**需要的库**：NumPy 、pandas 、IPython和Jupyter 、SciPy 、scikit-learn、

statsmodels

**构建词表：**

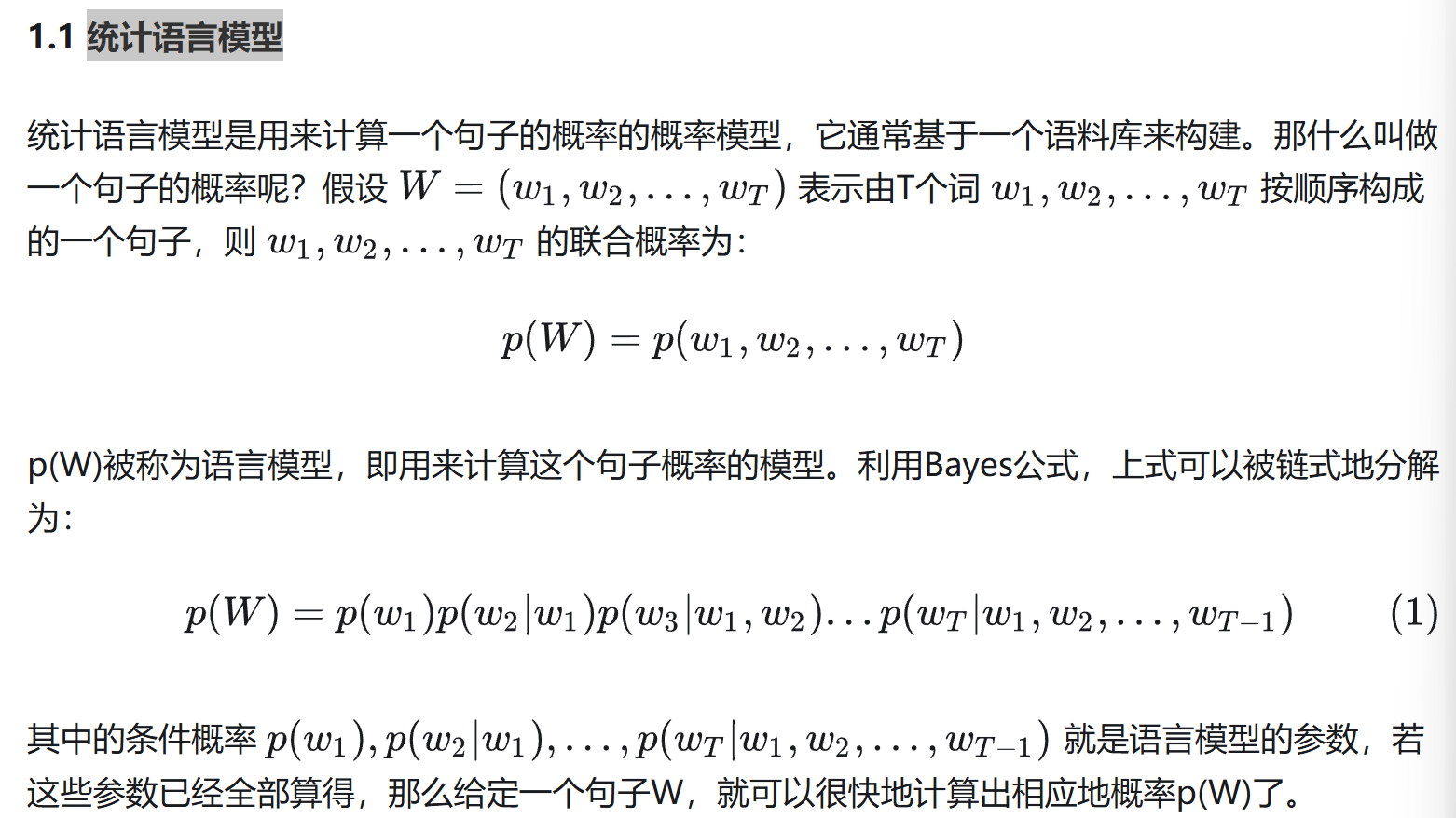
1. 构建text、label列表
2. 获取文件路径和路径下所有的子目录、文件用os库
3. 从文件中读取出text和label并通过append加入到列表中
4. 小写化 text.lower()
5. 去除符号
6. 分词spilt
7. 取出现数量最多的

