word2vec 中的数学

peghoty (peghoty@163.com)

2014 年 7 月

目 录

1 前言 **3**

2 预备知识 **4**

2.1 sigmoid 函数 5

2.2 逻辑回归 6

2.3 Bayes 公式 7

2.4 Huffman 编码 8

2.4.1 Huffman 树 8

2.4.2 Huffman 树的构造 8

2.4.3 Huffman 编码 9

3 背景知识 **11**

3.1 统计语言模型 12

3.2 n-gram 模型 13

3.3 神经概率语言模型 16

3.4 词向量的理解 19

4 基于 Hierarchical Softmax 的模型 **22**

4.1 CBOW 模型 23

4.1.1 网络结构 23

4.1.2 梯度计算 24

4.2 Skip-gram 模型 30

4.2.1 网络结构 30

4.2.2 梯度计算 30

5 基于 Negative Sampling 的模型 **33**

5.1 CBOW 模型 34

5.2 Skip-gram 模型 37

5.3 负采样算法 40

6 若干源码细节 42

6.1 $\sigma(x)$ 的近似计算 42
6.2 词典的存储 43
6.3 换行符 44
6.4 低频词和高频词 45
6.5 窗口及上下文 46
6.6 自适应学习率 47
6.7 参数初始化与训练 48
6.8 多线程并行 49
6.9 几点疑问和思考 50

§1 前言

word2vec 是 Google 于 2013 年开源推出的一个用于获取 word vector 的工具包, 它简单、高效, 因此引起了很多人的关注. 由于 word2vec 的作者 Tomas Mikolov 在两篇相关的论文 ([3], [4]) 中并没有谈及太多算法细节, 因而在一定程度上增加了这个工具包的神秘感. 一些按捺不住的人于是选择了通过解剖源代码的方式来一窥究竟.

第一次接触 word2vec 是 2013 年的 10 月份, 当时读了复旦大学郑骁庆老师发表的论文 [7], 其主要工作是将 SENNA 的那套算法 ([8]) 搬到中文场景. 觉得挺有意思, 于是做了一个实现 (可参见 [20]), 但苦于其中字向量的训练时间太长, 便选择使用 word2vec 来提供字向量, 没想到中文分词效果还不错, 立马对 word2vec 刮目相看了一把, 好奇心也随之增长.

后来, 陆陆续续看到了 word2vec 的一些具体应用, 而 Tomas Mikolov 团队本身也将其推广到了句子和文档 ([6]), 因此觉得确实有必要对 word2vec 里的算法原理做个了解, 以便对他们的后续研究进行追踪. 于是, 沉下心来, 仔细读了一回代码, 算是基本搞明白里面的做法了. 第一个感觉就是, “明明是个很简单的浅层结构, 为什么被那么多人沸沸扬扬地说成是 Deep Learning 呢?”

解剖 word2vec 源代码的过程中, 除了算法层面的收获, 其实编程技巧方面的收获也颇多. 既然花了功夫来读代码, 还是把理解到的东西整理成文, 给有需要的朋友提供点参考吧.

在整理本文的过程中, 和深度学习群的群友 北流浪子 ([15, 16]) 进行了多次有益的讨论, 在此表示感谢. 另外, 也参考了其他人的一些资料, 都列在参考文献了, 在此对他们的工作也一并表示感谢.

§2 预备知识

本节介绍 word2vec 中将用到的一些重要知识点, 包括 sigmoid 函数、Beyes 公式和 Huffman 编码等.

§2.1 sigmoid 函数

sigmoid 函数是神经网络中常用的激活函数之一, 其定义为

$$\sigma(x) = \frac{1}{1+e^{-x}},$$

该函数的定义域为 $(-\infty, +\infty)$, 值域为 $(0, 1)$. 图 1 给出了 sigmoid 函数的图像.

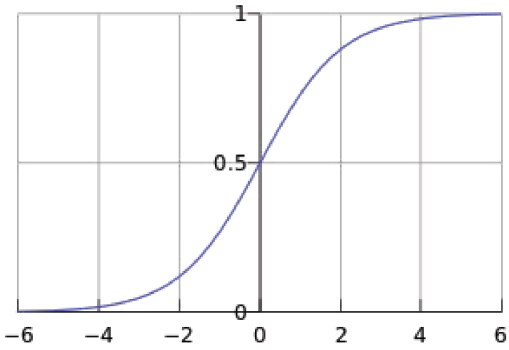

图 1 sigmoid 函数的图像

sigmoid 函数的**导函数**具有以下形式

$$\sigma'(x) = \sigma(x)[1-\sigma(x)],$$

由此易得, 函数 $\log\sigma(x)$ 和 $\log(1-\sigma(x))$ 的**导函数**分别为

$$[\log\sigma(x)]' = 1-\sigma(x), \quad [\log(1-\sigma(x))]' = -\sigma(x), \tag{2.1}$$

公式 (2.1) 在后面的推导中将用到.

§2.2 逻辑回归

生活中经常会碰到**二分类问题**, 例如, 某封电子邮件是否为垃圾邮件, 某个客户是否为潜在客户, 某次在线交易是否存在欺诈行为, 等等. 设 $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ 为一个二分类问题的样本数据, 其中 $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \{0, 1\}$, 当 $y_i = 1$ 时称相应的样本为**正例**, 当 $y_i = 0$ 时称相应的样本为**负例**.

利用 sigmoid 函数, 对于任意样本 $\mathbf{x} = (x_1, x_2, \cdots, x_n)^\top$, 可将二分类问题的 hypothesis 函数写成

$$h_\theta(\mathbf{x}) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n),$$

其中 $\theta = (\theta_0, \theta_1, \cdots, \theta_n)^\top$ 为待定参数. 为了符号上简化起见, 引入 $x_0 = 1$ 将 $\mathbf{x}$ 扩展为 $(x_0, x_1, x_2, \cdots, x_n)^\top$, 且在不引起混淆的情况下仍将其记为 $\mathbf{x}$. 于是, h_θ 可简写为

$$h_\theta(\mathbf{x}) = \sigma(\theta^\top \mathbf{x}) = \frac{1}{1 + e^{-\theta^\top \mathbf{x}}}.$$

取阀值 $T = 0.5$, 则二分类的判别公式为

$$y(\mathbf{x}) = \begin{cases} 1, & h_\theta(\mathbf{x}) \geq 0.5; \\ 0, & h_\theta(\mathbf{x}) < 0.5. \end{cases}$$

那参数 θ 如何求呢? 通常的做法是, 先确定一个形如下式的**整体损失函数**

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m cost(\mathbf{x}_i, y_i),$$

然后对其进行优化, 从而得到最优的参数 θ^*.

实际应用中, 单个样本的损失函数 $cost(\mathbf{x}_i, y_i)$ 常取为**对数似然函数**

$$cost(\mathbf{x}_i, y_i) = \begin{cases} -\log\left(h_\theta(\mathbf{x}_i)\right), & y_i = 1; \\ -\log\left(1 - h_\theta(\mathbf{x}_i)\right), & y_i = 0. \end{cases}$$

注意, 上式是一个分段函数, 也可将其写成如下的整体表达式

$$cost(\mathbf{x}_i, y_i) = -y_i \cdot \log\left(h_\theta(\mathbf{x}_i)\right) - (1 - y_i) \cdot \log\left(1 - h_\theta(\mathbf{x}_i)\right).$$

§2.3 Bayes 公式

贝叶斯公式是英国数学家**贝叶斯** (Thomas Bayes) 提出来的, 用来描述两个条件概率之间的关系. 若记 $P(A)$, $P(B)$ 分别表示事件 A 和事件 B 发生的概率, $P(A|B)$ 表示事件 B 发生的情况下事件 A 发生的概率, $P(A,B)$ 表示事件 A, B 同时发生的概率, 则有

$$P(A|B)=\frac{P(A,B)}{P(B)},\quad P(B|A)=\frac{P(A,B)}{P(A)},$$

利用上式, 进一步可得

$$P(A|B)=P(A)\frac{P(B|A)}{P(B)},$$

这就是 **Bayes 公式**.

§2.4 Huffman 编码

本节简单介绍 Huffman 编码 (具体内容主要来自百度百科的词条, [10]), 为此, 首先介绍 Huffman 树的定义及其构造算法.

§2.4.1 Huffman 树

在计算机科学中, **树**是一种重要的非线性数据结构, 它是数据元素 (在树中称为**结点**) 按分支关系组织起来的结构. 若干棵互不相交的树所构成的集合称为**森林**. 下面给出几个与树相关的常用概念.

- **路径**和**路径长度**

  在一棵树中, 从一个结点往下可以达到的孩子或孙子结点之间的通路, 称为路径. 通路中分支的数目称为路径长度. 若规定根结点的层号为 1, 则从根结点到第 L 层结点的路径长度为 $L-1$.

- **结点的权**和**带权路径长度**

  若为树中结点赋予一个具有某种含义的 (非负) 数值, 则这个数值称为该结点的权. 结点的带权路径长度是指, 从根结点到该结点之间的路径长度与该结点的权的乘积.

- **树的带权路径长度**

  树的带权路径长度规定为所有叶子结点的带权路径长度之和.

二叉树是每个结点最多有两个子树的有序树. 两个子树通常被称为 "**左子树**" 和 "**右子树**", 定义中的 "有序" 是指两个子树有左右之分, 顺序不能颠倒.

给定 n 个权值作为 n 个叶子结点, 构造一棵二叉树, 若它的带权路径长度达到最小, 则称这样的二叉树为**最优二叉树**, 也称为 **Huffman 树**.

§2.4.2 Huffman 树的构造

给定 n 个权值 $\{w_1, w_2, \cdots, w_n\}$ 作为二叉树的 n 个叶子结点, 可通过以下算法来构造一颗 Huffman 树.

算法 2.1 (*Huffman 树构造算法*)

(1) 将 $\{w_1, w_2, \cdots, w_n\}$ 看成是有 n 棵树的森林 (每棵树仅有一个结点).

(2) 在森林中选出两个根结点的权值最小的树合并, 作为一棵新树的左、右子树, 且新树的根结点权值为其左、右子树根结点权值之和.

(3) 从森林中删除选取的两棵树, 并将新树加入森林.

(4) 重复 (2)、(3) 步, 直到森林中只剩一棵树为止, 该树即为所求的 *Huffman* 树.

接下来, 给出算法 2.1 的一个具体实例.

例 2.1 假设 *2014* 年世界杯期间, 从新浪微博中抓取了若干条与足球相关的微博, 经统计, “我”、“喜欢”、“观看”、“巴西”、“足球”、“世界杯” 这六个词出现的次数分别为 *15, 8, 6, 5, 3, 1*. 请以这 *6* 个词为叶子结点, 以相应词频当权值, 构造一棵 *Huffman* 树.

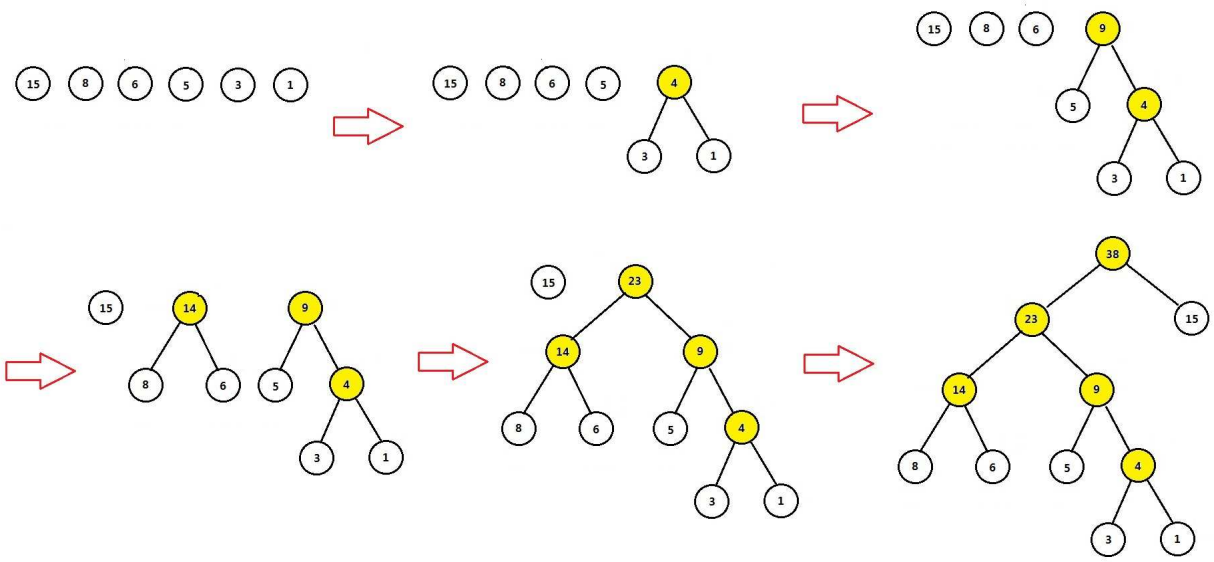

图 2 Huffman 树的构造过程

利用算法 *2.1*, 易知其构造过程如图 *2* 所示. 图中第六步给出了最终的 *Huffman* 树, 由图可见**词频越大的词离根结点越近**.

构造过程中, 通过合并新增的结点被标记为黄色. 由于每两个结点都要进行一次合并, 因此, 若叶子结点的个数为 n, 则构造的 *Huffman* 树中新增结点的个数为 $n-1$. 本例中 $n=6$, 因此新增结点的个数为 *5*.

注意, 前面有提到, 二叉树的两个子树是分左右的, 对于某个非叶子结点来说, 就是其两个孩子结点是分左右的, 在本例中, 统一**将词频大的结点作为左孩子结点, 词频小的作为右孩子结点**. 当然, 这只是一个约定, 你要将词频大的结点作为右孩子结点也没有问题.

§2.4.3 Huffman 编码

在数据通信中, 需要将传送的文字转换成二进制的字符串, 用 0, 1 码的不同排列来表示字符. 例如, 需传送的报文为 “AFTER DATA EAR ARE ART AREA”, 这里用到的字符集为 “A, E, R, T, F, D”, 各字母出现的次数为 8, 4, 5, 3, 1, 1. 现要求为这些字母设计编码.

要区别 6 个字母, 最简单的二进制编码方式是**等长编码**, 固定采用 3 位二进制 ($2^3=8>6$), 可分别用 000、001、010、011、100、101 对 “A, E, R, T, F, D” 进行编码发送, 当对方接收报文时再按照三位一分进行译码.

显然编码的长度取决报文中不同字符的个数. 若报文中可能出现 26 个不同字符, 则固定编码长度为 5 ($2^5=32>26$). 然而, 传送报文时总是希望总长度尽可能短. 在实际应用中,

各个字符的**出现频度**或**使用次数**是不相同的, 如 A、B、C 的使用频率远远高于 X、Y、Z, 自然会想到设计编码时, 让使用频率高的用短码, 使用频率低的用长码, 以优化整个报文编码.

为使**不等长编码**为**前缀编码** (即要求一个字符的编码不能是另一个字符编码的前缀), 可用字符集中的每个字符作为叶子结点生成一棵编码二叉树, 为了获得传送报文的最短长度, 可将每个字符的出现频率作为字符结点的权值赋予该结点上, 显然字使用频率越小权值越小, 权值越小叶子就越靠下, 于是频率小编码长, 频率高编码短, 这样就保证了此树的最小带权路径长度, 效果上就是传送报文的最短长度. 因此, 求传送报文的最短长度问题转化为求由字符集中的所有字符作为叶子结点, 由字符出现频率作为其权值所产生的 Huffman 树的问题. 利用 Huffman 树设计的二进制前缀编码, 称为 **Huffman 编码**, 它既能满足前缀编码的条件, 又能保证报文编码总长最短.

