## 中文分词

中文分词的这个应用，有多种处理方法，包括基于词典的方法、隐马尔可夫模型（HMM）、最大熵模型、条件随机场（CRF）、深度学习模型（双向 LSTM 等）和一些无监督学习的方法（基于凝聚度与自由度）。

# word2vec

google2013年开源的用于获取word vector的工具包。作者 Tomas Mikolov

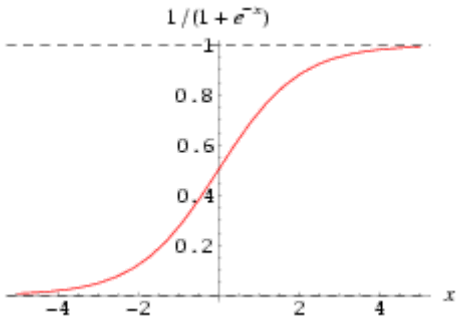
## 1. 相关知识

### 1.1 sigmod函数

**sigmod函数**是神经网络中常用的激活函数之一，其定义为：

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

函数定义域为$(-\infty，\ +\infty)$, 值域为(0,1)




由此，可得sigmod函数的导函数为：

$$\sigma`(x) = \sigma(x)[1-\sigma(x)]$$

而函数$log\sigma(x)$和函数$log(1-\sigma(x))$的导函数分别为：

$$[log\sigma(x)]`= 1-\sigma(x)$$
$$[log(1-\sigma(x)]`= -\sigma(x)$$

### 1.2 逻辑回归

**逻辑回归（Logistic Regression,LR）**通常用于机器学习二分类任务中，如邮件是否为垃圾邮件，客户是否贷款逾期等等。
假设数据样本i用$(x_i, y_i)$表示，其中$x_i$为样本特征向量$(x_1, x_2, \ldots, x_n)$,$y_i$为样本分类别标签（0,1）。
逻辑回归函数即是在样本线性函数的基础上做了sigmod变换：

$$y(x) = w_0 + \sum_i w_i x_i$$

变换后：

$$f(x) = \frac{1}{1+e^{-y(x)}}$$

其中，$W = (w_1, w_2, \ldots, w_n)^T$为待定参数。

### 1.3 Bayes公式

**贝叶斯公式**描述了两个条件概率之间的关系：

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

由此可得：

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

## 1.4 Huffman编码

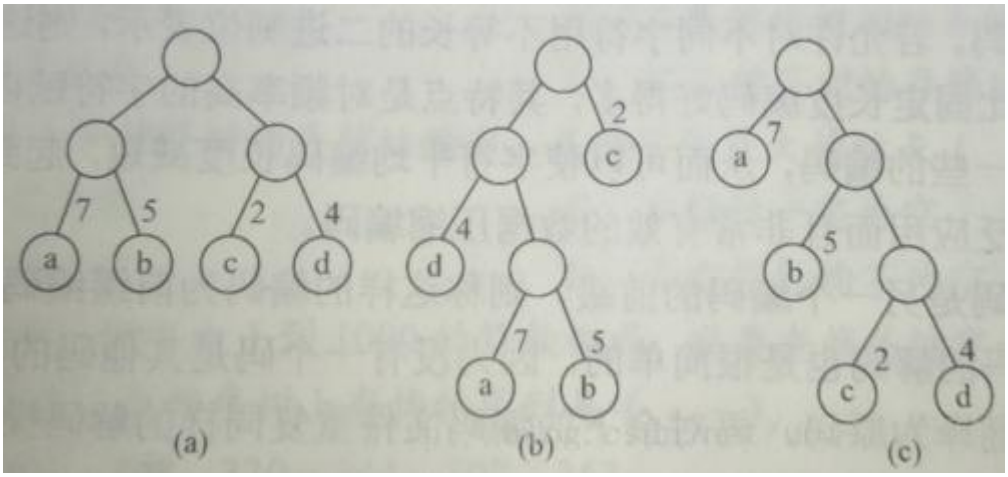
**树**
在计算机中树是非常重要的非线性数据结构，它是数据元素（在树中称为节点）按分支关系组织起来的结构。
树的常用概念：

- **路径和路径长度**
  在树中，一个节点往下可以到达的子节点之间的通路，称为路径，通路中分支的数目称为路径长度。
  若根节点层号是1，则从根节点到第L层子节点的路径长度为L-1。
- **节点的权和带权路径长度**
  若为树中节点赋予一个具有某种含义的数值，则这个数值可称为改节点的权。
  节点的带权路径长度是指。从根节点到该节点之间路径长度与该节点的权的乘积。
- **树的带权路径长度**
  树的带权路径长度是所有叶子节点的带权路径长度之和。