**包装数据集**：dataset dataloader

Dataset：\_\_init\_\_初始化 \_\_len\_\_数据集长度 \_\_getitem\_\_根据索引取出数据集内元素

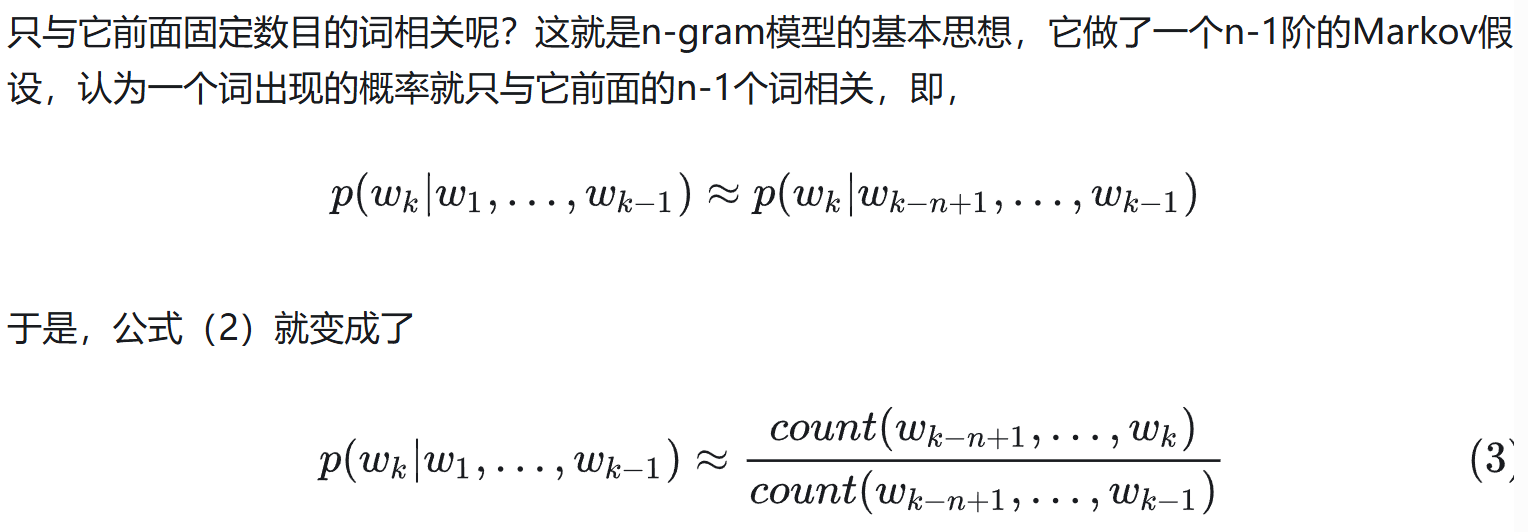
Dataloader：batch\_size 处理数据的批次大小 shuffle 是否打乱数据（训练集通常为True）

**Word2vec**：用来生成词向量



**计算参数的方法**：n-gram模型和神经网络

**N-gram模型的基本思想**：

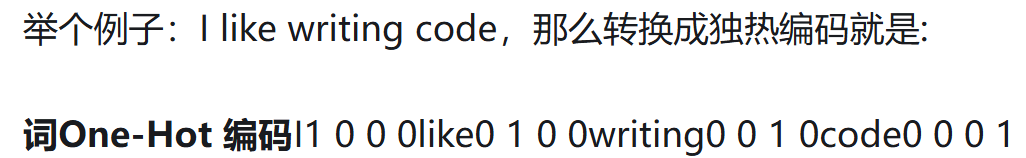


其主要工作是在语料中统计各种词串出现的次数以及平滑化处理。概率值计算好之后就存储起来，下次需要计算一个句子的概率时，只需找到相关的概率参数，将它们连乘起来就好了。

**神经概率语言模型：**

词语之间的相似性可以通过词向量来体现。

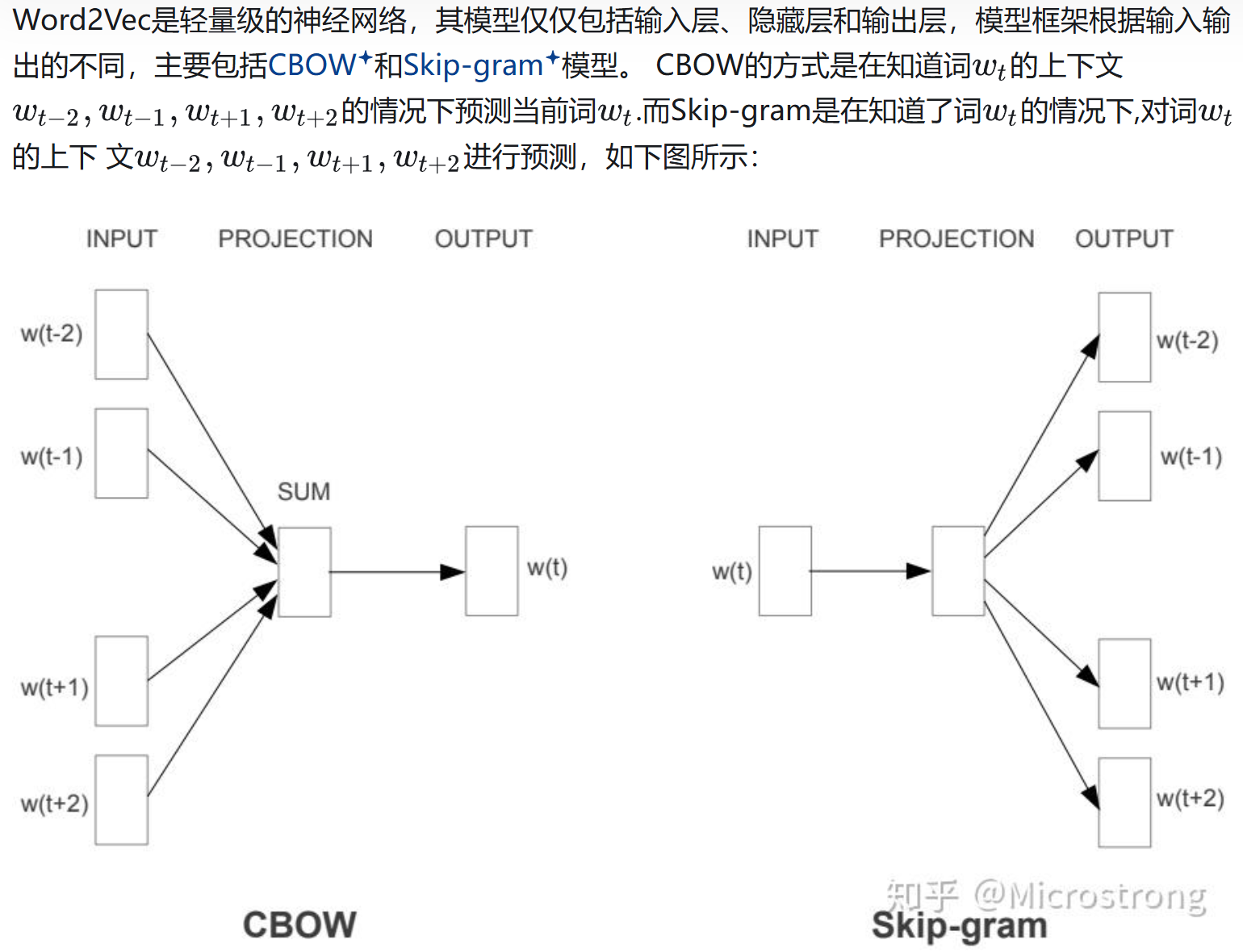
**词向量的编码方式**：One-Hot Representation 和 Distributed Representation。

