Word2vec

1. 如何表示词语的含义?

1.1 WordNet

WordNet是一个依靠专家的知识建立的大型英语词汇数据库(lexical database), 其中包含了同义词(synonym)和IS-A关系词(hypernym)。但是，WordNet的缺点是显而易见的。

- 不能把握词语细微的差别。如"proficient"和"good"在WordNet中是同义词，但是它们在不同语境下意思未必完全一样。
- 不能与时俱进，一些新词语的含义根本就没有。比如"wizard",“ninja”,“ipad”
- 需要大量的专家人力来完成，而且主观性很强
- 不能够比较词语的相似性，这是非常致命的一点

1.2 离散式语义(bag of words, 2012年以前)

在传统NLP中，每个词被看作一个离散的symbol，这就是 localist representation。每个词都可以被表示成一个one-hot向量，如：

```
                                                        $motel=
[0,0,0,0,0,0,0,1,0,0,0,0] \\ hotel=[0,0,1,0,0,0,0,0,0,0,0,0]  $
```

向量的维度就是词汇表vocab的大小，比如500000维。

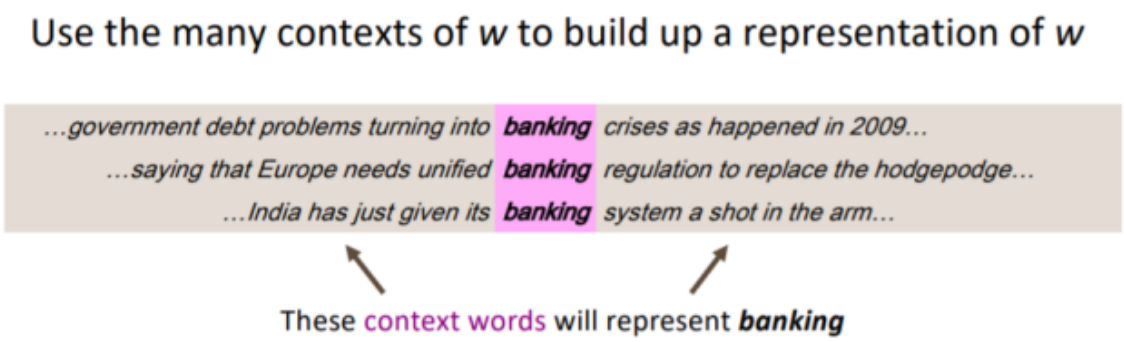
离散式语义的缺点：

- 语料稍微大一点，词语对应的矩阵就会非常大且稀疏。
- 不能衡量两个词语之间的相似度。从"motel"和"hotel"的例子中可以看出，每两个词语的表示都是正交的，也就是similarity=0. 这显然是很糟糕的。比如，我们在搜索"Seattle motel"的时候，也希望能显示"Seattle hotel"的结果。

1.3 分布式语义(distributed representation, representing words by their context)

一个词语的含义是由它周围的词来决定的(a word's meaning is given by the words that frequently appear close-by)。

分布式的意思意味着，一个dense vector的每一位可以表示多个特征、一个特征也可以由很多位来表示。



我们将每个词语都表示成一个dense vector，使得周围词(context word)相近的两个词的dense vector也相似。这样就可以衡量词语的相似性了。

一个好的word representation 能够把握住词语的syntactic(句法，如主谓宾)与semantic(词语的语义含义)信息，例如，一个优秀的词语表示可以做到：

```
                                $$WR(“China”) - WR(“Beijing”) + WR(“Tokyo”) =
WR(“Japan”)\\ WR(“King”) - WR(“Queen”) + WR(“Woman”) = WR(“Man”) $$
```

