主要参考博客http://blog.csdn.net/zhoubl668/article/details/23271225

语言模型就是判断一句话是不是正常人说出来的。其形式化描述就是给定一个字符串，看它是自然语言的概率P(w1,w2,…,wt)。w1 到 wt 依次表示这句话中的各个词。有个很简单的推论是：

P(w1,w2,…,wt)=P(w1)×P(w2|w1)×P(w3|w1,w2)×…×P(wt|w1,w2,…,wt-1)

n-gram语言模型认为一个词只有前n个词有关，比如2元语言模型(bigram)，认为一个词只有前一个词有关，那么上述概率公式可简化成

P(w1,w2,…,wt)=P(w1)×P(w2|w1)×P(w3|w2)×…×P(wt|wt-1)

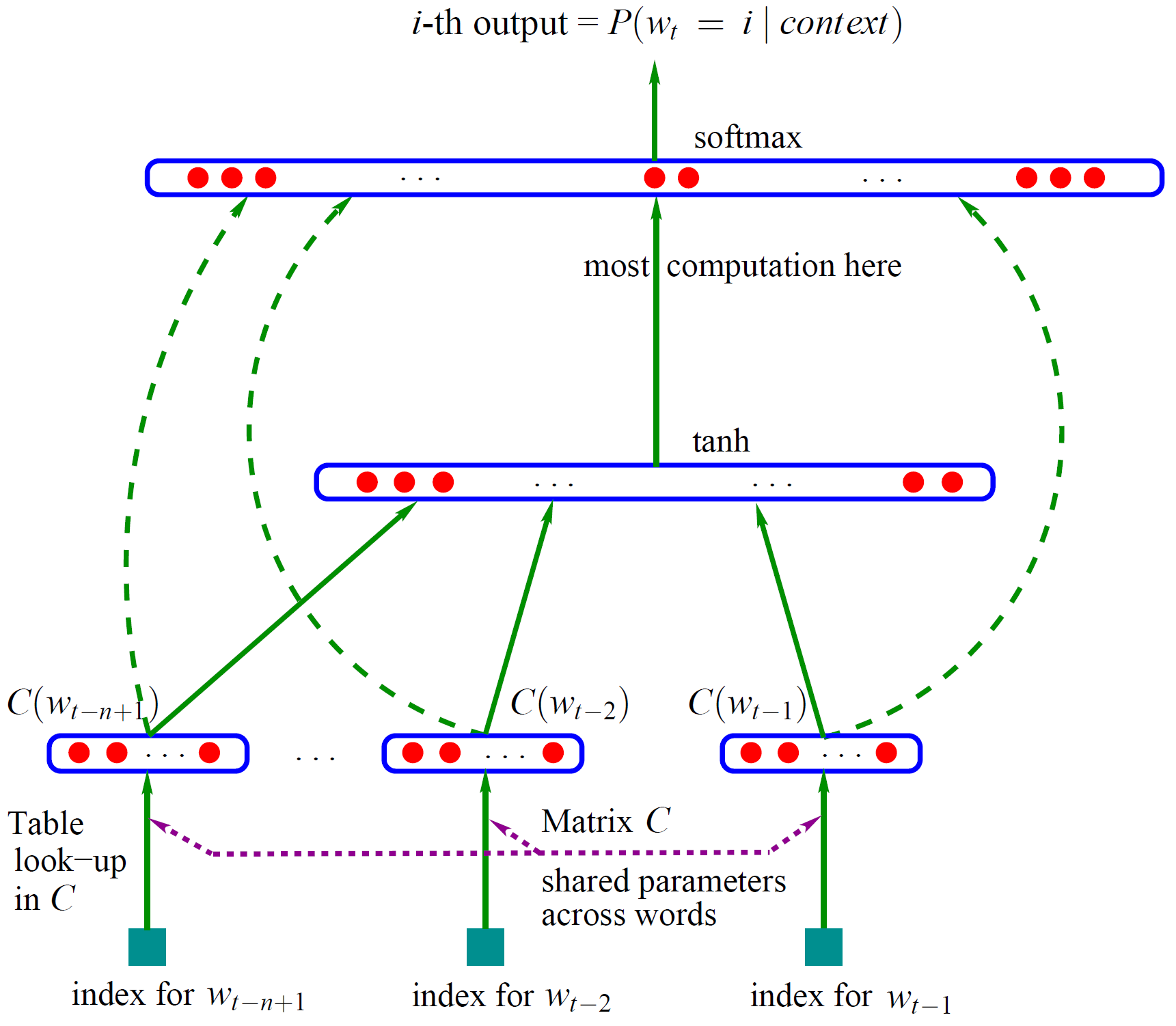
因为P(w3|w1,w2)中w3根w1无关，所以P(w3|w1,w2)=P(w3|w2)。

n-gram的计算全部由语料库中的概率计算得到。

n-gram可能会出现其中一项是0的情况，比如某个词在训练语料库中根本不存在，只存在测试语料库中，因此需要对概率平滑化，一种方法是统计词频时不从0开始计数而是从1开始。

论文《A Neural Probabilistic Language Model》提出了用神经网络训练语言模型的方案。

　Bengio 用了一个三层的神经网络来构建语言模型，同样也是 n-gram 模型。如下图



图中最下方的 wt−n+1,…,wt−2,wt−1 就是前 n−1 个词。现在需要根据这已知的 n−1 个词预测下一个词 wt。C(w) 表示词 w 所对应的词向量，整个模型中使用的是一套唯一的词向量，存在矩阵 C（一个 |V|×m 的矩阵）中。其中 |V| 表示词表的大小（语料中的总词数），m 表示词向量的维度。w 到 C(w) 的转化就是从矩阵中取出一行。

　　网络的第一层（输入层）是将 C(wt−n+1),…,C(wt−2),C(wt−1) 这 n−1 个向量首尾相接拼起来，形成一个 (n−1)m 维的向量，下面记为 x。

　　网络的第二层（隐藏层）就如同普通的神经网络，直接使用 d+Hx 计算得到。d 是一个偏置项。在此之后，使用 tanh 作为激活函数。

　　网络的第三层（输出层）一共有 |V| 个节点，每个节点 yi 表示 下一个词为 i 的未归一化 log 概率。最后使用 softmax 激活函数将输出值 y 归一化成概率。最终，y 的计算公式为：

y=b+Wx+Utanh(d+Hx)

　　式子中的 U（一个 |V|×h 的矩阵）是隐藏层到输出层的参数，整个模型的多数计算集中在 U 和隐藏层的矩阵乘法中。后文的提到的 3 个工作，都有对这一环节的简化，提升计算的速度。

　　式子中还有一个矩阵 W（|V|×(n−1)m），这个矩阵包含了从输入层到输出层的直连边。直连边就是从输入层直接到输出层的一个线性变换，好像也是神经网络中的一种常用技巧（没有仔细考察过）。如果不需要直连边的话，将 W 置为 0 就可以了。在最后的实验中，Bengio 发现直连边虽然不能提升模型效果，但是可以少一半的迭代次数。同时他也猜想如果没有直连边，可能可以生成更好的词向量。

　　现在万事俱备，用随机梯度下降法把这个模型优化出来就可以了。需要注意的是，一般神经网络的输入层只是一个输入值，而在这里，输入层 x 也是参数（存在 C 中），也是需要优化的。优化结束之后，词向量有了，语言模型也有了。

　　这样得到的语言模型自带平滑，无需传统 n-gram 模型中那些复杂的平滑算法。Bengio 在 APNews 数据集上做的对比实验也表明他的模型效果比精心设计平滑算法的普通 n-gram 算法要好 10% 到 20%。