**二叉树**
二叉树是子节点最多只有两个的有序树，子节点被称为左子树和右子树，有序是指节点的两个子树有左右之分，顺序不能颠倒。

**Huffman树**
给定n个节点，且带有n个权值，构造一颗二叉树，若该树的带权路径长度达到最小，则称这样的数为**最优二叉树，即Huffman树**

**例如**：给定4个叶子节点a, b, c 和 d, 分别带权 7, 5, 2 和 4，如下图所示：



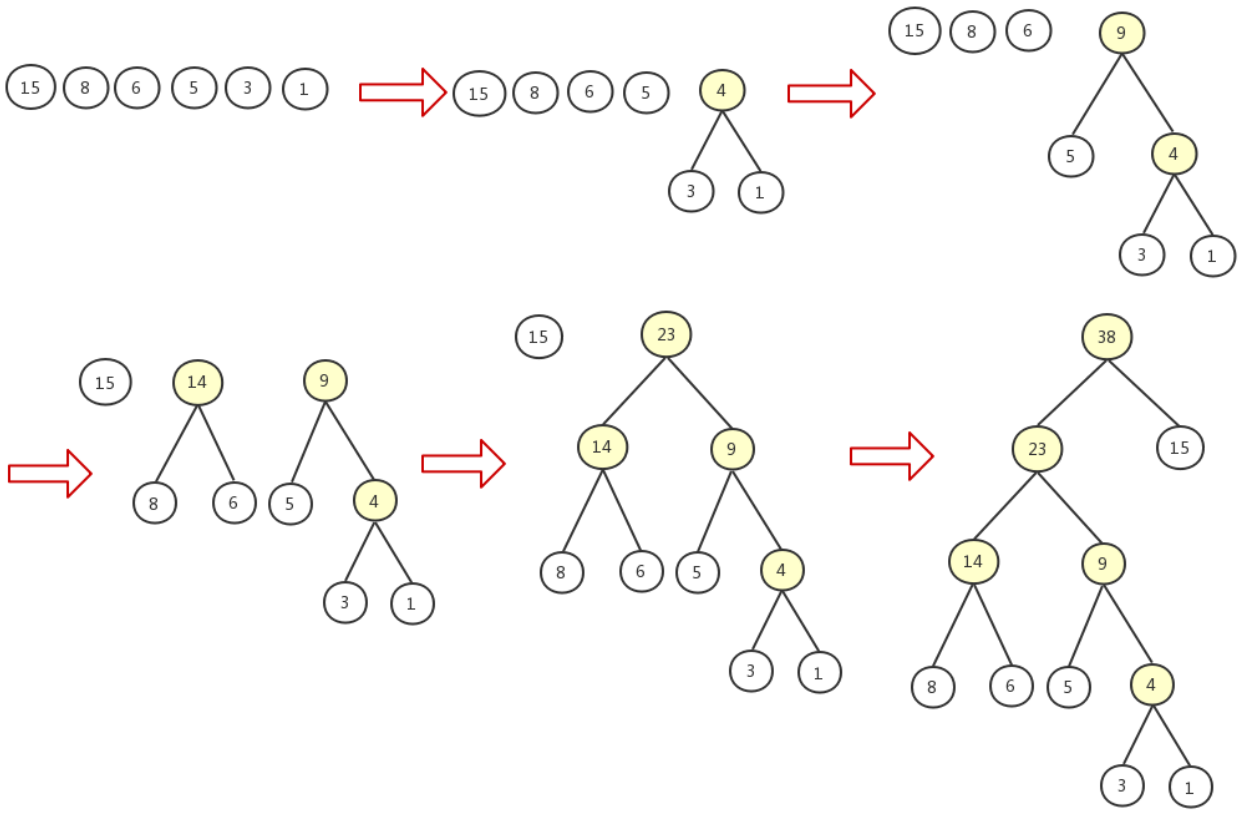

根据树的带权路径长度的定义，从左到右，三棵树的带权路径长度分别为:
(a) 7∗2+5∗2+2∗2+4∗2=36
(b) 7∗3+5∗3+4∗2+2∗1=46
(c) 7∗1+5∗2+4∗3+2∗3=35
可以看到当权重越小的节点离根节点越远，而权重越大的节点离根节点越近时，得到树的带权路径长度会越小。

**Huffman树的构造算法**
n个节点的权值为$\{w_1, w_2, \ldots, w_n\}$可以通过如下算法构造Huffman树：

- 将给定的n个权值看做n棵只有根节点（无左右孩子）的二叉树，组成一个森林，每棵树的权值为该节点的权值。

- 从森林中选出2棵权值最小的二叉树，组成一棵新的二叉树，其权值为这2棵二叉树的权值之和。
- 将步骤2中选出的2棵二叉树删去，同时将新得到的二叉树加入到森林中。
- 重复步骤2和步骤3，直到森林中只含一棵树，这棵树便是赫夫曼树。

