深入浅出Word2Vec原理解析

原创 Microstrong Microstrong 2020-03-30

收录于话题

#自然语言处理模型汇总

16个

本文概览：

1. 背景知识

Word2Vec是语言模型中的一种，它是从大量文本预料中以无监督方式学习语义知识的模型，被广泛地应用于自然语言处理中。

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。因此，我们先来了解一些语言模型方面的知识。

1.1 统计语言模型

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 $W = (w_1, w_2, \ldots, w_T)$ 表示由T个词 $w_1, w_2, \ldots, w_T$ 按顺序构成的一个句子，则 $w_1, w_2, \ldots, w_T$ 的联合概率为：

$$p(W) = p(w_1, w_2, \ldots, w_T)$$

$p(W)$被称为语言模型，即用来计算这个句子概率的模型。利用Bayes公式，上式可以被链式地分解为：

$$p(W) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)\ldots p(w_T|w_1, w_2, \ldots, w_{T-1}) \tag{1}$$

其中的条件概率 $p(w_1), p(w_2|w_1), \ldots, p(w_T|w_1, w_2, \ldots, w_{T-1})$ 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，就可以很快地计算出相应地概率$p(W)$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。不防假设语料库对应词典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，但这个量级还是有点吓人。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、最大熵马尔可夫模型、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。

1.2 N-gram模型

考虑 $p(w_k|w_1, \ldots, w_{k-1})$ 的近似计算。利用Bayes公式，有：

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

根据大数定理，当语料库足够大时， $p(w_k|w_1, \ldots, w_{k-1})$ 可以近似地表示为：

$$p(w_k|w_1, \ldots, w_{k-1}) \approx \frac{count(w_1, \ldots, w_k)}{count(w_1, \ldots, w_{k-1})} \tag{2}$$

其中， $count(w_1, \ldots, w_k)$ 表示词串 $w_1, \ldots, w_k$ 在语料中出现的次数， $count(w_1, \ldots, w_{k-1})$ 表示词串 $w_1, \ldots, w_{k-1}$ 在语料中出现的次数。可想而知，当k很大时， $count(w_1, \ldots, w_k)$ 和 $count(w_1, \ldots, w_{k-1})$ 的统计将会多么的耗时。

从公式（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，它做了一个$n-1$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，

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

于是，公式（2）就变成了

$$p(w_k|w_1,\ldots,w_{k-1}) \approx \frac{count(w_{k-n+1},\ldots,w_k)}{count(w_{k-n+1},\ldots,w_{k-1})} \tag{3}$$

以$n=2$为例，就有

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

这样简化，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。

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

表1：模型参数数量与n的关系

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（具体可以参考吴军在《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），考虑两个问题：

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

显然不能，但这是一个无法回避的问题，哪怕你的预料库有多么大。平滑化技术就是用来处理这个问题的，这里不展开讨论。

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解决问题的方法：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：

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

其中，C表示语料(Corpus)，$Context(w)$表示词w的上下文，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 $Context(w_i = w_{i-n+1}, \ldots, w_{i-1})$ 。

当然，实际应用中常采用最大对数似然，即把目标函数设为

$$L = \sum_{w \in C} logp(w|Context(w)) \tag{4}$$

然后对这个函数进行最大化。

从公式（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，一旦对公式（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的构建了。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。

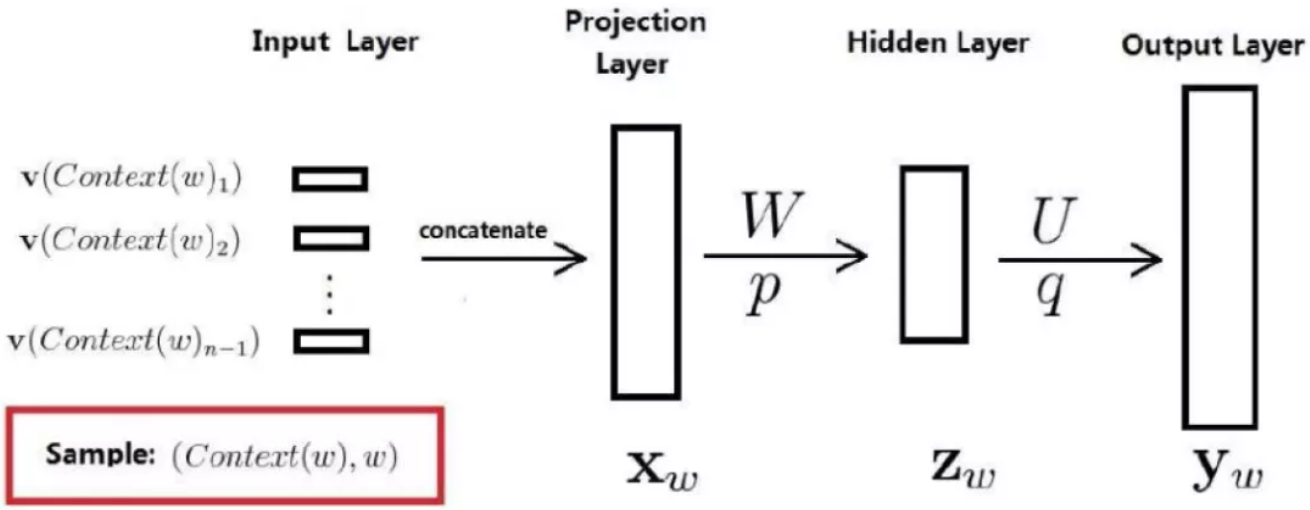
1.3 神经概率语言模型

本小节介绍 Bengio 等人于2003年在论文《A Neural Probabilistic Language Model》中提出的一种神经概率语言模型。该论文首次提出用神经网络来解决语言模型的问题，虽然在当时并没有得到太多的重视，却为后来深度学习在解决语言模型问题甚至很多别的nlp问题时奠定了坚实的基础，后人站在Yoshua Bengio的肩膀上，做出了更多的成就。包括Word2Vec的作者

Tomas Mikolov在NNLM的基础上提出了RNNLM和后来的Word2Vec。文中也较早地提出将word表示一个低秩的向量，而不是One-Hot。word embedding作为一个language model的副产品，在后面的研究中起到了关键作用，为研究者提供了更加宽广的思路。值得注意的是Word2Vec的概念也是在该论文中提出的。

什么是词向量呢？简单来说就是，对词典D中的任意词w，指定一个固定长度的实值向量$v(w) \in \Re^m$, $v(w)$ 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节来讲解。

既然是神经概率语言模型，其中当然要用到神经网络了。 下图给出了神经网络的结构示意图。模型一共三层，第一层是映射层，将n个单词映射为对应word embeddings的拼接，其实这一层就是MLP的输入层；第二层是隐藏层，激活函数用$tanh$；第三层是输出层，因为是语言模型，需要根据前n个单词预测下一个单词，所以是一个多分类器，用$Softmax$。整个模型最大的计算量集中在最后一层上，因为一般来说词汇表都很大，需要计算每个单词的条件概率，是整个模型的计算瓶颈。



经过上面步骤的计算得到的 $y_w = (y_{w,1}, y_{w,2}, \ldots, y_{w,N})^T$ 只是一个长度为N的向量，其分量不能表示概率。如果想要 y_w 的分量 $y_{w,i}$ 表示当上下文为 $Context(w)$ 时下一个词恰为词点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：

$$p(w|Context(w)) = \frac{e^{y_{w,i_w}}}{\sum_{i=1}^{N} e^{y_{w,i}}} \tag{5}$$

其中 i_w 表示词w在词典D中的索引。

这里，需要注意的是需要提前初始化一个word embedding矩阵，每一行表示一个单词的向量。词向量也是训练参数，在每次训练中进行更新。这里可以看出词向量是语言模型的一个副产物，因为语言模型本身的工作是为了估计给定的一句话有多像人类的话，但从后来的研究发现，语言模型成了一个非常好的工具。

Softmax是一个非常低效的处理方式，需要先计算每个单词的概率，并且还要计算指数，指数在计算机中都是用级数来近似的，计算复杂度很高，最后再做归一化处理。此后很多研究都针对这个问题进行了优化，比如层级softmax、softmax tree。

当然NNLM的效果在现在看来并不算什么，但对于后面的相关研究具有非常重要的意义。论文中的Future Work提到了用RNN来代替MLP作为模型可能会取得更好的效果，在后面Tomas Mikolov的博士论文中得到了验证，也就是后来的RNNLM。

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：

- **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 $S_1 =$" $Adogisrunningintheroom$ "出现了10000次，而 $S_2 =$" $Acatisrunningintheroom$ "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 的唯一区别在与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 $p(S_2)$ 应该很接近才对。事实上，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， 只要在语料库中出现一个，其他句子的概率也会相应的增大。

