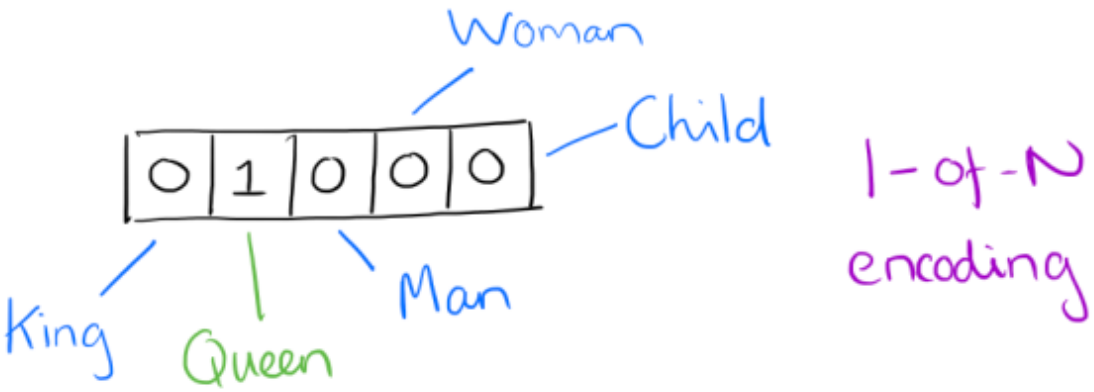
声明：本笔记主要源自于word2vec原理(一) CBOW与Skip-Gram模型基础 、word2vec原理(二) 基于Hierarchical Softmax的模型、word2vec原理(三) 基于Negative Sampling的模型 ，作者为刘建平Pinard，笔者修改和添加了部分内容。

Word2Vec是由Google在2013年推出的一款自然语言处理（NLP）工具。其核心优势在于能够将文本中的词语转换成数值化的向量形式，从而使计算机能够理解和处理人类语言。通过将词语转化为向量，Word2Vec不仅能够度量词语间的相似性，还能够捕捉词语间复杂的语义关系，如性别、复数等概念。虽然源码是开源的，但是谷歌的代码库国内无法访问，因此本文的讲解word2vec原理以Github上的word2vec代码为准。

1. 词向量基础

在Word2Vec出现之前，词向量的概念已经存在，但早期的实现方式较为粗略。最早的词向量是很冗长的，它使用是词向量维度大小为整个词汇表的大小，对于每个具体的词汇表中的词，将对应的位置置为1。比如我们有下面的5个词组成的词汇表，词"Queen"的序号为2， 那么它的词向量就是$(0,1,0,0,0)$。同样的道理，词"Woman"的词向量就是 $(0,0,0,1,0)$。这种词向量的编码方式我们一般叫做**1-of-N representation或者one hot representation**.



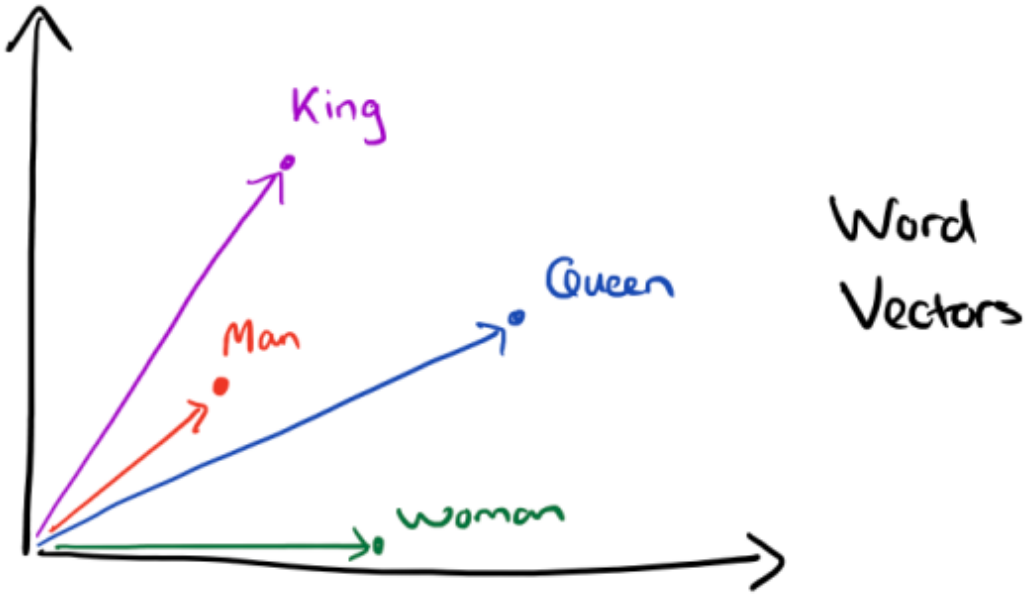
One hot representation用来表示词向量非常简单，但是问题是我们的词汇表一般非常大，可能达到百万级别，这样每个词都用百万维的向量来表示简直是内存的灾难。这样的向量其实除了一个位置是1，其余的位置全部都是0，表达的效率不高，能不能把词向量的维度变小呢？

为了克服这些问题，研究人员提出了**分布式表示**（Distributed Representation）。它的思路是通过训练，将每个词都映射到一个较短的词向量上来。所有的这些词向量就构成了向量空间，进而可以用普通的统计学的方法来研究词与词之间的关系。这个较短的词向量维度是多大呢？这个一般需要我们在训练时自己来指定。