本文将介绍的 word2vec 工具中也将用到 Huffman 编码, 它把训练语料中的词当成叶子结点, 其在语料中出现的次数当作权值, 通过构造相应的 Huffman 树来对每一个词进行 Huffman 编码.

图 3 给出了例 2.1 中六个词的 Huffman 编码, 其中约定 (词频较大的) 左孩子结点编码为 1, (词频较小的) 右孩子编码为 0. 这样一来, “我”、“喜欢”、“观看”、“巴西”、“足球”、“世界杯” 这六个词的 Huffman 编码分别为 0, 111, 110, 101, 1001 和 1000.

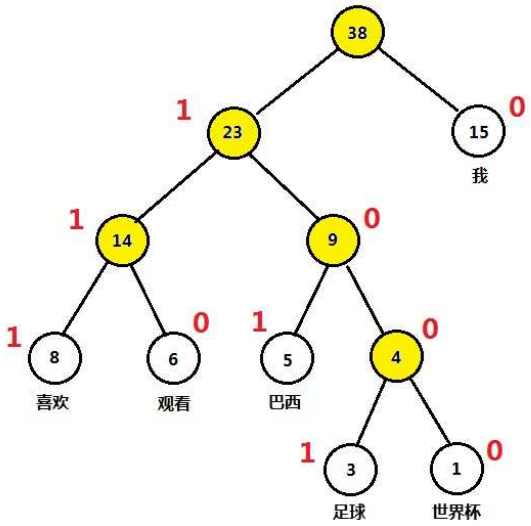

图 3 Huffman 编码示意图

注意, 到目前为止, 关于 Huffman 树和 Huffman 编码, 有两个约定: (1) 将权值大的结点作为左孩子结点, 权值小的作为右孩子结点; (2) 左孩子结点编码为 1, 右孩子结点编码为 0. 在 word2vec 源码中将权值较大的孩子结点编码为 1, 较小的孩子结点编码为 0. 为与上述约定统一起见, **下文中提到的 “左孩子结点” 都是指权值较大的孩子结点**.

§3 背景知识

word2vec 是用来生成词向量的工具, 而词向量与语言模型有着密切的关系, 为此, 不妨先来了解一些语言模型方面的知识.

§3.1 统计语言模型

当今的互联网迅猛发展, 每天都在产生大量的文本、图片、语音和视频数据, 要对这些数据进行处理并从中挖掘出有价值的信息, 离不开自然语言处理 (Nature Language Processing, NLP) 技术, 其中**统计语言模型** (Statistical Language Model) 就是很重要的一环, 它是所有 NLP 的基础, 被广泛应用于语音识别、机器翻译、分词、词性标注和信息检索等任务.

例 3.1 *在语音识别系统中, 对于给定的语音段 $Voice$, 需要找到一个使概率 $p(Text|Voice)$ 最大的文本段 $Text$. 利用 Bayes 公式, 有*

$$p(Text|Voice) = \frac{p(Voice|Text) \cdot p(Text)}{p(Voice)},$$

*其中 $p(Voice|Text)$ 为**声学模型**, 而 $p(Text)$ 为**语言模型** ([18]).*

简单地说, 统计语言模型是用来计算一个句子的概率的**概率模型**, 它通常基于一个语料库来构建. 那什么叫做一个句子的概率呢? 假设 $W = w_1^T := (w_1, w_2, \cdots, w_T)$ 表示由 T 个词 $w_1, w_2, \cdots, w_T$ 按顺序构成的一个句子, 则 $w_1, w_2, \cdots, w_T$ 的联合概率

$$p(W) = p(w_1^T) = p(w_1, w_2, \cdots, w_T)$$

就是这个句子的概率. 利用 **Bayes 公式**, 上式可以被链式地分解为

$$p(w_1^T) = p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|w_1^2) \cdots p(w_T|w_1^{T-1}), \tag{3.1}$$

其中的 (条件) 概率 $p(w_1)$, $p(w_2|w_1)$, $p(w_3|w_1^2)$, $\cdots$, $p(w_T|w_1^{T-1})$ 就是**语言模型的参数**, 若这些参数已经全部算得, 那么给定一个句子 w_1^T, 就可以很快地算出相应的 $p(w_1^T)$ 了.

看起来好像很简单, 是吧? 但是, 具体实现起来还是有点麻烦. 例如, 先来看看**模型参数的个数**. 刚才是考虑一个给定的长度为 T 的句子, 就需要计算 T 个参数. 不妨假设语料库对应词典 $\mathcal{D}$ 的大小 (即词汇量) 为 N , 那么, 如果考虑长度为 T 的任意句子, 理论上就有 N^T 种可能, 而每种可能都要计算 T 个参数, 总共就需要计算 TN^T 个参数. 当然, 这里只是简单估算, 并没有考虑重复参数, 但这个量级还是有蛮吓人. 此外, 这些概率计算好后, 还得保存下来, 因此, 存储这些信息也需要很大的内存开销.

此外, **这些参数如何计算呢?** 常见的方法有 n-gram 模型、决策树、最大熵模型、最大熵马尔科夫模型、条件随机场、神经网络等方法. 本文只讨论 **n-gram 模型**和**神经网络**两种方法. 首先来看看 n-gram 模型.

§3.2 n-gram 模型

考虑 $p(w_k|w_1^{k-1})$ $(k>1)$ 的近似计算. 利用 Bayes 公式, 有

$$p(w_k|w_1^{k-1}) = \frac{p(w_1^k)}{p(w_1^{k-1})},$$

根据大数定理, 当语料库足够大时, $p(w_k|w_1^{k-1})$ 可近似地表示为

$$p(w_k|w_1^{k-1}) \approx \frac{count(w_1^k)}{count(w_1^{k-1})}, \tag{3.2}$$

其中 $count(w_1^k)$ 和 $count(w_1^{k-1})$ 分别表示词串 w_1^k 和 w_1^{k-1} 在语料中出现的次数. 可想而知, 当 k 很大时, $count(w_1^k)$ 和 $count(w_1^{k-1})$ 的统计将会多么耗时.

从公式 (3.1) 可以看出: 一个词出现的概率与它前面的所有词都相关. 如果假定一个词出现的概率只与它前面固定数目的词相关呢? 这就是 **n-gram 模型**的基本思想, 它作了一个 $n-1$ 阶的 **Markov 假设**, 认为一个词出现的概率就只与它前面的 $n-1$ 个词相关, 即

$$p(w_k|w_1^{k-1}) \approx p(w_k|w_{k-n+1}^{k-1}),$$

于是, (3.2) 就变成了

$$p(w_k|w_1^{k-1}) \approx \frac{count(w_{k-n+1}^k)}{count(w_{k-n+1}^{k-1})}. \tag{3.3}$$

以 $n=2$ 为例, 就有

$$p(w_k|w_1^{k-1}) \approx \frac{count(w_{k-1}, w_k)}{count(w_{k-1})}.$$

这样一简化, 不仅使得单个参数的统计变得更容易 (统计时需要匹配的词串更短), 也使得参数的总数变少了.

那么, n-gram 中的参数 n 取多大比较合适呢? 一般来说, n 的选取需要同时考虑计算复杂度和模型效果两个因素.

表 1 模型参数数量与 n 的关系

n	模型参数数量
1 (unigram)	2×10^5
2 (bigram)	4×10^{10}
3 (trigram)	8×10^{15}
4 (4-gram)	16×10^{20}

在**计算复杂度**方面, 表 1 给出了 n-gram 模型中模型参数数量随着 n 的逐渐增大而变化的情况, 其中假定词典大小 $N = 200000$ (汉语的词汇量大致是这个量级). 事实上, 模型参数

的量级是 N 的指数函数 ($O(N^n)$), 显然 n 不能取得太大, 实际应用中最多的是采用 $n=3$ 的三元模型.

在**模型效果**方面, 理论上是 n 越大, 效果越好. 现如今, 互联网的海量数据以及机器性能的提升使得计算更高阶的语言模型 (如 $n>10$) 成为可能, 但需要注意的是, 当 n 大到一定程度时, 模型效果的提升幅度会变小. 例如, 当 n 从 1 到 2, 再从 2 到 3 时, 模型的效果上升显著, 而从 3 到 4 时, 效果的提升就不显著了 (具体可参考吴军在《数学之美》中的相关章节). 事实上, 这里还涉及到一个**可靠性**和**可区别性**的问题, 参数越多, 可区别性越好, 但同时单个参数的实例变少从而降低了可靠性, 因此需要在可靠性和可区别性之间进行折中.

另外, n-gram 模型中还有一个叫做**平滑化**的重要环节. 回到公式 (3.3), 考虑两个问题:

1. 若 $count(w_{k-n+1}^k)=0$, 能否认为 $p(w_k|w_1^{k-1})$ 就等于 0 呢?
2. 若 $count(w_{k-n+1}^k)=count(w_{k-n+1}^{k-1})$, 能否认为 $p(w_k|w_1^{k-1})$ 就等于 1 呢?

显然不能! 但这是一个无法回避的问题, 哪怕你的语料库有多么大. 平滑化技术就是用来处理这个问题的, 这里不展开讨论, 具体可参考 [11].

总结起来, n-gram 模型是这样一种模型, 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.

然而, 在机器学习领域有一种通用的招数是这样的: 对所考虑的问题建模后先为其构造一个目标函数, 然后对这个目标函数进行优化, 从而求得一组最优的参数, 最后利用这组最优参数对应的模型来进行预测.

对于统计语言模型而言, 利用**最大似然**, 可把目标函数设为

$$\prod_{w\in\mathcal{C}} p(w|Context(w)).$$

其中 $\mathcal{C}$ 表示语料 (Corpus), $Context(w)$ 表示词 w 的**上下文** (Context), 即 w 周边的词的集合. 当 $Context(w)$ 为空时, 就取 $p(w|Context(w))=p(w)$. 特别地, 对于前面介绍的 n-gram 模型, 就有 $Context(w_i)=w_{i-n+1}^{i-1}$.

注 3.1 *语料 $\mathcal{C}$ 和词典 $\mathcal{D}$ 的区别: 词典 $\mathcal{D}$ 是从语料 $\mathcal{C}$ 中抽取出来的, 不存在重复的词; 而语料 $\mathcal{C}$ 是指所有的文本内容, 包括重复的词.*

当然, 实际应用中常采用**最大对数似然**, 即把目标函数设为

$$\mathcal{L}=\sum_{w\in\mathcal{C}} \log p(w|Context(w)), \tag{3.4}$$

然后对这个函数进行最大化.

从 (3.4) 可见, 概率 $p(w|Context(w))$ 已被视为关于 w 和 $Context(w)$ 的**函数**, 即

$$p(w|Context(w))=F(w,Context(w),\theta),$$

其中 θ 为**待定参数集**. 这样一来, 一旦对 (3.4) 进行优化得到最优参数集 θ^* 后, F 也就唯一被确定了, 以后任何概率 $p(w|Context(w))$ 就可以通过函数 $F(w, Context(w), \theta^*)$ 来计算了. 与 n-gram 相比, 这种方法不需要 (事先计算并) 保存所有的概率值, 而是通过直接计算来获取, 且通过选取合适的模型可使得 θ 中参数的个数远小于 n-gram 中模型参数的个数.

很显然, 对于这样一种方法, 最关键的地方就在于**函数 F 的构造**了. 下一小节将介绍一种通过神经网络来构造 F 的方法. 之所以特意介绍这个方法, 是因为它可以视为 word2vec 中算法框架的前身或者说基础.

§3.3 神经概率语言模型

本小节介绍 **Bengio** 等人在文《A neural probabilistic language model》(2003) 中提出的一种神经概率语言模型 ([2]). 该模型中用到了一个重要的工具 — **词向量**.

什么是词向量呢? 简单来说就是, 对词典 $\mathcal{D}$ 中的任意词 w, 指定一个固定长度的实值向量 $\mathbf{v}(w) \in \mathbb{R}^m$, $\mathbf{v}(w)$ 就称为 w 的词向量, m 为词向量的长度. 关于词向量的进一步理解将放到下一小节来专门讲解.

既然是神经概率语言模型, 其中当然要用到一个神经网络啦. 图 4 给出了这个神经网络的结构示意图, 它包括四个层: **输入** (Input) 层、**投影** (Projection) 层、**隐藏** (Hidden) 层和**输出** (Output) 层. 其中 W, U 分别为投影层与隐藏层以及隐藏层与输出层之间的权值矩阵, $\mathbf{p}, \mathbf{q}$ 分别为隐藏层和输出层上的偏置向量.

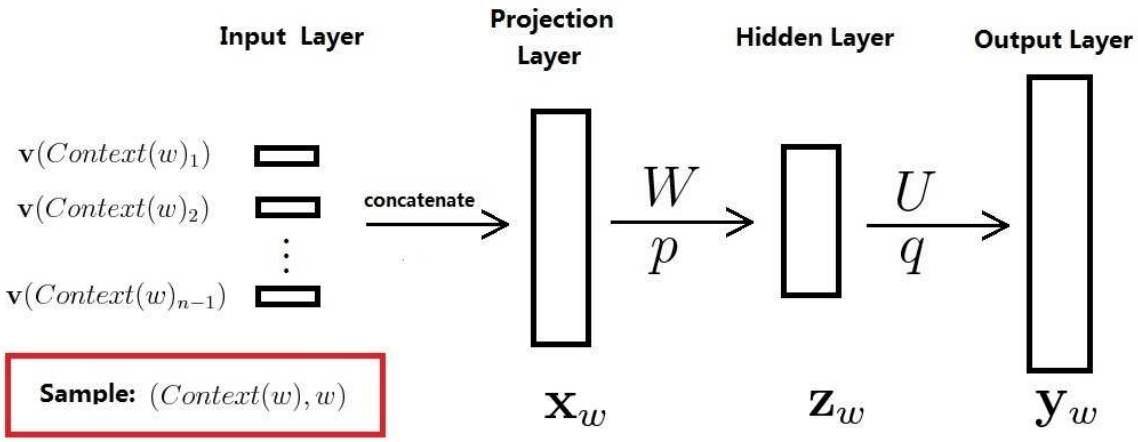

图 4 神经网络结构示意图

注 3.2 当提及文 [2] 中的神经网络时, 人们更多将其视为如图 5 所示的三层结构. 本文将其描述为如图 4 所示的四层结构, 一方面是便于描述, 另一方面是便于和 *word2vec* 中使用的网络结构进行对比.

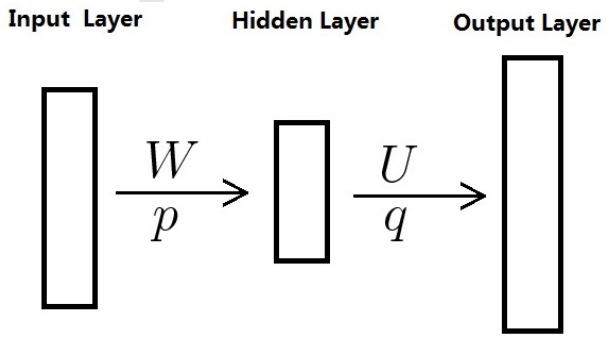

图 5 三层神经网络结构示意图

注 3.3 作者在文 [2] 中还考虑了投影层和输出层的神经元之间有边相连的情形, 因而也会多出一个相应的权值矩阵, 本文忽略了这种情形, 但这并不影响对算法本质的理解. 在数值实

验中, 作者发现引入投影层和输出层之间的权值矩阵虽然不能提高模型效果, 但可以减少训练的迭代次数.