**One-Hot Representation**：就是用一个很长的向量来表示一个词，向量的长度为词典的大小，向量中只有一个 1 ， 其他全为 0 ，1 的位置对应该词在词典中的位置。

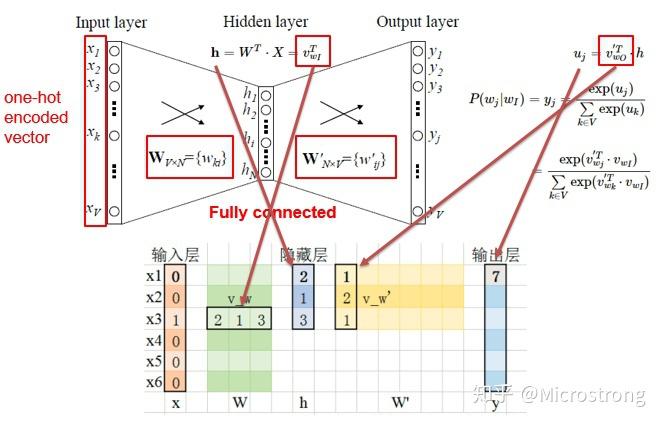
缺点：1.维数灾难 2.不能很好地刻画词与词之间的相似性 3.强稀疏性，0太多了不好计算

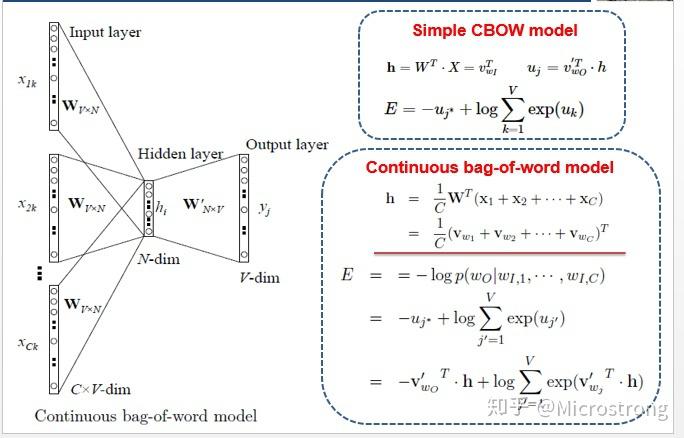
**Distributed Representation：**通过训练将某种语言中的每一个词 映射成一个固定长度的短向量（当然这里的“短”是相对于One-Hot Representation的“长”而言的），所有这些向量构成一个词向量空间，而每一个向量则可视为 该空间中的一个点，在这个空间上引入“距离”，就可以根据词之间的距离来判断它们之间的语法、语义上的相似性了。Word2Vec中采用的就是这种Distributed Representation 的词向量。

