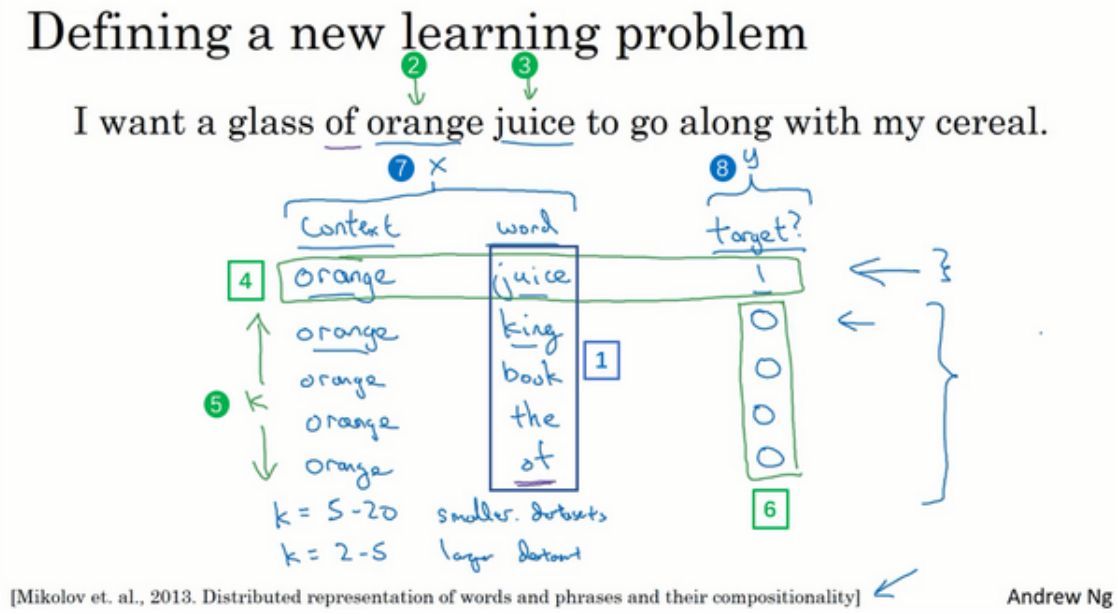
在上个视频中，你见到了 **Skip-Gram** 模型如何帮助你构造一个监督学习任务，把上下文映射到了目标词上，它如何让你学到一个实用的词嵌入。但是它的缺点就在于 **softmax** 计算起来很慢。在本视频中，你会看到一个改善过的学习问题叫做负采样，它能做到与你刚才看到的 **Skip-Gram** 模型相似的事情，但是用了一个更加有效的学习算法，让我们来看看这是怎么做到的。

在本视频中大多数的想法源于 **Tomas Mikolov**，**Ilya Sutskever**，**Kai Chen**，**Greg Corrado** 和 **Jeff Dean**。

（**Mikolov T, Sutskever I, Chen K, et al. Distributed Representations of Words and Phrases and their Compositionality[J]. 2013, 26:3111-3119.**）

我们在这个算法中要做的是构造一个新的监督学习问题，那么问题就是给定一对单词，比如 **orange** 和 **juice**，我们要去预测这是否是一对**上下文词-目标词（context-target）**。

