# 什么是文本表示？

-简单表示就是词到向量的处理，将文本以向量化方式进行文本表征。

# 为什么要进行文本表示？

-1：方便计算机处理（一维是向量，二维是矩阵，三维是张量）

-2：便于机器学习

-3：良好的文本表示形式可以极大的提升算法效果

# 文本表示的分类：

**-1基于粒度：**

-（1）：长文本表示

-（2）：短文本表示（句子）

-（3）：词表示

**-2 基于表示方法：**

**-（1）：离散表示：**

**-one-hot表示**

-词典中包含的每个单词都有唯一的索引，并且在词典中的顺序和在句子中的顺序没有关联。

**-Multi-hot表示**

**-Bag of Words**（词袋子模型BOW）：

概述：将字符串视为一个“装满字符（词）的袋子”。比较的是两句话的词语重复程度，和顺序无关。

优点：简单，方便，快捷，在预料充足的前提下，对简答自然语言处理任务效果好。

缺点：无法关注词语之间的顺序关系；没有考虑到词的重要性。

-**TF-IDF：（**主要是针对BOW的缺点[词的重要性]做出的改进**）**

公式：词频(TF)\*权重(IDF)

TF-IDF值越大，则这个词成为一个关键词的概率就越大。

IDF = log(1+N/n) N:文档总数，n:含有词的文档数

例如：有100篇关于汽车的文档，其中10篇涉及到车险，那么汽车的IDF=log((1+100)/(1+100))+1，大约等于1，而车险的IDF=log((1+100)/(1+10))+1，大约等于2.

**- N-Gram的原理:**

N-Gram是基于一个假设：第n个词出现与前n-1个词相关，而与其他任何词不相关。（这也是隐马尔可夫当中的假设。）整个句子出现的概率就等于各个词出现的概率乘积。

模型公式理解：

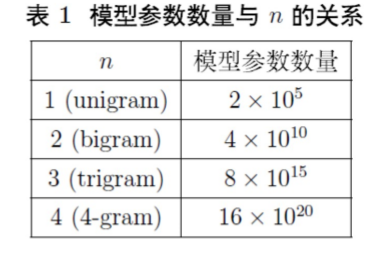
假设句子T是有词序列w1,w2,w3...wn组成，用公式表示N-Gram语言模型如下：

P(T) = p(w1)\*p(w2)\*…..\*p(wn)=p(w1)\*p(w2|w1)\*p(w3|w1w2)\*…\*p(wn|w1w2ww3…)

一般常用的N-Gram模型是Bi-Gram和Tri-Gram。分别用公式表示如下：

Bi-Gram:P(T)=p(w1|begin)\*p(w2|w1)\*p(w3|w2)\*\*\*p(wn|wn-1)

Tri-Gram:P(T)=p(w1|begin1,begin2)\*p(w2|w1,begin1)\*p(w3|w2w1)\*\*\*p(wn|wn-1,wn-2)