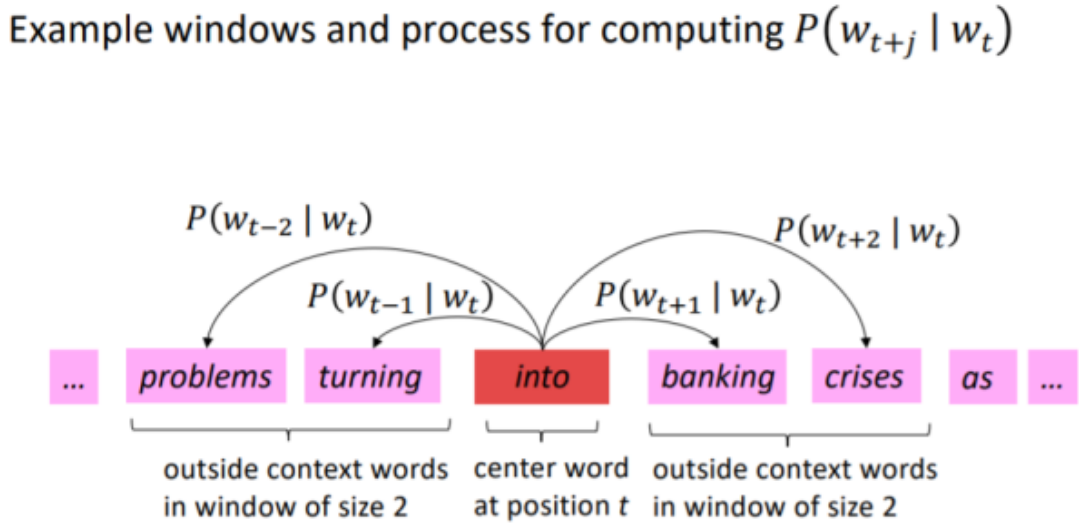
2. Word2vec

2.1 概述

Word2vec (Mikolov et al. 2013)是学习词向量的一种方法。

思想：

- 我们有一个非常大的语料，需要把这个语料中的每个词表示成词向量
- 遍历语料的每个位置 t，都有一个中心词 c (center word) 和 上下文词 o (context word)。算法刚开始的时候，每个词的词向量都随机初始化。
- 计算给定上下文词 o 的条件下中心词为 c 的概率，即 $p(c|o)$, 这就是 CBOW(continuous bag of words)的思想；或者相反，计算给定中心词 c 的条件下周围词为 o 的概率，这就是skip-gram的思想。
- 一直调整每个词的词向量，使得上面说的这个概率最大。