> 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

- **基于词向量的模型自带平滑化功能（由公式（5）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，下一节将对其作进一步阐述。

2. 词向量

自然语言处理相关任务中要将自然语言交给机器学习中的算法来处理，通常需要将语言数学化，因为机器不是人，机器只认数学符号。向量是人把自然界的东西抽象出来交给机器处理的东西，基本上可以说向量是人对机器输入的主要方式了。

词向量就是用来将语言中的词进行数学化的一种方式，顾名思义，词向量就是把一个词表示成一个向量。 我们都知道词在送到神经网络训练之前需要将其编码成数值变量，常见的编码方式有两种：One-Hot Representation 和 Distributed Representation。

2.1 One-Hot Representation

一种最简单的词向量方式是One-Hot编码 ，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。

举个例子：I like writing code，那么转换成独热编码就是:

词	**One-Hot** 编码
I	1 0 0 0
like	0 1 0 0
writing	0 0 1 0
code	0 0 0 1

这种One Hot编码如果采用稀疏方式存储，会是非常的简洁：也就是给每个 词分配一个数字 ID 。比如上面的例子中，code记为 1 ，like记为 4 。 如果要编程实现的话，用 Hash 表给每个词分配一个编号就可以了。这么简洁的表示方法配 合上最大熵、 SVM 、 CRF 等等算法已经能很好地完成 NLP 领域的各种主流任务。

但这种词表示有三个缺点：

> （1）容易受维数灾难的困扰，尤其是将其用于 Deep Learning的一些算法时；

当你的词汇量达到千万甚至上亿级别的时候,你会遇到一个更加严重的问题,维度爆炸了.这里举例使用的是4个词,你会发现,我们使用了四个维度,当词数量达到1千万的时候,词向量的大小变成了1千万维,不说别的,光内存你都受不了这么大的词向量,假设你使用一个bit来表示每一维,那么一个单词大概需要0.12GB的内存,但是注意这只是一个词,一共会有上千万的词,这样内存爆炸了。

> （ 2 ）词汇鸿沟，不能很好地刻画词与词之间的相似性；

任意两个词之间都是孤立的，从这两个向量中看不出两个词是否有关系。比如说，I、like之间的关系和like、writing之间的关系,通过0001和0010和0010和0100怎么表现,通过距离?通过1的位置?你会发现独热编码完全没法表现单词之间的任何关系。

> （3）强稀疏性；

当维度过度增长的时候,你会发现0特别多,这样造成的后果就是整个向量中有用的信息特别少,几乎就没法做计算。

由于One-hot编码存在以上种种问题，所以研究者就会寻求发展，用另外的方式表示，就是Distributed Representation。

2.2 Distributed Representation

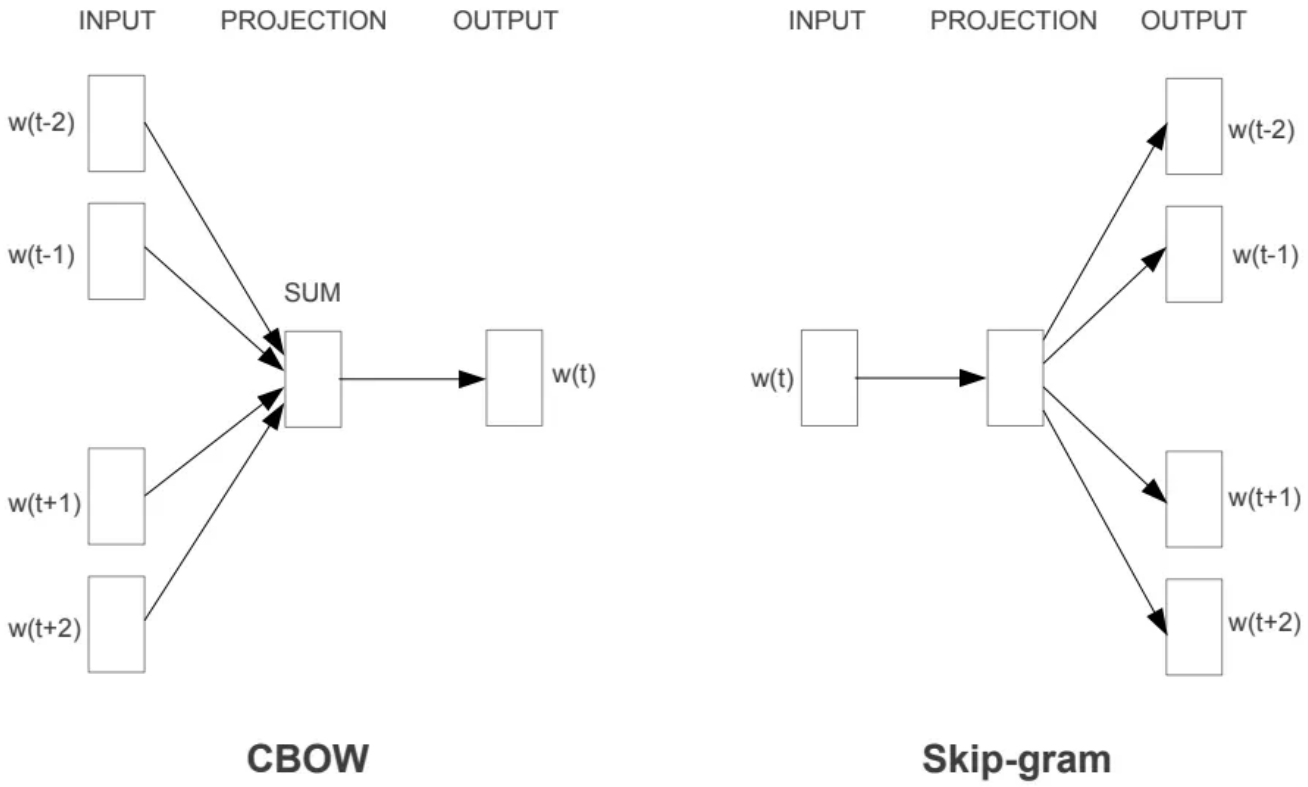
Distributed Representation最早是Hinton于1986年提出的，可以克服One-Hot Representation的上述缺点。其基本想法是：通过训练将某种语言中的每一个词 映射成一个固定长度的短向量（当然这里的“短”是相对于One-Hot Representation的“长”而言的），所有这些向量构成一个词向量空间，而每一个向量则可视为 该空间中的一个点，在这个空间上引入“距离”，就可以根据词之间的距离来判断它们之间的语法、语义上的相似性了。Word2Vec中采用的就是这种Distributed Representation 的词向量。

为什么叫做 Distributed Representation？很多人问到这个问题。一个简单的解释是这样的：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。

如何获取词向量呢？有很多不同模型可以用来估计词向量，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。在用神经网络训练语言模型方面，最经典的论文就是Bengio于2003年发表的《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。

3. Word2Vec的网络结构

Word2Vec是轻量级的神经网络，其模型仅仅包括输入层、隐藏层和输出层，模型框架根据输入输出的不同，主要包括CBOW和Skip-gram模型。 CBOW的方式是在知道词w_t的上下文$w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$的情况下预测当前词$w_t$.而Skip-gram是在知道了词$w_t$的情况下,对词$w_t$的上下文$w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$进行预测，如下图所示：