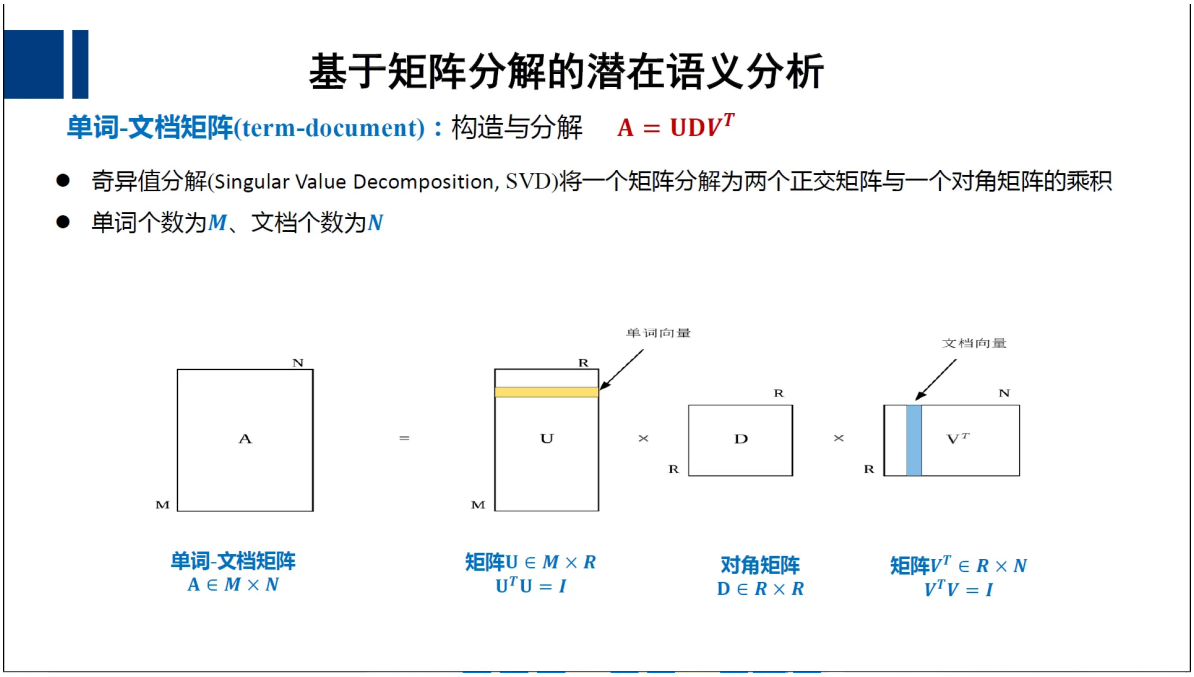
比如下图我们将词汇表里的词用"Royalty","Masculinity", "Femininity"和"Age"4个维度来表示，King这个词对应的词向量可能是 $(0.99,0.99,0.05,0.7)$。当然在实际情况中，我们并不能对词向量的每个维度做一个很好的解释。

有了用Distributed Representation表示的较短的词向量，我们就可以较容易的分析词之间的关系了，比如我们将词的维度降维到二维，有一个有趣的研究表明，用下图的词向量表示我们的词时，我们可以发现：

$$\vec{King} - \vec{Man} + \vec{Woman} = \vec{Queen}$$



当然除了Word2Vec之外，还有一种重要的词向量生成方法是**基于矩阵分解的潜在语义分析**（Latent Semantic Analysis, LSA）。LSA通过将文档-词矩阵分解为低秩矩阵，从而提取出潜在的主题或语义信息，这里不多提及。



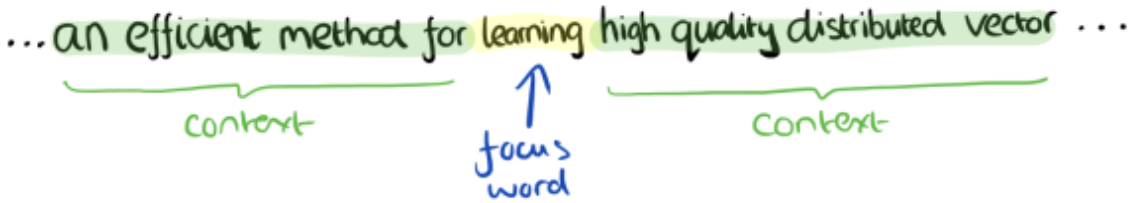
2. CBOW与Skip-Gram用于神经网络语言模型

在word2vec出现之前，已经有用神经网络DNN来用训练词向量进而处理词与词之间的关系了。采用的方法一般是一个三层的深度神经网络结构，分为输入层，隐藏层和输出层(softmax层)。

那这个模型是如何定义数据的输入和输出呢？一般分为**CBOW(Continuous Bag-of-Words)**与**Skip-Gram**两种模型。

CBOW模型的训练输入是某一个特征词的上下文相关的词对应的词向量，而输出就是这特定的一个词的词向量。比如下面这段话，我们的上下文大小取值为4，特定的这个词是"Learning"，也就是我们需要的输出词向量，上下文对应的词有8个，前后各4个，这8个词是我们模型的输入。由于CBOW使用的是**词袋模型**，因此这8个词都是平等的，也就是不考虑他们和我们关注的词之间的距离大小，只要在我们上下文之内即可。

> **“词袋模型”**（Bag of Words, BoW）是一种简化的方法，用于表示文本数据。在词袋模型中，文本（如句子或文档）被表示为词的集合，而不考虑词的顺序和结构。具体来说，词袋模型将每个词看作是一个独立的元素，而忽略了它们在原始文本中的位置和顺序信息。
>
> 在Continuous Bag of Words (CBOW) 模型中，目标是根据一个词的上下文词来预测该词。
>
> ### 示例
>
> 假设我们有一个句子 "The cat sat on the mat"，并且定义了一个大小为2的上下文窗口。对于中心词 "sat"，其上下文词为 "cat" 和 "on"。在CBOW模型中，这两个词会被视为平等的输入，用来预测中心词 "sat"。
>
> ### 与Skip-gram的对比
>
> - **CBOW**：使用上下文词来预测中心词，上下文词被视为平等的。
> - **Skip-gram**：使用中心词来预测上下文词，更注重词与词之间的关系和距离。



这样我们这个CBOW的例子里，我们的输入是8个词向量，输出是所有词的softmax概率（训练的目标是期望训练样本特定词对应的softmax概率最大），对应的CBOW神经网络模型输入层有8个神经元，输出层有词汇表大小个神经元。隐藏层的神经元个数我们可以自己指定。通过DNN的反向传播算法，我们可以求出DNN模型的参数，同时得到所有的词对应的词向量。这样当我们有新的需求，要求出某8个词对应的最可能的输出中心词时，我们可以通过一次DNN前向传播算法并通过softmax激活函数找到概率最大的词对应的神经元即可。