**例如**: 假设在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个词为叶子节点，以相应词频当权值，构造一棵Huffman树：




上例中，合并后新增节点用黄色表示，若节点个数是n则新增节点个数为n-1。

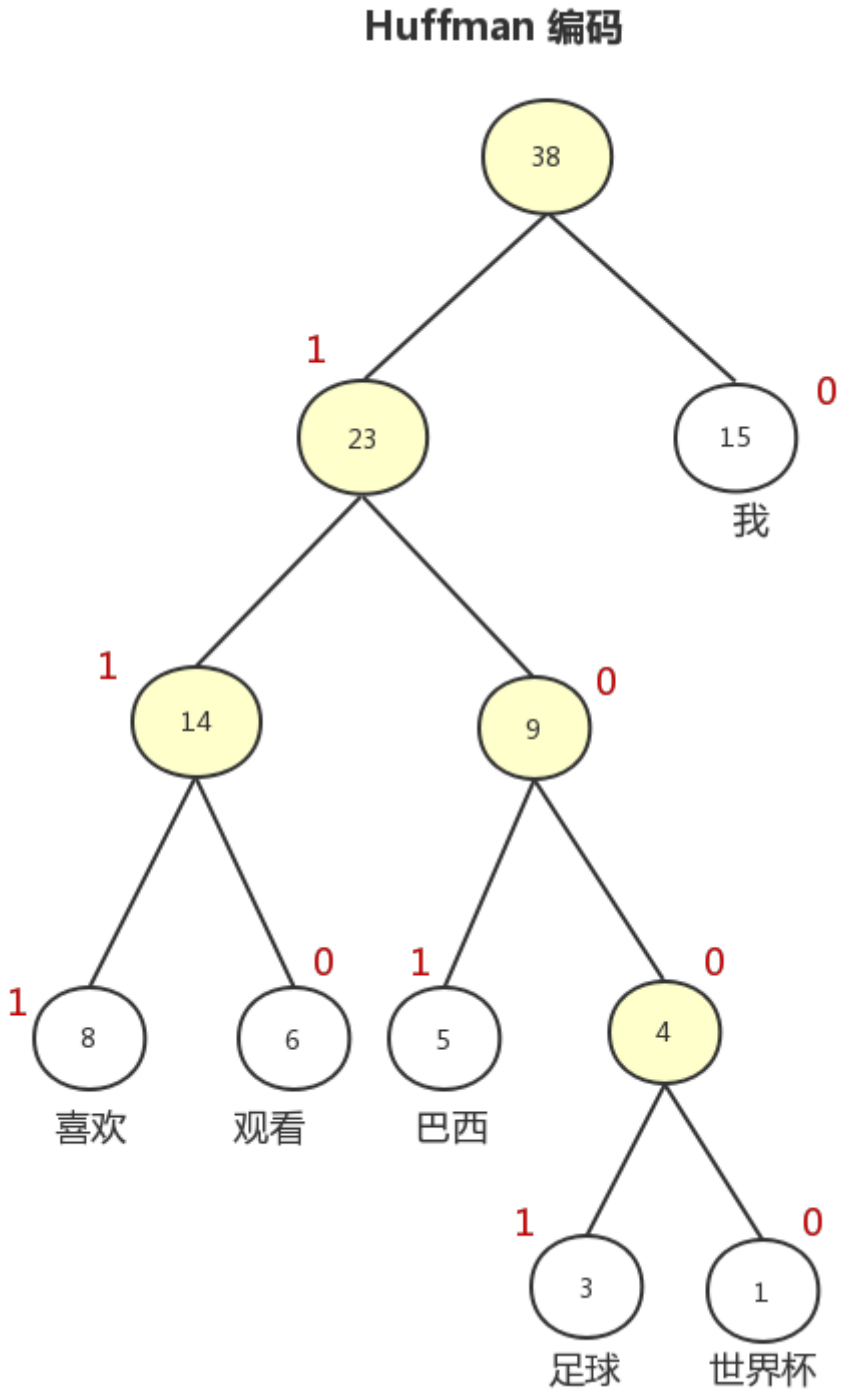
**Huffman编码**

- **二进制编码**：在通信中需要将字符转换为二进制字符串，在实际应用中各个字符出现的频次和使用频率是不相同的，我们希望能用对使用频率高的字符用短码，使用次数低的用长码，以优化通信传输的报文。
- 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），可将字符集中的字符作为二叉树的节点，字符出现的频率为节点权值，构造Huffman树。用Huffman树设计的二进制前缀编码，称为**Huffman编码。它能满足前缀编码，同时使二进制报文总长度最短**

**word2vec中的Huffman编码**

word2vec中也用到了Huffman编码，它将训练预料中的词作为二叉树的叶子节点，其在预料库中出现的次数作为节点权值，通过构造Huffman树来对每个词进行二进制编码。

如上例中，**约定词频较大的左孩子节点编码为1，较小的右孩子节点编码为0**，则“我”、“喜欢”、“观看”、“巴西”、“足球”、“世界杯”6个词的Huffman编码分别为：0,111,110,101,1001,1000




## 2. 语言模型相关知识

## 2.1 统计语言模型

**统计语言模型**是NLP的基础，它通常基于一个语料库，来计算一个句子的概率，是一个**概率模型**。 假设$W = w_1^T = (w_1, w_2, \ldots, w_T$表示由T个词$(w_1, w_2, \ldots, w_T$按顺序组成的句子. 则句子出现的概率：

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

利用贝叶斯公式，上式可以被链式的分解为：

$$p(w_1^T) = p(w_1) * p(w_2|w_1) * p(w_3|w_1^2) \cdots p(w_T|w_1^{T-1})$$

其中，概率$p(w_1), p(w_2|w_1), p(w_3|w_1^2)$.即是语言模型的参数，若提前基于语料库算出这些参数，则给定任意句子$w_1^T$都可以计算出相应的$p(w_1^T)$。

如何计算这些参数呢？常见的方法有n-gram模型，决策树，最大熵模型，条件随机场，神经网络等方法。

## 2.2 n-gram 模型

**n-gram**基本思想是，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

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

上式成立在**大数定理**下。

**n-gram**其主要工作是在语料中统计各种词串出现的次数以及**平滑处理**。概率值计算好之后存储起来，在需要计算一个句子概率时。只需找到相关的概率参数连乘即可。

**可靠性与可区别性** 当n增大时，参数越多，模型的可区别性越好，但同时单个参数的实例变少从而降低了模型的可靠性。因此n的选择需要在可靠性和可区别性之间进行折中。

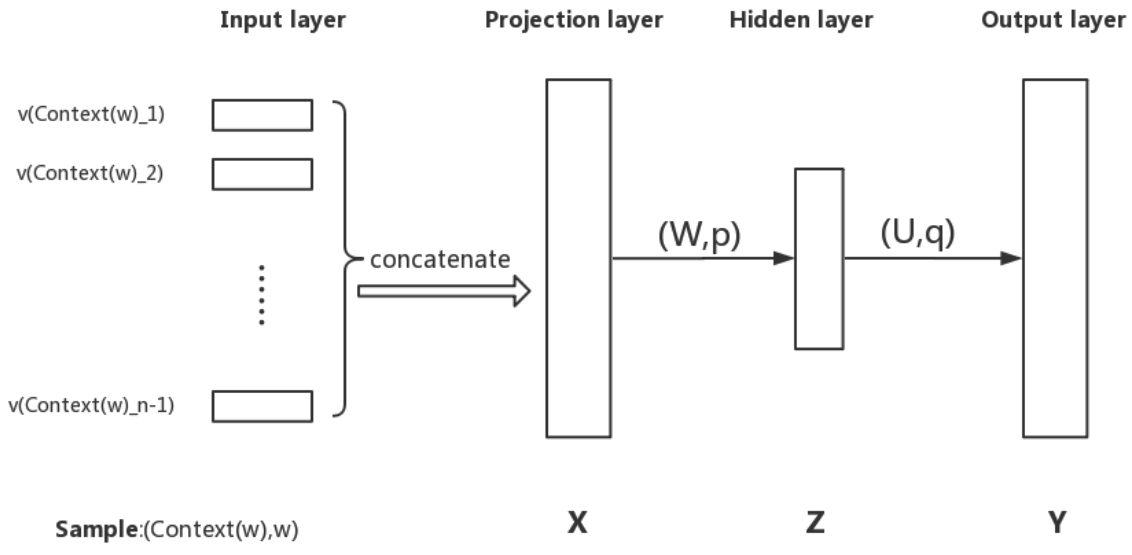
语料库要足够大，参数数量随着n的增加而呈指数型增长。实际应用中通常采用n=3的三元模型。