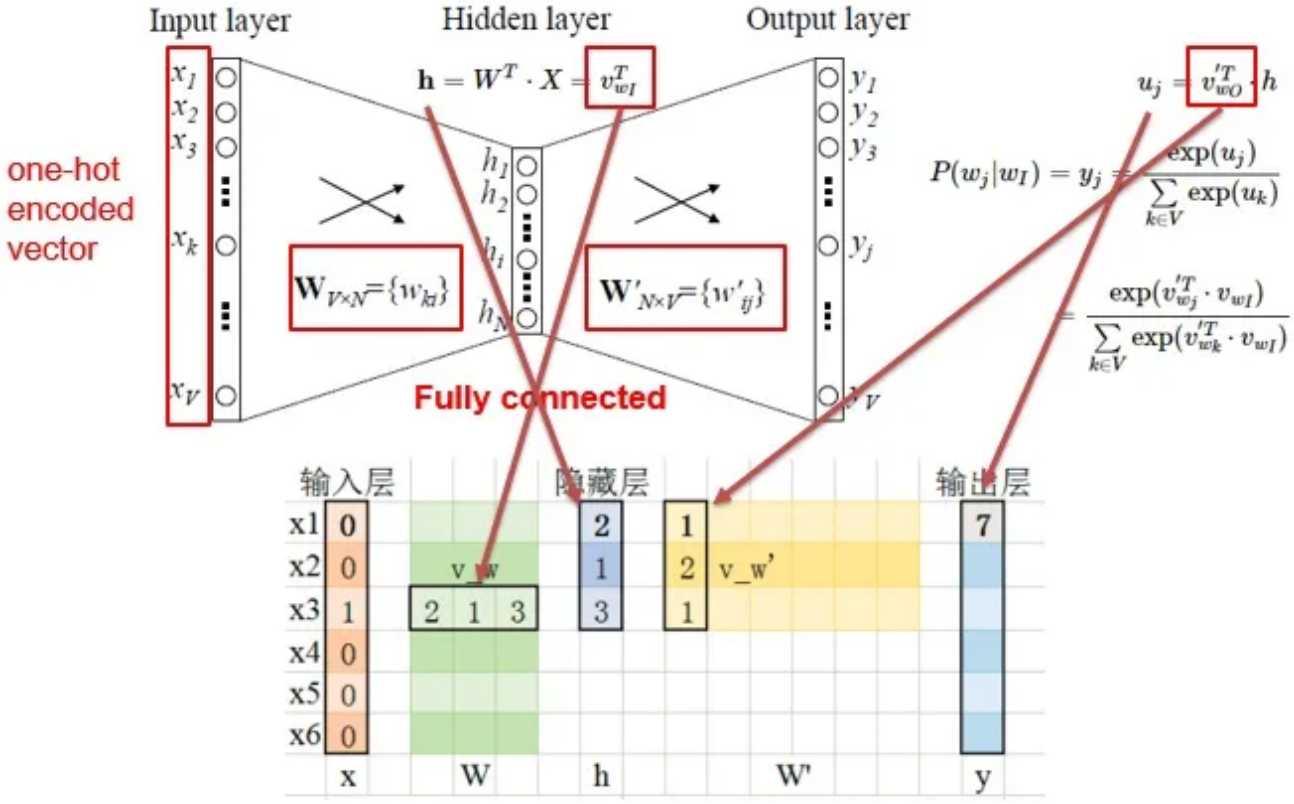


3.1 CBOW

3.1.1 Simple CBOW Model

为了更好的了解模型深处的原理，我们先从Simple CBOW model（仅输入一个词，输出一个词）框架说起。

如上图所示：



- input layer输入的X是单词的one-hot representation（考虑一个词表V，里面的每一个词 w_i 都有一个编号$i \in 1,\ldots,|V|$，那么词 w_i 的one-hot表示就是一个维度为$|V|$的向量，其中第i个元素值非零，其余元素全为0，例如： $w_2 = [0,1,0,\ldots,0]^T$ ）；
- 输入层到隐藏层之间有一个权重矩阵W，隐藏层得到的值是由输入X乘上权重矩阵得到的（细心的人会发现，$0-1$向量乘上一个矩阵，就相当于选择了权重矩阵的某一行，如图：输入的向量X是$[0，\ 0，\ 1，\ 0，\ 0，\ 0]$，W的转置乘上X就相当于从矩阵中选择第3行$[2,1,3]$作为隐藏层的值）；
- 隐藏层到输出层也有一个权重矩阵W'，因此，输出层向量y的每一个值，其实就是隐藏层的向量点乘权重向量W'的每一列，比如输出层的第一个数7，就是向量$[2,1,3]$和列向量$[1，\ 2，\ 1]$点乘之后的结果；
- 最终的输出需要经过softmax函数，将输出向量中的每一个元素归一化到$0-1$之间的概率，概率最大的，就是预测的词。

了解了Simple CBOW model的模型框架之后，我们来学习一下其目标函数。

$$\begin{aligned} L &= max\ log\, p(w|Context(w)) \\ &= max \log\ (y_j^*) \\ &= max\ log(\frac{exp(u_j^*)}{\sum exp(u_k)}) \end{aligned}$$

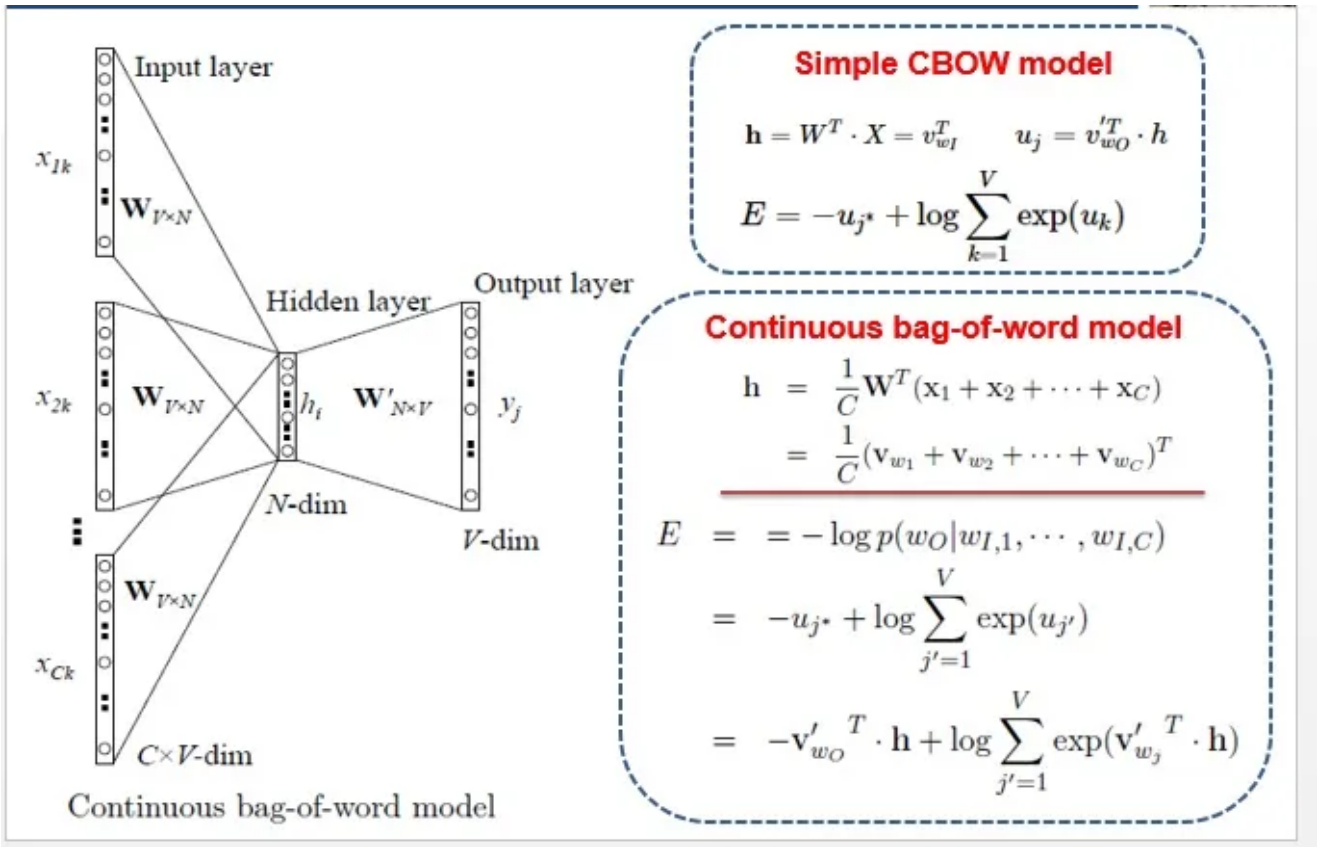
输出层通过softmax归一化，u代表的是输出层的原始结果。通过下面公式，我们的目标函数可以转化为现在这个形式：

$$a^{loga(N)} = N$$

$$max\ log(\frac{exp(u_j^*)}{\sum exp(u_k)}) = max\ u_j^* - log \sum_{k=1}^{V} exp(u_k)$$

