cs224N 笔记 1: 词向量的表示

nghuyong

December 13, 2018

1 独热表示

独热表示 (One-hot) 是最简单的词向量表示方案. 假设字典的大小为 V, 那么每个词都可以表示成一个维度为 $\mathbb{R}^{|V|\times 1}$ 的向量. 其中只有这个单词所在字典中索引的那一位是 1，其余都是 0. 如下所示:

$$w^a = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \tag{1}$$

这种表示在传统的机器学习算法中广泛使用，比如结合 SVM 就可以做垃圾邮件的分类. 但是这种表示方案不包含任何语义信息，也就是任何两个词都是正交的，乘积均为 0.

2 基于共现矩阵的表示

直觉上，如果两个单词经常共同出现在同一篇文档中，或者同一句话中，我们就可以认为这两个单词在语义上有一定的联系. 比如"bank" 和"stock" 会经常共现，而和"banana" 就很少出现在一起. 所以我们可以遍历语料，统计字典中单词两两共现次数，构成一个共现矩阵 X.

注意, 根据在什么范围内共现, 又可以分为, 在一篇文档中共现 ($|V|\times M$,M 是文档的个数) 和在一个窗口中共现 ($|V|\times|V|$).

比如现在的语料是三个句子，"I enjoy flying .","I like NLP .","I like deep learning ." 并且窗口大小为 1，那么遍历语料，获得的共现矩阵如下所示:

$$
X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array}
\begin{array}{c}
\begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\
\left[\begin{array}{cccccccc}
0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}\right]
\end{array}
$$

其实基于这个矩阵，每一行就已经可以作为这个单词的表示了，比如对于 I 可以表示为 $(0,2,1,0,0,0,0,0)^T$，但是这种表示存在几个问题: 1) 词向量的维度过大，依然是整个字典的大小.2) 表示过于的稀疏，存在大量的 0，不具备鲁棒性.3) 加入一个新词，所有的词向量都要重新计算.

解决的办法就是通过奇异值分解 (SVD) 进行降维，并转换成稠密的表示. 如下图所示:

$$
\begin{array}{c} |V| \\ \\ |V| \end{array}
\begin{array}{c} |V| \\ \left[\begin{array}{c} \\ X \\ \\ \end{array}\right] \end{array}
=
\begin{array}{c} |V| \\ \left[\begin{array}{cccc} | & | & & \\ u_1 & u_2 & \cdots & \\ | & | & & \end{array}\right] \end{array}
\begin{array}{c} |V| \\ \left[\begin{array}{ccc} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{array}\right] \end{array}
\begin{array}{c} |V| \\ \left[\begin{array}{ccc} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{array}\right] \end{array}
$$

$$
\begin{array}{c} |V| \\ \\ |V| \end{array}
\begin{array}{c} |V| \\ \left[\begin{array}{c} \\ \hat{X} \\ \\ \end{array}\right] \end{array}
=
\begin{array}{c} k \\ \left[\begin{array}{cccc} | & | & & \\ u_1 & u_2 & \cdots & \\ | & | & & \end{array}\right] \end{array}
\begin{array}{c} k \\ \left[\begin{array}{ccc} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{array}\right] \end{array}
\begin{array}{c} |V| \\ \left[\begin{array}{ccc} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{array}\right] \end{array}
$$

如上通过分解，$\hat{X} = USV^T$, 其中 $U \in \mathbb{R}^{|V| \times K}$ 就可以作为词向量矩阵. 每个单词通过一个 k 维的向量来表示.

这种方案比较简单，而且效果也很好，并且有一些改进的措施，比如: 1) 矩阵中对于 the,a,an 这些词的词频会非常高，所以可以设置一个词频的上限. 确保不会出现过大的值. 2) 根据到中心词的距离不同，每个词的贡献值也不同，靠的近的就比较大，远的就比较小.

3 Word2Vector

上面基于共现矩阵的方案可以认为是全局信息，其实我们还可以仅仅利用局部的信息，并通过迭代的方法来获得词向量. 直觉就是相近的词会有相近的上下文.

通过这个想法，我们构建一个神经网络的模型，模型的参数就是词向量，每次训练迭代的过程，就是根据目标函数，更新参数. 最终当模型训练完毕，模型的参数就是我们得到的词向量.

Word2Vector 就是根据这个思想构建的一个软件包，具体来说可以分成两个算法: 1)Continuous bag-of-words(CBOW) 根据周围的词预测中间的词. 2)Skip-gram，根据中间的词预测周围的词. 和两个训练方法: 1) 负采样法，在目标函数中引入负样本 2) 层级 softmax，通过树结构来计算目标函数 (存疑).