## 2.3 神经概率语言模型

Bengio等人在2003论文《A Neural Probabilistic Language Model》中提出了一种神经概率语言模型，用到了重要工具--词向量
**词向量**：简单来说对于词典$D$中的每一个词$w$，指定一个固定长度的实值向量$v(w) \in R^m$，$v(w)$即称为词w的词向量，m是向量的长度。
词嵌入(word embeding)方法：

- **one-hot Representation** 向量长度为词典大小，向量的分量只有一个为1，其他全为0,1的位置对应该词在词典中的索引
- **Distributed Representation** 通过模型训练将语言中的词语映射成一个固定长度的短向量，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，这些点在空间中的距离可以反映词语在语义上的相似性。(word2vec思想基础)

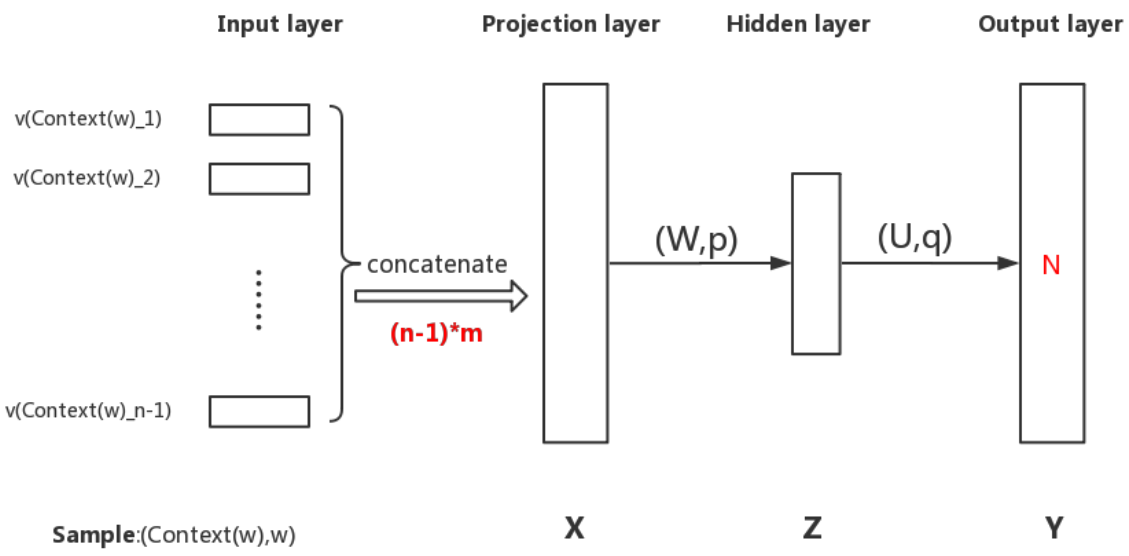
**神经概率语言模型**
对于语料C中的任意一个词$w$，$Context(w)$为其前面的n-1个词，则$(Context(w), w)$就是一个训练样本，在神经概率语言模型中：




神经网络结构示意图

- 模型包括四层：输入层，投影层，隐藏层，和输出层
- 其中$W,U$是投影层与隐藏层以及隐藏层与输出层之间的权值矩阵，$p$，$q$分别是隐藏层和输出层上的偏置向量。

一旦语料库C和词向量长度m给定后，模型投影层和输出层的规模即确定:




神经网络结构示意图

- 投影层由输入层的n-1个m维的向量串联而成。若词w前不足n-1个词，则人工增加填充向量
- 输出层的规模为N=|D|，即语料库$C$的词汇量大小
- 隐藏层规模是可调参数，由用户指定，它决定了权值矩阵的规模

在模型中，输入层到隐藏层以及隐藏层到输出层的计算过程如下：

$$\begin{cases} z_w = tanh(Wx_w + p) \\ \\ y_w = softmax(Uz_w + q) \end{cases}$$

- $tanh$为双曲正切函数，作为隐藏层的激活函数，作用于词向量的每一个分量上。

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- $softmax$作为归一化函数，使得$y_w$的分量$y_{w,i}$能表示当上下文为$Context(w)$时，下一个词恰是词典D中第i个词的概率：

$$p(w_i|Context(w)) = \frac{e^{y_{wi}}}{\sum_{k=1}^{N} e^{y_{wk}}}$$

模型的目标函数为：

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

**Softmax函数**

或称归一化指数函数，它能将一个含任意实数的K维向量 z “压缩”到另一个K维实向量$\sigma(z)$中，使得每一个元素的范围都在 (0,1)之间，并且所有元素的和为1。该函数的形式通常按下面的式子给出：

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, for j = 1, \dots, K.$$

Softmax函数在包括多项逻辑回归，多项线性判别分析，朴素贝叶斯分类器和人工神经网络等的多种**基于概率的多分类问题**方法中都有着广泛应用。

**模型运算规模**
如图所示，神经网络中，投影层，隐藏层，输出层的规模分别为：$(n-1)*m, n_h, N$, 其所涉及的参数量为：

- n是上下文词数，通常不超过5
- m是词向量长度，通常是$10^1 - 10^2$量级
- $n_h$由用户指定，通常是$10^2$
- N是语料库词汇量大小，与语料有关，通常是$10^4 - 10^5$