3.1.2 CBOW Multi-Word Context Model

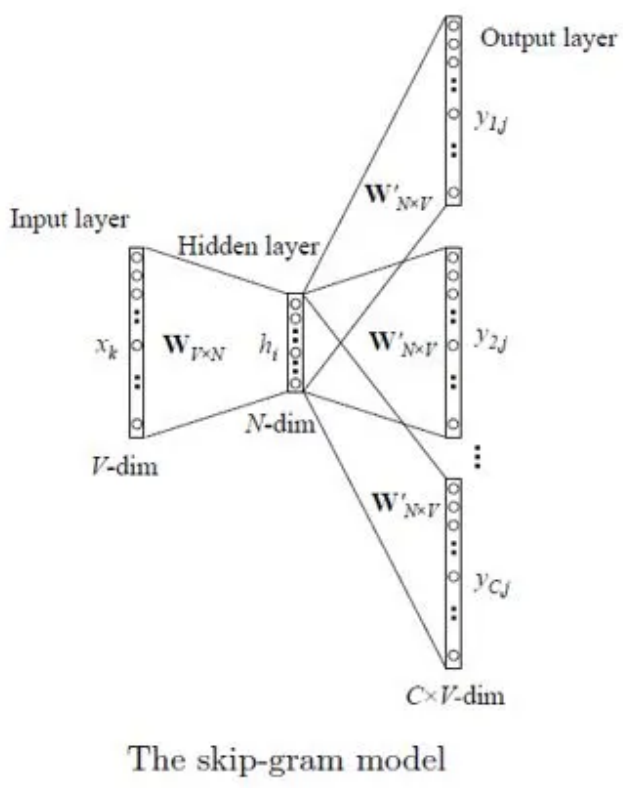
了解了Simple CBOW model之后，扩展到CBOW就很容易了，只是把单个输入换成多个输入罢了（划红线部分）。



对比可以发现，和simple CBOW不同之处在于，输入由1个词变成了C个词，每个输入 X_{ik} 到达隐藏层都会经过相同的权重矩阵W，隐藏层h的值变成了多个词乘上权重矩阵之后加和求平均值。

3.2 Skip-gram Model

有了CBOW的介绍，对于Skip-gram model 的理解应该会更快一些。



The skip-gram model

input → hidden

$$\mathbf{h} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I}$$

hidden → output

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V}\exp(u_{j'})}$$

Share the same weights：

$$u_{c,j} = u_j = \mathbf{v}'^{T}_{w_j} \cdot \mathbf{h}, \text{ for } c = 1, 2, \cdots, C$$

Loss function：

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C}|w_I) \\ &= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j^*_c})}{\sum_{j'=1}^{V}\exp(u_{j'})} \\ &= -\sum_{c=1}^{C} u_{j^*_c} + C \cdot \log \sum_{j'=1}^{V} \exp(u_{j'}) \end{aligned}$$

如上图所示，Skip-gram model是通过输入一个词去预测多个词的概率。输入层到隐藏层的原理和simple CBOW一样，不同的是隐藏层到输出层，损失函数变成了C个词损失函数的总和，权重矩阵W'还是共享的。

一般神经网络语言模型在预测的时候，输出的是预测目标词的概率，也就是说我每一次预测都要基于全部的数据集进行计算，这无疑会带来很大的时间开销。不同于其他神经网络，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。

4. 基于Hierarchical Softmax的模型

基于层次Softmax的模型主要包括输入层、投影层（隐藏层）和输出层，非常的类似神经网络结构。对于Word2Vec中基于层次Softmax的CBOW模型，我们需要最终优化的目标函数是：

$$L = \sum \log p(w|Context(w)) \tag{6}$$

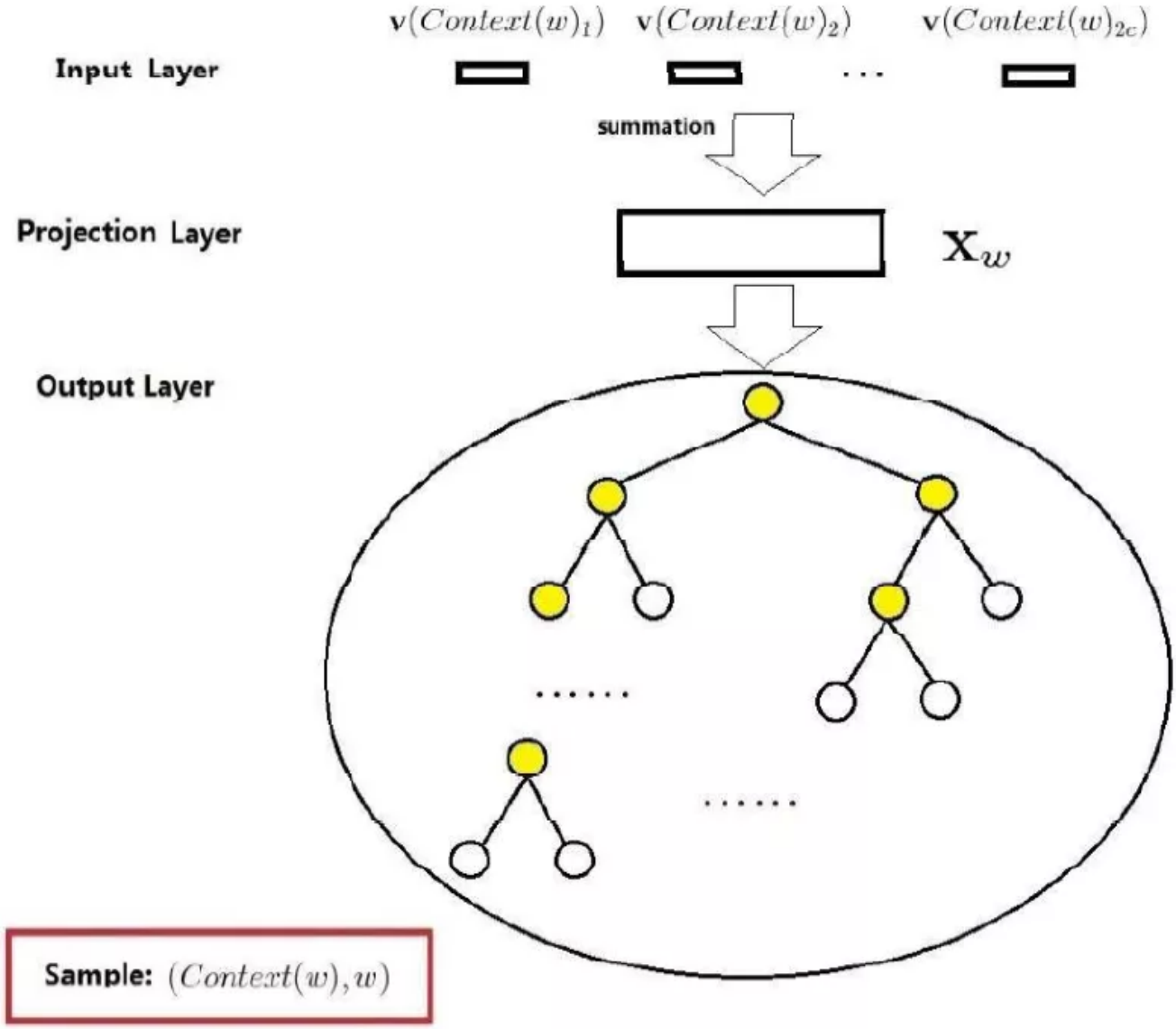
其中 $Context(w)$ 表示的是单词 w 的的上下文单词。而基于层次Softmax的Skip-gram模型的最终需要优化的目标函数是：

$$L = \sum \log p(Context(w)|w) \tag{7}$$

4.1 基于Hierarchical Softmax的CBOW

4.1.1 CBOW模型网络结构

下图给出了基于层次Softmax的CBOW的整体结构，首先它包括输入层、投影层和输出层：



- 输入层：是指 $Context(w)$ 中所包含的 $2c$ 个词的词向量 $v(Context(w)_1), v(Context(w)_2), \ldots, v(Context(w)_{2c})$ 。
- 投影层：指的是直接对 $2c$ 个词向量进行累加，累加之后得到下式：