Skip-Gram模型和CBOW的思路是反着来的，即输入是特定的一个词的词向量，而输出是特定词对应的上下文词向量。还是上面的例子，我们的上下文大小取值为4， 特定的这个词"Learning"是我们的输入，而这8个上下文词是我们的输出。

这样我们这个Skip-Gram的例子里，我们的输入是特定词， 输出是softmax概率排前8的8个词，对应的Skip-Gram神经网络模型输入层有1个神经元，输出层有词汇表大小个神经元。隐藏层的神经元个数我们可以自己指定。通过DNN的反向传播算法，我们可以求出DNN模型的参数，同时得到所有的词对应的词向量。这样当我们有新的需求，要求出某1个词对应的最可能的8个上下文词时，我们可以通过一次DNN前向传播算法得到概率大小排前8的softmax概率对应的神经元所对应的词即可。

以上就是**神经网络语言模型**中如何用CBOW与Skip-Gram来训练模型与得到词向量的大概过程。但是这和word2vec中用CBOW与Skip-Gram来训练模型与得到词向量的过程有很多的不同。

word2vec为什么不用现成的DNN模型，要继续优化出新方法呢？**最主要的问题是DNN模型的这个处理过程非常耗时。**我们的词汇表一般在百万级别以上，这意味着我们DNN的输出层需要进行softmax计算各个词的输出概率的的计算量很大。

3. word2vec基础之霍夫曼树

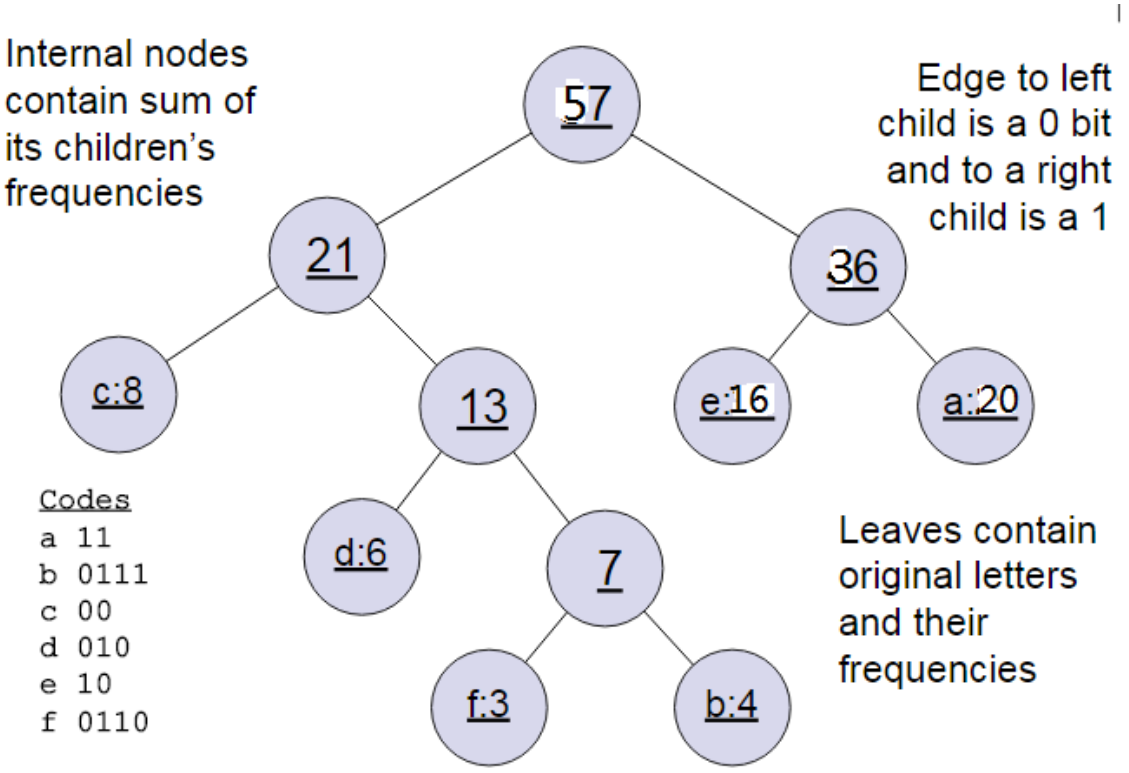
word2vec也使用了CBOW与Skip-Gram来训练模型与得到词向量，但是并没有使用传统的DNN模型。最先优化使用的数据结构是用**霍夫曼树**来代替隐藏层和输出层的神经元，霍夫曼树的叶子节点起到输出层神经元的作用，叶子节点的个数即为词汇表的小大。 而内部节点则起到隐藏层神经元的作用。

这里我们先复习下霍夫曼树。霍夫曼树的建立其实并不难，过程如下：

> 输入：权值为 $(w_1, w_2, \ldots, w_n)$ 的 n 个节点
>
> 输出：对应的霍夫曼树
>
> 1）将 $(w_1, w_2, \ldots, w_n)$ 看做是有 n 棵树的森林，每个树仅有一个节点。
>
> 2）在森林中选择根节点权值最小的两棵树进行合并，得到一个新的树，这两颗树分布作为新树的左右子树。新树的根节点权重为左右子树的根节点权重之和。
>
> 3） 将之前的根节点权值最小的两棵树从森林删除，并把新树加入森林。
>
> 4）重复步骤2）和3）直到森林里只有一棵树为止。

下面我们用一个具体的例子来说明霍夫曼树建立的过程，我们有 (a, b, c, d, e, f) 共 6 个节点，节点的权值分布是 $(20, 4, 8, 6, 16, 3)$。

首先是最小的 b 和 f 合并，得到的新树根节点权重是 7 。此时森林里 5 棵树，根节点权重分别是$20, 8, 6, 16, 7$。此时根节点权重最小的 $6, 7$ 合并，得到新子树，依次类推，最终得到下面的霍夫曼树。