对于语料 $\mathcal{C}$ 中的任意一个词 w, 将 $Context(w)$ 取为其前面的 $n-1$ 个词 (类似于 n-gram), 这样二元对 $(Context(w), w)$ 就是一个**训练样本**了. 接下来, 讨论样本 $(Context(w), w)$ 经过如图 4 所示的神经网络时是如何参与运算的. 注意, 一旦语料 $\mathcal{C}$ 和词向量长度 m 给定后, 投影层和输出层的规模就确定了, 前者为 $(n-1)m$, 后者为 $N = |\mathcal{D}|$ 即语料 $\mathcal{C}$ 的词汇量大小. 而隐藏层的规模 n_h 是**可调参数**由用户指定.

为什么投影层的规模是 $(n-1)m$ 呢? 因为输入层包含 $Context(w)$ 中 $n-1$ 个词的词向量, 而投影层的向量 $\mathbf{x}_w$ 是这样构造的: 将输入层的 $n-1$ 个词向量按顺序首尾相接地拼起来形成一个长向量, 其长度当然就是 $(n-1)m$ 了. 有了向量 $\mathbf{x}_w$, 接下来的计算过程就很平凡了, 具体为

$$\begin{cases} \mathbf{z}_w = \tanh(W\mathbf{x}_w + \mathbf{p}), \\ \mathbf{y}_w = U\mathbf{z}_w + \mathbf{q}, \end{cases} \tag{3.5}$$

其中 tanh 为**双曲正切函数**, 用来做隐藏层的**激活函数**, 上式中, tanh 作用在向量上表示它作用在向量的每一个分量上.

注 3.4 有读者可能要问: 对于语料中的一个给定句子的前几个词, 其前面的词不足 $n-1$ 个怎么办? 此时, 可以人为地添加一个 (或几个) 填充向量就可以了, 它们也参与训练过程.

经过上述两步计算得到的 $\mathbf{y}_w = (y_{w,1}, y_{w,2}, \cdots, y_{w,N})^\top$ 只是一个长度为 N 的向量, 其分量不能表示概率. 如果想要 $\mathbf{y}_w$ 的分量 $y_{w,i}$ 表示当上下文为 $Context(w)$ 时下一个词恰为词典 $\mathcal{D}$ 中第 i 个词的概率, 则还需要做一个 **softmax** 归一化, 归一化后, $p(w|Context(w))$ 就可以表示为

$$p(w|Context(w)) = \frac{e^{y_{w,i_w}}}{\sum_{i=1}^{N} e^{y_{w,i}}}, \tag{3.6}$$

其中 i_w 表示词 w 在词典 $\mathcal{D}$ 中的索引.

公式 (3.6) 给出了概率 $p(w|Context(w))$ 的函数表示, 即找到了上一小节中提到的函数 $F(w, Context(w), \theta)$, 那么其中待确定的参数 θ 有哪些呢? 总结起来, 包括两部分

- 词向量: $\mathbf{v}(w) \in \mathbb{R}^m, w \in \mathcal{D}$ 以及填充向量.
- 神经网络参数: $W \in \mathbb{R}^{n_h \times (n-1)m}$, $\mathbf{p} \in \mathbb{R}^{n_h}$; $U \in \mathbb{R}^{N \times n_h}$, $\mathbf{q} \in \mathbb{R}^N$,

这些参数均通过训练算法得到. 值得一提的是, 通常的机器学习算法中, 输入都是已知的, 而在上述神经概率语言模型中, 输入 $\mathbf{v}(w)$ 也需要通过训练才能得到.

接下来, 简要地分析一下上述模型的运算量. 在如图 4 所示的神经网络中, 投影层、隐藏层和输出层的规模分别为 $(n-1)m$, n_h, N, 依次看看其中涉及的参数:

(1) n 是一个词的上下文中包含的词数, 通常不超过 5;
(2) m 是词向量长度, 通常是 $10^1 \sim 10^2$ 量级;
(3) n_h 由用户指定, 通常不需取得太大, 如 10^2 量级;
(4) N 是语料词汇量的大小, 与语料相关, 但通常是 $10^4 \sim 10^5$ 量级.

再结合 (3.5) 和 (3.6), 不难发现, 整个模型的大部分计算集中在隐藏层和输出层之间的矩阵向量运算, 以及输出层上的 softmax 归一化运算. 因此后续的相关研究工作中, 有很多是针对这一部分进行优化的, 其中就包括了 word2vec 的工作.

与 n-gram 模型相比, 神经概率语言模型有什么**优势**呢? 主要有以下两点:

1. 词语之间的相似性可以通过词向量来体现.

   举例来说, 如果某个 (英语) 语料中 $S_1 =$ "A dog is running in the room" 出现了 10000 次, 而 $S_2 =$ "A cat is running in the room" 只出现了 1 次. 按照 n-gram 模型的做法, $p(S_1)$ 肯定会远大于 $p(S_2)$. 注意, S_1 和 S_2 的唯一区别在于 dog 和 cat, 而这两个词无论是句法还是语义上都扮演了相同的角色, 因此, $p(S_1)$ 和 $p(S_2)$ 应该很相近才对.

   然而, 由神经概率语言模型算得的 $p(S_1)$ 和 $p(S_2)$ 是大致相等的. 原因在于: (1) 在神经概率语言模型中假定了 "相似的" 的词对应的词向量也是相似的; (2) 概率函数关于词向量是光滑的, 即词向量中的一个小变化对概率的影响也只是一个小变化. 这样一来, 对于下面这些句子

   ```
   A dog is running in the room
   A cat is running in the room
   The cat is running in a room
   A dog is walking in a bedroom
   The dog was walking in the room
      ...
   ```

   只要在语料库中出现一个, 其他句子的概率也会相应地增大.

2. 基于词向量的模型自带平滑化功能 (由 (3.6) 可知, $p(w|Context(w)) \in (0,1)$ 不会为零), 不再需要像 n-gram 那样进行额外处理了.

最后, 我们回过头来想想, 词向量在整个神经概率语言模型中扮演了什么角色呢? 训练时, 它是用来帮助构造目标函数的辅助参数, 训练完成后, 它也好像只是语言模型的一个副产品. 但这个副产品可不能小觑, 下一小节将对其作进一步阐述.

§3.4 词向量的理解

通过上一小节的讨论, 相信大家对词向量已经有一个初步的认识了. 接下来, 对词向量做进一步介绍.

在 NLP 任务中, 我们将自然语言交给机器学习算法来处理, 但机器无法直接理解人类的语言, 因此首先要做的事情就是将语言数学化, 如何对自然语言进行数学化呢? 词向量提供了一种很好的方式.

一种最简单的词向量是 **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 $\mathcal{D}$ 的大小 N, 向量的分量只有一个 1, 其它全为 0, 1 的位置对应该词在词典中的索引. 但这种词向量表示有一些缺点, 如容易受维数灾难的困扰, 尤其是将其用于 Deep Learning 场景时; 又如, 它不能很好地刻画词与词之间的相似性.

另一种词向量是 **Distributed Representation**, 它最早是 Hinton 于 1986 年提出的 ([1]), 可以克服 one-hot representation 的上述缺点. 其基本想法是: 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 (词法、语义上的) 相似性了. word2vec 中采用的就是这种 Distributed Representation 的词向量.

为什么叫做 Distributed Representation? 很多人问到这个问题. 我的一个理解是这样的: 对于 one-hot representation, 向量中只有一个非零分量, 非常集中 (有点孤注一掷的感觉); 而对于 Distributed Representation, 向量中有大量非零分量, 相对分散 (有点风险平摊的感觉), 把词的信息分布到各个分量中去了. 这一点, 跟并行计算里的分布式并行很像.

为更好地理解上述思想, 我们来举一个通俗的例子.

例 3.2 假设在二维平面上分布有 a 个不同的点, 给定其中的某个点, 现在想在平面上找到与这个点最相近的一个点.

我们是怎么做的呢? 首先, 建立一个直角坐标系, 基于该坐标系, 其上的每个点就唯一地对应一个坐标 (x, y); 接着引入欧氏距离; 最后分别计算这个点与其他 $a-1$ 个点之间的距离, 对应最小距离值的那个 (或那些) 点便是我们要找的点了.

上面的例子中, 坐标 (x, y) 的地位就相当于词向量, 它用来将平面上一个点的位置在数学上作量化. 坐标系建立好以后, 要得到某个点的坐标是很容易的. 然而, 在 NLP 任务中, 要得到词向量就复杂得多了, 而且词向量并不唯一, 其质量依赖于训练语料、训练算法等因素.

如何获取词向量呢? 有很多不同模型可用来估计词向量, 包括有名的 LSA (Latent Semantic Analysis) 和 LDA (Latent Dirichlet Allocation). 此外, 利用神经网络算法也是一种常用的方法, 上一小节介绍的神经概率语言模型就是一个很好的实例. 当然, 在那个模型中, 目标是生成语言模型, 词向量只是一个副产品. 事实上, 大部分情况下, 词向量和语言模型都是捆绑在一起的, 训练完成后两者同时得到. 用神经网络来训练语言模型的思想最早由百度 IDL (深度学习研究院) 的**徐伟**提出 ([14]). 这方面最经典的文章要数 **Bengio** 于 2003 年发

表在 JMLR 上的《A Neural Probabilistic Language Model》, 其后有一系列相关的研究工作, 其中也包括谷歌 **Tomas Mikolov** 团队的 word2vec.

Task	Benchmark		Performance	Timing (s)
Part of Speech (POS)	(Toutanova et al, 2003)	(Accuracy)	97.29%	3
Chunking (CHK)	CoNLL 2000	(F1)	94.32%	2
Name Entity Recognition (NER)	CoNLL 2003	(F1)	89.59%	2
Semantic Role Labeling (SRL)	CoNLL 2005	(F1)	75.49%	36
Syntactic Parsing (PSG)	Penn Treebank	(F1)	87.92%	74

图 6 SENNA performance in per-word accuracy for POS, and F1 score for all the other tasks. Timing corresponds to the time needed by SENNA to pass over the given test data set (Macbook Pro i7, 2.8GHz, Intel MKL). For PSG, F1 score is the one over all sentences.

一份好的词向量是很有价值的, 例如, **Ronan Collobert** 团队在软件包 **SENNA** ([12]) 中利用词向量进行了 POS、CHK、NER 等任务, 且取得了不错的效果 (见图 6 中的表格). 最近了解到词向量在机器翻译领域的一个应用 ([13]), 报道是这样的:

> 谷歌的 Tomas Mikolov 团队开发了一种词典和术语表的自动生成技术, 能够把一种语言转变成另一种语言. 该技术利用数据挖掘来构建两种语言的结构模型, 然后加以对比. 每种语言词语之间的关系集合即“语言空间”, 可以被表征为数学意义上的向量集合. 在向量空间内, 不同的语言享有许多共性, 只要实现一个向量空间向另一个向量空间的映射和转换, 语言翻译即可实现. 该技术效果非常不错, 对英语和西语间的翻译准确率高达 90%.

文 [5] 在引言中介绍算法原理时举了一个简单的例子, 可以帮助我们更好地理解词向量的工作原理, 特将其介绍如下.

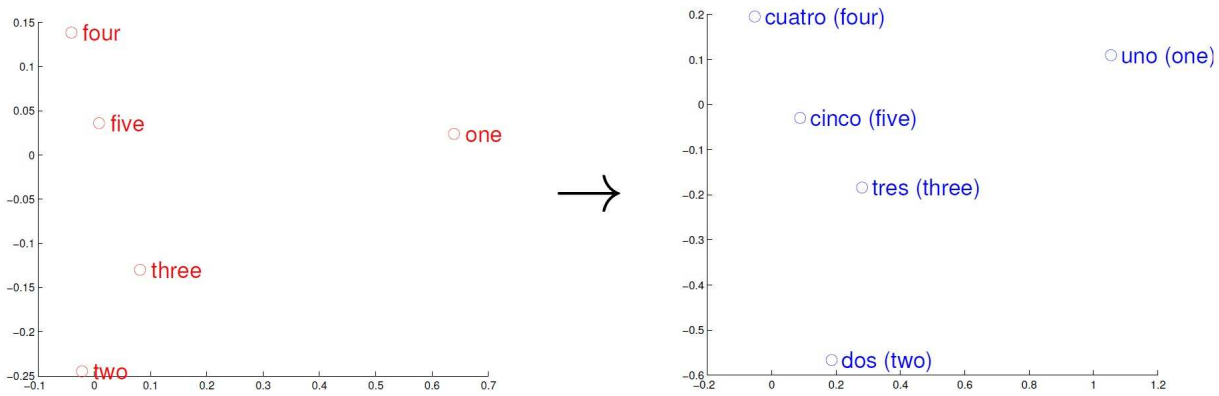

图 7 五个词在两个向量空间中的位置 (左: E 右: S)

考虑英语和西班牙语两种语言, 通过训练分别得到它们对应的词向量空间 E(nglish) 和 S(panish). 从英语中取出五个词 one, two, three, four, five, 设其在 E 中对应的词向量

分别为 u_1, u_2, u_3, u_4, u_5, 为方便作图, 利用主成分分析 (PCA) 降维, 得到相应的二维向量 v_1, v_2, v_3, v_4, v_5, 在二维平面上将这五个点描出来, 如图 7 左图所示.

类似地, 在西班牙语中取出 (与 one, two, three, four, five 对应的) uno, dos, tres, cuatro, cinco, 设其在 S 中对应的词向量分别为 s_1, s_2, s_3, s_4, s_5, 用 PCA 降维后的二维向量分别为 t_1, t_2, t_3, t_4, t_5, 将它们在二维平面上描出来 (可能还需作适当的旋转), 如图 7 右图所示.

观察左、右两幅图, 容易发现: 五个词在两个向量空间中的相对位置差不多, 这说明两种不同语言对应向量空间的结构之间具有相似性, 从而进一步说明了在词向量空间中利用距离刻画词之间相似性的合理性.

注意, 词向量只是针对 “词” 来提的, 事实上, 我们也可以针对更细粒度或更粗粒度来进行推广, 如**字向量** ([7]), **句子向量**和**文档向量** ([6]), 它们能为字、句子、文档等单元提供更好的表示.

§4 基于 Hierarchical Softmax 的模型

有了前面的准备, 本节开始正式介绍 word2vec 中用到的两个重要模型 — CBOW 模型 (Continuous Bag-of-Words Model) 和 Skip-gram 模型 (Continuous Skip-gram Model). 关于这两个模型, 作者 Tomas Mikolov 在文 [5] 给出了如图 8 和图 9 所示的示意图.

由图可见, 两个模型都包含三层: **输入层、投影层**和**输出层**. 前者是在已知当前词 w_t 的上下文 $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$ 的前提下预测当前词 w_t (见图 8); 而后者恰恰相反, 是在已知当前词 w_t 的前提下, 预测其上下文 $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$ (见图 9).