$$X_w = \sum_{i=1}^{2c} v(Context(w)_i)$$

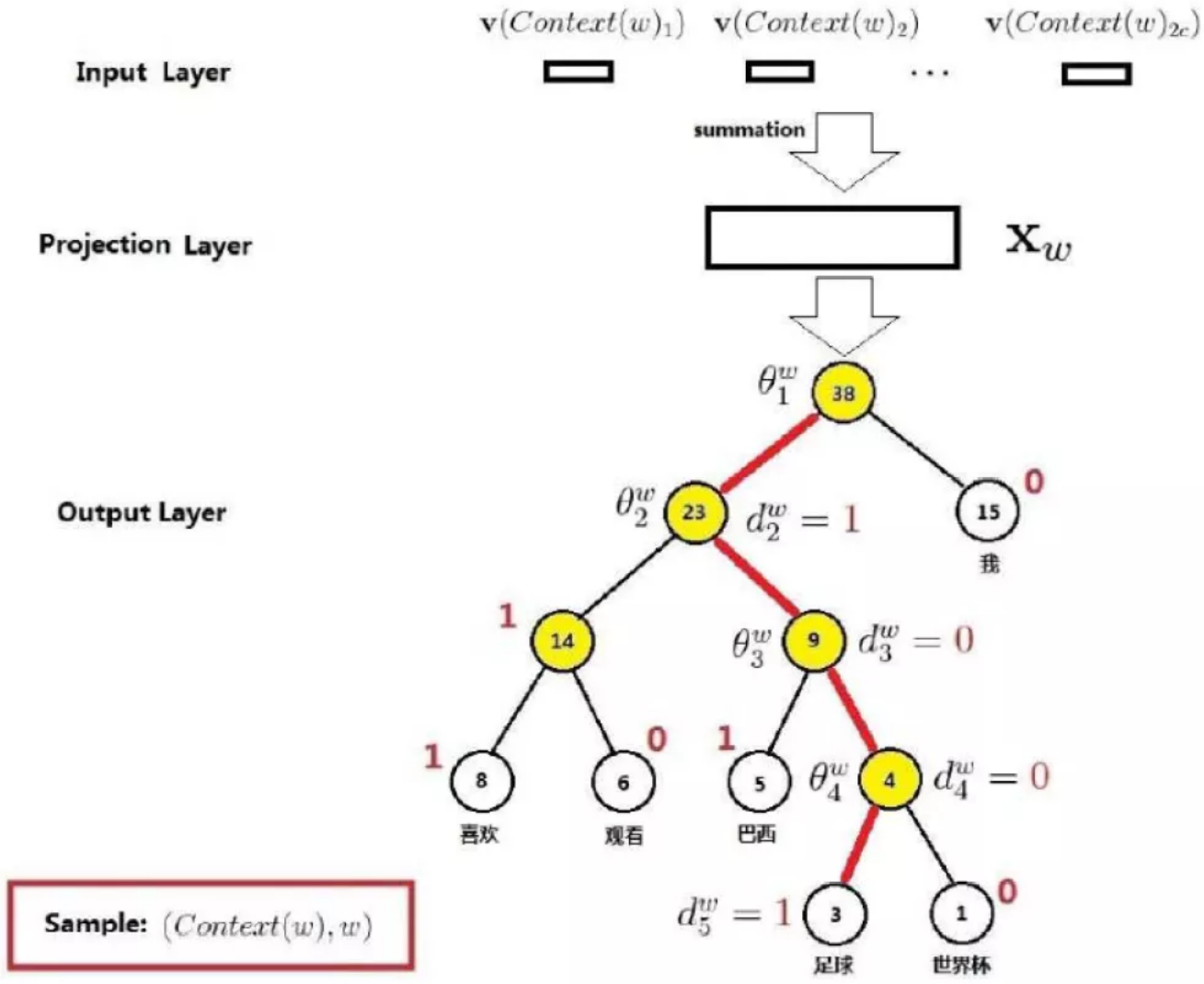
- 输出层：是一个Huffman树，其中叶子节点共N个，对应于N个单词，非叶子节点$N-1$个（对应上图中标成黄色的结点）。Word2Vec基于层次Softmax的方式主要的精华部分都集中在了哈夫曼树这部分，下面详细介绍。

4.1.2 CBOW的目标函数

为了便于下面的介绍和公式的推导，这里需要预先定义一些变量：

- p^w：从根节点出发，然后到达单词 w 对应叶子结点的路径。
- l^w：路径 p^w 中包含的结点的个数。
- $p_1^w, p_2^w, \ldots, p_{l^w}^w$：路径 p^w 中对应的各个结点，其中 p_1^w 代表根结点，而 $p_{l^w}^w$ 代表的是单词 w 对应的结点。
- $d_2^w, d_3^w \ldots, d_{l^w}^w \in \{0,1\}$：单词 w 对应的哈夫曼编码，一个词的哈夫曼编码是由 $l^w - 1$ 位构成的，d_j^w 表示路径 p^w 中的第j个单词对应的哈夫曼编码，根结点不参与对应的编码。
- $\theta_1^w, \theta_2^w, \ldots, \theta_{l^w-1}^w \in \Re^m$：路径 p^w 中非叶子节点对应的向量，θ_j^w 表示路径 p^w 中第 j 个非叶子节点对应的向量。 这里之所以给非叶子节点定义词向量，是因为这里的非叶子节点的词向量会作为下面的一个辅助变量进行计算，在下面推导公式的时候就会发现它的作用了。

既然已经引入了那么多符号，我们通过一个简单的例子把它们落到实处吧，我们考虑单词w="足球"的情形。下图中红色线路就是我们的单词走过的路径，整个路径上的5个结点就构成了路径 p^w，其长度 $l^w = 5$，然后 $p_1^w, p_2^w, p_3^w, p_4^w, p_5^w$ 就是路径 p^w 上的五个结点，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"对应的哈夫曼编码就是1001。最后 $\theta_1^w, \theta_2^w, \theta_3^w, \theta_4^w$ 就是路径 p^w 上的4个非叶子节点对应的向量。



下面我们需要开始考虑如何构建条件概率函数 $p(w|Context(w))$，以上面的w="足球"为例，从根节点到"足球"这个单词，经历了4次分支，也就是那4条红色的线，而对于这个哈夫曼树而

言，每次分支相当于一个二分类。

既然是二分类，那么我们可以定义一个为正类，一个为负类。我们的"足球"的哈夫曼编码为1001，这个哈夫曼编码是不包含根节点的，因为根节点没法分为左还是右子树。那么根据哈夫曼编码，我们一般可以把正类就认为是哈夫曼编码里面的1，而负类认为是哈夫曼编码里面的0。不过这只是一个约定而已，因为哈夫曼编码和正类负类之间并没有什么明确要求对应的关系。事实上，Word2Vec将编码为1的认定为负类，而编码为0的认定为正类，也就是说如果分到了左子树，就是负类；分到了右子树，就是正类。那么我们可以定义一个正类和负类的公式：

$$Label(p_i^w) = 1 - d_i^w, i = 2, 3, 4, \dots, l^w$$

简而言之就是，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。

在进行二分类的时候，这里选择了Sigmoid函数。那么，一个结点被分为正类的概率就是：

$$\sigma(x_w^T\theta) = \frac{1}{1+e^{-x_w^t\theta}}$$

被分为负类的概率就是 $1-\sigma(x_w^T\theta)$ 。注意，公式里面包含的有 θ ，这个就是非叶子对应的向量 θ_i^w 。

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

- 第一次分类： $p(d_2^w|x_w, \theta_1^w) = 1 - \sigma(x_w^T\theta_1^w)$
- 第二次分类： $p(d_3^w|x_w, \theta_2^w) = \sigma(x_w^T\theta_2^w)$
- 第三次分类： $p(d_4^w|x_w, \theta_3^w) = \sigma(x_w^T\theta_3^w)$
- 第四次分类： $p(d_5^w|x_w, \theta_4^w) = 1 - \sigma(x_w^T\theta_4^w)$

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

$$p(\text{足球}\ |Context(\text{足球})) = \prod_{j=2}^{5} p(d_j^w|x_w, \theta_{j-1}^w)$$

至此，通过w="足球"的小例子，Hierarchical Softmax的基本思想就已经介绍完了。小结一下：对于词典中的任意一个单词 w ，哈夫曼树中肯定存在一条从根节点到单词 w 对应叶子结点的路径 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，就是所需要的 $p(w|Context(w))$ 。

条件概率 $p(w|Context(w))$ 的一般公式可写为：

$$p(w|Context(w)) = \prod_{j=2}^{l^w} p(d_j^w|x_w, \theta_{j-1}^w) \tag{8}$$

其中：

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

或者写成整体表达式：

$$p(d_j^w|x_w,\theta_{j-1}^w)=[\sigma(x_w^T\theta_{j-1}^w)]^{1-d_j^w}\cdot[1-\sigma(x_w^T\theta_{j-1}^w)]^{d_j^w}$$

将公式(8)带入公式(6)中，得到:

$$\begin{aligned}L&=\sum_{w\in C}\log\prod_{j=2}^{l^w}[\sigma(x_w^T\theta_{j-1}^w)]^{1-d_j^w}\cdot[1-\sigma(x_w^T\theta_{j-1}^w)]^{d_j^w}\\&=\sum_{w\in C}\sum_{j=2}^{l^w}(1-d_j^w)\cdot\log[\sigma(x_w^T\theta_{j-1}^w)]+d_j^w\cdot\log[1-\sigma(x_w^T\theta_{j-1}^w)]\end{aligned}\tag{9}$$