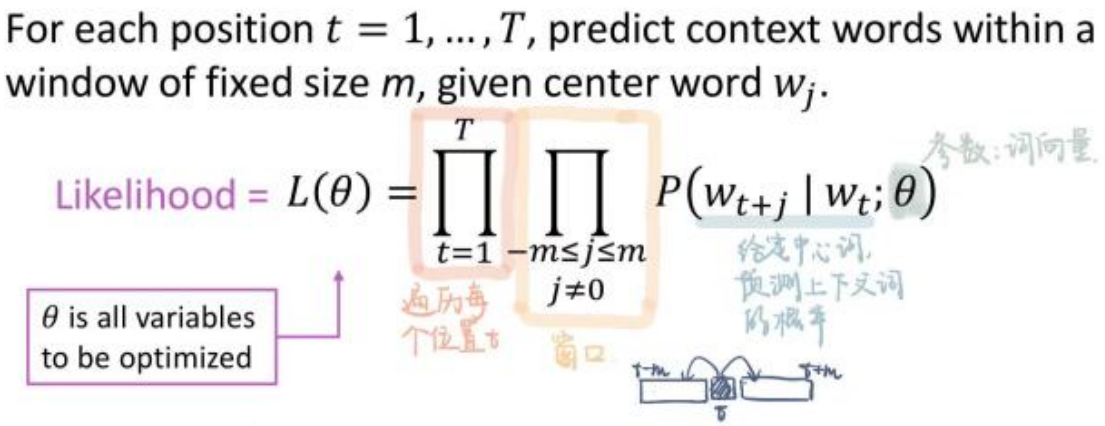


skip-gram 示意图

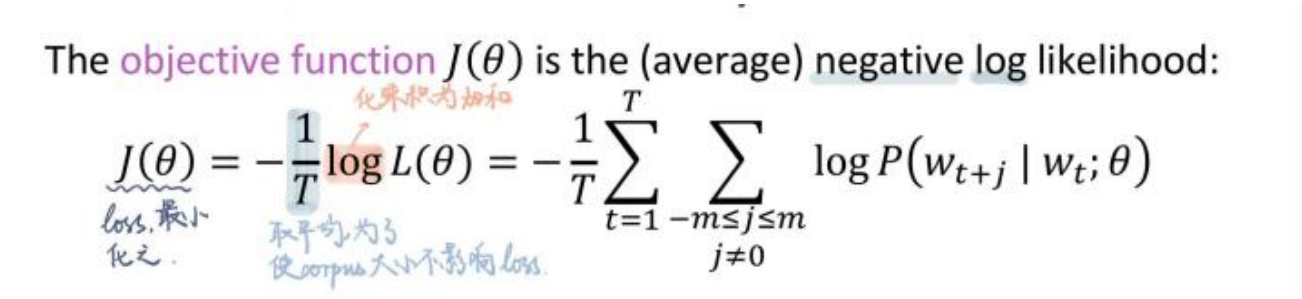
2.2 word2vec的目标函数

2.2.1 目标函数的表示

对于语料库中的每一个位置 $t = 1, 2, \ldots T$，都计算一下给定中心词 w_t 的条件下，窗口大小为 m 的上下文词出现的概率。这个概率(也叫似然度)就是：



损失函数就是：



最小化损失函数就相当于最大化概率。

2.2.2 如何计算目标函数

上文已经推导过了，损失函数为

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\neq 0} logP(w_{t+j}|w_t;\theta)$$

那么如何计算

$$logP(w_{t+j}|w_t;\theta)$$

呢?

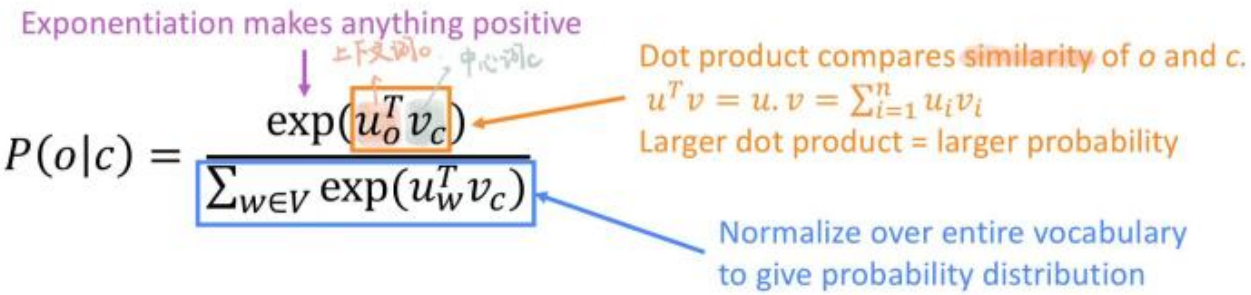
解决办法是，我们为每个词语

$$w$$

都训练两个向量:

- v_w 是当 w 为中心词时，w 的表示向量
- u_w 是当 w 为上下文词时，w 的表示向量

最后的词向量结果可以是 v_w 和 u_w 的平均。

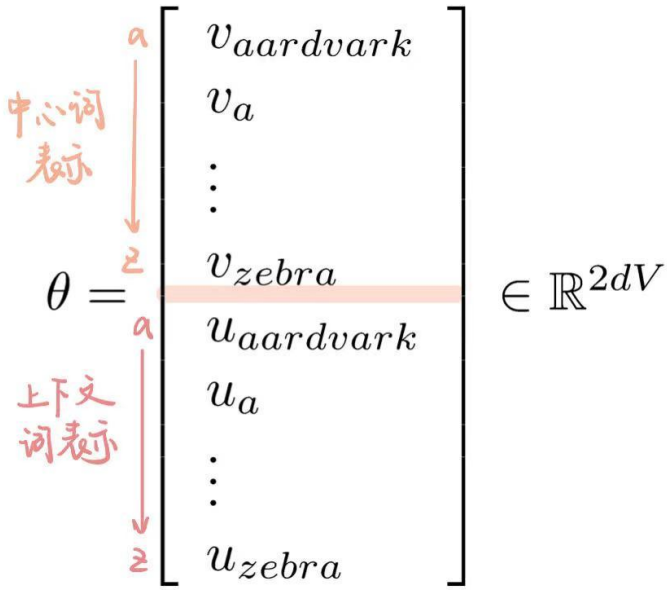


这个概率 $P(o|c)$ 的计算方法是一个典型的【**softmax**】方法:

- "soft": 对于那些不是最大值的值，softmax也给了它们一定的小值，而不是像hardmax那样采用one-hot编码，让非最大值的值都为0.
- "max": 表示它使最大的值更加放大(exp函数特点)

2.2.3 梯度下降 训练模型

为了训练模型，需要每一步调整参数使得loss最小化。这里说的参数，实际上就是所有个词语的词向量表示：



$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

参数的维度是 $2dV$，是因为每个词语都有两个表示(作为中心词时的 v_{word}、作为周围词时的 u_{word}，每个表示的维度为d。总共有V个词语，所以，总共需要有2dV个参数。

使用随机梯度下降法(SGD)来更新参数，损失函数 $J(\theta)$ 关于参数 θ 的梯度为：

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

这是因为如果采用随机梯度下降法(小批量梯度下降法也会遇到这个问题)，每次只考虑一个位置 t，这样只会更新2m+1个词语的词向量(m为窗口大小)。这样，梯度是一个非常稀疏的向量！

解决这个问题的方法是：由于每次只会更新我们这轮迭代见到的2m+1个词语的词向量，其他词语的词向量根本不会发生变化，所以可以维护一个词语->词向量的哈希，这样每次只改变那2m+1个词语的词向量就可以了。尤其是当我们有非常多的词语的时候，绝对不能每次都传一个巨大的、稀疏的梯度向量！

2.2.4 Skip-gram 模型的优化 – negative sampling

skip-gram模型就是我们一直在讲的方法，即在给定中心词 c 的条件下，求上下文词 o 出现的概率，并不断调整模型参数使得这个概率最大化的一种方法。

之前我们用朴素的softmax来衡量每个上下文词 o 出现的概率，分母是vocab中所有词语的概率总和。这显然需要太长的时间来计算，因为词汇表中的词语实在是太多了！所以，实际上我们使用negative sampling的方法。

- Overall objective function (they maximize): $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j\sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$$

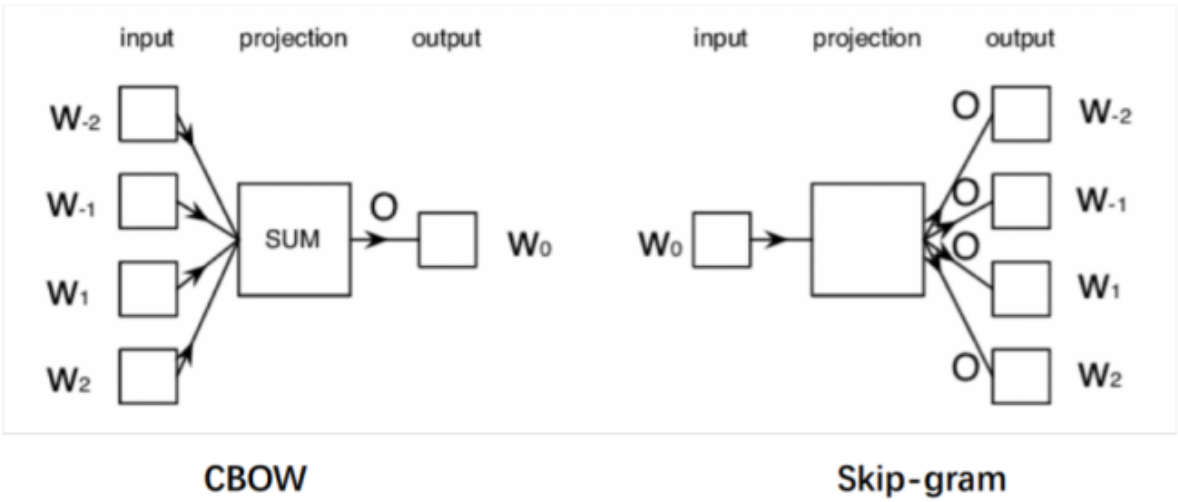
真实的上下文词o　maximize　按词频随机选　minimize $\log\sigma(u_j^T v_c)$