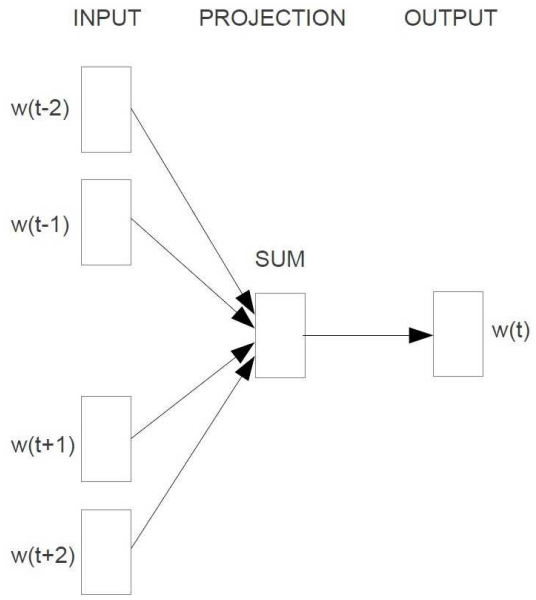

图 8 CBOW 模型

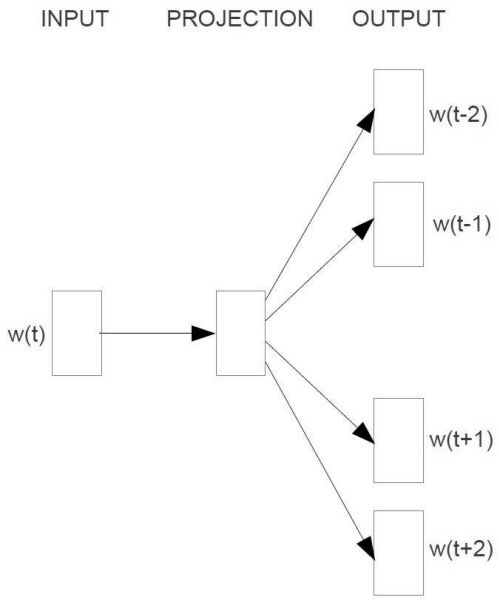

图 9 Skip-gram 模型

对于 CBOW 和 Skip-gram 两个模型, word2vec 给出了两套框架, 它们分别基于 Hierarchical Softmax 和 Negative Sampling 来进行设计. 本节介绍基于 Hierarchical Softmax 的 CBOW 和 Skip-gram 模型.

在 §3.2 中, 我们提到, 基于神经网络的语言模型的目标函数通常取为如下**对数似然函数**

$$\mathcal{L} = \sum_{w \in \mathcal{C}} \log p(w|Context(w)), \tag{4.1}$$

其中的关键是条件概率函数 $p(w|Context(w))$ 的构造, 文 [2] 中的模型就给出了这个函数的一种构造方法 (见 (3.6) 式).

对于 word2vec 中基于 Hierarchical Softmax 的 CBOW 模型, 优化的目标函数也形如 (4.1); 而对于基于 Hierarchical Softmax 的 Skip-gram 模型, 优化的目标函数则形如

$$\mathcal{L} = \sum_{w \in \mathcal{C}} \log p(Context(w)|w), \tag{4.2}$$

因此, 讨论过程中我们应将重点放在 $p(w|Context(w))$ 或 $p(Context(w)|w)$ 的构造上, 意识到这一点很重要, 因为它可以让我们目标明确、心无旁骛, 不致于陷入到一些繁琐的细节当中去. 接下来将从数学的角度对这两个模型进行详细介绍.

§4.1 CBOW 模型

本小节介绍 word2vec 中的第一个模型 — CBOW 模型.

§4.1.1 网络结构

图 10 给出了 CBOW 模型的网络结构, 它包括三层: 输入层、投影层和输出层. 下面以样本 $(Context(w), w)$ 为例 (这里假设 $Context(w)$ 由 w 前后各 c 个词构成), 对这三个层做简要说明.

1. **输入层**: 包含 $Context(w)$ 中 $2c$ 个词的词向量 $\mathbf{v}(Context(w)_1)$, $\mathbf{v}(Context(w)_2)$, $\cdots$, $\mathbf{v}(Context(w)_{2c}) \in \mathbb{R}^m$. 这里, m 的含义同上表示词向量的长度.

2. **投影层**: 将输入层的 $2c$ 个向量做求和累加, 即 $\mathbf{x}_w = \sum_{i=1}^{2c} \mathbf{v}(Context(w)_i) \in \mathbb{R}^m$.

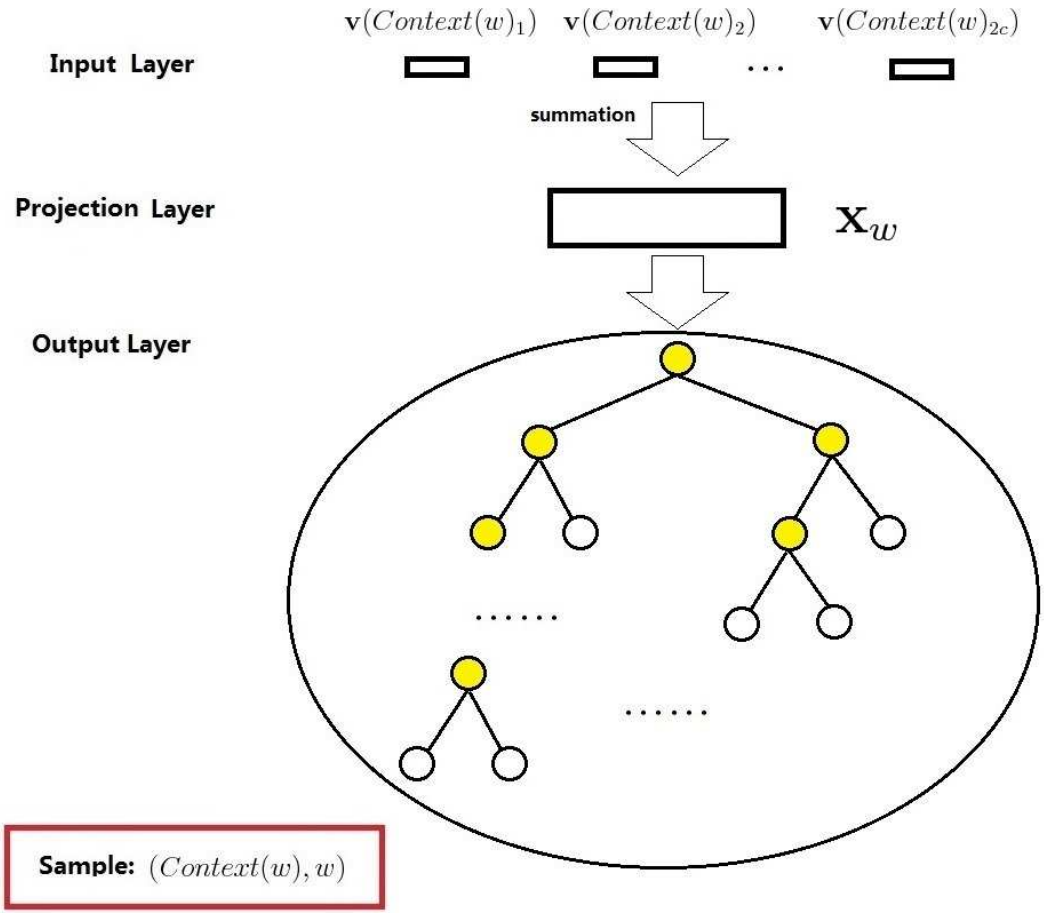

图 10 CBOW 模型的网络结构示意图

3. **输出层**: 输出层对应一棵二叉树, 它是以语料中出现过的词当叶子结点, 以各词在语料中出现的次数当权值构造出来的 Huffman 树. 在这棵 Huffman 树中, 叶子结点共 N $(= |\mathcal{D}|)$ 个, 分别对应词典 $\mathcal{D}$ 中的词, 非叶子结点 $N-1$ 个 (图中标成黄色的那些结点).

对比 §3.3 中神经概率语言模型的网络图 (见图 4) 和 CBOW 模型的结构图 (见图 10), 易知它们主要有以下三处不同:

1. (从输入层到投影层的操作) 前者是通过拼接, 后者通过**累加求和**.
2. (隐藏层) 前者有隐藏层, 后者**无隐藏层**.
3. (输出层) 前者是线性结构, 后者是**树形结构**.

在 §3.3 介绍的神经概率语言模型中, 我们指出, 模型的大部分计算集中在隐藏层和输出层之间的矩阵向量运算, 以及输出层上的 softmax 归一化运算. 而从上面的对比中可见, CBOW 模型对这些计算复杂度高的地方有针对性地进行了改变, 首先, 去掉了隐藏层, 其次, 输出层改用了 Huffman 树, 从而为利用 Hierarchical softmax 技术奠定了基础.

§4.1.2 梯度计算

Hierarchical Softmax 是 wor2vec 中用于提高性能的一项关键技术. 为描述方便起见, 在具体介绍这个技术之前, 先引入若干相关记号. 考虑 Huffman 树中的某个叶子结点, 假设它对应词典 $\mathcal{D}$ 中的词 w, 记

1. p^w: 从根结点出发到达 w 对应叶子结点的路径.
2. l^w: 路径 p^w 中包含结点的个数.
3. $p_1^w, p_2^w, \cdots, p_{l^w}^w$: 路径 p^w 中的 l^w 个结点, 其中 p_1^w 表示根结点, $p_{l^w}^w$ 表示词 w 对应的结点.
4. $d_2^w, d_3^w, \cdots, d_{l^w}^w \in \{0,1\}$: 词 w 的 Huffman 编码, 它由 $l^w - 1$ 位编码构成, d_j^w 表示路径 p^w 中第 j 个结点对应的编码 (根结点不对应编码).
5. $\theta_1^w, \theta_2^w, \cdots, \theta_{l^w-1}^w \in \mathbb{R}^m$: 路径 p^w 中**非叶子结点**对应的向量, θ_j^w 表示路径 p^w 中第 j 个非叶子结点对应的向量.

注 4.1 按理说, 我们要求的是词典 $\mathcal{D}$ 中每个词 (即 *Huffman* 树中所有叶子节点) 的向量, 为什么这里还要为 *Huffman* 树中每一个非叶子结点也定义一个同长的向量呢? 事实上, 它们只是算法中的辅助向量, 具体用途在下文中将会为大家解释清楚.

好了, 引入了这么一大堆抽象的记号, 接下来, 我们还是通过一个简单的例子把它们落到实处吧, 看图 11, 仍以预备知识中例 2.1 为例, 考虑词 $w =$ “足球” 的情形.

图 11 中由 4 条红色边串起来的 5 个节点就构成路径 p^w, 其长度 $l^w = 5$. $p_1^w, p_2^w, p_3^w, p_4^w, p_5^w$ 为路径 p^w 上的 5 个结点, 其中 p_1^w 对应根结点. $d_2^w, d_3^w, d_4^w, d_5^w$ 分别为 1, 0, 0, 1, 即 “足球” 的 Huffman 编码为 1001. 此外, $\theta_1^w, \theta_2^w, \theta_3^w, \theta_4^w$ 分别表示路径 p^w 上 4 个非叶子结点对应的向量.

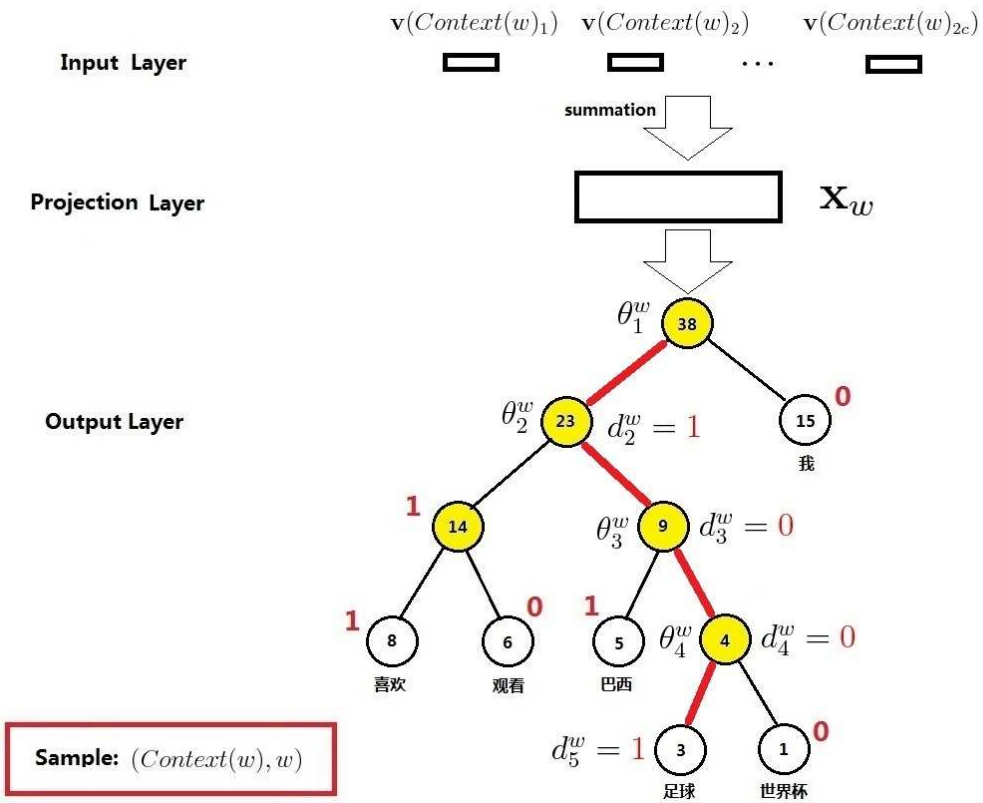

图 11 $w =$ “足球” 时的相关记号示意图

那么, 在如图 10 所示的网络结构下, 如何定义条件概率函数 $p(w|Contex(w))$ 呢? 更具体地说, 就是如何利用向量 $\mathbf{x}_w \in \mathbb{R}^m$ 以及 Huffman 树来定义函数 $p(w|Contex(w))$ 呢?

以图 11 中词 $w =$ “足球” 为例. 从根结点出发到达 “足球” 这个叶子节点, 中间共经历了 4 次分支 (每条红色的边对应一次分支), 而每一次分支都可视为进行了一次**二分类**.

既然是从二分类的角度来考虑问题, 那么对于每一个非叶子结点, 就需要为其左右孩子结点指定一个类别, 即哪个是正类 (标签为 1), 哪个是负类 (标签为 0). 碰巧, 除根结点以外, 树中每个结点都对应了一个取值为 0 或 1 的 Huffman 编码. 因此, 一种最自然的做法就是将 Huffman 编码为 1 的结点定义为正类, 编码为 0 的结点定义为负类. 当然, 这只是个约定而已, 你也可以将编码为 1 的结点定义为负类, 而将编码为 0 的结点定义为正类. 事实上, word2vec 选用的就是后者, 为方便读者对照着文档看源码, 下文中统一采用后者, 即约定

$$Label(p_i^w) = 1 - d_i^w, i = 2, 3, \cdots, l^w.$$

简言之就是, **将一个结点进行分类时, 分到左边就是负类, 分到右边就是正类.**

根据预备知识 §2.2 中介绍的逻辑回归, 易知, 一个结点被分为正类的概率是

$$\sigma(\mathbf{x}_w^\top \theta) = \frac{1}{1 + e^{-\mathbf{x}_w^\top \theta}},$$

被分为负类的概率当然就等于

$$1 - \sigma(\mathbf{x}_w^\top \theta),$$

注意, 上式中有个叫 θ 的向量, 它是待定参数, 显然, 在这里非叶子结点对应的那些向量 θ_i^w 就可以扮演参数 θ 的角色 (这也是为什么将它们取名为 θ_i^w 的原因).