为了梯度推导方便起见，将上式中双重求和符号下的内容简记为 $L(w,j)$ ，即:

$$L(w,j)=(1-d_j^w)\cdot\log[\sigma(x_w^T\theta_{j-1}^w)]+d_j^w\cdot\log[1-\sigma(x_w^T\theta_{j-1}^w)]$$

至此，已经推导出了CBOW模型的目标函数公式(9)，接下来就是讨论如何优化它，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，进行反向传播。

4.1.3 参数更新

首先考虑$L(w,j)$关于 θ_{j-1}^w 的梯度计算：

$$\begin{aligned}\frac{\Delta L(w,j)}{\Delta\theta_{j-1}^w}&=\left\{(1-d_j^w)[1-\sigma(x_w^T\theta_{j-1}^w)]x_w-d_j^w\sigma(x_w^T\theta_{j-1}^w)\right\}x_w\\&=[1-d_j^w-\sigma(x_w^T\theta_{j-1}^w)]x_w\end{aligned}$$

于是， θ_{j-1}^w的更新公式可写为：

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

接下来考虑$L(w,j)$关于 x_w 的梯度：

$$\frac{\Delta L(w,j)}{\Delta x_w}=[1-d_j^w-\sigma(x_w^T\theta_{j-1}^w)]\theta_{j-1}^w$$

到了这里，我们已经求出来了 x_w 的梯度，但是我们想要的其实是每次进行运算的每个单词的梯度，而 x_w 是 $Context(w)$ 中所有单词累加的结果，那么我们怎么使用 x_w 来对$Context(w)$中的每个单词 $v(\widetilde{w})$ 进行更新呢？这里原作者选择了一个简单粗暴的方式，直接使用 x_w 的梯度累加对$v(\widetilde{w})$进行更新：

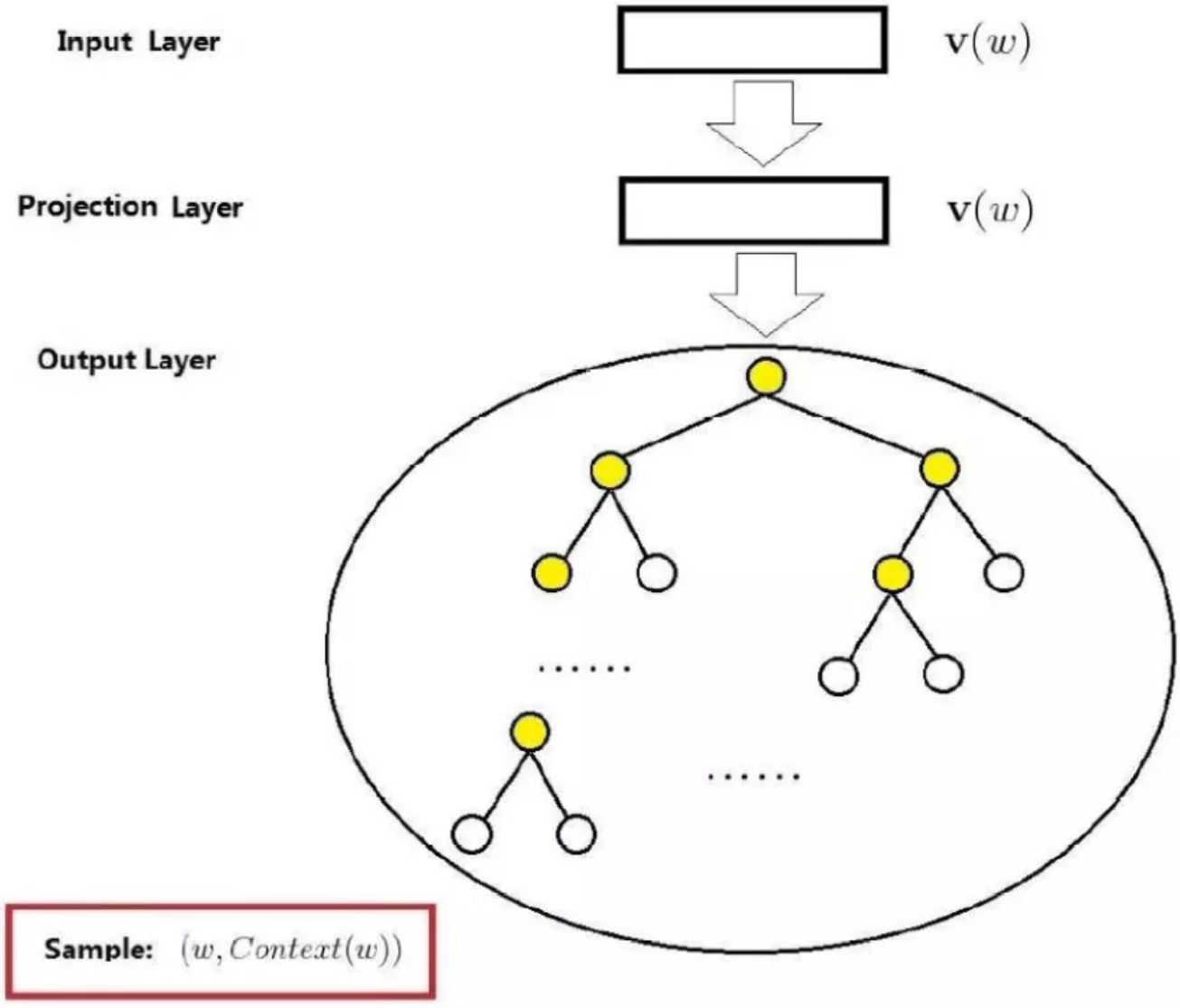
$$v(\widetilde{w})=v(\widetilde{w})+\eta\sum_{j=2}^{l^w}\frac{\Delta L(w,j)}{\Delta x_w},\ \ \widetilde{w}\in Context(w)$$

4.2 基于Hierarchical Softmax的Skip-gram

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 Skip-gram模型网络结构

下图给出了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，对这三层做简要说明。



- 输入层：只含当前样本的中心词 w 的词向量 $v(w) \in \Re^m$ 。
- 投影层：这是个恒等投影，把 $v(w)$ 投影到 $v(w)$ 。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。
- 输出层：和CBOW模型一样，输出层也是一颗Huffman树。

4.2.2 Skip-gram的目标函数

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公式（7），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 $p(u|w)$可按照上小节介绍的Hierarchical Softmax思想，类似于公式（8），可写为：

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

其中：

$$p(d_j^u|v(w), \theta_{j-1}^u) = [\sigma(v(w)^T\theta_{j-1}^u)]^{1-d_j^w} \cdot [1-\sigma(v(w)^T\theta_{j-1}^u)]^{d_j^u} \tag{10}$$

将公式（10）依次代回，可得对数似然函数公式（7）的具体表达式：

$$\begin{aligned} L &= \sum_{w \in C} \log \prod_{u \in Context(w)} \prod_{j=2}^{l^u} [\sigma(v(w)^T\theta_{j-1}^u)]^{1-d_j^w} \cdot [1-\sigma(v(w)^T\theta_{j-1}^u)]^{d_j^u} \\ &= \sum_{w \in C} \sum_{u \in Context(w)} \sum_{j=2}^{l^u} (1-d_j^u) \cdot \log[\sigma(v(w)^T\theta_{j-1}^u)] + d_j^u \log[1-\sigma(v(w)^T\theta_{j-1}^u)] \end{aligned} \tag{11}$$

同样，为了梯度推导方便，将三重求和符号里的内容简记为$L(w,u,j)$ ，即：

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

至此，已经推导出了Skip-gram模型的目标函数（公式11），接下来同样利用随机梯度上升法对其进行优化。而梯度类算法的关键是给出相应的梯度计算公式，进行反向传播。

4.2.3 参数更新

首先考虑$L(w,u,j)$关于 θ_{j-1}^u 的梯度计算：

$$\begin{aligned} \frac{\Delta L(w,u,j)}{\Delta \theta_{j-1}^u} &= \left\{(1-d_j^u)(1-\sigma(v(w)^T\theta_{j-1}^u))v(w) - d_j^u\sigma(v(w)^T\theta_{j-1}^u)x\right\}v(w) \\ &= [1-d_j^u-\sigma(v(w)^T\theta_{j-1}^u]v(w) \end{aligned}$$

于是， θ_{j-1}^u的更新公式可写为：

$$\theta_{j-1}^u = \theta_{j-1}^u + \eta[1-d_j^u-\sigma(v(w)^T\theta_{j-1}^u]v(w)$$

同理,根据对称性,可以很容易得到 $L(w,u,j)$ 关于 $v(w)$ 的梯度：

$$\frac{\Delta L(w,u,j)}{\Delta v(w)} = [1-d_j^u-\sigma(v(w)^T\theta_{j-1}^u]\theta_{j-1}^u$$