- $P(w)=U(w)^{3/4}/Z$,
  the unigram distribution U(w) raised to the 3/4 power
  (We provide this function in the starter code).
- The power makes less frequent words be sampled more often

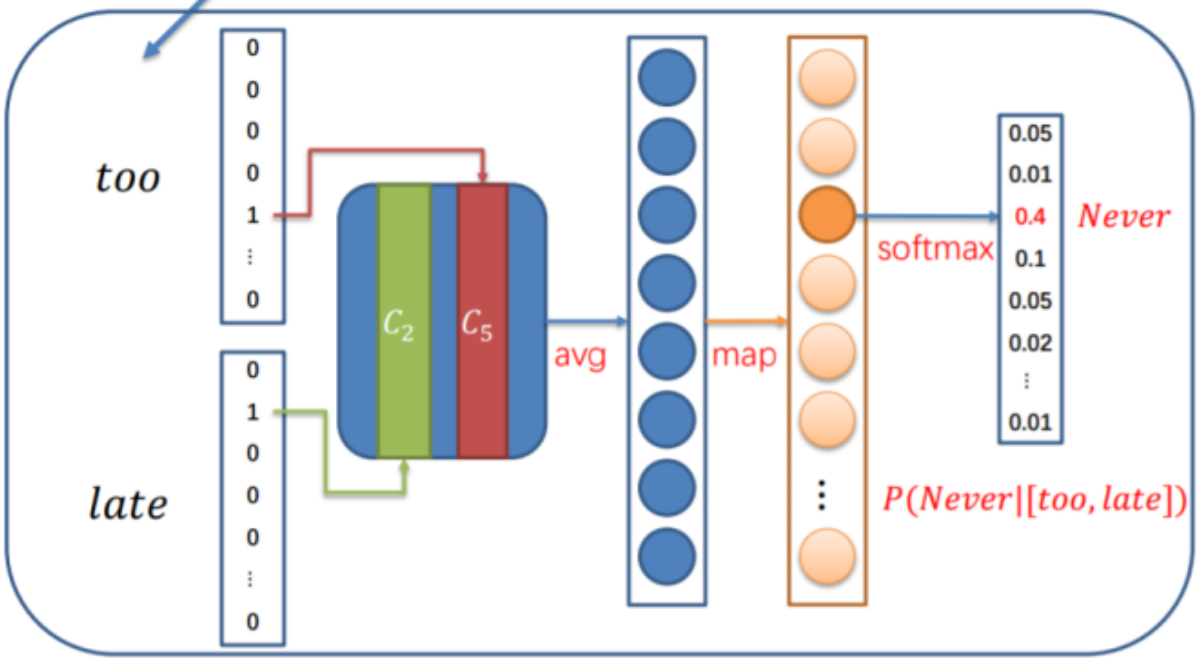
使用negative sampling的方法，每次只需采样k个负样本，并不需要计算所有词汇表，大大节省了计算时间。