对于从根结点出发到达 “足球” 这个叶子节点所经历的 4 次二分类, 将每次分类结果的概率写出来就是

1. 第 1 次: $p(d_2^w|\mathbf{x}_w,\theta_1^w)=1-\sigma(\mathbf{x}_w^\top\theta_1^w)$;
2. 第 2 次: $p(d_3^w|\mathbf{x}_w,\theta_2^w)=\sigma(\mathbf{x}_w^\top\theta_2^w)$;
3. 第 3 次: $p(d_4^w|\mathbf{x}_w,\theta_3^w)=\sigma(\mathbf{x}_w^\top\theta_3^w))$;
4. 第 4 次: $p(d_5^w|\mathbf{x}_w,\theta_4^w)=1-\sigma(\mathbf{x}_w^\top\theta_4^w)$,

但是, 我们要求的是 $p(\text{足球}|Contex(\text{足球}))$, 它跟这 4 个概率值有什么关系呢? 关系就是

$$p(\text{足球}|Contex(\text{足球}))=\prod_{j=2}^{5}p(d_j^w|\mathbf{x}_w,\theta_{j-1}^w).$$

至此, 通过 $w=$ “足球” 的小例子, **Hierarchical Softmax** 的基本思想其实就已经介绍完了. 小结一下: **对于词典 $\mathcal{D}$ 中的任意词 w, Huffman 树中必存在一条从根结点到词 w 对应结点的路径 p^w (且这条路径是唯一的). 路径 p^w 上存在 l^w-1 个分支, 将每个分支看做一次二分类, 每一次分类就产生一个概率, 将这些概率乘起来, 就是所需的 $p(w|Context(w))$.**

条件概率 $p(w|Context(w))$ 的一般公式可写为 (对应文 [4] 中的 (3) 式)

$$p(w|Context(w))=\prod_{j=2}^{l^w}p(d_j^w|\mathbf{x}_w,\theta_{j-1}^w), \tag{4.3}$$

其中

$$p(d_j^w|\mathbf{x}_w,\theta_{j-1}^w)=\begin{cases}\sigma(\mathbf{x}_w^\top\theta_{j-1}^w), & d_j^w=0;\\ 1-\sigma(\mathbf{x}_w^\top\theta_{j-1}^w), & d_j^w=1,\end{cases}$$

或者写成整体表达式

$$p(d_j^w|\mathbf{x}_w,\theta_{j-1}^w)=[\sigma(\mathbf{x}_w^\top\theta_{j-1}^w)]^{1-d_j^w}\cdot[1-\sigma(\mathbf{x}_w^\top\theta_{j-1}^w)]^{d_j^w}.$$

注 4.2 按理说, 这里的整体表达式也可以写为

$$p(d_j^w|\mathbf{x}_w,\theta_{j-1}^w)=(1-d_j^w)[\sigma(\mathbf{x}_w^\top\theta_{j-1}^w)]+d_j^w[1-\sigma(\mathbf{x}_w^\top\theta_{j-1}^w)],$$

但如果这样的话, 后续对 $p(d_j^w|\mathbf{x}_w,\theta_{j-1}^w)$ 作对数操作就麻烦了.

注 4.3 在 §3.3 中, 最后得到的条件概率为

$$p(w|Context(w)) = \frac{e^{y_{w,i_w}}}{\sum_{i=1}^{N} e^{y_{w,i}}}.$$

具体见 (3.6) 式, 由于这里有个归一化操作, 因此显然成立

$$\sum_{w\in\mathcal{D}} p(w|Context(w)) = 1.$$

然而, 对于由 (4.3) 定义的概率, 是否也能满足上式呢? 这个问题留给读者思考.

将 (4.3) 代入对数似然函数 (4.1), 便得

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \log \prod_{j=2}^{l^w} \left\{ [\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{1-d_j^w} \cdot [1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{d_j^w} \right\} \\ &= \sum_{w\in\mathcal{C}} \sum_{j=2}^{l^w} \left\{ (1-d_j^w)\cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] \right\}, \end{aligned} \tag{4.4}$$

为下面梯度推导方便起见, 将上式中花括号里的内容简记为 $\mathcal{L}(w,j)$, 即

$$\mathcal{L}(w,j) = (1-d_j^w)\cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]. \tag{4.5}$$

至此, 已经推导出对数似然函数 (4.4), 这就是 CBOW 模型的目标函数, 接下来讨论它的优化, 即如何将这个函数最大化. word2vec 里面采用的是**随机梯度上升法***. 而梯度类算法的关键是给出相应的梯度计算公式, 因此接下来重点讨论梯度的计算.

随机梯度上升法的做法是: 每取一个样本 $(Context(w), w)$, 就对目标函数中的所有 (相关) 参数做一次刷新. 观察目标函数 $\mathcal{L}$ 易知, 该函数中的参数包括向量 $\mathbf{x}_w, \theta_{j-1}^w$, $w\in\mathcal{C}$, $j=2,\cdots,l^w$. 为此, 先给出函数 $\mathcal{L}(w,j)$ 关于这些向量的梯度.

首先考虑 $\mathcal{L}(w,j)$ 关于 θ_{j-1}^w 的梯度计算.

$$\begin{aligned}\frac{\partial \mathcal{L}(w,j)}{\partial \theta_{j-1}^w} &= \frac{\partial}{\partial \theta_{j-1}^w}\left\{ (1-d_j^w)\cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] \right\} \\ &= (1-d_j^w)[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]\mathbf{x}_w - d_j^w \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)\mathbf{x}_w \quad \text{(利用 (2.1) 式)} \\ &= \left\{ (1-d_j^w)[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] - d_j^w \sigma(\mathbf{x}_w^\top \theta_{j-1}^w) \right\} \mathbf{x}_w \quad \text{(合并)} \\ &= \left[1 - d_j^w - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w) \right] \mathbf{x}_w. \end{aligned}$$

于是, θ_{j-1}^w 的更新公式可写为

$$\theta_{j-1}^w := \theta_{j-1}^w + \eta \left[1 - d_j^w - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w) \right] \mathbf{x}_w,$$

*求最小值用 (随机) 梯度下降法, 求最大值用 (随机) 梯度上升法, 这种基于梯度的方法通常统称为 (随机) 梯度下降法. 这里强调 "上升" 是想提醒大家这是在求最大值.

其中 η 表示**学习率**, 下同.

接下来考虑 $\mathcal{L}(w,j)$ 关于 $\mathbf{x}_w$ 的梯度. 观察 (4.5) 可发现, $\mathcal{L}(w,j)$ 中关于变量 $\mathbf{x}_w$ 和 θ_{j-1}^w 是**对称**的 (即两者可交换位置), 因此, 相应的梯度 $\frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}$ 也只需在 $\frac{\partial \mathcal{L}(w,j)}{\partial \theta_{j-1}^w}$ 的基础上对这两个向量交换位置就可以了, 即

$$\frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w} = \left[1 - d_j^w - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)\right] \theta_{j-1}^w.$$

到这里, 细心的读者可能已经看出问题来了: 我们的最终目的是要求词典 $\mathcal{D}$ 中每个词的词向量, 而这里的 $\mathbf{x}_w$ 表示的是 $Context(w)$ 中各词词向量的累加. 那么, 如何利用 $\frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}$ 来对 $\mathbf{v}(\widetilde{w}), \widetilde{w} \in Context(w)$ 进行更新呢? word2vec 中的做法很简单, 直接取

$$\mathbf{v}(\widetilde{w}) := \mathbf{v}(\widetilde{w}) + \eta \sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}, \quad \widetilde{w} \in Context(w),$$

即把 $\sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}$ 贡献到 $Context(w)$ 中每一个词的词向量上. 这个应该很好理解, 既然 $\mathbf{x}_w$ 本身就是 $Context(w)$ 中各词词向量的累加, 求完梯度后当然也应该将其贡献到每个分量上去.

注 4.4 当然, 读者这里需要考虑的是: 采用**平均**贡献会不会更合理 ? 即使用公式

$$\mathbf{v}(\widetilde{w}) := \mathbf{v}(\widetilde{w}) + \frac{\eta}{|Context(w)|} \sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}, \quad \widetilde{w} \in Context(w),$$

其中 $|Context(w)|$ 表示 $Context(w)$ 中词的个数.

下面以样本 $(Context(w), w)$ 为例, 给出 CBOW 模型中采用随机梯度上升法更新各参数的伪代码.

1. $\mathbf{e} = \mathbf{0}$.
2. $\mathbf{x}_w = \sum_{u \in Context(w)} \mathbf{v}(u)$.
3. **FOR** $j = 2 : l^w$ **DO**
   {
   3.1 $q = \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)$
   3.2 $g = \eta(1 - d_j^w - q)$
   3.3 $\mathbf{e} := \mathbf{e} + g\theta_{j-1}^w$
   3.4 $\theta_{j-1}^w := \theta_{j-1}^w + g\mathbf{x}_w$
   }
4. **FOR** $u \in Context(w)$ **DO**
   {
   $\mathbf{v}(u) := \mathbf{v}(u) + \mathbf{e}$
   }

注意, 步 3.3 和步 3.4 不能交换次序, 即 θ_{j-1}^w 应等贡献到 $\mathbf{e}$ 后再做更新.

注 4.5 结合上面的伪代码, 简单给出其与 *word2vec* 源码中的对应关系如下: *syn0* 对应 $\mathbf{v}(\cdot)$, *syn1* 对应 θ_{j-1}^{w}, *neu1* 对应 $\mathbf{x}_w$, *neu1e* 对应 $\mathbf{e}$.

§4.2 Skip-gram 模型

本小节介绍 word2vec 中的另一个模型 — Skip-gram 模型, 由于推导过程与 CBOW 大同小异, 因此会沿用上小节引入的记号.

§4.2.1 网络结构

图 12 给出了 Skip-gram 模型的网络结构, 同 CBOW 模型的网络结构一样, 它也包括三层: 输入层、投影层和输出层. 下面以样本 $(w, Context(w))$ 为例, 对这三个层做简要说明.

1. **输入层**: 只含当前样本的中心词 w 的词向量 $\mathbf{v}(w) \in \mathbb{R}^m$.

2. **投影层**: 这是个恒等投影, 把 $\mathbf{v}(w)$ 投影到 $\mathbf{v}(w)$. 因此, **这个投影层其实是多余的**, 这里之所以保留投影层主要是方便和 CBOW 模型的网络结构做对比.

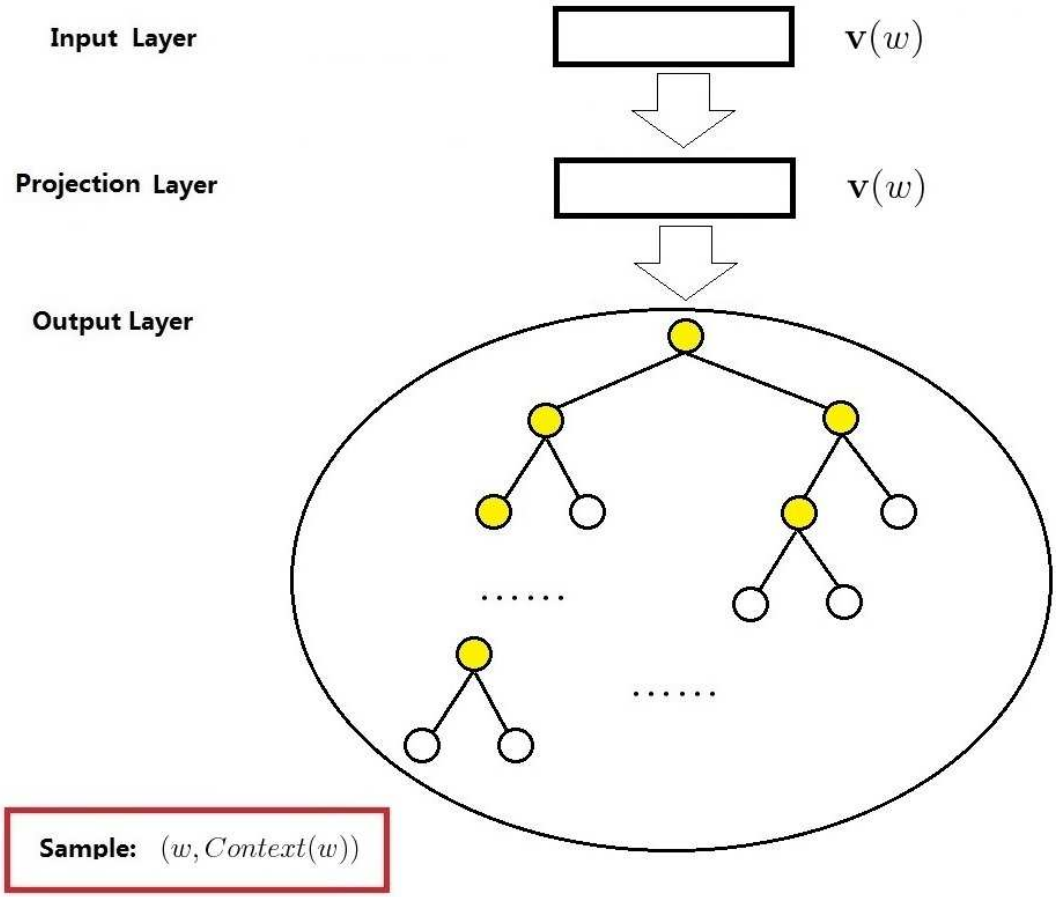

图 12 Skip-gram 模型的网络结构示意图

3. **输出层**: 和 CBOW 模型一样, 输出层也是一棵 Huffman 树.

§4.2.2 梯度计算

对于 Skip-gram 模型, 已知的是当前词 w, 需要对其上下文 $Context(w)$ 中的词进行预测, 因此目标函数应该形如 (4.2), 且关键是条件概率函数 $p(Context(w)|w)$ 的构造, Skip-gram

模型中将其定义为

$$p(Context(w)|w) = \prod_{u \in Context(w)} p(u|w),$$

上式中的 $p(u|w)$ 可按照上小节介绍的 Hierarchical Softmax 思想, 类似于 (4.3) 地写为

$$p(u|w) = \prod_{j=2}^{l^u} p(d_j^u|\mathbf{v}(w), \theta_{j-1}^u),$$

其中

$$p(d_j^u|\mathbf{v}(w), \theta_{j-1}^u) = [\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{1-d_j^u} \cdot [1 - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{d_j^u}. \tag{4.6}$$

将 (4.6) 依次代回, 可得对数似然函数 (4.2) 的具体表达式

$$\begin{aligned} \mathcal{L} &= \sum_{w \in \mathcal{C}} \log \prod_{u \in Context(w)} \prod_{j=2}^{l^u} \left\{ [\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{1-d_j^u} \cdot [1 - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{d_j^u} \right\} \\ &= \sum_{w \in \mathcal{C}} \sum_{u \in Context(w)} \sum_{j=2}^{l^u} \left\{ (1 - d_j^u) \cdot \log[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] + d_j^u \cdot \log[1 - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] \right\}. \end{aligned} \tag{4.7}$$

同样, 为下面梯度推导方便起见, 将上式中花括号里的内容简记为 $\mathcal{L}(w, u, j)$, 即

$$\mathcal{L}(w, u, j) = (1 - d_j^u) \cdot \log[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] + d_j^u \cdot \log[1 - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)].$$

至此, 已经推导出对数似然函数的表达式 (4.7), 这就是 Skip-gram 模型的目标函数. 接下来同样利用**随机梯度上升法**对其进行优化, 关键是要给出两类梯度.