**Word2vec：**

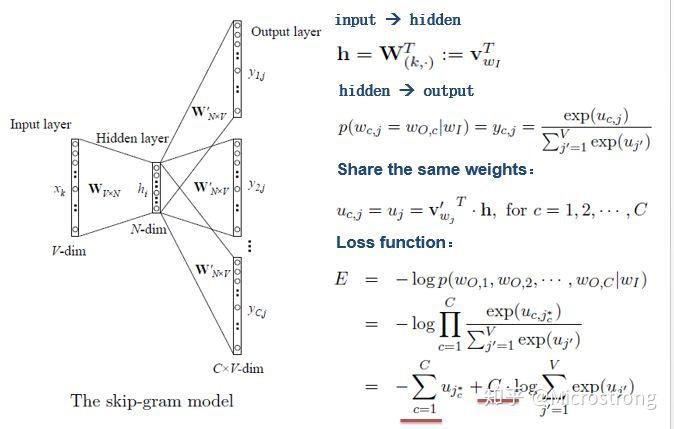


CBOW:

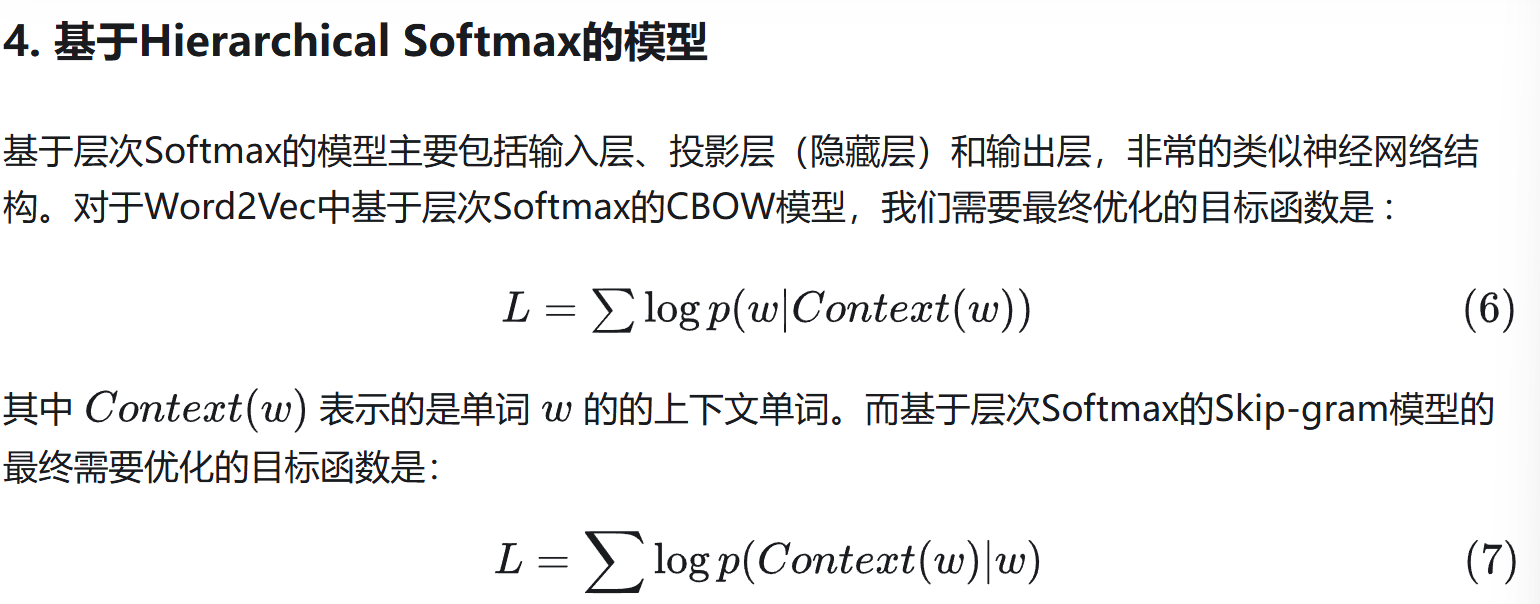