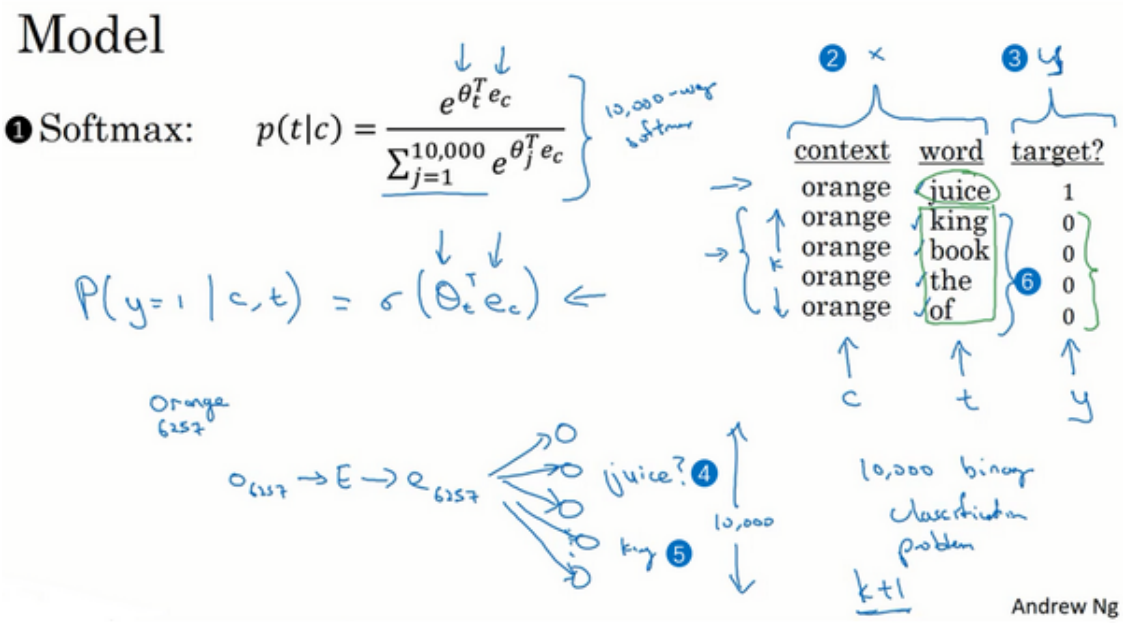
在这个例子中 **orange** 和 **juice** 就是个正样本，那么 **orange** 和 **king** 就是个负样本，我们把它标为 0。我们要做的就是采样得到一个上下文词和一个目标词，在这个例子中就是 **orange** 和 **juice**，我们用 1 作为标记，我把中间这列（下图编号 1 所示）叫做词（**word**）。这样生成一个正样本，正样本跟上个视频中生成的方式一模一样，先抽取一个上下文词，在一定词距内比如说正负 10 个词距内选一个目标词，这就是生成这个表的第一行，即 **orange–juice -1** 的过程。然后为了生成一个负样本，你将用相同的上下文词，再在字典中随机选一个词，在这里我随机选了单词 **king**，标记为 0。然后我们再拿 **orange**，再随机从词汇表中选一个词，因为我们设想，如果随机选一个词，它很可能跟 **orange** 没关联，于是 **orange–book–0**。我们再选点别的，**orange** 可能正好选到 **the**，然后是 0。还是 **orange**，再可能正好选到 **of** 这个词，再把这个标记为 0，注意 **of** 被标记为 0，即使 **of** 的确出现在 **orange** 词的前面。




总结一下，生成这些数据的方式是我们选择一个上下文词（上图编号 2 所示），再选一个目标词（上图编号 3 所示），这（上图编号 4 所示）就是表的第一行，它给了一个正样本，上下文，目标词，并给定标签为 1。然后我们要做的是给定几次，比如$K$次（上图编号 5 所示），我们将用相同的上下文词，再从字典中选取随机的词，**king**、**book**、**the**、**of** 等，从词典中任意选取的词，并标记 0，这些就会成为负样本（上图编号 6 所示）。出现以下情况也没关系，就是如果我们从字典中随机选到的词，正好出现在了词距内，比如说在上下文词 **orange** 正负 10 个词之内。

接下来我们将构造一个监督学习问题，其中学习算法输入$x$，输入这对词（上图编号 7 所示），要去预测目标的标签（上图编号 8 所示），即预测输出$y$。因此问题就是给定一对词，像 **orange** 和 **juice**，你觉得它们会一起出现么？你觉得这两个词是通过对靠近的两个词采样获得的吗？或者你觉得我是分别在文本和字典中随机选取得到的？这个算法就是要分辨这两种不同的采样方式，这就是如何生成训练集的方法。

那么如何选取$K$？**Mikolov** 等人推荐小数据集的话，$K$从 5 到 20 比较好。如果你的数据集很大，$K$就选的小一点。对于更大的数据集$K$就等于 2 到 5，数据集越小$K$就越大。那么在这个例子中，我们就用$K = 4$。




下面我们讲讲学习从$x$映射到$y$的监督学习模型，这（上图编号 1 所示:$Softmax: p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$）的 **softmax** 模型。这是我们从上张幻灯片中得到的训练集，这个（上图编号 2 所示）将是新的输入$x$，这个（上图编号 3 所示）将是你要预测的值$y$。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，我再用$y$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：