首先考虑 $\mathcal{L}(w, u, j)$ 关于 θ_{j-1}^u 的梯度计算 (与 CBOW 模型对应部分的推导完全类似).

$$\begin{aligned} \frac{\partial \mathcal{L}(w, u, j)}{\partial \theta_{j-1}^u} &= \frac{\partial}{\partial \theta_{j-1}^u} \left\{ (1 - d_j^u) \cdot \log[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] + d_j^u \cdot \log[1 - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] \right\} \\ &= (1 - d_j^u)[1 - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]\mathbf{v}(w) - d_j^u \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)\mathbf{v}(w) \quad \text{(利用 (2.1) 式)} \\ &= \left\{ (1 - d_j^u)[1 - \sigma(\mathbf{v}(w)^\top \mathbf{v}_{j-1}^u)] - d_j^u \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u) \right\} \mathbf{v}(w) \quad \text{(合并)} \\ &= \left[1 - d_j^u - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u) \right] \mathbf{v}(w). \end{aligned}$$

于是, θ_{j-1}^u 的更新公式可写为

$$\theta_{j-1}^u := \theta_{j-1}^u + \eta \left[1 - d_j^u - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u) \right] \mathbf{v}(w).$$

接下来考虑 $\mathcal{L}(w, u, j)$ 关于 $\mathbf{v}(w)$ 的梯度. 同样利用 $\mathcal{L}(w, u, j)$ 中 $\mathbf{v}(w)$ 和 θ_{j-1}^w 的**对称性**, 有

$$\frac{\partial \mathcal{L}(w, u, j)}{\partial \mathbf{v}(w)} = \left[1 - d_j^u - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u) \right] \theta_{j-1}^u.$$

于是, $\mathbf{v}(w)$ 的更新公式可写为

$$\mathbf{v}(w) := \mathbf{v}(w) + \eta \sum_{u \in Context(w)} \sum_{j=2}^{l^u} \frac{\partial \mathcal{L}(w,u,j)}{\partial \mathbf{v}(w)}.$$

下面以样本 $(w, Context(w))$ 为例, 给出 Skip-gram 模型中采用随机梯度上升法更新各参数的伪代码.

$\mathbf{e} = \mathbf{0}$
FOR $u \in Context(w)$ **DO**
{
  FOR $j = 2 : l^u$ **DO**
  {
    1. $q = \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)$
    2. $g = \eta(1 - d_j^u - q)$
    3. $\mathbf{e} := \mathbf{e} + g\theta_{j-1}^u$
    4. $\theta_{j-1}^u := \theta_{j-1}^u + g\mathbf{v}(w)$
  }
}
$\mathbf{v}(w) := \mathbf{v}(w) + \mathbf{e}$

但是, word2vec 源码中, 并不是等 $Context(w)$ 中的所有词都处理完后才刷新 $\mathbf{v}(w)$, 而是, 每处理完 $Context(w)$ 中的一个词 u, 就及时刷新一次 $\mathbf{v}(w)$, 具体为

FOR $u \in Context(w)$ **DO**
{
  $\mathbf{e} = \mathbf{0}$
  FOR $j = 2 : l^u$ **DO**
  {
    1. $q = \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)$
    2. $g = \eta(1 - d_j^u - q)$
    3. $\mathbf{e} := \mathbf{e} + g\theta_{j-1}^u$
    4. $\theta_{j-1}^u := \theta_{j-1}^u + g\mathbf{v}(w)$
  }
  $\mathbf{v}(w) := \mathbf{v}(w) + \mathbf{e}$
}

同样, 需要注意的是, 循环体内的步 3 和步 4 不能交换次序, 即 θ_{j-1}^u 要等贡献到 $\mathbf{e}$ 后才更新.

注 4.6 结合上面的伪代码, 简单给出其与 *word2vec* 源码中的对应关系如下: *syn0* 对应 $\mathbf{v}(\cdot)$, *syn1* 对应 θ_{j-1}^u, *neu1e* 对应 $\mathbf{e}$.

§5 基于 Negative Sampling 的模型

本节将介绍基于 Negative Sampling 的 CBOW 和 Skip-gram 模型. **Negative Sampling** (简称为 **NEG**) 是 Tomas Mikolov 等人在文 [4] 中提出的, 它是 **NCE** (Noise Contrastive Estimation) 的一个简化版本, 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.

注 5.1 NCE 的细节有点复杂, 其本质是利用已知的概率密度函数来估计未知的概率密度函数. 简单来说, 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. 具体可参考文献 [9].

§5.1 CBOW 模型

在 CBOW 模型中, 已知词 w 的上下文 $Context(w)$, 需要预测 w, 因此, 对于给定的 $Context(w)$, 词 w 就是一个**正样本**, 其它词就是**负样本**了. 负样本那么多, 该如何选取呢? 这个问题比较独立, 我们放到后面再进行介绍.

假定现在已经选好了一个关于 w 的负样本子集 $NEG(w) \neq \emptyset$. 且对 $\forall\ \widetilde{w} \in \mathcal{D}$, 定义

$$L^w(\widetilde{w}) = \begin{cases} 1, & \widetilde{w} = w; \\ 0, & \widetilde{w} \neq w, \end{cases}$$

表示词 $\widetilde{w}$ 的标签, 即正样本的标签为 1, 负样本的标签为 0.

对于一个给定的正样本 $(Context(w), w)$, 我们希望最大化

$$g(w) = \prod_{u \in \{w\} \cup NEG(w)} p(u|Context(w)), \tag{5.1}$$

其中

$$p(u|Context(w)) = \begin{cases} \sigma(\mathbf{x}_w^\top \theta^u), & L^w(u) = 1; \\ 1 - \sigma(\mathbf{x}_w^\top \theta^u), & L^w(u) = 0, \end{cases}$$

或者写成整体表达式

$$p(u|Context(w)) = \left[\sigma(\mathbf{x}_w^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right]^{1 - L^w(u)}, \tag{5.2}$$

这里 $\mathbf{x}_w$ 仍表示 $Context(w)$ 中各词的词向量之和, 而 $\theta^u \in \mathbb{R}^m$ 表示词 u 对应的一个 (辅助) 向量, 为待训练参数.

为什么要最大化 $g(w)$ 呢? 让我们先来看看 $g(w)$ 的表达式, 将 (5.2) 代入 (5.1), 有

$$g(w) = \sigma(\mathbf{x}_w^\top \theta^w) \prod_{u \in NEG(w)} \left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right],$$

其中 $\sigma(\mathbf{x}_w^\top \theta^w)$ 表示当上下文为 $Context(w)$ 时, 预测中心词为 w 的概率, 而 $\sigma(\mathbf{x}_w^\top \theta^u)$, $u \in NEG(w)$ 则表示当上下文为 $Context(w)$ 时, 预测中心词为 u 的概率 (这里可看成一个二分类问题, 具体可参见预备知识中的逻辑回归). 从形式上看, 最大化 $g(w)$, 相当于最大化 $\sigma(\mathbf{x}_w^\top \theta^w)$, 同时最小化所有的 $\sigma(\mathbf{x}_w^\top \theta^u)$, $u \in NEG(w)$. 这不正是我们希望的吗? **增大正样本的概率同时降低负样本的概率**. 于是, 对于一个给定的语料库 $\mathcal{C}$, 函数

$$G = \prod_{w \in \mathcal{C}} g(w)$$

就可以作为整体优化的目标. 当然, 为计算方便, 对 G 取对数, 最终的目标函数 (为和前面章

节统一起见, 这里仍将其记为 $\mathcal{L}$) 就是

$$\begin{aligned}\mathcal{L} &= \log G = \log \prod_{w\in\mathcal{C}} g(w) = \sum_{w\in\mathcal{C}} \log g(w) \\ &= \sum_{w\in\mathcal{C}} \log \prod_{u\in\{w\}\cup NEG(w)} \left\{ \left[\sigma(\mathbf{x}_w^\top \theta^u)\right]^{L^w(u)} \cdot \left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right]^{1-L^w(u)} \right\} \\ &= \sum_{w\in\mathcal{C}} \sum_{u\in\{w\}\cup NEG(w)} \left\{ L^w(u)\cdot \log\left[\sigma(\mathbf{x}_w^\top \theta^u)\right] + \left[1-L^w(u)\right]\cdot \log\left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right] \right\}.\end{aligned} \tag{5.3}$$

注 5.2 将 (5.3) 进行改写, 可得

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \sum_{u\in\{w\}\cup NEG(w)} \left\{ L^w(u)\cdot \log\left[\sigma(\mathbf{x}_w^\top \theta^u)\right] + \left[1-L^w(u)\right]\cdot \log\left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right] \right\} \\ &= \sum_{w\in\mathcal{C}} \left\{ \log\left[\sigma(\mathbf{x}_w^\top \theta^w)\right] + \sum_{u\in NEG(w)} \log\left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right] \right\} \\ &= \sum_{w\in\mathcal{C}} \left\{ \log\left[\sigma(\mathbf{x}_w^\top \theta^w)\right] + \sum_{u\in NEG(w)} \log\left[\sigma(-\mathbf{x}_w^\top \theta^u)\right] \right\} \quad (\text{利用了等式 } 1-\sigma(x)=\sigma(-x)\ )\end{aligned}$$

上式中花括号中的项就是 Tomas Mikolov 等人在文 [4] 中提到的目标函数 (4) 式.

作者 Yoav Goldberg 和 Omer Levy 对文 [4] 中只给出 (4) 式不是很满意, 于是在文 [21] 中针对 Skip-gram 模型尝试对 (4) 式进行了推导, 有兴趣的读者可以参考一下.

为下面梯度推导方便起见, 将 (5.3) 式中花括号里的内容简记为 $\mathcal{L}(w,u)$, 即

$$\mathcal{L}(w,u) = L^w(u)\cdot \log\left[\sigma(\mathbf{x}_w^\top \theta^u)\right] + \left[1-L^w(u)\right]\cdot \log\left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right]. \tag{5.4}$$

接下来利用**随机梯度上升法**对 (5.3) 进行优化, 关键是要给出 $\mathcal{L}$ 的两类梯度. 首先考虑 $\mathcal{L}(w,u)$ 关于 θ^u 的梯度计算.

$$\begin{aligned}\frac{\partial \mathcal{L}(w,u)}{\partial \theta^u} &= \frac{\partial}{\partial \theta^u}\left\{ L^w(u)\cdot \log\left[\sigma(\mathbf{x}_w^\top \theta^u)\right] + \left[1-L^w(u)\right]\cdot \log\left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right] \right\} \\ &= L^w(u)[1-\sigma(\mathbf{x}_w^\top \theta^u)]\mathbf{x}_w - [1-L^w(u)]\sigma(\mathbf{x}_w^\top \theta^u)\mathbf{x}_w \quad (\text{利用 (2.1) 式}) \\ &= \left\{ L^w(u)[1-\sigma(\mathbf{x}_w^\top \theta^u)] - [1-L^w(u)]\sigma(\mathbf{x}_w^\top \theta^u) \right\}\mathbf{x}_w \quad (\text{合并}) \\ &= \left[L^w(u) - \sigma(\mathbf{x}_w^\top \theta^u) \right]\mathbf{x}_w.\end{aligned} \tag{5.5}$$

于是, θ^u 的更新公式可写为

$$\theta^u := \theta^u + \eta\left[L^w(u) - \sigma(\mathbf{x}_w^\top \theta^u) \right]\mathbf{x}_w. \tag{5.6}$$

接下来考虑 $\mathcal{L}(w,u)$ 关于 $\mathbf{x}_w$ 的梯度. 同样利用 $\mathcal{L}(w,u)$ 中 $\mathbf{x}_w$ 和 θ^u 的**对称性**, 有

$$\frac{\partial \mathcal{L}(w,u)}{\partial \mathbf{x}_w} = \left[L^w(u) - \sigma(\mathbf{x}_w^\top \theta^u) \right]\theta^u. \tag{5.7}$$

于是, 利用 $\frac{\partial \mathcal{L}(w,u)}{\partial \mathbf{x}_w}$, 可得 $\mathbf{v}(\widetilde{w}), \widetilde{w} \in Context(w)$ 的更新公式为 (至于为什么可以这么做请参考上一节基于 Hierarchical Softmax 的 CBOW 模型对应部分的解释)

$$\mathbf{v}(\widetilde{w}) := \mathbf{v}(\widetilde{w}) + \eta \sum_{u \in \{w\} \cup NEG(w)} \frac{\partial \mathcal{L}(w,u)}{\partial \mathbf{x}_w}, \quad \widetilde{w} \in Context(w). \tag{5.8}$$

下面以样本 $(Context(w), w)$ 为例, 给出基于 Negative Sampling 的 CBOW 模型中采用随机梯度上升法更新各参数的伪代码.

1. $\mathbf{e} = \mathbf{0}$.
2. $\mathbf{x}_w = \sum_{u \in Context(w)} \mathbf{v}(u)$.
3. **FOR** $u = \{w\} \cup NEG(w)$ **DO**
   {
   3.1 $q = \sigma(\mathbf{x}_w^\top \theta^u)$
   3.2 $g = \eta(L^w(u) - q)$
   3.3 $\mathbf{e} := \mathbf{e} + g\theta^u$
   3.4 $\theta^u := \theta^u + g\mathbf{x}_w$
   }
4. **FOR** $u \in Context(w)$ **DO**
   {
   $\mathbf{v}(u) := \mathbf{v}(u) + \mathbf{e}$
   }

注意, 步 3.3 和步 3.4 不能交换次序, 即 θ^u 要等贡献到 $\mathbf{e}$ 后才更新.

注 5.3 结合上面的伪代码, 简单给出其与 *word2vec* 源码中的对应关系如下: *syn0* 对应 $\mathbf{v}(\cdot)$, *syn1neg* 对应 θ^u, *neu1* 对应 $\mathbf{x}_w$, *neu1e* 对应 $\mathbf{e}$.

§5.2 Skip-gram 模型

本小节介绍基于 Negative Sampling 的 Skip-gram 模型.

有了 Hierarchical Softmax 框架下由 CBOW 模型过渡到 Skip-gram 模型的推导经验, 这里, 我们仍然可以这样来做. 首先, 将优化目标函数由原来的

$$G = \prod_{w\in\mathcal{C}} g(w)$$

改写为

$$G = \prod_{w\in\mathcal{C}} \prod_{u\in Context(w)} g(u), \tag{5.9}$$

这里, $\prod_{u\in Context(w)} g(u)$ 表示对于一个给定的样本 $(w, Context(w))$, 我们希望最大化的量, $g(u)$ 类似于上一节的 $g(w)$, 定义为

$$g(u) = \prod_{z\in\{u\}\cup NEG(u)} p(z|w), \tag{5.10}$$

其中 $NEG(u)$ 表示处理词 u 时生成的负样本子集, 条件概率

$$p(z|w) = \begin{cases} \sigma(\mathbf{v}(w)^\top\theta^z), & L^u(z)=1; \\ 1-\sigma(\mathbf{v}(w)^\top\theta^z), & L^u(z)=0, \end{cases}$$

或者写成整体表达式

$$p(z|w) = \left[\sigma(\mathbf{v}(w)^\top\theta^z)\right]^{L^u(z)} \cdot \left[1-\sigma(\mathbf{v}(w)^\top\theta^z)\right]^{1-L^u(z)}. \tag{5.11}$$

同样, 我们取 G 的对数, 最终的目标函数就是

$$\begin{aligned}
\mathcal{L} &= \log G = \log \prod_{w\in\mathcal{C}} \prod_{u\in Context(w)} g(u) = \sum_{w\in\mathcal{C}} \sum_{u\in Context(w)} \log g(u) \\
&= \sum_{w\in\mathcal{C}} \sum_{u\in Context(w)} \log \prod_{z\in\{u\}\cup NEG(u)} p(z|w) \\
&= \sum_{w\in\mathcal{C}} \sum_{u\in Context(w)} \sum_{z\in\{u\}\cup NEG(u)} \log p(z|w) \\
&= \sum_{w\in\mathcal{C}} \sum_{u\in Context(w)} \sum_{z\in\{u\}\cup NEG(u)} \log\left\{\left[\sigma(\mathbf{v}(w)^\top\theta^z)\right]^{L^u(z)} \cdot \left[1-\sigma(\mathbf{v}(w)^\top\theta^z)\right]^{1-L^u(z)}\right\} \\
&= \sum_{w\in\mathcal{C}} \sum_{u\in Context(w)} \sum_{z\in\{u\}\cup NEG(u)} \\
&\quad \left\{L^u(z)\cdot\log\left[\sigma(\mathbf{v}(w)^\top\theta^z)\right] + [1-L^u(z)]\cdot\log\left[1-\sigma(\mathbf{v}(w)^\top\theta^z)\right]\right\}.
\end{aligned} \tag{5.12}$$

到这里, 接下来的梯度计算与参数更新公式的推导都可以按部就班地来进行了, 具体推导过程请自行完成.