可以发现，整个模型的计算量集中在隐藏层与输出层之间的矩阵向量运算以及输出层的$softmax$归一化运算上。
后续的优化如word2vec也是针对这一部分进行。

**优点**
与n-gram相比，神经概率语言模式的副产品词向量可以体现出词语之间的语义相似性；此外还自带平滑处理功能。

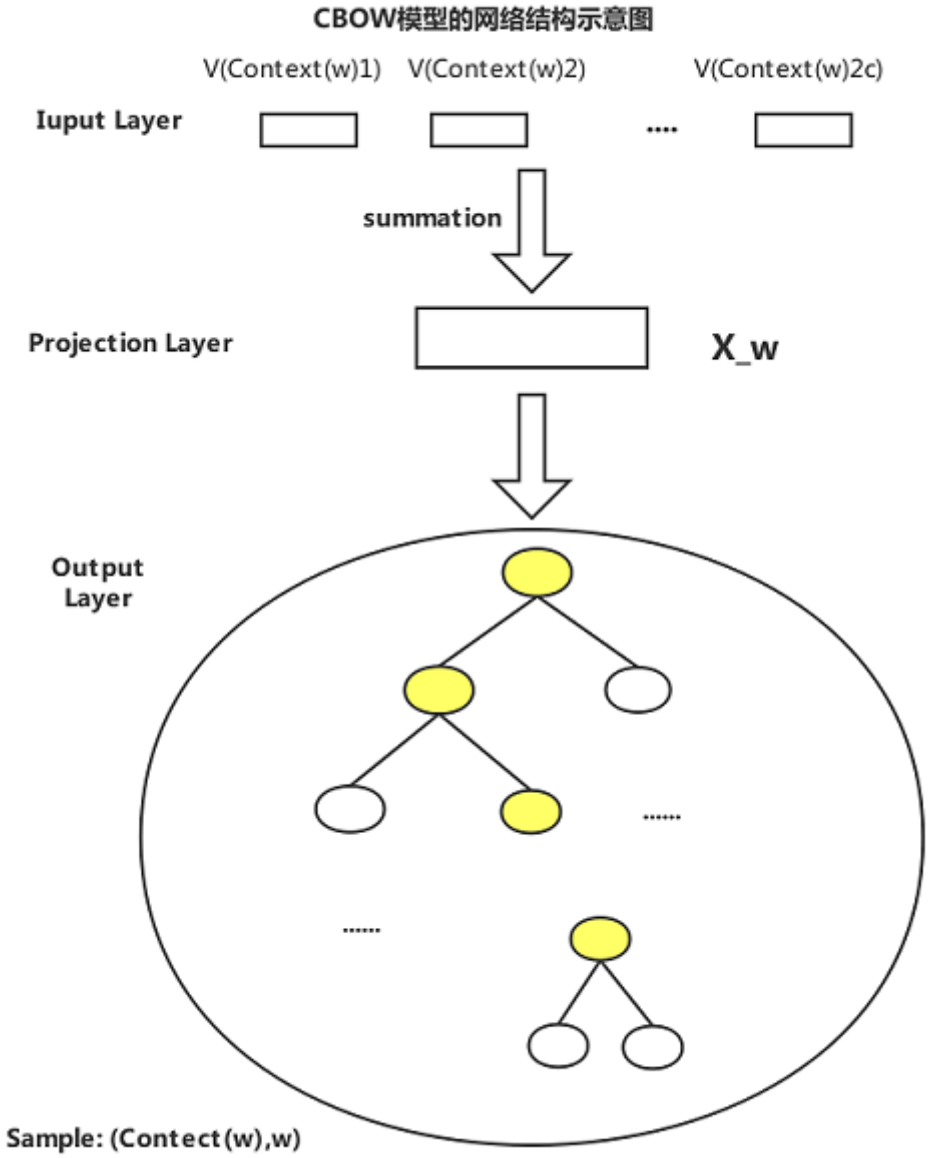
## 3. 基于Hierarchical Softmax的模型

在神经概率语言模型的基础上，Tomas Mikolov提出了word2vec。
word2vec中用到的两个重要模型：

- **CBOW模型**(Continous Bag-of-Words Model)和**Skip-gram**模型(Continous Skip-gram Model)
- 对于这两个模型,word2vec给出了两套框架，分别是基于**Hierarchical Softmax**和基于**Negative Sampling**来进行设计。

## 3.1 CBOW模型

**CBOW模型**包括三层：**输入层，投影层，输出层**
以样本(Context(w),w)为例，设Context(w)由w的前后各c各词构成，则CBOW结构如下：




- **输入层**：包含$Context(w)$中2c个词的词向量$v(Context(w)_1), v(Context(w)_2), \dots, v(Context(w)_{2c}) \in R^n$，$n$表示词向量的长度
- **投影层**：将输入层的词向量做**求和累加**，即$X_w = \sum_{i=1}^{2c} v(Context(w)_i)$
- **输出层**：对应一颗二叉树，**以词库中的每个词为叶子节点，以语料中词语的词频作为权值构造的Huffman树**，叶子节点($N = |D|$)对应词典$D$中的词，非叶子节点$N - 1$个(图中黄色节点)

对比神经概率语言模型：

- 输入层到投影层，一个是拼接串联，一个是累加求和 （ CBOW减少了输入层维度 ）
- 神经概率语言模型有隐藏层
- 输出层，一个是线性结构，一个是树形结构（ CBOW输出层的Huffman树形结构为Hierarchical softmax奠定了基础 ）