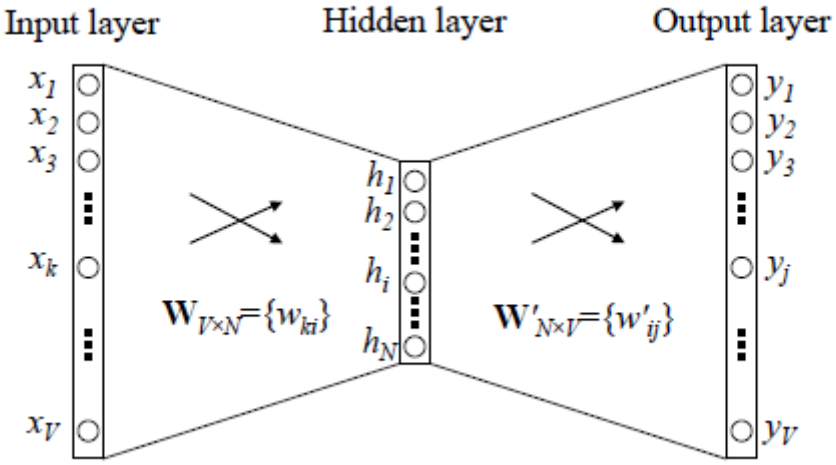
霍夫曼树有什么好处呢？一般得到霍夫曼树后我们会对叶子节点进行霍夫曼编码，由于权重高的叶子节点越靠近根节点，而权重低的叶子节点会远离根节点，这样我们的**高权重节点编码值较短**，而**低权重值编码值较长**。这保证的树的带权路径最短，也符合我们的信息论，即我们希望越常用的词拥有更短的编码。如何编码呢？一般对于一个霍夫曼树的节点（根节点除外），可以约定左子树编码为 0 ，右子树编码为 1 。上图则可以指定 c 的编码为 00。

在word2vec中，约定编码方式和上面的例子相反，即约定左子树编码为 1 ，右子树编码为 0 ，同时约定**左子树的权重不小于右子树的权重**。

word2vec有两种改进方法，一种是基于**Hierarchical Softmax**的，另一种是基于**Negative Sampling**的。以下是基于**Hierarchical Softmax**的探讨部分：

4. 基于Hierarchical Softmax的模型概述

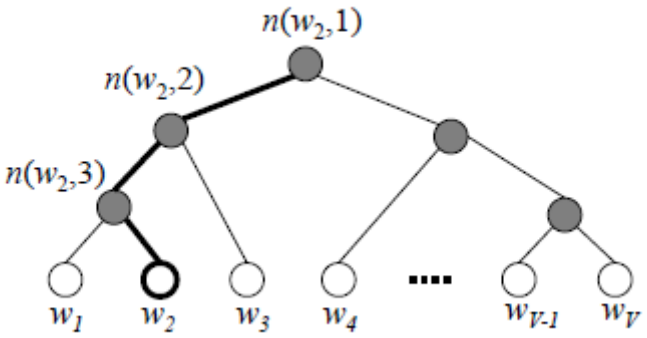
传统的神经网络词向量语言模型通常包含三层：输入层（词向量）、隐藏层和输出层（softmax层）。这种模型的主要问题是隐藏层到输出层的softmax计算量非常大，因为需要计算所有词的softmax概率，再找出概率最大的值。这在词汇表非常大时尤其耗时。



word2vec对这个模型做了改进，首先，对于从输入层到隐藏层的映射，没有采取神经网络的线性变换加激活函数的方法，而是采用简单的**对所有输入词向量求和并取平均**的方法。比如输入的是三个4维词向量：$(1,2,3,4),(9,6,11,8),(5,10,7,12)$，那么我们word2vec映射后的词向量就是$(5,6,7,8)$。由于这里是从多个词向量变成了一个词向量。

第二个改进就是从**隐藏层到输出的softmax层的计算量**上的改进。为了避免要计算所有词的softmax概率，word2vec采样了霍夫曼树来代替从隐藏层到输出softmax层的映射。具体来说，word2vec将softmax概率计算转换为沿着霍夫曼树的路径进行计算。

由于我们把之前所有都要计算的从输出softmax层的概率计算变成了一颗二叉霍夫曼树，那么我们的softmax概率计算只需要沿着树形结构进行就可以了。如下图所示，我们可以沿着霍夫曼树从根节点一直走到我们的叶子节点的词 w_2。



和之前的神经网络语言模型相比，我们的霍夫曼树的所有内部节点就类似之前神经网络隐藏层的神经元,其中，根节点的词向量对应我们的投影后的词向量，而所有叶子节点就类似于之前神经网络softmax输出层的神经元，叶子节点的个数就是词汇表的大小。在霍夫曼树中，隐藏层到输出层的softmax映射不是一下子完成的，而是沿着霍夫曼树一步步完成的，因此这种softmax取名为"Hierarchical Softmax"。

如何沿着霍夫曼树的路径进行计算呢？在word2vec中，我们采用了二元逻辑回归的方法，即规定沿着左子树走，那么就是负类（霍夫曼树编码 1），沿着右子树走，那么就是正类（霍夫曼树编码 0）。判别正类和负类的方法是使用sigmoid函数，即：

$$P(+)=\sigma(x_w^T\theta)=\frac{1}{1+\exp\left(-x_w^T\theta\right)}$$

其中 x_w 是当前内部节点的词向量，而 θ 则是我们需要从训练样本求出的逻辑回归的模型参数。

使用霍夫曼树有什么好处呢？首先，由于是二叉树，之前计算量为 V，现在变成了 $\log_2 V$。其次，由于使用霍夫曼树是高频的词靠近树根，这样高频词需要更少的时间会被找到，这符合我们的贪心优化思想。

不难理解，被划分为左子树而成为负类的概率为 $P(-) = 1 - P(+)$。在某一个内部节点，要判断是沿左子树还是右子树走的标准就是**看 $P(-), P(+)$ 谁的概率值大。**而控制 $P(-), P(+)$ 谁的概率值大的因素一个是当前节点的词向量，另一个是当前节点的模型参数 θ。

对于上图中的 w_2，如果它是一个训练样本的输出，那么我们期望对于里面的隐藏节点 $n(w_2, 1)$ 的 $P(-)$ 概率大，$n(w_2, 2)$ 的 $P(-)$ 概率大，$n(w_2, 3)$ 的 $P(+)$ 概率大。