值得一提的是, word2vec 源代码中基于 Negative Sampling 的 Skip-gram 模型并不是基于目标函数 (5.12) 来编程的. 判断依据很简单, 如果基于 (5.12) 来编程的话, 对于每一个样本 $(w, Context(w))$, 需要针对 $Context(w)$ 中的每一个词进行负采样, 而 word2vec 源码中只是针对 w 进行了 $|Context(w)|$ 次负采样.

那么, word2vec 源码在这一块的根据是什么呢?

通过代码细节反推回来, 我得到了如下一个理解 (不一定是源码作者的初衷, 或者还有其他理解方式, 将自己的想法写在这里供读者参考, 欢迎交流): 它本质上用的还是 CBOW 模型, 只是将原来通过求和累加做整体用的上下文 $Contex(w)$ 拆成一个一个来考虑. 此时, 对于一个给定的样本 $(w, Context(w))$, 我们希望最大化

$$g(w) = \prod_{\widetilde{w} \in Context(w)} \prod_{u \in \{w\} \cup NEG^{\widetilde{w}}(w)} p(u|\widetilde{w}),$$

其中

$$p(u|\widetilde{w}) = \begin{cases} \sigma(\mathbf{v}(\widetilde{w})^\top \theta^u), & L^w(u) = 1; \\ 1 - \sigma(\mathbf{v}(\widetilde{w})^\top \theta^u), & L^w(u) = 0, \end{cases}$$

或者写成整体表达式

$$p(u|\widetilde{w}) = \left[\sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right]^{1 - L^w(u)},$$

这里 $NEG^{\widetilde{w}}(w)$ 表示处理词 $\widetilde{w}$ 时生成的负样本子集. 于是, 对于一个给定的语料库 $\mathcal{C}$, 函数

$$G = \prod_{w \in \mathcal{C}} g(w)$$

就可以作为整体优化的目标. 同样, 我们取 G 的对数, 最终的目标函数就是

$$\begin{aligned} \mathcal{L} &= \log G = \log \prod_{w \in \mathcal{C}} g(w) = \sum_{w \in \mathcal{C}} \log g(w) \\ &= \sum_{w \in \mathcal{C}} \log \prod_{\widetilde{w} \in Context(w)} \prod_{u \in \{w\} \cup NEG^{\widetilde{w}}(w)} \left\{ \left[\sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right]^{1 - L^w(u)} \right\} \\ &= \sum_{w \in \mathcal{C}} \sum_{\widetilde{w} \in Context(w)} \sum_{u \in \{w\} \cup NEG^{\widetilde{w}}(w)} \\ &\quad \left\{ L^w(u) \cdot \log \left[\sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right] + [1 - L^w(u)] \cdot \log \left[1 - \sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right] \right\}. \end{aligned} \tag{5.13}$$

为下面梯度推导方便起见, 将三重求和符号下花括号里的内容简记为 $\mathcal{L}(w, \widetilde{w}, u)$, 即

$$\mathcal{L}(w, \widetilde{w}, u) = L^w(u) \cdot \log \left[\sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right] + [1 - L^w(u)] \cdot \log \left[1 - \sigma(\mathbf{v}(\widetilde{w})^\top \theta^u)\right].$$

接下来利用**随机梯度上升法**对 (5.13) 进行优化, 关键是要给出 $\mathcal{L}$ 的两类梯度. 首先考虑 $\mathcal{L}(w,\widetilde{w},u)$ 关于 θ^u 的梯度计算.

$$\begin{aligned}
&\frac{\partial \mathcal{L}(w,\widetilde{w},u)}{\partial \theta^u}\\
=\ & \frac{\partial}{\partial \theta^u}\left\{L^w(u)\cdot \log\left[\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)\right]+[1-L^w(u)]\cdot\log\left[1-\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)\right]\right\}\\
=\ & L^w(u)[1-\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)]\mathbf{v}(\widetilde{w})-[1-L^w(u)]\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)\mathbf{v}(\widetilde{w}) \quad \text{(利用 (2.1) 式)}\\
=\ & \left\{L^w(u)[1-\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)]-[1-L^w(u)]\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)\right\}\mathbf{v}(\widetilde{w}) \quad \text{(合并)}\\
=\ & \left[L^w(u)-\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)\right]\mathbf{v}(\widetilde{w}).
\end{aligned}\tag{5.14}$$

于是, θ^u 的更新公式可写为

$$\theta^u := \theta^u+\eta\left[L^w(u)-\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)\right]\mathbf{v}(\widetilde{w}).\tag{5.15}$$

接下来考虑 $\mathcal{L}(w,\widetilde{w},u)$ 关于 $\mathbf{v}(\widetilde{w})$ 的梯度. 利用 $\mathcal{L}(w,\widetilde{w},u)$ 中 $\mathbf{v}(\widetilde{w})$ 和 θ^u 的**对称性**, 有

$$\frac{\partial \mathcal{L}(w,\widetilde{w},u)}{\partial \mathbf{v}(\widetilde{w})}=\left[L^w(u)-\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)\right]\theta^u,$$

于是, $\mathbf{v}(\widetilde{w})$ 的更新公式可写为

$$\mathbf{v}(\widetilde{w}) := \mathbf{v}(\widetilde{w})+\eta\sum_{u\in\{w\}\cup NEG^{\widetilde{w}}(w)}\frac{\partial \mathcal{L}(w,\widetilde{w},u)}{\partial \mathbf{v}(\widetilde{w})}.$$

下面以样本 $(w, Context(w))$ 为例, 给出基于 Negative Sampling 的 Skip-gram 模型中采用随机梯度上升法更新各参数的伪代码.

FOR $\widetilde{w}=Context(w)$ **DO**
{
  $\mathbf{e}=\mathbf{0}$.
  FOR $u=\{w\}\cup NEG^{\widetilde{w}}(w)$ **DO**
  {
    $q=\sigma(\mathbf{v}(\widetilde{w})^\top\theta^u)$
    $g=\eta(L^w(u)-q)$
    $\mathbf{e}:=\mathbf{e}+g\theta^u$
    $\theta^u:=\theta^u+g\mathbf{v}(\widetilde{w})$
  }
  $\mathbf{v}(\widetilde{w}):=\mathbf{v}(\widetilde{w})+\mathbf{e}$
}

注意, 步 3.3 和步 3.4 不能交换次序, 即 θ^u 要等贡献到 $\mathbf{e}$ 后才更新.

注 5.4 结合上面的伪代码, 简单给出其与 *word2vec* 源码中的对应关系如下: *syn0* 对应 $\mathbf{v}(\cdot)$, *syn1neg* 对应 θ^u, *neu1e* 对应 $\mathbf{e}$.

§5.3 负采样算法

顾名思义, 在基于 Negative Sampling 的 CBOW 和 Skip-gram 模型中, 负采样是个很重要的环节, 对于一个给定的词 w, 如何生成 $NEG(w)$ 呢?

词典 $\mathcal{D}$ 中的词在语料 $\mathcal{C}$ 中出现的次数有高有低, 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. 这就是我们对采样过程的一个大致要求, 本质上就是一个**带权采样问题**, 相关的算法有很多, 我之前在博文 [19] 中就给出过两个具体算法.

下面先用一段通俗的描述来帮助读者理解带权采样的机理.

> 设词典 $\mathcal{D}$ 中的每一个词 w 对应一个线段 $l(w)$, 长度为
>
> $$len(w) = \frac{\text{counter}(w)}{\sum\limits_{u\in\mathcal{D}} \text{counter}(u)}, \tag{5.16}$$
>
> 这里 counter$(\cdot)$ 表示一个词在语料 $\mathcal{C}$ 中出现的次数 (分母中的求和项用来做归一化). 现在将这些线段首尾相连地拼接在一起, 形成一个长度为 1 的单位线段. 如果随机地往这个单位线段上打点, 则其中长度越长的线段 (对应高频词) 被打中的概率就越大.

接下来再谈谈 word2vec 中的具体做法. 记 $l_0 = 0$, $l_k = \sum\limits_{j=1}^{k} len(w_j)$, $k = 1, 2, \cdots, N$, 这里 w_j 表示词典 $\mathcal{D}$ 中第 j 个词, 则以 $\{l_j\}_{j=0}^{N}$ 为剖分节点可得到区间 $[0, 1]$ 上的一个**非等距剖分**, $I_i = (l_{i-1}, l_i]$, $i = 1, 2, \cdots, N$ 为其 N 个剖分区间. 进一步引入区间 $[0, 1]$ 上的一个**等距离剖分**, 剖分节点为 $\{m_j\}_{j=0}^{M}$, 其中 $M >> N$, 具体见图 13 给出的示意图.

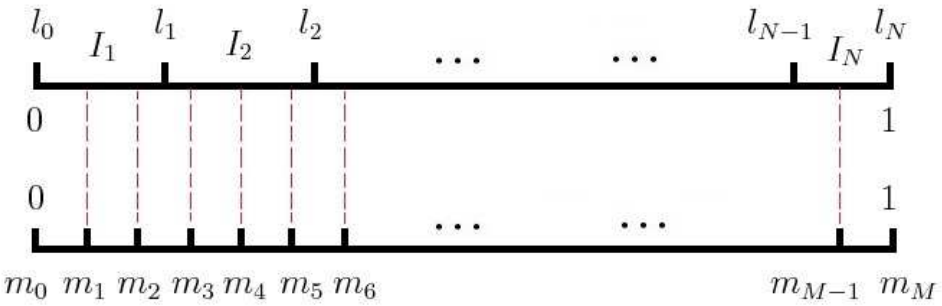

图 13 Table$(\cdot)$ 映射的建立示意图

将内部剖分节点 $\{m_j\}_{j=1}^{M-1}$ 投影到非等距剖分上, 如图 13 中的红色虚线所示, 则可建立 $\{m_j\}_{j=1}^{M-1}$ 与区间 $\{I_j\}_{j=1}^{N}$ (或者说 $\{w_j\}_{j=1}^{N}$) 的映射关系

$$\text{Table}(i) = w_k, \quad \text{where} \quad m_i \in I_k, \quad i = 1, 2, \cdots, M-1. \tag{5.17}$$

有了这个映射, 采样就简单了: 每次生成一个 $[1, M-1]$ 间的随机整数 r, Table(r) 就是一个样本. 当然, 这里还有一个细节, 当对 w_i 进行负采样时, 如果碰巧选到 w_i 自己怎么办? 那就跳过去呗 :-), 代码中也是这么处理的.

值得一提的是, word2vec 源码中为词典 $\mathcal{D}$ 中的词设置权值时, 不是直接用 $\text{counter}(w)$, 而是对其作了 α 次幂, 其中 $\alpha = \frac{3}{4}$, 即 (5.16) 变成了

$$len(w) = \frac{[\text{counter}(w)]^{\frac{3}{4}}}{\sum\limits_{u\in\mathcal{D}}[\text{counter}(u)]^{\frac{3}{4}}}.$$

此外, 代码中取 $M = 10^8$ (对应源码中变量 table_size), 而映射 (5.17) 则是通过一个名为 InitUnigramTable 的函数来完成.

§6 若干源码细节

§6.1 $\sigma(x)$ 的近似计算

由图像 1 可见, sigmoid 函数 $\sigma(x)$ 在 $x = 0$ 附近变化剧烈, 往两边逐渐趋于平缓, 当 $x < -6$ 或者 $x > 6$ 时函数值就基本不变了, 前者趋于 0, 后者趋于 1.

如果在某种应用场景中需要计算大量不同 x 对应的 $\sigma(x)$, 且对 $\sigma(x)$ 值的精确度不是非常严格, 那么, 基于上述观察, 我们就可以采用这样一种**近似计算方法**.

将区间 $[-6, 6]$ 等距剖分为 K 等份, 剖分节点分别记为 $x_0, x_1, \cdots, x_K$, 它们可表为 $x_i = x_0 + ih$, 其中 $x_0 = -6$, 步长 $h = \frac{12}{K}$.

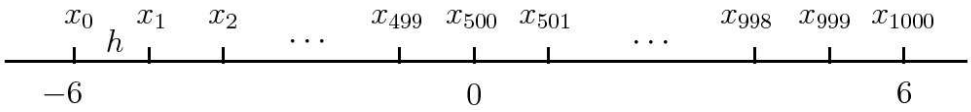

图 14 $K = 1000$ 时区间 $[-6, 6]$ 的等距剖分示意图

事先将 $\sigma(x_i)$ 的值计算好并保存起来, 计算 $\sigma(x)$ 时, 采用如下近似公式

$$\sigma(x) \approx \begin{cases} 0, & x \le -6 \\ \sigma(x_k), & x \in (-6, 6) \\ 1, & x \ge 6 \end{cases} \tag{6.1}$$

其中, k 等于 $\frac{x-x_0}{h}$ 对小数点后第一位四舍五入后的整数值 (即 x_k 表示与 x 距离最近的剖分节点), 或者简单地取为 $\frac{x-x_0}{h}$ 的向上或向下取整都可以. 公式 (6.1) 有点像查表, word2vec 源码中采用了这种技巧来提高效率.

采用查表方式为什么就能提高运算效率呢? 因为 $\sigma(x)$ 中涉及到指数运算 e^z, 它是一个初等超越函数, 内部实现时通常利用幂级数展开

$$e^z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \frac{z^3}{3!} + \ldots$$

取其前 n 项做近似计算. 其优点是当 z 较小 ($|z| < 1$) 时, 很小的 n 就可以做很好的近似, 因此计算速度较快; 但当 z 较大时, 展开项数较多, 运算量也较大. 查表的方法是一次性计算好一批节点的值, 以后就只需进行匹配查询了.

注 6.1 *关于 $\sigma(x)$ 的近似计算, word2vec 源码中有一处不严谨的地方, 见下面的代码, $i <$ EXP_TABLE_SIZ 应改为 $i <=$ EXP_TABLE_SIZE.*

```
for (i = 0; i < EXP_TABLE_SIZE; i++)
{
    expTable[i] = exp((i / (real)EXP_TABLE_SIZE * 2 - 1) * MAX_EXP);
    expTable[i] = expTable[i] / (expTable[i] + 1);
}
```

§6.2 词典的存储

在 word2vec 源码中, 词典 $\mathcal{D}$ 是通过**哈希**技术来存储的. 为简单起见, 这里直接用代码中用到的变量来进行描述.