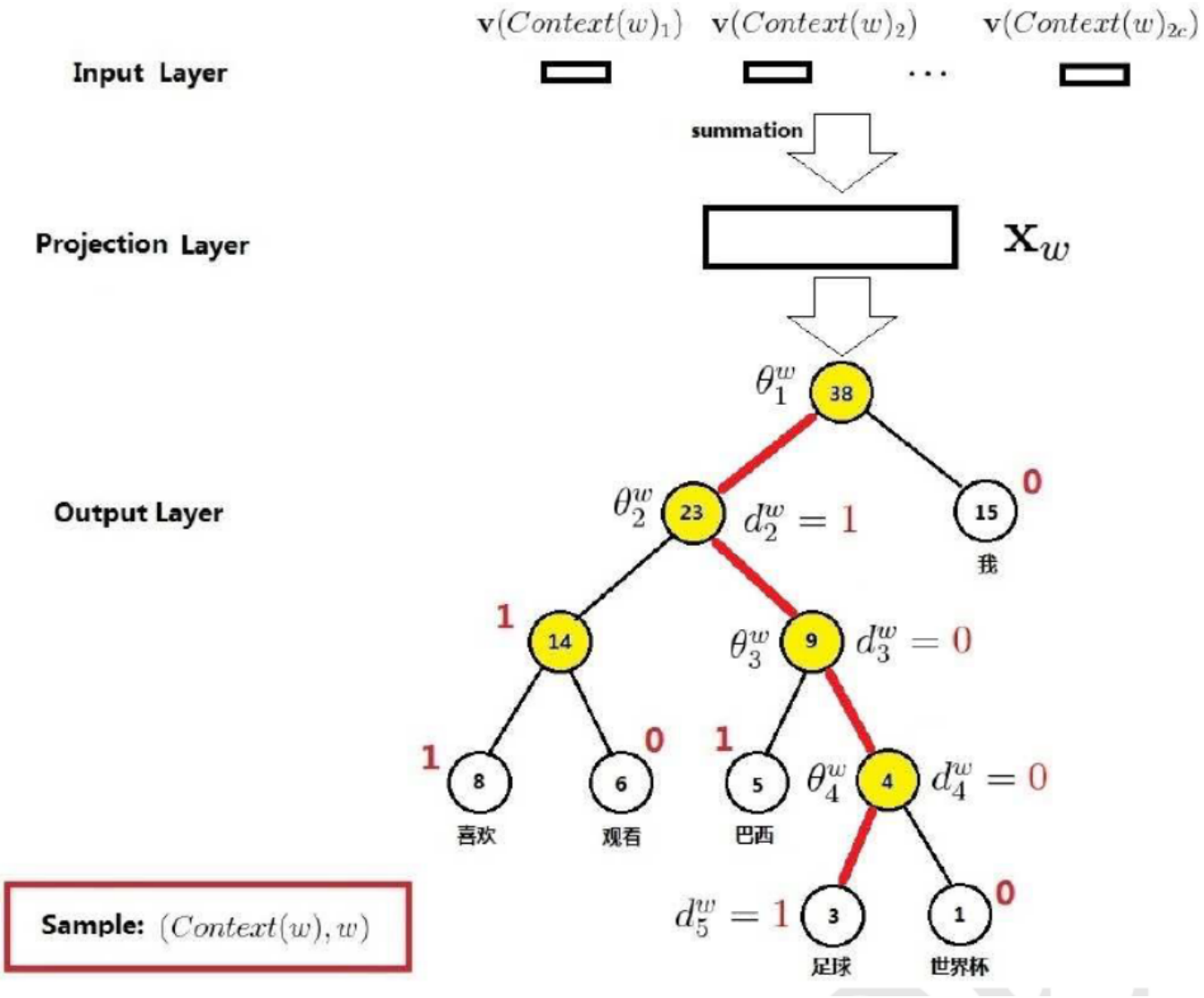
### 3.1.1 符号定义

设Huffman树中的某个叶子节点，它即代表样本$(Context(w), w)$中的w，那么定义：

- $p^w$：表示从根节点出发到达词w的对应节点组成的**路径**
- $l^w$：表示$p^w$中包含的**节点个数**
- $p_1^w, p_2^w, \dots, p_{l^w}^w$: 表示路径中的每个**节点**
- $d_1^w, d_2^w, \dots, d_{l^w}^w \in 0, 1$: 表示词w对应的**Huffman编码**，由$l^w - 1$位01编码构成（根节点不对应编码）

- $\theta_1^w, \theta_2^w, \ldots, \theta_{l^w-1}^w \in R^m$: 表示路径中**非叶子节点对应的向量**，$\theta_j^w$表示第j个非叶子节点对应的向量

例：sample(context(w),w) = ('我，喜欢，看，巴西，世界杯'，足球)，其中：
路径长度$l^w$=5，节点 $p^w$={$p_1^w, p_2^w, p_3^w$， $p_4^w$， $p_5^w$}
Huffman编码 $d_2^w, d_3^w, d_4^w, d_5^w$=1001
$\theta_1^w, \theta_2^w, \theta_3^w, \theta_4^w$分别表示路径上的4个非叶子节点对应的向量




### 3.1.2 模型

- 输出层二叉树的每一次都可视为一次二分类，那么可知$Label(p_i^w) \in 0, 1$
- 用逻辑回归来拟合二分类任务，可知属于正类概率为：

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

负类概率为：

$$1 - \sigma(X_w^T\ \theta)$$

从根节点到分支叶子节点“足球”的概率为：

$$p = (\text{足球}|context(\text{足球})) = \prod_{j=2}^{l^w} \{[\sigma(X_w^T\ \theta)]^{1-d_j^w} * [1 - \sigma(X_w^T\ \theta)]^{d_j^w}\}$$