> 为了更好地理解霍夫曼树在word2vec中的应用，我们可以通过一个具体的例子来说明。
>
> 假设我们有一个小型词汇表，包含以下5个词及其频率：
>
> | 词 | 频率 |
> | --- | --- |
> | the | 100 |
> | cat | 50 |
> | sat | 30 |
> | on | 20 |
> | mat | 10 |
>
> ## 构建霍夫曼树
>
> 根据这些词的频率，我们可以构建一棵霍夫曼树。频率高的词（如 "the"）会被放在靠近根节点的位置，路径较短；频率低的词（如 "mat"）会被放在远离根节点的位置，路径较长。
>
> ```
>         (210)
>       /      \
>  (150)        (60)
>   / \         /  \
> (100) (50) (30)  (30)
> the  cat  sat    |
>                / \
>              (20) (10)
>               on  mat
> ```
>
> ## 路径编码
>
> 每个词从根节点到叶子节点的路径可以用二进制序列表示。例如：
>
> - "the": 11
> - "cat": 10
> - "sat": 01
> - "on": 001
> - "mat": 000

> ## 层次softmax的计算
>
> 假设我们要预测词 "sat"，其路径为 110。路径上的内部节点为：
>
> - 第一个内部节点（根节点到第一个分支）：0
> - 第二个内部节点（第一个分支到第二个分支）：1
>
> ### 前向传播
>
> 对于路径上的每个内部节点，计算其概率：
>
> 1. **第一个内部节点**：
>
> $$P(0|v_{sat}) = \sigma(-v_1^T v_{sat})$$
>
> 其中，v_1 是第一个内部节点的权重向量，v_{sat} 是 `sat` 的向量表示。
>
> 2. **第二个内部节点**：
>
> $$P(1|v_{sat}) = \sigma(-v_2^T v_{sat})$$
>
> 其中，$v2$ 是第二个内部节点的权重向量。
>
> ### 损失函数
>
> 目标是最小化路径上所有内部节点的负对数似然：
>
> $$-[\log P(0|v_{sat}) + \log P(1|v_{sat})]$$
>
> 在传统的softmax方法中，每次更新参数时需要计算所有 V 个词的概率，计算复杂度为 $O(V)$。使用层次softmax时，只需要计算从根节点到目标词路径上的内部节点的概率。假设路径长度为 L，计算复杂度为 $O(L)$。通常 L 远小于 V，因此计算量大大减少。

回到基于Hierarchical Softmax的word2vec本身，我们的目标就是找到合适的所有节点的词向量和所有内部节点θ，使训练样本达到最大似然。那么如何达到最大似然呢？

5. 基于Hierarchical Softmax的模型梯度计算

我们使用最大似然法来寻找所有节点的词向量和所有内部节点 θ。先拿上面的 w_2 例子来看，我们期望最大化下面的似然函数：

$$\prod_{i=1}^{3} P(n(w_i), i) = \left(1 - \frac{1}{1+\exp\left(-x_w^T\theta_1\right)}\right)\left(1 - \frac{1}{1+\exp\left(-x_w^T\theta_2\right)}\right)\left(1 - \frac{1}{1+\exp\left(-x_w^T\theta_3\right)}\right)$$

对于所有的训练样本，我们期望最大化所有样本的似然函数乘积。

为了便于我们后面一般化的描述，我们定义输入的词为 w，其从输入层词向量求和平均后的霍夫曼树根节点词向量为 x_w，从根节点到 w 所在的叶子节点，包含的节点总数为 l_w，w 在霍夫曼树中从根节点开始，经过的第 i 个节点表示为 p_i^w，对应的霍夫曼编码为 $d_i^w \in 0, 1$，其中 $i = 2, 3, \cdots, l_w$。而该节点对应的模型参数表示为θ_i^w，其中 $i = 1, 2, \cdots, l_w - 1$，没有 $i = l_w$ 是因为模型参数仅仅针对于霍夫曼树的内部节点。

定义w经过的霍夫曼树某一个节点j的逻辑回归概率为 $P(d_j^w|x_w, \theta_{j-1}^w)$，其表达式为：

$$P(d_j^w|x_w, \theta_{j-1}^w) = \begin{cases} \sigma(x_w^T\theta_{j-1}^w) & , d_j^w = 0 \\ 1 - \sigma(x_w^T\theta_{j-1}^w) & , d_j^w = 1 \end{cases}$$

那么对于某一个目标输出词 w，其最大似然为：

$$\prod_{j=2}^{l_w} P(d_j^w|x_w,\theta_{j-1}^w) = \prod_{j=2}^{l_w} [\sigma(x_w^T\theta_{j-1}^w)]^{1-d_j^w}[1-\sigma(x_w^T\theta_{j-1}^w)]^{d_j^w}$$

在word2vec中，由于使用的是随机梯度上升法，所以并没有把所有样本的似然乘起来得到真正的训练集最大似然，仅仅每次只用一个样本更新梯度，这样做的目的是减少梯度计算量。这样我们可以得到 w 的对数似然函数 L 如下：

$$L = \log\prod_{j=2}^{l_w} P(d_j^w|x_w,\theta_{j-1}^w) = \sum_{j=2}^{l_w}(1-d_j^w)\log\sigma(x_w^T\theta_{j-1}^w) + d_j^w\log\left[1-\sigma(x_w^T\theta_{j-1}^w)\right]$$

要得到模型中 w 词向量和内部节点的模型参数 θ, 我们使用梯度上升法即可。首先我们求模型参数θ_{j-1}^w 的梯度：

$$\begin{aligned}\frac{\partial L}{\partial\theta_{j-1}^w} &= (1-d_j^w)\frac{\sigma(x_w^T\theta_{j-1}^w)(1-\sigma(x_w^T\theta_{j-1}^w)}{\sigma(x_w^T\theta_{j-1}^w)}x_w - d_j^w\frac{\sigma(x_w^T\theta_{j-1}^w)(1-\sigma(x_w^T\theta_{j-1}^w)}{1-\sigma(x_w^T\theta_{j-1}^w)}x_w \\ &= (1-d_j^w)(1-\sigma(x_w^T\theta_{j-1}^w))x_w - d_j^w\sigma(x_w^T\theta_{j-1}^w)x_w \\ &= (1-d_j^w-\sigma(x_w^T\theta_{j-1}^w))x_w\end{aligned}$$

同样的方法，可以求出 x_w 的梯度表达式如下：

$$\frac{\partial L}{\partial x_w} = \sum_{j=2}^{l_w}(1-d_j^w-\sigma(x_w^T\theta_{j-1}^w))\theta_{j-1}^w$$

有了梯度表达式，我们就可以用梯度上升法进行迭代来一步步的求解我们需要的所有的 θ_{j-1}^w 和 x_w。

6. 基于Hierarchical Softmax的CBOW模型

基于CBOW模型时，Hierarchical Softmax该如何使用呢?