我们也可以得到关于$v(w)$的更新公式:

$$v(w) = v(w) + \eta \sum_{u \in Context(w)} \sum_{j=2}^{l^w} \frac{\Delta L(w,u,j)}{\Delta v(w)}$$

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等人在论文《Distributed Representations of Words and Phrases and their Compositionality》中提出的，它是NCE（Noise Contrastive Estimation）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。

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。具体可以参考论文《 Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics》。

5.1 负采样算法简单介绍

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？

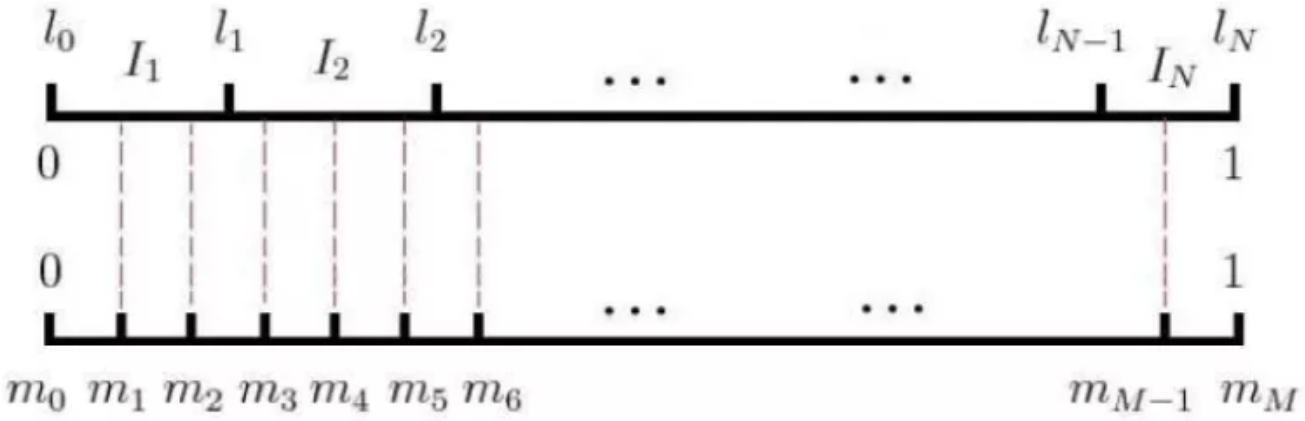
词典D中的词在语料C中出现的次数有高有底，对于那些高频词，被选为负样本的概率就应该比较大，反之，对于那些低频词，其被选中的概率就应该比较小。这就是我们对采样过程的一个大致要求，本质上就是一个带权采样问题。

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D中的每一个词w对应一个线段 $len(w)$ ，长度为：

$$len(w) = \frac{counter(w)}{\sum_{u \in C} counter(u)} \tag{12}$$

这里 $counter(.)$ 表示一个词在语料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 $l_0 = 0, \ldots, l_k = \sum_{j=1}^{k} len(w_j), \quad k = 1, 2, \ldots, N$ ，这里 w_j 表示词典中的第j个单词，则以 $\{l_i\}_{j=0}^{N}$ 为剖分结点可得到区间 $[0, 1]$ 上的一个非等距剖分， $I_i = \{l_{i-1}, l_i\}, i = 1, 2, \ldots, N$ 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，剖分结点为 $\{m_j\}_{j=0}^{M}$ ，其中 $M >> N$ ，具体见下面给出的示意图。



图：$Table(.)$映射的建立示意图

将内部剖分结点 $\{m_j\}_{j=1}^{M-1}$ 投影到非等距剖分上，如上图中的红色虚线所示，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。

$$Table(i) = w_k,\ \ where\ \ m_i \in I_k,\ \ i = 1, 2, \ldots, M-1 \tag{13}$$

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 自己该怎么办？那就跳过去，Word2Vec的代码中也是这么处理的。

值得一提的是，Word2Vec源码中为词典D中的词设置权值时，不是直接用$counter(w)$，而是对其做了 α 次幂，其中 $\alpha = 0.75$ ，即公式（12）变成了：

$$\begin{aligned} len(w) &= \frac{counter(w)^{\alpha}}{\sum_{u \in C}[counter(u)]^{\alpha}} \\ &= \frac{counter(w)^{0.75}}{\sum_{u \in C}[counter(u)]^{0.75}} \end{aligned}$$

此外，代码中取 $M = 10^8$ ，源代码中是变量$table_size$。而映射公式（13）则是通过一个名为InitUnigramTable的函数来完成。

5.2 基于Negative Sampling的CBOW

上面已经介绍完了负采样算法，下面开始推导出基于Negative Sampling的CBOW的目标函数。首先我们先选好一个关于 $Context(w)$ 的负样本子集 $NEG(w) \neq \oslash$ ，对于 $\forall \tilde{w} \in D$ ，我们定义单词 $\tilde{w}$ 的标签为:

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

上式表示词 $\tilde{w}$ 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 $(Context(w), w)$，我们希望最大化的目标函数是：

$$g(w) = \prod_{u \in w \cup NEG(w)} p(u|Context(w)) \tag{14}$$

其中:

$$p(u|Context(w)) = \begin{cases} \sigma(x_w^T\theta^u), & L^w(u)=1 \\ 1-\sigma(x_w^T\theta^u), & L^w(u)=0 \end{cases} \\ = [\sigma(x_w^T\theta^u)]^{L^w(u)} \cdot [1-\sigma(x_w^T)\theta^u]^{1-L^w(u)} \tag{15}$$

这里 x_w 仍表示 $Context(w)$ 中各个词的词向量之和，而 $\theta^u \in R^m$ 在这里作为一个辅助向量，为待训练的参数。

为什么要最大化 $g(w)$ 呢？让我们先来看看$g(w)$的表达式，将公式（15）带入公式（14），有：

$$g(w) = \sigma(x_w^T\theta^w) \prod_{u \in NEG(w)} [1-\sigma(x_w^T\theta^u)]$$

其中， $\sigma(x_w^T\theta^w)$ 表示当上下文为 $Context(w)$ 时，预测中心词为w的概率，而 $\sigma(x_w^T\theta^u),\ u \in NEG(w)$ 则表示当上下文为$Context(w)$ 时，预测中心词为u的概率，这里可以看一个二分类问题。从形式上看，最大化 $g(w)$ ，相当于最大化 $\sigma(x_w^T\theta^w)$ ，同时最小化所有的 $\sigma(x_w^T\theta^u),\ u \in NEG(w)$ 。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C，有函数：

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

可以作为最终的整体优化目标。当然，这里为了求导方便，对G取对数，最终的目标函数就是：

$$\begin{aligned} L &= \log G = \sum_{w \in C} \log g(w) \\ &= \sum_{w \in C} \sum_{u \in w \cup NEG(w)} \log\, [\sigma(x_w^T\theta^u)]^{L^w(u)} \cdot [1-\sigma(x_w^T)\theta^u]^{1-L^w(u)} \\ &= \sum_{w \in C} \sum_{u \in w \cup NEG(w)} L^w(u) \cdot \log[\sigma(x_w^T\theta^u) + [1-L^w(u)] \cdot \log[1-\sigma(x_w^T\theta^u)]] \end{aligned}$$

同样，为了求导方便,我们还是取 $L(w,u)$ ：

$$L(w,u) = L^w(u) \cdot \log[\sigma(x_w^T\theta^u) + [1-L^w(u)] \cdot \log[1-\sigma(x_w^T\theta^u)]]$$

接下来，利用随机梯度上升法求梯度。首先考虑$L(w,u)$关于 θ^u 的梯度计算：

$$\begin{aligned} \frac{\Delta L(w,u)}{\Delta \theta^u} &= \left\{L^w(u)[1-\sigma(x_w^T\theta^u)]x_w - [1-L^w(u)] \cdot \sigma(x_w^T\theta^u)\right\}x_w \\ &= [L^w(u) - \sigma(x_w^T\theta^u)]x_w \end{aligned}$$

那么 θ^u 的更新公式可以写成：

$$\theta^u = \theta^u + \eta[L^w(u) - \sigma(x_w^T\theta^u)]x_w$$

同时根据对称性，可以得到 x_w 的梯度：

$$\frac{\Delta L(w,u)}{\Delta x_w} = [L^w(u) - \sigma(x_w^T\theta^u)]\theta^u$$

那么 $v(w)$ 的更新公式可以写成：

$$v(\tilde{w}) = v(\tilde{w}) + \eta \sum_{u \in w \cup NEG(w)} \frac{\Delta L(w,u)}{\Delta x_w}, \ \tilde{w} \in Context(w)$$