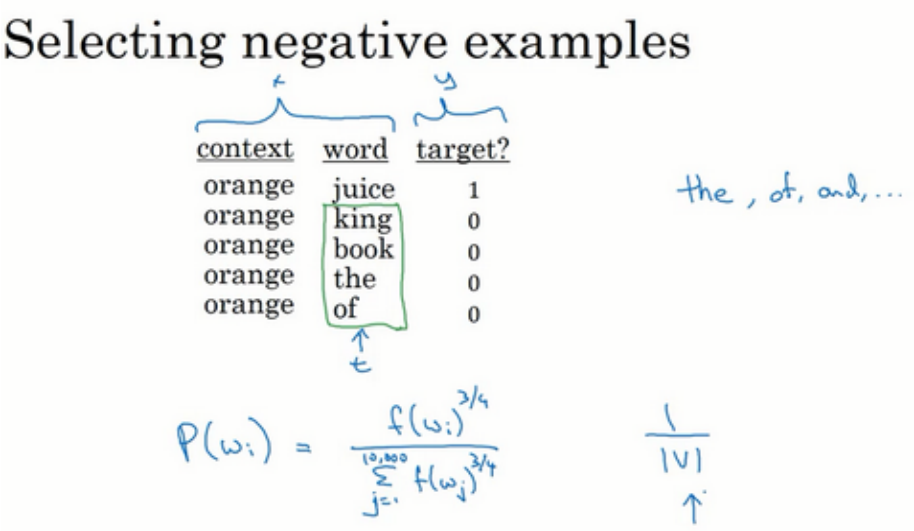
$$P(y = 1|c, t) = \sigma(\theta_t^T e_c)$$

这个模型基于逻辑回归模型，但不同的是我们将一个 **sigmoid** 函数作用于$\theta_t^T e_c$，参数和之前一样，你对每一个可能的目标词有一个参数向量$\theta_t$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。如果你有$K$个样本，你可以把这个看作$\frac{1}{K}$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。

我们把这个画成一个神经网络，如果输入词是 **orange**，即词 6257，你要做的就是输入 **one-hot** 向量，再传递给$E$，通过两者相乘获得嵌入向量$e_{6257}$，你就得到了 10,000 个可能的逻辑回归分类问题，其中一个（上图编号 4 所示）将会是用来判断目标词是否是 **juice** 的分类器，还有其他的词，比如说可能下面的某个分类器（上图编号 5 所示）是用来预测 **king** 是否是目标词，诸如此类，预测词汇表中这些可能的单词。把这些看作 10,000 个二分类逻辑回归分类器，但并不是每次迭代都训练全部 10,000 个，我们只训练其中的 5 个，我们要训

练对应真正目标词那一个分类器，再训练 4 个随机选取的负样本，这就是$K = 4$的情况。所以不使用一个巨大的 10,000 维度的 **softmax**，因为计算成本很高，而是把它转变为 10,000 个二分类问题，每个都很容易计算，每次迭代我们要做的只是训练它们其中的 5 个，一般而言就是$K + 1$个，其中$K$个负样本和 1 个正样本。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。

你也会在本周的编程练习中用到这个算法，这个技巧就叫负采样。因为你做的是，你有一个正样本词 **orange** 和 **juice**，然后你会特意生成一系列负样本，这些（上图编号 6 所示）是负样本，所以叫负采样，即用这 4 个负样本训练，4 个额外的二分类器，在每次迭代中你选择 4 个不同的随机的负样本词去训练你的算法。




这个算法有一个重要的细节就是如何选取负样本，即在选取了上下文词 **orange** 之后，你如何对这些词进行采样生成负样本？一个办法是对中间的这些词进行采样，即候选的目标词，你可以根据其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。所以论文的作者 **Mikolov** 等人根据经验，他们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，他们采用以下方式：

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=1}^{10,000} f(w_j)^{\frac{3}{4}}}$$