首先我们要定义词向量的维度大小 M 以及CBOW的上下文大小 $2c$，这样我们对于训练样本中的每一个词，其前面的 c 个词和后面的 c 个词作为了CBOW模型的输入，该词本身作为样本的输出，期望softmax概率最大。

在做CBOW模型前，我们需要先将词汇表建立成一颗霍夫曼树。

对于从输入层到隐藏层（投影层），这一步比较简单，就是对 w 周围的 $2c$ 个词向量求和取平均即可，即：

$$x_w = \frac{1}{2c}\sum_{i=1}^{2c}x_i$$

第二步，通过梯度上升法来更新我们的 θ_{j-1}^w 和 x_w，注意这里的 x_w 是由 $2c$ 个词向量相加而成，我们做梯度更新完毕后会用梯度项直接更新原始的各个 $x_i(i=1,2\ldots,2c)$ ，即：

$$\theta_{j-1}^w = \theta_{j-1}^w + \eta(1-d_j^w-\sigma(x_w^T\theta_{j-1}^w))x_w$$

$$x_i = x_i + \eta\sum_{j=2}^{l_w}(1-d_j^w-\sigma(x_w^T\theta_{j-1}^w))\theta_{j-1}^w\,(i=1,2..,2c)$$

其中 η 为梯度上升法的步长。

这里总结下基于Hierarchical Softmax的CBOW模型算法流程，梯度迭代使用了随机梯度上升法：

> 输入：基于CBOW的语料训练样本，词向量的维度大小 M，CBOW的上下文大小 $2c$，步长 η
>
> 输出：霍夫曼树的内部节点模型参数 θ，所有的词向量 w
>
> 1. 基于语料训练样本建立霍夫曼树。
> 2. 随机初始化所有的模型参数 θ，所有的词向量 w
> 3. 进行梯度上升迭代过程，对于训练集中的每一个样本 $(context(w), w)$ 做如下处理：
>
> a) $e = 0$，计算 $x_w = \frac{1}{2c}\sum_{i=1}^{2c} x_i$
>
> b) for $j = 2$ to l_w，计算：
>
> $$f = \sigma(x_w^T \theta_{j-1}^w)$$
> $$g = (1 - d_j^w - f)\eta$$
> $$e = e + g\theta_{j-1}^w$$
> $$\theta_{j-1}^w = \theta_{j-1}^w + g x_w$$
>
> c) 对于 $context(w)$ 中的每一个词向量 x_i（共 $2c$ 个）进行更新：
>
> $$x_i = x_i + e$$
>
> d) 如果梯度收敛，则结束梯度迭代，否则回到步骤3继续迭代。

7. 基于Hierarchical Softmax的Skip-Gram模型

现在我们先看看基于Skip-Gram模型时， Hierarchical Softmax如何使用。此时输入的只有一个词 w，输出的为 $2c$ 个词向量 $context(w)$。

我们对于训练样本中的每一个词，该词本身作为样本的输入， 其前面的 c 个词和后面的 c 个词作为了Skip-Gram模型的输出,，期望这些词的softmax概率比其他的词大。

在做CBOW模型前，我们需要先将词汇表建立成一颗霍夫曼树。

对于从输入层到隐藏层（投影层），这一步比CBOW简单，由于只有一个词，所以，即 x_w 就是词 w 对应的词向量。

第二步，通过梯度上升法来更新我们的 θ_{j-1}^w 和 x_w，注意这里的 x_w 周围有 $2c$ 个词向量，此时如果我们期望 $P(x_i|x_w), i = 1, 2, \cdots, 2c$ 最大。此时我们注意到由于上下文是相互的，在期望 $P(x_i|x_w), i = 1, 2, \cdots, 2c$ 最大化的同时，反过来我们也期望 $P(x_w|x_i), i = 1, 2, \cdots, 2c$ 最大。那么是使用$P(x_i|x_w)$ 好还是 $P(x_w|x_i)$ 好呢，word2vec使用了后者，这样做的好处就是在一个迭代窗口内，我们不是只更新 x_w 一个词，而是 $xi, i = 1, 2, \ldots, 2c$ 共 $2c$ 个词。这样整体的迭代会更加的均衡。因为这个原因，Skip-Gram模型并没有和CBOW模型一样对输入进行迭代更新，而是对 $2c$ 个输出进行迭代更新。

这里总结下基于Hierarchical Softmax的Skip-Gram模型算法流程，梯度迭代使用了随机梯度上升法：

> 输入：基于Skip-Gram的语料训练样本，词向量的维度大小 M，Skip-Gram的上下文大小 $2c$，步长 η
>
> 输出：霍夫曼树的内部节点模型参数 θ，所有的词向量 w

1. 基于语料训练样本建立霍夫曼树。

2. 随机初始化所有的模型参数 θ，所有的词向量 w

3. 进行梯度上升迭代过程，对于训练集中的每一个样本 $(context(w), w)$ 做如下处理：

a) for $i = 1$ to $2c$:

i) $e = 0$

ii) for $j = 2$ to l_w，计算：

$$f = \sigma(x_i^T \theta_{j-1}^w)$$
$$g = (1 - d_j^w - f)\eta$$
$$e = e + g\theta_{j-1}^w$$
$$\theta_{j-1}^w = \theta_{j-1}^w + gx_i$$

iii) $x_i = x_i + e$

b) 如果梯度收敛，则结束梯度迭代，算法结束，否则回到步骤a继续迭代。

8. Hierarchical Softmax的模型源码和算法的对应

这里给出上面算法和word2vec源码https://github.com/tmikolov/word2vec/blob/master/word2vec.c中的变量对应关系。