5.3 基于Ngative Sampling的Skip-gram

本小节介绍基于Negative Sampling的Skip-gram模型。它和上一小节介绍的CBOW模型所用的思想是一样的，因此，这里我们直接从目标函数出发，且沿用之前的记号。

对于一个给定的样本 $(w, Context(w))$ ， 我们希望最大化：

$$g(w) = \prod_{\tilde{w} \in Context(w)} \prod_{u \in w \cup NEU^{\tilde{w}}(w)} p(Context|\tilde{w})$$

其中：

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

或者写成整体表达式：

$$p(u|\tilde{w}) = [\sigma(v(\tilde{w})^T]^{L^w(u)} \cdot [1 - \sigma(v(\tilde{w})^T]^{1-L^w(u)}$$

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

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

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：

$$\begin{aligned} L &= \log G = \sum_{w \in C} \log g(w) \\ &= \sum_{w \in C} \sum_{\tilde{w} \in Context(w)} \sum_{u \in w \cup NEU^{\tilde{w}}(w)} L^w(u) \log[\sigma(v(\tilde{w})^T \theta^u)] + [1 - L^w(u)] \log[1 - \sigma(v(\tilde{w})^T \theta^u)] \end{aligned}$$

为了梯度推导的方便，我们依旧将三重求和符号下的内容提取出来，记为 $L(w, \tilde{w}, u)$ ，即：

$$L(w, \tilde{w}, u) = L^w(u) \log[\sigma(v(\tilde{w})^T \theta^u)] + [1 - L^w(u)] \log[1 - \sigma(v(\tilde{w})^T \theta^u)]$$

接下来，利用随机梯度上升法求梯度。首先考虑 $L(w, \tilde{w}, u)$ 关于 θ^u 的梯度计算：

$$\begin{aligned} \frac{\Delta L(w, \tilde{w}, u)}{\Delta \theta^u} &= \left\{ L^w(u)[1 - \sigma(v(\tilde{w})_w^T \theta^u)] v(\tilde{w}) - [1 - L^w(u)] \cdot \sigma(v(\tilde{w})_w \theta^u) \right\} v(\tilde{w}) \\ &= [L^w(u) - \sigma(v(\tilde{w})^T \theta^u)] v(\tilde{w}) \end{aligned}$$

然后得到 θ^u 的更新公式：

$$\theta^u = \theta^u + \eta = [L^w(u) - \sigma(v(\tilde{w})^T \theta^u)] v(\tilde{w})$$

同理根据对称性,得到：

$$\frac{\Delta L(w, \tilde{w}, u)}{\Delta v(\tilde{w})} = [L^w(u) - \sigma(v(\tilde{w})^T \theta^u)] \theta^u$$

然后得到 $v(\tilde{w})$ 的更新公式：

$$v(\tilde{w}) = v(\tilde{w}) + \eta \sum_{u \in w \cup NEU^{\tilde{w}}(w)} \frac{\Delta L(w, \tilde{w}, u)}{\Delta v(\tilde{w})}, \quad \tilde{w} \in Context(w)$$

6. 关于Word2Vec若干问题的思考

（1）Word2Vec两个算法模型的原理是什么，网络结构怎么画？

（2）网络输入输出是什么？隐藏层的激活函数是什么？输出层的激活函数是什么？

（3）目标函数/损失函数是什么？

（4）Word2Vec如何获取词向量？

（5）推导一下Word2Vec参数如何更新？

（6）Word2Vec的两个模型哪个效果好哪个速度快？为什么？

（7）Word2Vec加速训练的方法有哪些？

（8）介绍下Negative Sampling，对词频低的和词频高的单词有什么影响？为什么？

（9）Word2Vec和隐狄利克雷模型(LDA)有什么区别与联系？

以上问题可以通过本文和参考文章找到答案，这里不再详细解答。

（10）介绍下Hierarchical Softmax的计算过程，怎么把 Huffman 放到网络中的？参数是如何更新的？对词频低的和词频高的单词有什么影响？为什么？

Hierarchical Softmax利用了Huffman树依据词频建树，词频大的节点离根节点较近，词频低的节点离根节点较远，距离远参数数量就多，在训练的过程中，低频词的路径上的参数能够得到更多的训练，所以效果会更好。

（11）Word2Vec有哪些参数，有没有什么调参的建议？

- Skip-Gram 的速度比CBOW慢一点，小数据集中对低频次的效果更好；
- Sub-Sampling Frequent Words可以同时提高算法的速度和精度，Sample 建议取值为 $[10^{-5}, 10^{-3}]$；
- Hierarchical Softmax对低词频的更友好；
- Negative Sampling对高词频更友好；
- 向量维度一般越高越好，但也不绝对；
- Window Size，Skip-Gram一般10左右，CBOW一般为5左右。

（12）Word2Vec有哪些局限性？

Word2Vec作为一个简单易用的算法，其也包含了很多局限性：

- Word2Vec只考虑到上下文信息，而忽略的全局信息；
- Word2Vec只考虑了上下文的共现性，而忽略的了彼此之间的顺序性；

7. Word2Vec在工业界的应用

Word2Vec主要原理是根据上下文来预测单词，一个词的意义往往可以从其前后的句子中抽取出来。

而用户的行为也是一种相似的时间序列，可以通过上下文进行推断。当用户浏览并与内容进行交互时，我们可以从用户前后的交互过程中判断行为的抽象特征，这就使得我们可以把词向量模型应用到推荐、广告领域当中。

7.1 NLP领域

- Word2Vec学习到的词向量代表了词的语义，可以用来做分类、聚类、也可以做词的相似度计算。
- 把Word2Vec生成的向量直接作为深度神经网络的输入，可以做sentiment analysis等工作。

7.2 图嵌入

基于Word2Vec这一类的Graph Embedding方法有很多，具体可以参考论文：DeepWalk（是引入Word2Vec思想比较经典的图嵌入算法），node2vec，struc2vec 等等。

7.3 推荐领域

- Airbnb在论文《Real-time Personalization using Embeddings for Search Ranking at Airbnb》中提出将用户的浏览行为组成List，通过Word2Vec方法学习item的向量，其点击率提升了21%，且带动了99%的预定转化率。该论文主要是在Skip-gram 模型的基础上做了改进。

- Yahoo在论文《E-commerce in Your Inbox: Product Recommendations at Scale》中提出Yahoo邮箱从发送到用户的购物凭证中抽取商品并组成List，通过Word2Vec学习并为用户推荐潜在的商品；

7.4 广告领域

Yahoo在论文《Scalable Semantic Matching of Queries to Ads in Sponsored Search Advertising》中提出将用户的搜索查询和广告组成List，并为其学习特征向量，以便对于给定的搜索查询可以匹配适合的广告。

8. Reference

【1】Rong X. word2vec parameter learning explained[J]. arXiv preprint arXiv:1411.2738, 2014.
【2】【Paper】Word2Vec：词嵌入的一枚银弹，地址：https://mp.weixin.qq.com/s/7dsjfcOfm9uPheJrmB0Ghw
【3】Word2Vec详解-公式推导以及代码 - link-web的文章 - 知乎 https://zhuanlan.zhihu.com/p/86445394
【4】The Illustrated Word2vec，Jay Alammar，地址：https://jalammar.github.io/illustrated-word2vec/
【5】图解word2vec（原文翻译），地址：https://mp.weixin.qq.com/s/Yq_-1eS9UuiUBhNNAIxC-Q
【6】word2vec 相比之前的 Word Embedding 方法好在什么地方？ - 知乎 https://www.zhihu.com/question/53011711
【7】[NLP] 秒懂词向量Word2vec的本质 - 穆文的文章 - 知乎 https://zhuanlan.zhihu.com/p/26306795
【8】http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
【9】word2vec模型深度解析 - TianMin的文章 - 知乎 https://zhuanlan.zhihu.com/p/85998950
【10】 A Neural Probabilistic Language Model - 张俊的文章 - 知乎 https://zhuanlan.zhihu.com/p/21240807
【11】word2vec有什么应用？ - 知乎 https://www.zhihu.com/question/25269336

最后除引用文献外也推荐一些看过的资料：

【1】《深度学习word2vec学习笔记》，北流浪子。
【2】《Word2Vec中的数学》，peghoty。
【3】《Deep Learning实战之Word2Vec》，网易有道。

本文主要参考了peghoty的《Word2Vec中的数学》，写的非常棒， 强烈推荐大家学习。此外，我把自己学习Word2Vec时，收集到的优质资料已经整理好了，公众号后台回复【Word2Vec】领取。