2.2.5 CBOW (Continuous Bag of Words)

已知窗口中的一些上下文词，计算中心词为 c 的概率，并最大化之。根据bag of words假设，周围词出现的顺序不会影响它们预测中心词的结果。



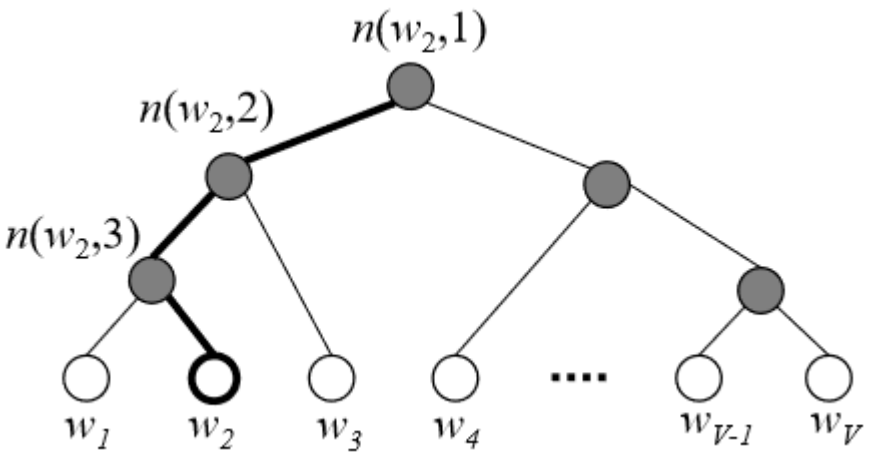
- e.g.
  - *Never too late to learn*



对于CBOW，输入层是上下文词的词向量，投影层对其求和（简单的向量加法），输出层输出最可能的中心词w。由于语料库中词汇量是固定的|V|个，所以上述过程其实可以看做一个多分类问题。

对于神经网络模型多分类，最朴素的做法是softmax回归，但是softmax回归需要对语料库中每个词语都计算一遍输出概率并进行归一化，在几十万词汇量的语料上无疑是令人头疼的。所以可以使用hierarchical softmax的方法，将复杂度由 $O(V)$ 降为 $O(logV)$

该模型用二叉树来表示词汇表中的所有单词。V个单词必须存储于二叉树的叶子节点，一共有V-1个内部节点。对于每个叶子节点，有一条唯一的路径可以从根节点到达该叶子节点；该路径被用来计算该叶子结点所代表的单词的概率。



用于分层softmax的二叉树示例。白色节点表示词汇表中的所有单词，黑色节点表示内部节点。

记内部节点 $n(w,j)$ 为从根节点到单词 w 的路径的第 j 个节点，它对应一个向量$v'_{n(w,j)}$。那么，一个单词作为输出词的概率被定义为：

$$p(w=w_O)=\prod_{j=1}^{L(w)-1}\sigma\left([\![n(w,j+1)=\mathrm{ch}(n(w,j))]\!]\cdot {\mathbf{v}'_{n(w,j)}}^T\mathbf{h}\right)$$

$ch(n)$是节点n的左侧子节点；$v'_{n(w,j)}$是隐节点$n(w,j)$的向量表示；h是softmax之前最后一层隐藏层的输出值（即窗口内词语embedding的avg pooling，h=$1/C\sum_{c=1}^{C}v_{w_c}$。$[[x]]$ 是一个特殊的函数，定义如下：

$$[\![x]\!]=\begin{cases}1 & \text{if } x \text{ is true;}\\ -1 & \text{otherwise.}\end{cases} \tag{38}$$

让我们通过一个例子来直观上理解一下这个公式。我们定义在当前内部节点 n 往左走的概率为：

$$p(n,\text{left})=\sigma\left({\mathbf{v}'_n}^T\cdot\mathbf{h}\right) \tag{39}$$

它是由内部节点向量和隐藏层输出值共同决定。容易得到，从内部节点 n 往右走的概率为：

$$p(n,\text{right})=1-\sigma\left({\mathbf{v}'_n}^T\cdot\mathbf{h}\right)=\sigma\left(-{\mathbf{v}'_n}^T\cdot\mathbf{h}\right) \tag{40}$$