在源代码中，基于Hierarchical Softmax的CBOW模型算法在435-463行，基于Hierarchical Softmax的Skip-Gram的模型算法在495-519行。

在源代码中，neule对应我们上面的 e，syn0对应我们的 x_w, syn1对应我们的 θ_{j-1}^i, layer1_size对应词向量的维度，window对应我们的 c。

另外，vocab[word].code[d]指的是，当前单词word的，第 d 个编码，编码不含Root结点。vocab[word].point[d]指的是，当前单词word，第 d 个编码下，前置的结点。

9. Hierarchical Softmax的缺点与改进

在讲基于Negative Sampling的word2vec模型前，我们先看看Hierarchical Softmax的的缺点。的确，使用霍夫曼树来代替传统的神经网络，可以提高模型训练的效率。但是如果我们的训练样本里的中心词 w 是一个很生僻的词，那么就得在霍夫曼树中辛苦的向下走很久了。能不能不用搞这么复杂的一颗霍夫曼树，将模型变的更加简单呢？

Negative Sampling就是这么一种求解word2vec模型的方法，它摒弃了霍夫曼树，采用了Negative Sampling（负采样）的方法来求解，下面我们就来看看Negative Sampling的求解思路。

10. 基于Negative Sampling的模型概述

既然名字叫Negative Sampling（负采样），那么肯定使用了采样的方法。采样的方法有很多种，比如之前讲到的大名鼎鼎的MCMC。我们这里的Negative Sampling采样方法并没有MCMC那么复杂。

比如我们有一个训练样本，中心词是 w，它周围上下文共有 $2c$ 个词，记为 $context(w)$。由于这个中心词 w，的确和 $context(w)$ 相关存在，因此它是一个真实的正例。通过Negative Sampling采样，我们得到neg个和 w 不同的中心词 $w_i, i = 1, 2, \cdots, neg$，这样 $context(w)$ 和 w_i 就组成了neg个并不真实存在的负例。利用这一个正例和neg个负例，我们进行二元逻辑回归，得到负采样对应每

个词 w_i 对应的模型参数 θ_i，和每个词的词向量。

从上面的描述可以看出，Negative Sampling由于没有采用霍夫曼树，每次只是通过采样neg个不同的中心词做负例，就可以训练模型，因此整个过程要比Hierarchical Softmax简单。

不过有两个问题还需要弄明白：1）如果通过一个正例和neg个负例进行二元逻辑回归呢？ 2）如何进行负采样呢？

11. 基于Negative Sampling的模型梯度计算

Negative Sampling也是采用了二元逻辑回归来求解模型参数，通过负采样，我们得到了neg个负例$(context(w), w_i), i = 1, 2, \cdots, neg$。为了统一描述，我们将正例定义为 w_0。

在逻辑回归中，我们的正例应该期望满足：

$$P(context(w_0), w_i) = \sigma(x_{w_0}^T \theta^{w_i}), y_i = 1, i = 0$$

我们的负例期望满足：

$$P(context(w_0), w_i) = 1 - \sigma(x_{w_0}^T \theta^{w_i}), y_i = 0, i = 1, 2, \ldots neg$$

我们期望可以最大化下式：

$$\prod_{i=0}^{neg} P(context(w_0), w_i) = \sigma(x_{w_0}^T \theta^{w_0}) \prod_{i=1}^{neg} (1 - \sigma(x_{w_0}^T \theta^{w_i}))$$

此时模型的似然函数为：

$$\prod_{i=0}^{neg} \sigma(x_{w_0}^T \theta^{w_i})^{y_i} (1 - \sigma(x_{w_0}^T \theta^{w_i}))^{1-y_i}$$

此时对应的对数似然函数为：

$$L = \sum_{i=0}^{neg} y_i log(\sigma(x_{w_0}^T \theta^{w_i})) + (1 - y_i) log(1 - \sigma(x_{w_0}^T \theta^{w_i}))$$

和Hierarchical Softmax类似，我们采用随机梯度上升法，仅仅每次只用一个样本更新梯度，来进行迭代更新得到我们需要的 $x_{w_i}, \theta^{w_i}, i = 0, 1, \cdots, neg$，这里我们需要求出 $x_{w_i}, \theta^{w_i}, i = 0, 1, \cdots, neg$ 的梯度。

首先我们计算 θ^{w_i} 的梯度：

$$\begin{aligned} \frac{\partial L}{\partial \theta^{w_i}} &= y_i (1 - \sigma(x_{w_0}^T \theta^{w_i})) x_{w_0} - (1 - y_i) \sigma(x_{w_0}^T \theta^{w_i}) x_{w_0} \\ &= (y_i - \sigma(x_{w_0}^T \theta^{w_i})) x_{w_0} \end{aligned}$$

同样的方法，我们可以求出 x_{w_0} 的梯度如下：

$$\frac{\partial L}{\partial x^{w_0}} = \sum_{i=0}^{neg} (y_i - \sigma(x_{w_0}^T \theta^{w_i})) \theta^{w_i}$$

有了梯度表达式，我们就可以用梯度上升法进行迭代来一步步的求解我们需要的 $x_{w_i}, \theta^{w_i}, i = 0, 1, \cdots, neg$。