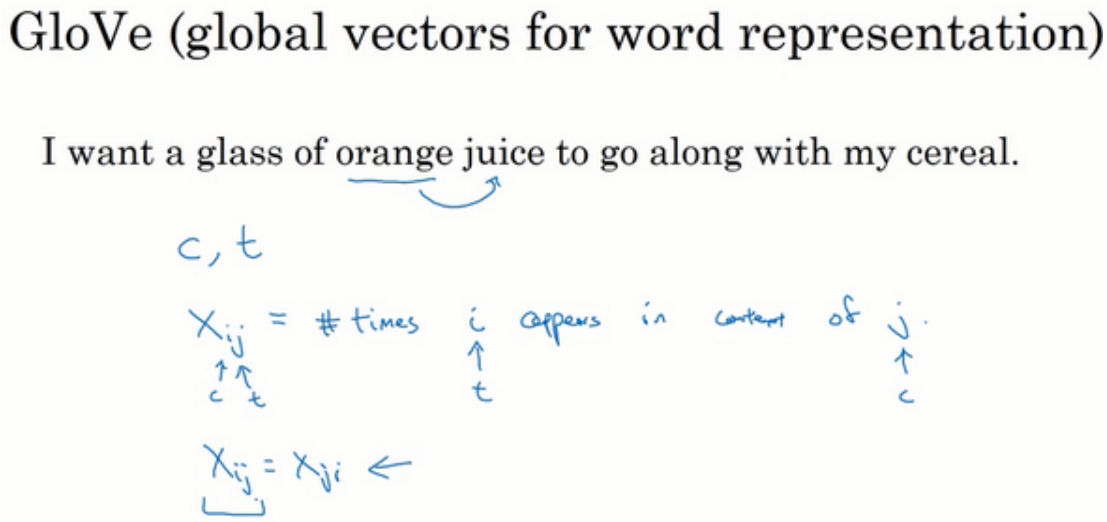
进行采样，所以如果$f(w_i)$是观测到的在语料库中的某个英文词的词频，通过$\frac{3}{4}$次方的计算，使其处于完全独立的分布和训练集的观测分布两个极端之间。我并不确定这是否有理论证明，但是很多研究者现在使用这个方法，似乎也效果不错。

总结一下，你已经知道了在 **softmax** 分类器中如何学到词向量，但是计算成本很高。在这个视频中，你见到了如何通过将其转化为一系列二分类问题使你可以非常有效的学习词向量。如果你使用这个算法，你将可以学到相当好的词向量。当然和深度学习的其他领域一样，有很多开源的实现，当然也有预训练过的词向量，就是其他人训练过的然后授权许可发布在网上的，所以如果你想要在 **NLP** 问题上取得进展，去下载其他人的词向量是很好的方法，在此基础上改进。

**Skip-Gram** 模型就介绍到这里，在下个视频中，我会跟你分享另一个版本的词嵌入学习算法 **GloVe**，而且这可能比你之前看到的都要简单。

## 2.8 GloVe 词向量（GloVe Word Vectors）

你已经了解了几个计算词嵌入的算法，另一个在 **NLP** 社区有着一定势头的算法是 **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，我认为可能是因为它简便吧，我们来看看这个算法。




**Glove** 算法是由 **Jeffrey Pennington**，**Richard Socher** 和 **Chris Manning** 发明的。

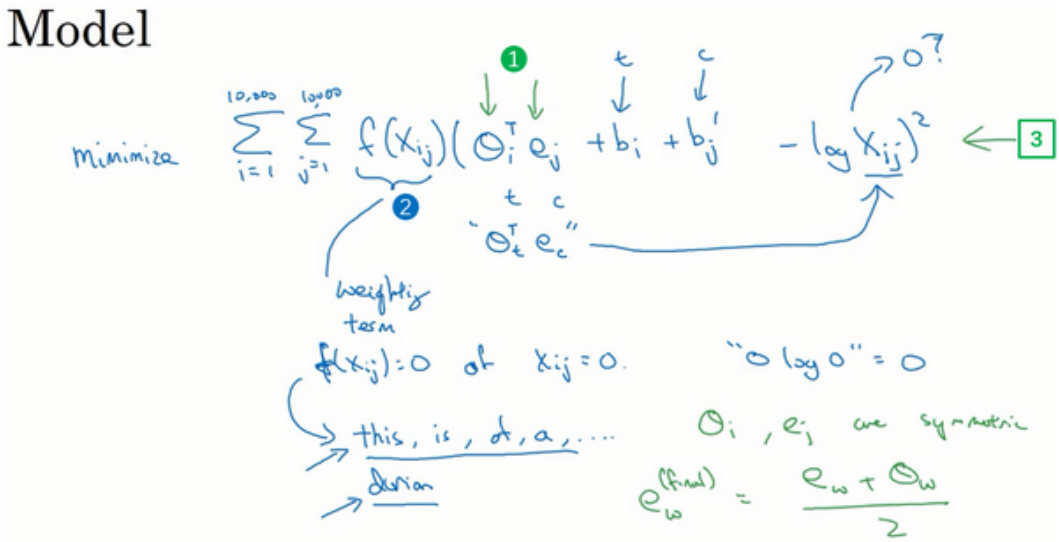
**(Pennington J, Socher R, Manning C. Glove: Global Vectors for Word Representation[C]// Conference on Empirical Methods in Natural Language Processing. 2014:1532-1543.)**