目前常用的基本只有2和3。

优点：考虑了词向量的顺序

缺点：词表的膨胀（词表的数据会非常大，增长的会很迅速）

离散表示总结：

1：无法衡量词向量之间的关系，词向量太稀疏，很难捕捉文本的含义

2：词表维度随着预料库增长膨胀

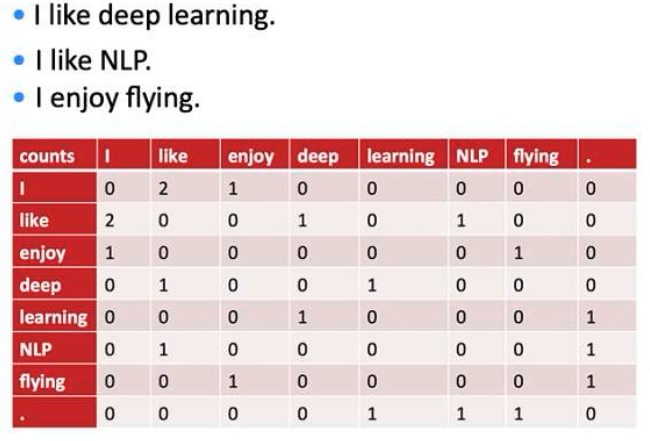
3：N-gram次序列随着语料库膨胀更快

-**（2）：分布式表示：**

用一个词附近的其他词来表示该词。

-**共现矩阵**：

使用的是滑动窗口的形式，比如对称窗口为1，就是考虑左边1个和右边1个（一般设置为5-10）。



缺点：

1：向量稀疏

2：向量维数随着词典的大小线性增长

3：词典占用空间大

改进：SVD降维（基本不用，问题太多）

**语言模型：**

概述：一句话（词组合）出现的概率和（衡量一句话的合理性）。

NNLM（神经网络语言模型）：

网络结构：输入层(N)🡪映射层🡪隐藏层(H)🡪输出层(N)

先将输入的字符串映射为词向量，然后进行tanh激活，再进行softmax分类。

投影层为点乘。

-基于矩阵表示：

-基于降维表示

-基于聚类表示

-**基于神经网络**（）

Word2Vec：就是通过学习文本来用词向量的方式表征词的语义信息，即通过一个嵌入空间（Embedding）使得语义上相似的单词在该空间内距离很近，Embedding其实就是一个映射，将单词从原先所属的空间映射到新的多维空间中，也就是把原先词缩在空间嵌入到一个新的空间中去。

例如：现在有猫，狗，男人，女人，汽车这几个词，在我们的理解中，可以认为猫和狗可以属于一类—动物，男人和女人可以属于一类—人类，汽车单独一类—机器，这样就可以降低词典的维度和稀疏性，并且可以很好的表示语义。

旅馆，旅社，宾馆，酒店这几个词的词向量表示不同，读取的语义也不同，但是使用Word2vec处理的话，这几个词的词向量所表示的非常相近，可以很好的解读语义。

**Word2vec两大模型：**

**-CBOW**

网络结构优化：

1：无隐藏层

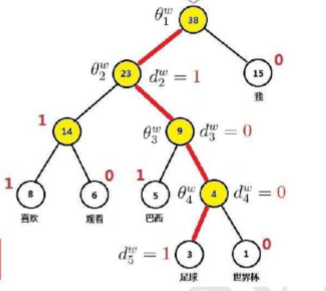
2：使用双向上下文窗口

3：投影层简化为求和

输出层的优化：

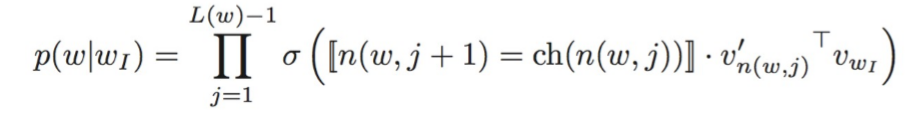
**1：层次softmax**（）：

（1）：使用Huffman Tree来编码输出层的词典。



（2）：只需要计算路径上所有非叶子节点词向量的贡献

（3）：计算量降为树的深度V =>log\_2(V)

计算公式：

二分类问题：使用了Sigmoid函数.

括号中间都是路径。

**2：负例采样**（）**：**

假如有1亿个输入，只有一个正样本，输出1亿的话不太好，所以这边就用到了负例采样，将这个词典划分成很多等分，比如10^8等分，然后随机生成一个1-10^8-1的数字，拿出这个刻度中的内容，然后重复这样的操作，每个负例只取一次，取到正例就跳过，知道取出想要输出的样本数。在这个过程中考虑到了每个词的权重的问题，所以这个过程不是等比例随机的。

**-Skip-gram：**

1：无隐藏层

2：投影层也可以省略

3：中间预测两遍。

Word2Vec存在的问题：

对多义词无法很好的表示和处理，因为使用了唯一的词向量，例如解，可以是jie,也可以是xie。

**Glove ：**

1：词嵌入可视化

2：随着迭代次数增加，精度会提升，Word2Vec未使用类似迭代次数的Epoch，用Negative Sample模拟