12. Negative Sampling负采样方法

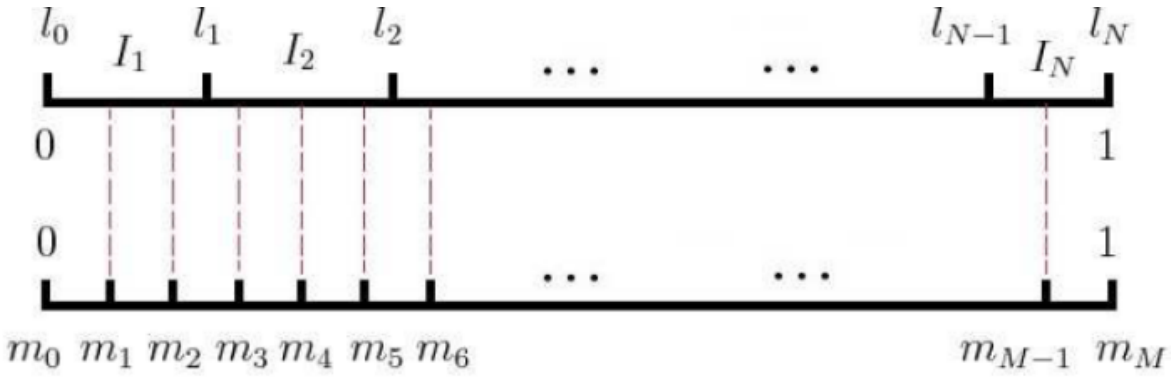
现在我们来看看如何进行负采样，得到neg个负例。word2vec采样的方法并不复杂，如果词汇表的大小为 V，那么我们就将一段长度为1的线段分成 V 份，每份对应词汇表中的一个词。当然每个词对应的线段长度是不一样的，高频词对应的线段长，低频词对应的线段短。每个词 w 的线段长度由下式决定：

$$len(w) = \frac{count(w)}{\sum_{u \in vocab} count(u)}$$

在word2vec中，分子和分母都取了 $\frac{3}{4}$ 次幂如下：

$$len(w) = \frac{count(w)^{3/4}}{\sum_{u \in vocab} count(u)^{3/4}}$$

在采样前，我们将这段长度为 1 的线段划分成 M 等份，这里 $M >> V$，这样可以保证每个词对应的线段都会划分成对应的小块。而 M 份中的每一份都会落在某一个词对应的线段上。在采样的时候，我们只需要从 M 个位置中采样出 neg 个位置就行，此时采样到的每一个位置对应到的线段所属的词就是我们的负例词。



在word2vec中，M 取值默认为 10^8。

13. 基于Negative Sampling的CBOW模型

有了上面Negative Sampling负采样的方法和逻辑回归求解模型参数的方法，我们就可以总结出基于Negative Sampling的CBOW模型算法流程了。梯度迭代过程使用了随机梯度上升法：

> 输入：基于CBOW的语料训练样本，词向量的维度大小 $Mcount$，CBOW的上下文大小 $2c$，步长 η， 负采样的个数neg
>
> 输出：词汇表每个词对应的模型参数 θ，所有的词向量 x_w
>
> 1. 随机初始化所有的模型参数 θ，所有的词向量 w
> 2. 对于每个训练样本 $(context(w_0), w_0)$，负采样出neg个负例中心词 $w_i, i = 1, 2, \cdots, neg$
> 3. 进行梯度上升迭代过程，对于训练集中的每一个样本 $(context(w_0), w_0, w_1, \cdots, w_{neg})$ 做如下处理：
>
> a) $e = 0$，计算 $x_{w_0} = \frac{1}{2c}\sum_{i=1}^{2c} x_i$
>
> b) for $j = 0$ to neg，计算：
>
> $$f = \sigma(x_{w_0}^T \theta^{w_i})$$
> $$g = (y_i - f)\eta$$
> $$e = e + g\theta^{w_i}$$
> $$\theta^{w_i} = \theta^{w_i} + g x_{w_0}$$
>
> c) 对于 $context(w)$ 中的每一个词向量 x_k （共 $2c$ 个）进行更新：

$$x_k = x_k + e$$

d) 如果梯度收敛，则结束梯度迭代，否则回到步骤3继续迭代。

14. 基于Negative Sampling的Skip-Gram模型

有了上一节CBOW的基础和上一篇基于Hierarchical Softmax的Skip-Gram模型基础，我们也可以总结出基于Negative Sampling的Skip-Gram模型算法流程了。梯度迭代过程使用了随机梯度上升法：

输入：基于Skip-Gram的语料训练样本，词向量的维度大小 $Mcount$，Skip-Gram的上下文大小 $2c$，步长 η，负采样的个数neg。

输出：词汇表每个词对应的模型参数 θ，所有的词向量 x_w

1. 随机初始化所有的模型参数 θ，所有的词向量 w

2. 对于每个训练样本 $(context(w_0), w_0)$，负采样出neg个负例中心词 $w_i, i = 1, 2, \cdots, neg$

3. 进行梯度上升迭代过程，对于训练集中的每一个样本 $(context(w_0), w_0, w_1, \cdots, w_{neg})$ 做如下处理：

a) for $i = 1$ to $2c$:

i) $e = 0$

ii) for $j = 0$ to neg，计算：

$$f = \sigma(x_{w_0 i}^T \theta^{w_j})$$
$$g = (y_i - f)\eta$$
$$e = e + g\theta^{w_j}$$
$$\theta^{w_j} = \theta^{w_j} + g x_{w_0 i}$$

iii) 词向量更新：

$$x_{w_{0i}} = x_{w_{0i}} + e$$

b)如果梯度收敛，则结束梯度迭代，算法结束，否则回到步骤a继续迭代。

15. Negative Sampling的模型源码和算法的对应

这里给出上面算法和word2vec源码中的变量对应关系。

在源代码中，基于Negative Sampling的CBOW模型算法在464-494行，基于Negative Sampling的Skip-Gram的模型算法在520-542行。大家可以对着源代码再深入研究下算法。

在源代码中，neule对应我们上面的 e，syn0对应我们的 x_w，syn1neg对应我们的 θ^{w_i}，layer1_size对应词向量的维度，window对应我们的 c。negative对应我们的neg，table_size对应我们负采样中的划分数 M。

另外，vocab[word].code[d]指的是，当前单词word的，第 d 个编码，编码不含Root结点。vocab[word].point[d]指的是，当前单词word，第 d 个编码下，前置的结点。这些和基于Hierarchical Softmax的是一样的。

以上就是基于Negative Sampling的word2vec模型，希望可以帮到大家，后面会讲解用gensim的python版word2vec来使用word2vec解决实际问题。

参考文献

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient estimation of word representations in vector space. *ICLR Workshop*, 2013

[2] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013

[3] From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions (Young et al., TACL 2014)