**GloVe** 代表用词表示的全局变量（**global vectors for word representation**）。在此之前，我们曾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，**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，那么这里$i$和$j$就和$t$和$c$的功能一样，所以你可以认为$X_{ij}$等同于$X_{tc}$。你也可以遍历你的训练集，然后数出单词$i$在不同单词$j$上下文中出现的个数，单词$t$在不同单词$c$的上下文中共出现多少次。根据上下文和目标词的定义，你大概会得出$X_{ij}$等于$X_{ji}$这个结论。事实上，如果你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。如果你对上下文的选择是，上下文总是目标词前一个单词的话，那么$X_{ij}$和$X_{ji}$就不会像这样对称了。不过对于 **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，我们将他们之间的差距进行最小化处理：

$$\text{minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij})(\theta_i^T e_j + b_i + b_j' - logX_{ij})^2$$

其中$\theta_i^T e_j$，想一下$i$和$j$与$t$和$c$的功能一样，因此这就和你之前看的有些类似了，即$\theta_t^T e_c$。同时对于这个（$\theta_t^T e_c$，下图编号 1 所示）来说，你想要知道的是告诉你这两个单词之间有多少联系，$t$和$c$之间有多紧密，$i$和$j$之间联系程度如何，换句话说就是他们同时出现的频率是多少，这是由这个$X_{ij}$影响的。然后，我们要做的是解决参数$\theta$和$e$的问题，然后准备用梯度下降来最小化上面的公式，你只想要学习一些向量，这样他们的输出能够对这两个单词同时出现的频率进行良好的预测。



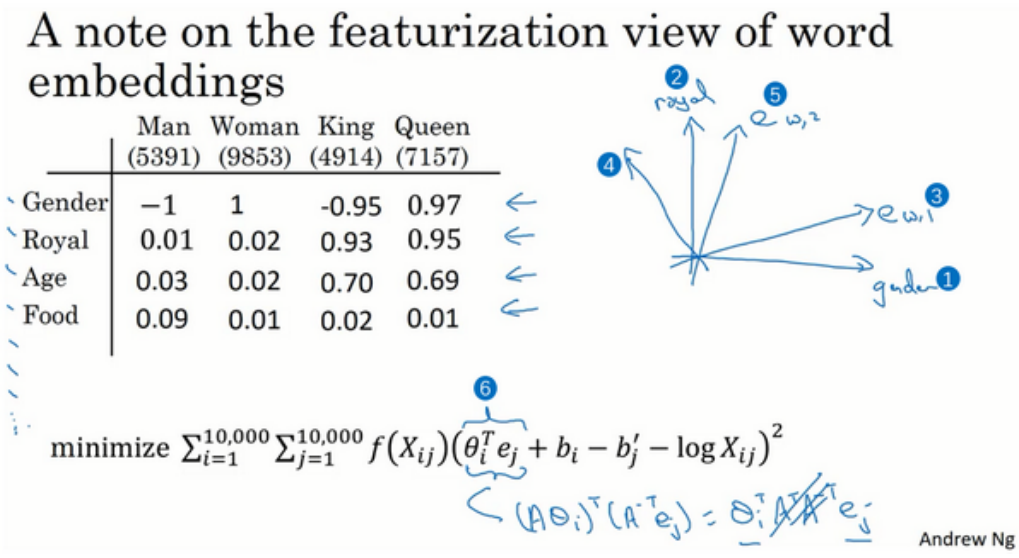

现在一些附加的细节是如果$X_{ij}$是等于 0 的话，那么$log0$就是未定义的，是负无穷大的，所以我们想要对$X_{ij}$为 0 时进行求和，因此要做的就是添加一个额外的加权项$f(X_{ij})$（上图编号 2 所示）。如果$X_{ij}$等于 0 的话，同时我们会用一个约定，即$0log0 = 0$，这个的意思是如果$X_{ij} = 0$，先不要进行求和，所以这个$log0$项就是不相关项。上面的求和公式表明，这个和仅是一个上下文和目标词关系里连续出现至少一次的词对的和。$f(X_{ij})$的另一个作用是，有些词在英语里出现十分频繁，比如说 **this**，**is**，**of**，**a** 等等，有些情况，这叫做**停止词**，但是在频繁词和不常用词之间也会有一个连续统（**continuum**）。不过也有一些不常用的词，比如 **durion**，你还是想将其考虑在内，但又不像那些常用词这样频繁。因此，这个加权因子$f(X_{ij})$就可以是一个函数，即使是像 **durion** 这样不常用的词，它也能给予大量有意义的运算，同时也能够给像 **this**，**is**，**of**，**a** 这样在英语里出现更频繁的词更大但不至于过分的权重。因此有一些对加权函数$f$的选择有着启发性的原则，就是既不给这些词（**this**，**is**，**of**，**a**）过分的权重，也不给这些不常用词（**durion**）太小的权值。如果你想要知道 f 是怎么能够启发性地