**模型中，词向量和非叶子节点向量都要先初始化，初始化方式有几种：0-1初始化，随机值初始化，Xavier 初始化，DL中通常优先选择Xavier初始化**

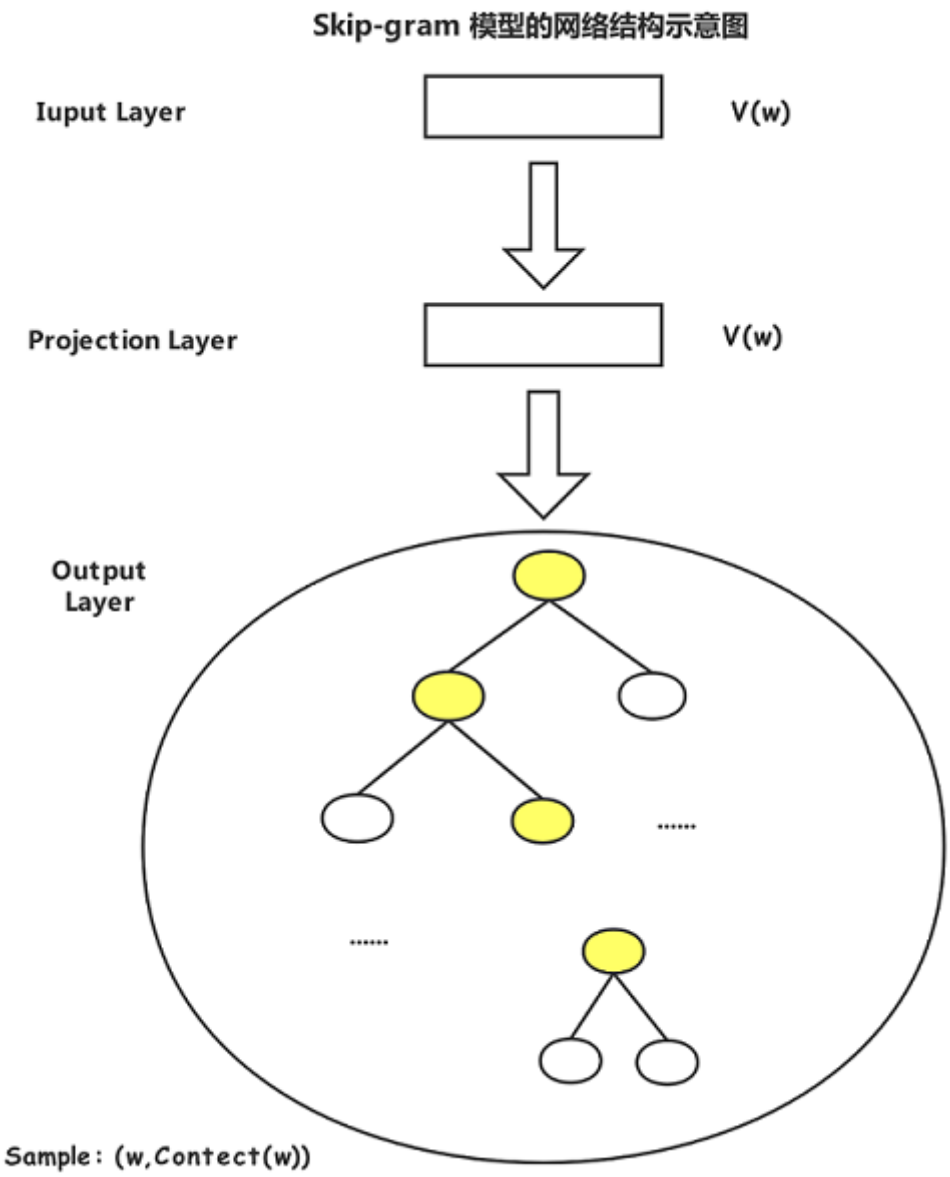
- 由此得到了CBOW模型的概率模型，目标是要使该模型概率值最大，即我们的优化目标函数为：

$$L = \sum_{w \in C} log \prod_{j=2}^{l^w} \{[\sigma(X_w^T \theta)]^{1-d_j^w} * [1 - \sigma(X_w^T \theta)]^{d_j^w}\}$$

word2vec中采用随机梯度上升法求目标函数最大值。

## 3.2 Skip-gram模型

**Skip-gram**模型核心思想是，已知当前中心词w，对其上下文$Context(w)$中心词进行预测，网络结构如下：




- **输入层**：只包含当前样本的中心词$w$的词向量$v(w) \in R^m$
- **输出层**：同CBOW一样，也是Huffman树。

### 3.2.1 模型

根据Skip-gram的思想，模型可以表示为：

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

其中$u$表示中心词w上下文中的其中一个，其概率公式类似于Hierarchical Softmax，为：

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

其中

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

对语料库所有样本建模后的目标函数：

$$L = \sum_{w\in C} log \prod_{u\in Context(w)} \prod_{j=2}^{l^u} \{[\sigma(v(w)^T\ \theta)]^{1-d_j^w} * [1-\sigma(v(w)^T\ \theta)]^{d_j^w}\}$$

## 4. 基于Negative Sampling的模型

与Hierarchical Softmax不同，Negative Sampling不再使用Huffman树，而是利用简单的随机负采样替代。
通过随机负采样，提高训练速度，改善所得词向量质量。

## 4.1 CBOW模型

在CBOW模型中，我们已知词$w$的上下文$Context(w)$,需要预测$w$ 因此可以把对应的w看做正样本，而词典中的其他词就是负样本。
假定通过负采样算法生成了关于w的负样本集$NEG(w) \neq \varnothing$, 则可看做一个二分类问题，对任意$u$有定义：

$$Label(u) = \begin{cases} 1, & u = w \\ 0, & u \neq w \end{cases}$$

则模型函数为：
单样本

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

所有样本：

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

- 其中$X_w$是$Context(w)$中所有词向量求和所得，$\theta^j$是样本中每个词向量对应的参数向量，$C$是语料库对给定的语料库C,最终的目标函数为：

$$L = \sum_{w\in C} log\ g(w)$$

## 4.2 Skip-gram模型

基于Skip-gram的核心思想，其模型即在CBOW的基础上，对于给定的$w$,其context(w)中的每个词都是一个正样本，都要为期随机采样出一个负样本集，所以模型函数为：

$$G = \prod_{w\in C} \prod_{u\in Context(w)} \prod_{z\in\{u\}\cup NEG(u)} p(z|w)$$