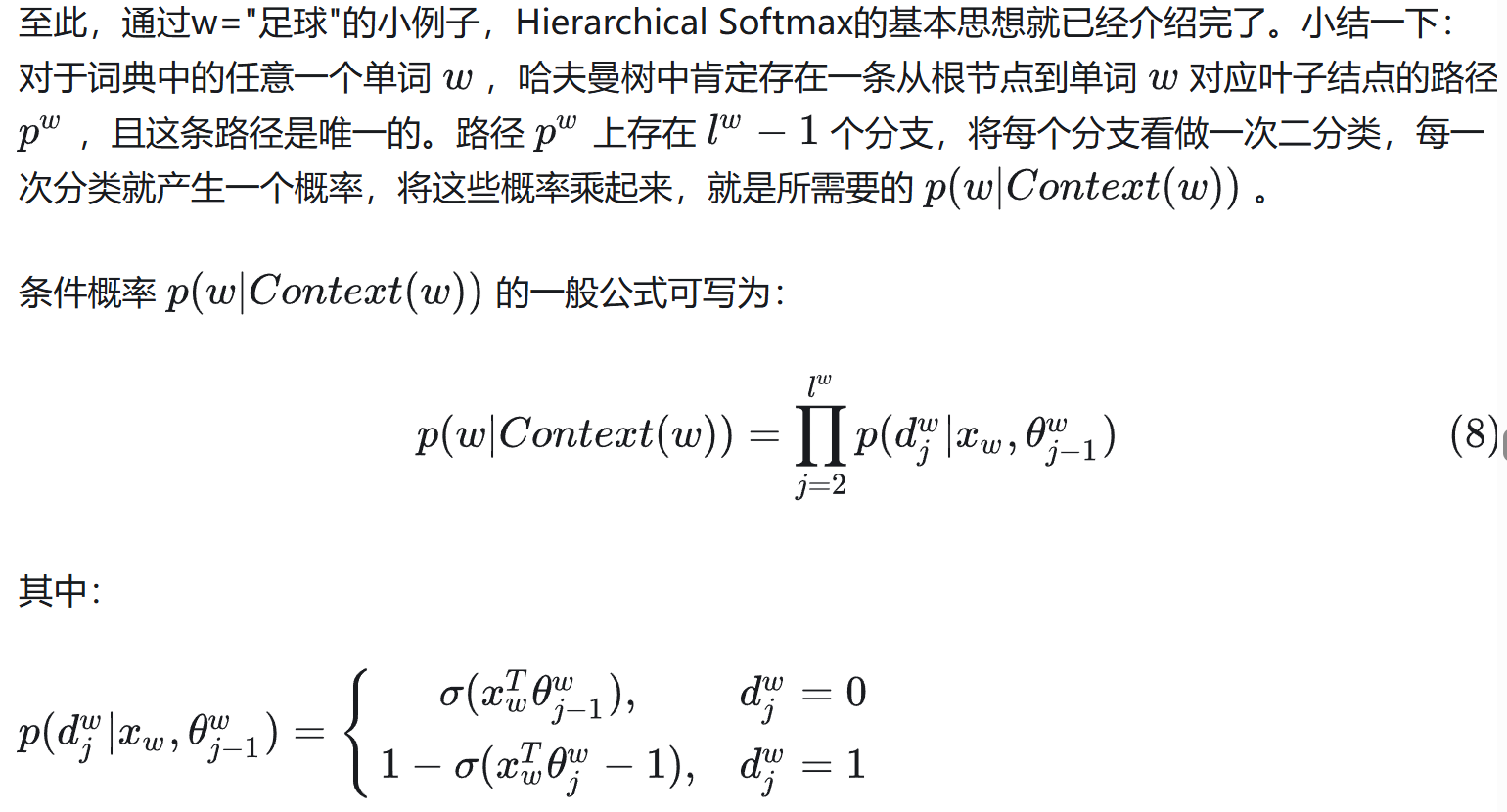
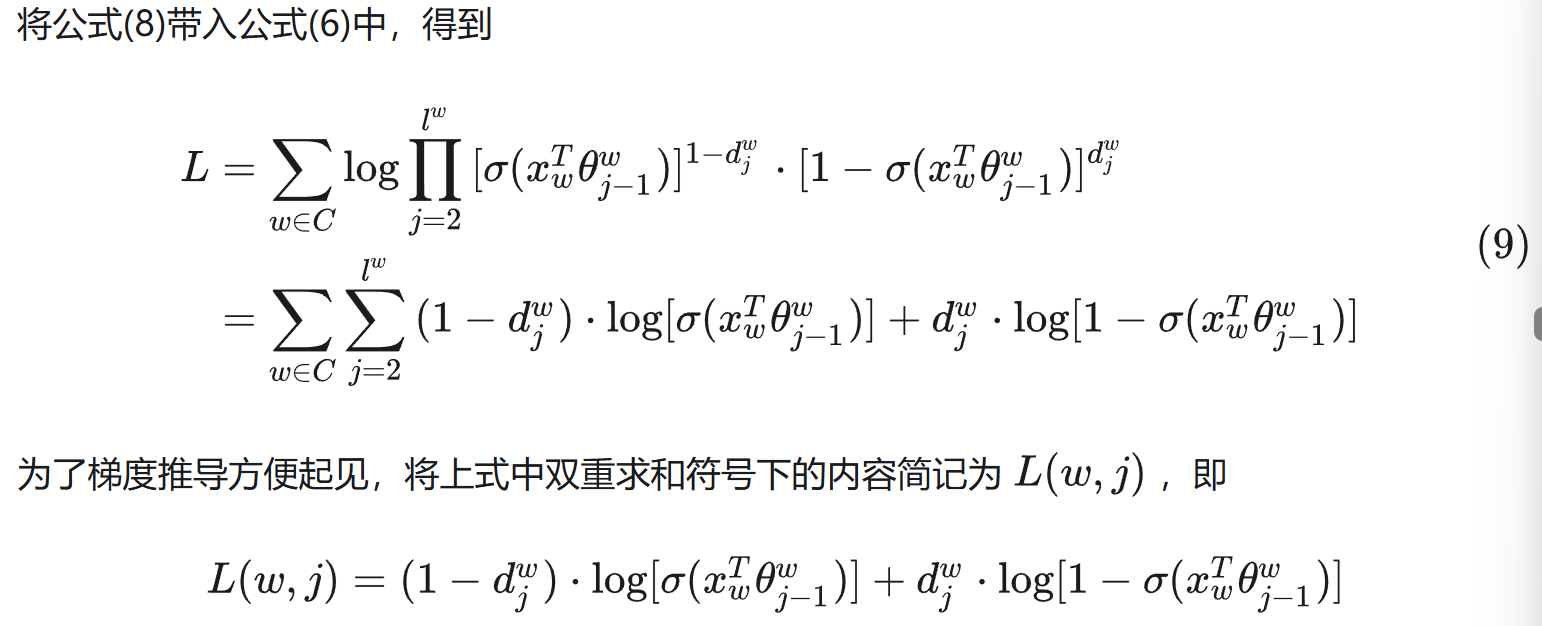
Skip-gram:



Word2Vec提出两种加快训练速度的方式，一种是Hierarchical softmax，另一种是Negative Sampling。

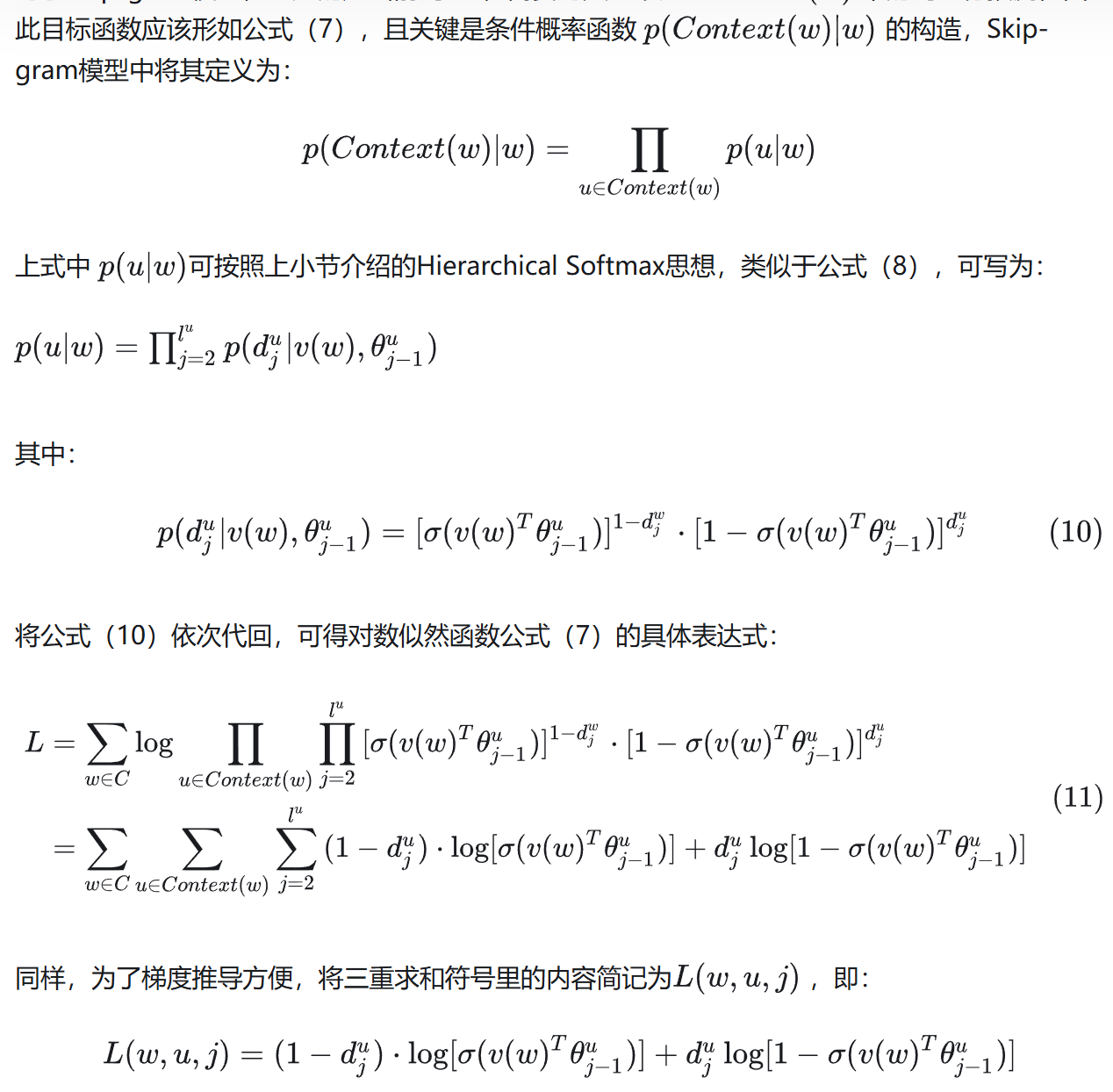


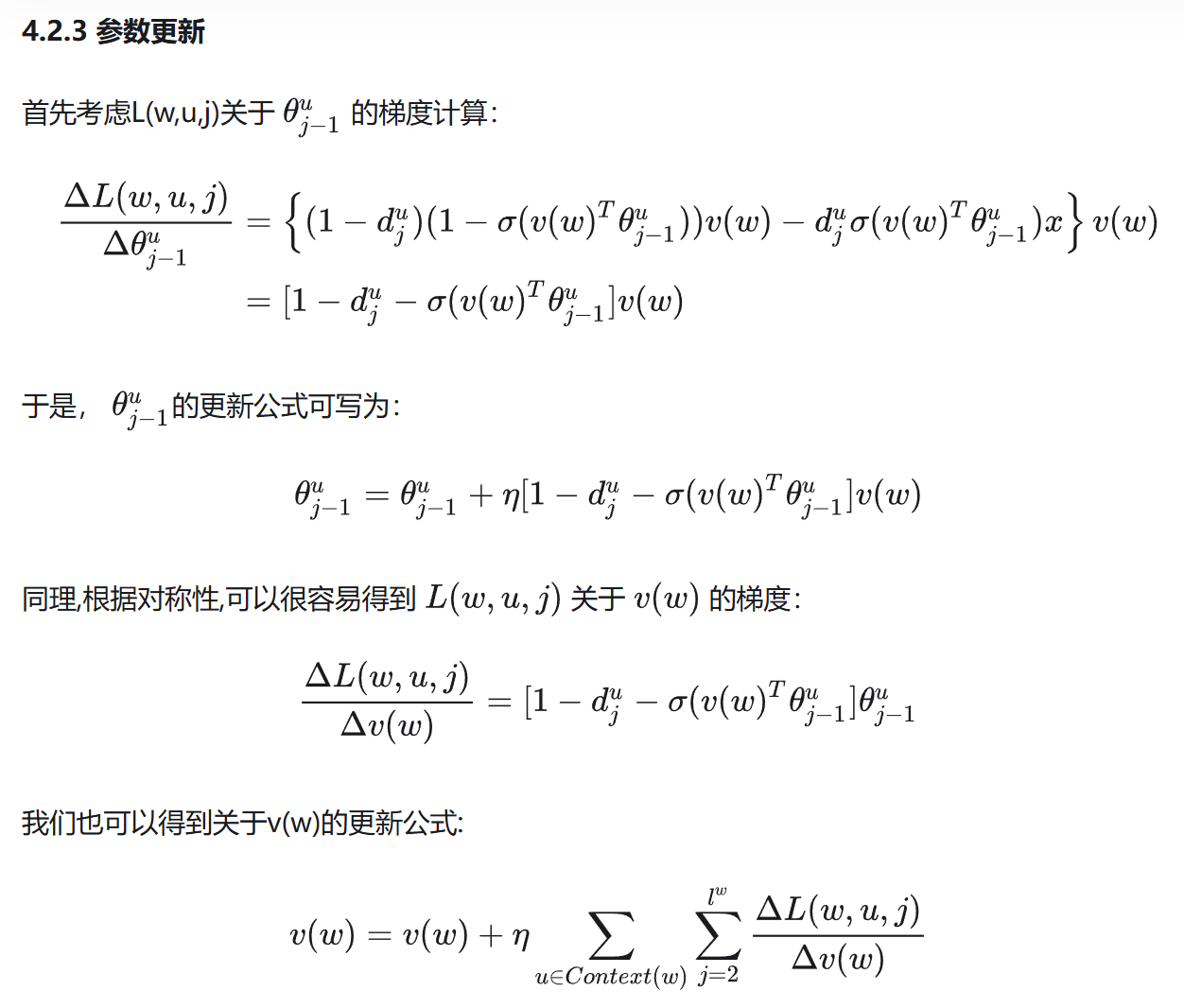
**基于Hierarchical Softmax的CBOW**

使用随机梯度上升法来优化，使目标函数最大化

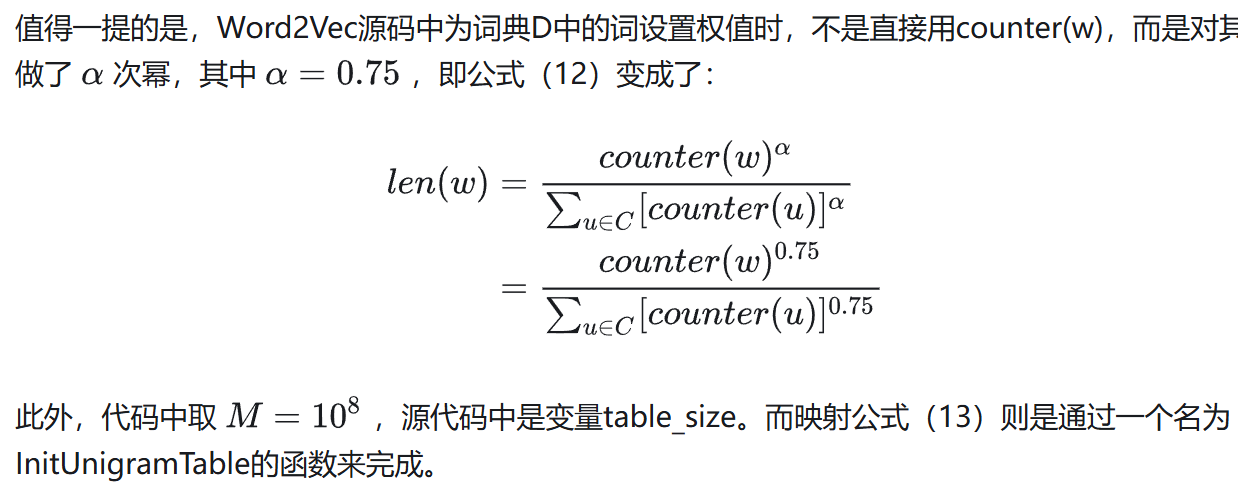