完成这个功能的话，你可以看一下我之前的幻灯片里引用的 **GloVe** 算法论文。

最后，一件有关这个算法有趣的事是$\theta$和$e$现在是完全对称的，所以那里的$\theta_i$和$e_j$就是对称的。如果你只看数学式的话，他们（$\theta_i$和$e_j$）的功能其实很相近，你可以将它们颠倒或者将它们进行排序，实际上他们都输出了最佳结果。因此一种训练算法的方法是一致地初始化$\theta$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$w$，你就会有$e_w^{(final)} = \frac{e_w+\theta_w}{2}$。因为$\theta$和$e$在这个特定的公式里是对称的，而不像之前视频里我们了解的模型，$\theta$和$e$功能不一样，因此也不能像那样取平均。

这就是 **GloVe** 算法的内容，我认为这个算法的一个疑惑之处是如果你看着这个等式，它实在是太简单了，对吧？仅仅是最小化，像这样的一个二次代价函数（上图编号 3 所示）是怎么能够让你学习有意义的词嵌入的呢？但是结果证明它确实有效，发明者们发明这个算法的过程是他们以历史上更为复杂的算法，像是 **newer language** 模型，以及之后的 **Word2Vec**、**Skip-Gram** 模型等等为基础，同时希望能够简化所有之前的算法才发明的。

在我们总结词嵌入学习算法之前，有一件更优先的事，我们会简单讨论一下。就是说，我们以这个特制的表格作为例子来开始学习词向量，我们说，第一行的嵌入向量是来表示 **Gender** 的，第二行是来表示 **Royal** 的，然后是是 **Age**，在之后是 **Food** 等等。但是当你在使用我们了解过的算法的一种来学习一个词嵌入时，例如我们之前的幻灯片里提到的 **GloVe** 算法，会发生一件事就是你不能保证嵌入向量的独立组成部分是能够理解的，为什么呢？




假设说有个空间，里面的第一个轴（上图编号 1 所示）是 **Gender**，第二个轴（上图编号 2 所示）是 **Royal**，你能够保证的是第一个嵌入向量对应的轴（上图编号 3 所示）是和这个轴（上面提到的第一和第二基轴，编号 1，2 所示）有联系的，它的意思可能是 **Gender**、

**Royal**、**Age** 和 **Food**。具体而言，这个学习算法会选择这个（上图编号 3 所示）作为第一维的轴，所以给定一些上下文词，第一维可能是这个轴（上图编号 3 所示），第二维也许是这个（上图编号 4 所示），或者它可能不是正交的，它也可能是第二个非正交轴（上图编号 5 所示），它可以是你学习到的词嵌入中的第二部分。当我们看到这个（上图编号 6 所示）的时候，如果有某个可逆矩阵$A$，那么这项（上图编号 6 所示）就可以简单地替换成$(A\theta_i)^T(A^{-T}e_j)$，因为我们将其展开：

$$(A\theta_i)^T\left(A^{-T}e_j\right) = \theta_i^T A^T A^{-T} e_j = \theta_i^T e_j$$

不必担心，如果你没有学过线性代数的话会，和这个算法一样有一个简单证明过程。你不能保证这些用来表示特征的轴能够等同于人类可能简单理解的轴，具体而言，第一个特征可能是个 **Gender**、**Roya**、**Age**、**Food Cost** 和 **Size** 的组合，它也许是名词或是一个行为动词和其他所有特征的组合，所以很难看出独立组成部分，即这个嵌入矩阵的单行部分，然后解释出它的意思。尽管有这种类型的线性变换，这个平行四边形映射也说明了我们解决了这个问题，当你在类比其他问题时，这个方法也是行得通的。因此尽管存在特征量潜在的任意线性变换，你最终还是能学习出解决类似问题的平行四边形映射。

这就是词嵌入学习的内容，你现在已经了解了一些学习词嵌入的算法了，你可以在本周的编程练习里更多地运用它们。下节课讲解怎样使用这些算法来解决情感分类问题。