其中（z|w）只是中心词w上下文中的一个，对应概率为：

$$p(z|w) = \prod_{j=2}^{l^z} \sigma(d_j * v(w)^T\ \theta_{j-1}^z)$$

目标函数为：

$$L = \sum_{w\in C} \sum_{u\in Context(w)} \sum_{z\in\{u\}\cup NEGz} logp(z|w)$$

## 4.3 负采样算法

在word2vec中，负采样是为了对于给定的词$w$,如何采集生成$NEG(w)$

- 词典D中的词被采集作为样本的概率与其在语料C中出现的频率有关，所以其实是带权采集：

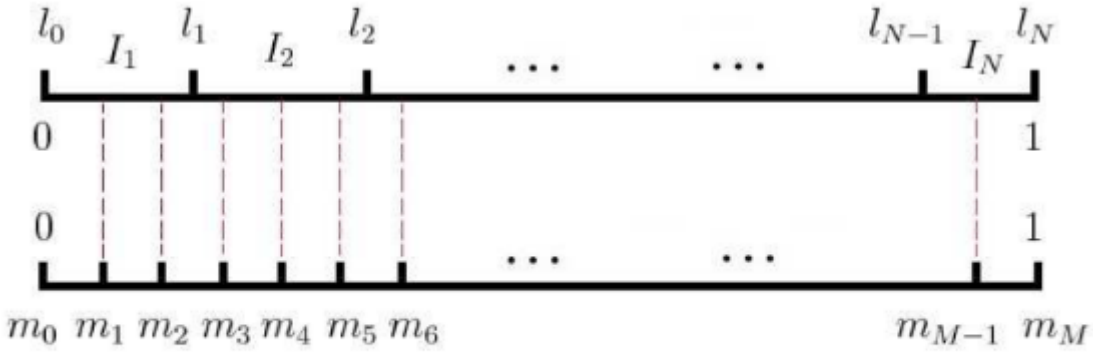
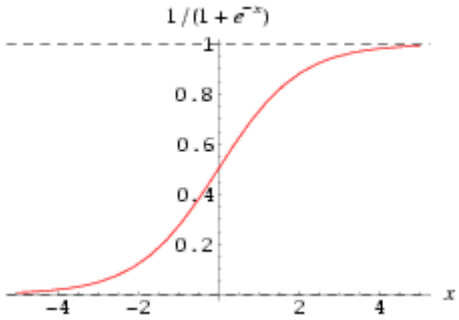



图 13 Table(·) 映射的建立示意图

## 5. 源码优化细节

代码优化主要目的是提高计算效率。

## 5.1 $\sigma(x)$ 的近似计算




由上图可见，sigmod函数$\sigma(x)$在0附件变化剧烈，越往两边越趋于平缓。
当x<-6或x>6时函数值基本保持不变，前者趋近0，后者趋近1。 word2vec中用近似方法来提高效率：

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

当$x \in (-6,6)$,$x_k$取向上或向下取整提前计算好一批节点的值，通过匹配查询提高效率。

## 5.2 词典的存储

**word2vec中词典的源码是通过哈希技术来存储的**
代码中定义了长度为$3 * 10^7$的整数数组vocab_hash,初始化为-1，为词典D建立如下映射：

$$vocab\_hash[hv(w_j)] = j$$

当出现冲突时：

$$hv(w_i) = hv(w_j), i \ne j$$

采用线性探测的开放定址法解决冲突，即当
$vocab_hash[hv(w_j)] \ne -1$时，线性顺序往下找，直到找到值为$-1$的位置，作为当前$j$的索引地址
**匹配**

当
$vocab_hash[hv(w_j)] = -1$表示词$w_j$不在词典$D$中，若$vocab_hash[hv(w_j)] \neq w_j$，则线性往下找，直到$vocc$
$-1$

## 5.3 低频词和高频词

**低频词处理**：视语料库规模而定，当D的规|D|>0.7×vocab_hash_size,则从语料库中删除出现次数小于等于min_reduce的词。
**高频词处理**：对于出现上千万次的词，往往是助词：的，是等词，这些词在众多样本进行训练时不会发生显著变化，因此对高频词采用Subsampling的技巧处理：

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

t为提前设定的参数，过程中将以概率$prod(w)$的概率删除高频词。

## 5.4 窗口及上下文

- 模型训练以行为单位，利用换行符分隔，当句子太长（词语数超过1000），则强行截断。
- word2vec中，取上下文词数c是不固定的，通常是每次构造Context(w)时，生成区间[1，window]中的随机整数c，然后Context(w)取w前后c个词。
- 与神经概率语言模型不同，word2vec输入层是对词向量的求和累加，并未采用拼接方式，这样即使是句首和句尾的词也不会需要填充向量。

## 5.5 自适应学习

**学习率**$\eta$:在word2vec中，学习率不是固定不变的，一般是设定初始学习率
$\eta_0$(默认$0.025$),然后每完成$10000$个词的训练，则更新一次学习率:

$$\eta = \eta_0(1 - \frac{word\_count\_actual}{train\_words + 1})$$

其中$word_count_actual$表示当前已处理过得词，$train_words$为训练总词数

- 由此，可知随着训练的进行，学习率$\eta$会逐渐减小
- 但当$\eta$小值阈值$\eta_{min}$时，会将$\eta$固定为$\eta_{min} = 10^{-4} \times \eta_0$

## 5.6 参数初始化与训练

**word2vec采用随机梯度上升法，且只对语料库遍历一次**
**参数初始化**： word2vec模型中参数包括，逻辑回归对应的参数向量，以及词典D中每个词的词向量。

- 逻辑回归对应的参数向量采用**零初始化**
- 词向量采用**随机初始化**：

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

其中m是词向量长度。可以看出词向量分量均落在区间$[-\frac{0.5}{m}, \frac{0.5}{m}]$

## 其他

- word2vec是在分好词的语料基础上进行训练，中文中其词向量受分词质量影响？
- word2vec训练词向量时，Hierarchical Softmax和Negative Sampling两套框架都用到了，词向量结果是两者的融合。
- 投影层的首尾串联和词向量相加的方式，一个会考虑词序，一个不考虑词序。若考虑词序词向量质量是否有提升。
- 如何进行增量训练

In [ ]:

```

```

In [ ]:

```

```