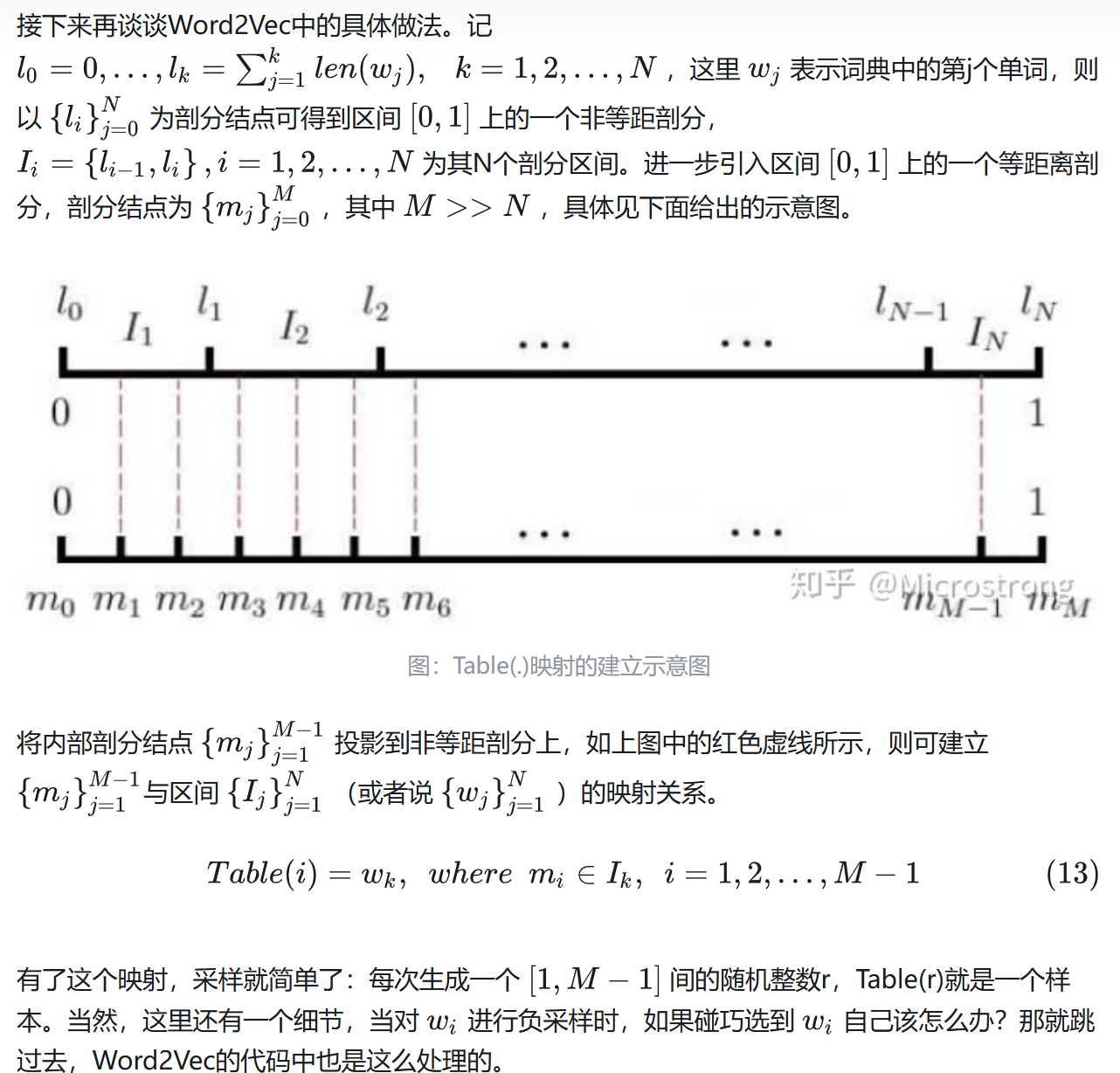
**基于Hierarchical Softmax的Skip-gram**





**基于Negative Sampling的模型**

它是NCE（Noise Contrastive Estimation）（利用已知的概率密度函数来估计未知的概率密度函数。假设未知的概率密度函数为X，已知的概率密度为Y，如果得到了X和Y的关系，那么X也就可以求出来了）的一个简化版，目的是用来提高训练速度并改善所得词向量的质量。与Hierarchical Softmax相比，NEG不再使用复杂的Huffman树，而是利用相对简单的随机负采样，能大幅度提高性能，因而可作为Hierarchical Softmax的一种替代。  
  
负采样算法：词典D中的词在语料C中出现的次数有高有底，对于那些高频词，被选为负样本的概率就应该比较大，反之，对于那些低频词，其被选中的概率就应该比较小。



用Word2vec计算两个句子之间的相似度：将句子分词、获取词分量、计算句子向量、计算句子向量之间的相似度

Self-Attention