在上图中，我们可以计算 w_2 是输出单词的概率为：

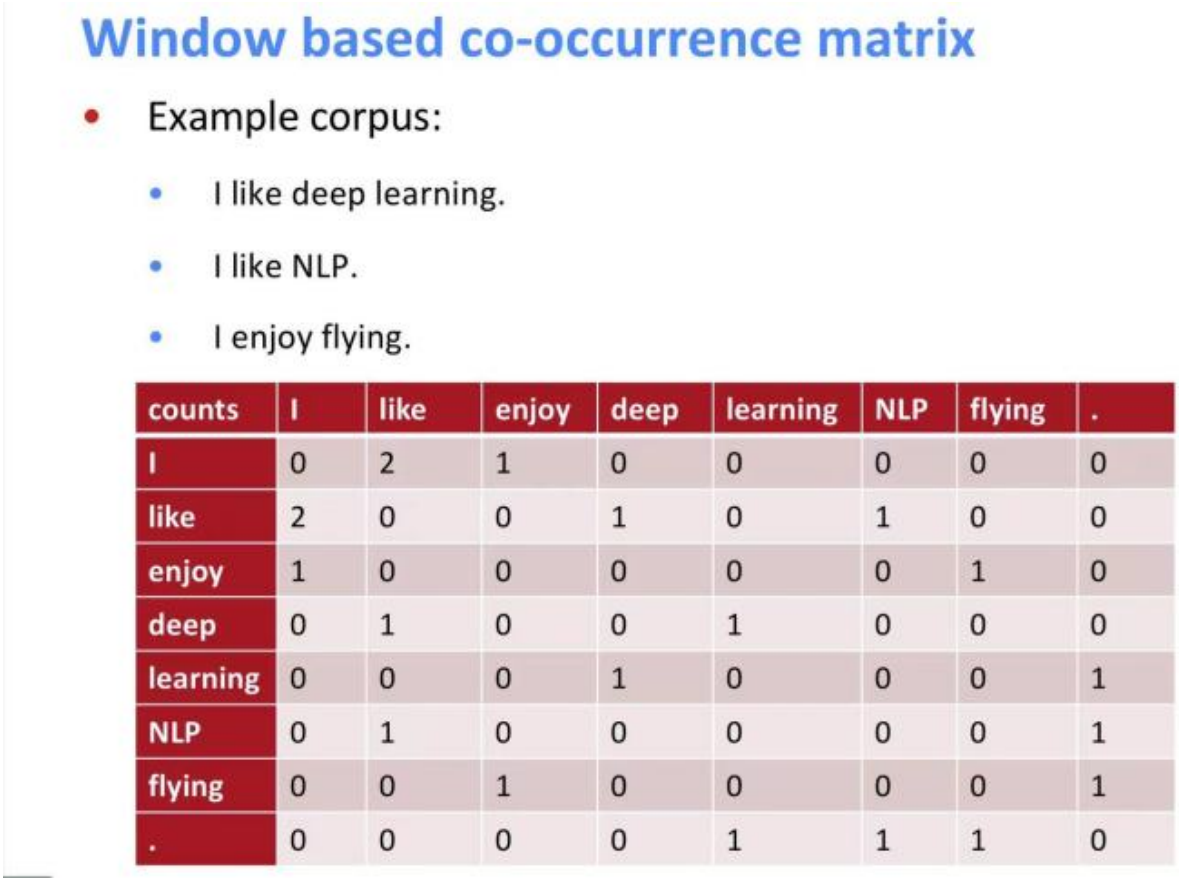
$$\begin{aligned}p(w_2=w_O) &= p\left(n(w_2,1),\text{left}\right)\cdot p\left(n(w_2,2),\text{left}\right)\cdot p\left(n(w_2,3),\text{right}\right) &(41)\\ &= \sigma\left({\mathbf{v}'_{n(w_2,1)}}^T\mathbf{h}\right)\cdot\sigma\left({\mathbf{v}'_{n(w_2,2)}}^T\mathbf{h}\right)\cdot\sigma\left(-{\mathbf{v}'_{n(w_2,3)}}^T\mathbf{h}\right) &(42)\end{aligned}$$

2.2.6 Trick

- 因为罕见词会表示更多、更独特的信息，所以应该更加关注罕见词、不要过度关注常见词。因此，可以对词语进行欠采样，以$1-\sqrt{t/f(w)}$的概率丢弃词语w，其中$f(w)$就是词语词频，t是一个可调阈值。
- Soft sliding window: 在一个窗口中，离target word较远的那些周围词应该具有较低的权重

3. 潜在语义分析 LSA

在2013年之前，人们用LSA（Latent Semantic Analysis, 潜在语义分析）来把握词语的共现关系，以此来得到词向量。例如，一个词语共现矩阵是这样的：



counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

使用SVD分解将其降维，就得到了词语的稠密表示。

4. 词向量的评估

有两种方法来衡量word embedding方法的好坏：

- intrinsic evaluation：直接看embedding值相似的两个词，究竟是否相近？或者，类似“man之于king相当于woman之于？”这样的analogy问题，看word embedding能否很好的回答。当然这需要和人工标记的结果做对比。
- extrinsic evaluation：看word embedding下游任务完成的好坏。