首先, 开设一个长度为 vocab_hash_size (默认值为 3×10^7) 的整型数组 vocab_hash, 并将每个分量初始化为 -1. 然后, 为词典 $\mathcal{D}$ 中的词建立如下映射

$$\text{vocab\_hash}[\text{hv}(w_j)] = j,$$

其中 $\text{hv}(w_j)$ 表示词 w_j (根据某个公式得到) 的哈希值. 当然, 可能出现

$$\text{hv}(w_i) = \text{hv}(w_j),\ i \neq j$$

的情形, 此时采用**线性探测的开放定址法**来解决冲突, 即如果计算 w_i 的哈希值 $\text{hv}(w_i)$ 后, 发现 vocab_hash 中的该位置已被其他词占用, 则顺序往下查找, 直到找到一个未被占用的位置 (若已到数组末尾, 则从头开始查找).

在字典中查找某个词 (如 w) 时, 先计算其哈希值 $\text{hv}(w)$, 然后在 vocab_hash 中找到 $\text{hv}(w)$ 对应的位置, 如果 $\text{vocab\_hash}[\text{hv}(w_j)] = -1$, 则表示 w 未被收录在词典中; 否则, 需将 w 和 $w_{\text{vocab\_hash}[\text{hv}(w_j)]}$ (即词典中第 $\text{vocab\_hash}[\text{hv}(w_j)]$ 个词) 进行比较, 如果相同, 则 w 在词典中的索引就是 $\text{vocab\_hash}[\text{hv}(w_j)]$, 如果不同, 则继续往下匹配, 直到 w 和某个 w_k 匹配上或者 vocab_hash 的值为 -1 为止, 前者表示 w 在词典中的索引就是 k, 后者表示 w 未被收录在词典中.

§6.3 换行符

构建词典 $\mathcal{D}$ 时, 其中包含一个特殊的词 $</s>$, 它被放在词典的第一个位置上. 该词代表语料中出现的换行符, 它虽然也对应一个词向量, 但这个向量在训练过程中并不参与运算. 其作用主要是用来判别一个句子的结束.

word2vec 源码的函数 SortVocab 中有一段代码是删除词典中出现次数小于 min_count 的低频词, 并重新为词典建立哈希映射. 由于这段代码前已对字典中的词按照出现次数进行了降序排列, 因此, 按理说, 只需对词典**从后往前**遍历, 剔除出现次数小于 min_count 的低频词即可, 但 word2vec 相应部分的代码是这样的:

```
for (a = 0; a < size; a ++)
{
   if (vocab[a].cn < min_count)
   {
      vocab_size --;
      free(vocab[vocab_size].word);
   }
   else
   {
      ......
   }
}
```

注意, 换行符 $</s>$ 对应的词并没有参与排序, 因此, 当碰巧换行符在语料中出现的次数小于 min_count, 则有可能多删除一个出现次数大于等于 min_count 的词.

§6.4 低频词和高频词

在 word2vec 源码中, 对语料中的低频词和高频词都进行了一些特殊的处理.

1. **低频词的处理**

利用语料 $\mathcal{C}$ 建立词典 $\mathcal{D}$ 时, 并不是每个出现过的词都能收录到词典中. 代码中引入一个叫做 **min_count** 的阀值参数 (默认取值为 5), 若某个词在语料中出现的次数小于它, 则将其从词典中删除. 如果不想做任何删除, 则只需将 min_count 取为 0 或 1 即可. 注意, 这些未能进入词典 $\mathcal{D}$ 的词在训练时也是不可见的. 可以简单地理解为训练前语料进行了这样一次预处理: 将所有出现次数小于 min_count 的词都挖走了.

此外, 为提高效率, 在利用语料构建词典时, 代码中会根据词典当前的规模来决定是否需要对词典中的低频词进行清理. 具体做法如下: 预先设定一个阀值参数 min_reduce (默认值为 1), 如果当前词典 $\mathcal{D}_{current}$ 的规模 $|\mathcal{D}_{current}|$ 满足

$$|\mathcal{D}_{current}| > 0.7 \cdot \text{ vocab\_hash\_size},$$

则从 $\mathcal{D}_{current}$ 删除所有出现次数小于等于 min_reduce 的词.

2. **高频词的处理**

文 [4] 中提到, 在一个大语料库中, 那些最常见的词 (the most frequent words) 将出现百万甚至千万次, 如 “的”、“是” 等. 同低频词相比, 这些词提供的有用信息更少. 而且这次词对应的词向量在众多样本上进行训练时也不会发生显著的变化. 因此, 文 [4] 中提出了一种叫做 **Subsampling** 的技巧, 用来提高训练速度 (提高 $2 \sim 10$ 倍, 而且可提高那些相对来说词频没那么高的词的词向量精度), 具体做法如下: 给定一个词频阀值参数 t (即 word2vec 源码中叫做 sample 的变量), 词 w 将以

$$\text{prob}(w) = 1 - \sqrt{\frac{t}{f(w)}} \tag{6.2}$$

的概率被舍弃, 其中

$$f(w) = \frac{\text{counter}(w)}{\sum\limits_{u \in \mathcal{D}} \text{counter}(u)}, \quad w \in \mathcal{D}$$

表示 w 的频率, 显然, Subsampling 只是针对那些满足 $f(w) > t$ 的所谓的高频词.

值得一提的是, word2vec 源码中实际用的公式并不是 (6.2), 而是

$$\text{prob}(w) = 1 - \left(\sqrt{\frac{t}{f(w)}} + \frac{t}{f(w)} \right).$$

具体实现是这样的: 假设当前处理词 w, 先计算 $ran = \sqrt{\frac{t}{f(w)}} + \frac{t}{f(w)}$ 的值, 然后产生一个 $(0, 1)$ 上的随机 (实) 数 r, 如果 $r > ran$, 则舍弃词 w. 由于在区间 $(0, 1)$ 上产生一个大于 ran 的随机数的概率是 $1 - ran$, 因此上述做法就等同于以 $1 - ran$ 的概率舍弃词 w.

§6.5 窗口及上下文

模型训练是以**行**为单位进行的 (因此, 如果语料文件每**行**存储一个句子, 则训练时每次就是处理一个句子), 利用特殊词 $< /s >$ 进行分割. 当然, 句子不能太长 (极端情况就是将整个语料存成一个超长的句子), 源码中设置了一个阀值参数 MAX_SENTENCE_LENGTH (默认值为 1000), 如果一**行**的词数超过 MAX_SENTENCE_LENGTH, 则强行进行截断.

对于一个给定的**行**, 设它包含 T 个词, 则可以得到 T 个训练样本, 因为每一个词都对应一个样本. 考虑该**行**中的某个词 w, 只需定义好 $Contex(w)$, 就可得到样本 $(Contex(w), w)$ 或 $(w, Contex(w))$.

那么如何定义 $Context(w)$ 呢? 前面的描述中采用的是在词 w 的前后各取 c 个词, 但 word2vec 中采用的策略是这样的: 事先设置一个窗口阀值参数 window (默认值为 5), 每次构造 $Context(w)$ 时, 首先生成区间 $[1, \text{window}]$ 上的一个随机 (整) 数 $\widetilde{c}$, w 前后各取 $\widetilde{c}$ 个词就构成了 $Context(w)$.

前文中对比 §3.3 中神经概率语言模型和 CBOW 时, 我们有提到它们在投影方式上有一个不同, 前者采用的是首尾相接, 后者则是直接累加. 采用直接相加的投影方式除了可使所得向量更短外, 还有一个好处: 对于一**行**的首尾的某些词, 其前、后的词数可能不足 $\widetilde{c}$ 个, 此时如果是首尾相接的方式则需要补充填充向量, 但直接相加就没有这个问题, 无非是求和项里少了几项罢了.

§6.6 自适应学习率

前文中谈及学习率时, 统一使用的是常数 η. 而在 word2vec 源码中, 用到了自适应技术. 具体是这样的: 预先设置一个初始的学习率 η_0 (默认值为 0.025), 每处理完 10000 (源码里直接指定, 没有设置参数, 但可以根据经验调整) 个词, 则按照以下公式对学习率进行一次调整

$$\eta = \eta_0 \left(1 - \frac{\text{word\_count\_actual}}{\text{train\_words} + 1}\right), \tag{6.3}$$

其中 word_count_actual 表示当前已处理过的词数, $\text{train\_words} = \sum_{w \in \mathcal{D}} \text{counter}(w)$, $+1$ 是为了防止分母为零.

由 (6.3) 可见, η 将随着训练的进行而逐渐变小, 且趋于 0. 但是, 学习率也不能过小, 因此, 源码中加入了一个阀值 η_{min}, 一旦 $\eta < \eta_{min}$, 则将 η 固定为 η_{min}, 其中 $\eta_{min} = 10^{-4} \cdot \eta_0$.

§6.7 参数初始化与训练

模型训练采用的是随机梯度上升法, 且**只对语料遍历一次**, 这也是其高效的原因之一.

模型中需要训练的参数包括逻辑回归对应的参数向量, 以及词典 $\mathcal{D}$ 中每个词的词向量. 在对这些参数进行初始化时, 前者采用的是**零初始化**, 后者采用的是**随机初始化**, word2vec 源码中的具体公式为

$$\frac{[\text{rand}()/\text{RAND\_MAX}] - 0.5}{m},$$

易知, 初始词向量的分量均落在区间 $[-\frac{0.5}{m}, \frac{0.5}{m}]$, 这里 m 仍表示词向量的长度.

关于词向量和参数向量, word2vec 源码中出现了 syn0, syn1 和 syn1neg 三个一维数组, 其中 syn0 对应 Huffman 树所有叶子结点的词向量, syn1 对应 Huffman 树中所有非叶子结点的参数向量, syn1neg 对应基于 Negative Sampling 的模型中与词相关的参数向量.

§6.8 多线程并行

word2vec 中支持多线程并行, 源码中有个参数 num_threads (默认取 1). 其中与线程相关的主要代码如下.

```
pthread_t *pt = (pthread_t *)malloc(num_threads*sizeof(pthread_t));
for (a = 0; a < num_threads; a++)
   pthread_create(&pt[a], NULL, TrainModelThread, (void *)a);
for (a = 0; a < num_threads; a++)
   pthread_join(pt[a], NULL);
```

其中 TrainModelThread 是单线程训练模块. 对于多线程情形, 需要对语料文件进行负载平衡地划分, 每个线程处理一部分, 具体为

```
fseek(fi, file_size/(long long)num_threads*(long long)id, SEEK_SET);
```

其中 file_size 表示整个训练语料文件的总字节数. 由此可见, 划分是基于字节而不是基于词.

当然, 我们也可以把 word2vec 作**分布式**并行实现. 不过呢, 文 [4] 在引言中介绍 Skip-gram 模型时有提到:

> This makes the training extremely efficient: an optimized single-machine implementation can train on more than 100 billion words in one day.

这样的效率应该已经能满足大多数人的要求了吧.

§6.9 几点疑问和思考

1. word2vec 源码的函数 TrainModelThread 中, 当走 Hierarchical Softmax 那一支时, 有一段代码是用来求近似 $\sigma(\cdot)$ 值的, 具体如下

```
if (f <= -MAX_EXP)
   continue;
else if (f >= MAX_EXP)
   continue;
else
   f = expTable[(int)((f+MAX_EXP)*(EXP_TABLE_SIZE/MAX_EXP/2))];

g = (1 - vocab[word].code[d] - f) * alpha;
......
```

   按照公式 (6.1), 当 f <= - MAX_EXP 时, 应该取 f = 0; 当 f >= MAX_EXP 时, 应该取 f = 1. 代码中的 continue 简单地跳过去了. 显然, 当 f 等于 0 或 1 时, 代入 g 中, 其值不一定为零, 因此, 这里应该是有一个近似. 但奇怪的是, 走 Negative Sampling 那一支的时候, 在相应部分作者又特别注意了这一点, 看以下代码片段.

```
if (f > MAX_EXP)
   g = (label - 1) * alpha;
else if (f < -MAX_EXP)
   g = (label - 0) * alpha;
else
   ......
```

   不知道作者这样处理是不是为了提高效率.

2. 前面讨论算法时, 都是假定语料已经分好词. 对于英文语料, 分词不是什么问题, 但对于中文语料, 分词的预处理则是必须的, 不同分词器会得到不同的分词结果, 那么 word2vec 所得词向量的质量是如何依赖于中文分词的质量的呢?

3. Hierarchical Softmax 和 Negative Sampling 是两大类不同的方法, 但 word2vec 源码中对这两个分支采用的并不是二择一的方式, 用户可以通过参数选取 (如取 negative = 5, hs = 1) 同时选中这两支, 相当于是一种混合方法, 在这种情况下, 训练得到的词向量质量如何呢?

4. 投影层采用累积相加的方式, 会使得 $Context(w)$ 中各词的顺序不再敏感, 例如 "我**爱**足球" 和 "足球**爱**我" 在训练过程中会视为同一样本. 因此, 如果在投影层换回首尾相接的方式会怎么样呢?

5. word2vec 几个模型中的目标函数中均未考虑正则项, 如果加入正则会怎么样呢?

6. 前文介绍的算法都是针对一个静态的语料库来讨论的, 如果用户的语料数据是动态阶段性地获取的, 譬如, 当前只有一批语料, 过一段时间后又有另一批语料, ..., 针对这种情形, 如果想进行增量训练, 该如何调整现有的框架?

参考文献

[1] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. **Learning representations by backpropagating errors.** Nature, 323(6088):533-536, 1986.

[2] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. **A neural probabilistic language model.** Journal of Machine Learning Research (JMLR), 3:1137-1155, 2003.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. **Efficient Estimation of Word Representations in Vector Space.** arXiv:1301.3781, 2013.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. **Distributed Representations of Words and Phrases and their Compositionality.** arXiv:1310.4546, 2013.

[5] Tomas Mikolov, Quoc V. Le, Ilya Sutskever. **Exploiting Similarities among Languages for Machine Translation.** arXiv:1309.4168v1, 2013.

[6] Quoc V. Le, Tomas Mikolov. **Distributed Representations of Sentences and Documents.** arXiv:1405.4053, 2014.

[7] Xiaoqing Zheng, Hanyang Chen, Tianyu Xu. **Deep Learning for Chinese Word Segmentation and POS tagging.** Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 647-657.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa. **Natural Language Processing (Almost) from Scratch.** Journal of Machine Learning Research (JMLR), 12:2493-2537, 2011.

[9] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. The Journal ofMachine Learning Research, 13:307-361, 2012.

[10] 百度百科中的 “哈夫曼树” 词条.

[11] 吴军. **《数学之美》**. 人民邮电出版社, 2012.

[12] http://ml.nec-labs.com/senna/

[13] http://www.looooker.com/archives/5621

[14] licstar. **Deep Learning in NLP (一) 词向量和语言模型**.
http://licstar.net/archives/328

[15] **深度学习 word2vec 笔记之基础篇**.
http://blog.csdn.net/mytestmy/article/details/26961315

[16] **深度学习 word2vec 笔记之算法篇**.
http://blog.csdn.net/mytestmy/article/details/26969149

[17] 邓澍军, 陆光明, 夏龙. **Deep Learning 实战之 word2vec**, 2014.

[18] 杨超. **Word2Vec 的一些理解**.
http://www.zhihu.com/question/21661274/answer/19331979

[19] **基于权值的微博用户采样算法研究**.
http://blog.csdn.net/itplus/article/details/9079297

[20] **利用 word2vec 训练的字向量进行中文分词**.
http://blog.csdn.net/itplus/article/details/17122431

[21] Yoav Goldberg, Omer Levy. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. arXiv: 1402.3722v1, 2014. (http://arxiv.org/pdf/1402.3722v1.pdf)