3.1 CBOW

CBOW 模型是根据周围的词来预测中间的词，比如现在给定周围的词为 {the cat <need to predict> over the puddle}，我们的任务就是预测出中间的词是 jump.

形式化的定义一下这个问题，设字典的大小为 $|V|$, 词向量的维度为 n.w_i 表示字典中的第 i 个单词, $x_i \in \mathbb{R}^{|V|}$ 是单词 w_i 的独热表示. v_i 表示 w_i 作为上下文单词时的词向量 (输入词向量),u_i 表示 w_i 作为中心单词时的词向量 (输出词向量), $V \in \mathbb{R}^{n\times|V|}$ 表示输入矩阵，v_i 就是 V 的第 i 行. $U \in \mathbb{R}^{|V|\times n}$ 表示输入矩阵，u_i 就是 U 的第 i 列.

整个算法具体的步骤如下，假设窗口的大小为 m

1. 提取上下文单词的独热表示 $(x_{c-m}, x_{c-m-1}, ..., x_{c-1}, x_{c+1}, ..., x_{c+m-1}, x_{c+m})$

2. 通过输入矩阵进行 embedding, $(Vx_{c-m}, Vx_{c-m-1}, ..., Vx_{c-1}, Vx_{c+1}, ..., Vx_{c+m-1}, Vx_{c+m})$, 得到 $(v_{c-m}, v_{c-m-1}, ..., v_{c-1}, v_{c+1}, ..., v_{c+m-1}, v_(c+m))$

3. 计算上下文词向量的均值，并作为整个上下文的表示 $\hat{v} = \frac{v_{c-m}+..+v_{c+m}}{2m}$

4. 计算评分向量 z,$z = U\hat{v}$, 这里用点乘计算得分, 与上下文表示相似的输出向量得分就会比较高.

5. 计算得分的 softmaxt，转换成概率 $\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$

6. 最后计算 loss，采用交叉熵，$loss = H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j log(\hat{y}_j)$，其实只需要计算中心词，因为 y 是一个 one-hot 向量.

训练可以采用随机梯度下降进行训练.

最后单词 w_i 的词向量可以是 u_i 和 u_i 的拼接或者相加.

3.2 Skip-gram

Skip-gram 和 CBOW 正好相反，这是通过中心词来预测周围的词，比如现在给定中心词为"jump"，我们的任务是需要能够预测出周围的词是 {the cat <need to predict> over the puddle}.

这里使用的数学记号与 CBOW 中定义的一样，所以整个算法的具体步骤如下:

1. 获取中心词的独热表示 $x_c \in \mathbb{R}^{|V|}$.

2. 通过输入矩阵进行 embedding，$V_c = Vc \in \mathbb{R}^n$

3. 计算评分向量，$z = Uc \in \mathbb{R}^{|V|}$

4. 计算得分的 sotfmax，转换成概率 $\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$, 这里的 $\hat{y}_{c-m}, ..., \hat{y}_{c-1}, \hat{y}_{c+1}, \hat{y}_{c+m}$ 对于上下文词，模型认为也是上下文词的概率

5. 最后计算 loss，采用交叉熵，$loss = H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j log(\hat{y}_j)$，其实只需要计算上下文词，因为 y 只有上下文词是 1（$2m$ 个），其他均为 0.

3.3 负采样

对于 skip-gram 来说，计算 $p(o|c) = \frac{exp(u_o^T v_c)}{\sum_w^{|V|} exp(u_w^T v_c)}$，可见每次计算都是在整个字典空间 $|V|$ 上进行的，所以非常耗费时间.

实际上真正属于上下文单词的仅仅有几个，大量的均不是上下文单词，所以我们只需要采样一部分负样本即可.

并且改造一下我们的目标函数，不再使用交叉熵，我们采用最大似然的思想.

首先对于一对词 (w, c)，我们定义 $P(D = 1|w, c, \theta) = \sigma(u_w v_c)$ 为 w 是 c 上下文单词的概率，所以 $P(D = 0|w, c, \theta) = 1 - P(D = 1|w, c, \theta)$.

根据我们假设每个单词都是独立分别的，根据最大似然的思想就是让整体

概率最大, 所以:

$$\begin{aligned}
\theta &= \arg\max_{\theta} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \hat{D}} P(D=0|w,c,\theta) \\
&= \arg\max_{\theta} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \hat{D}} (1-P(D=1|w,c,\theta) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} logP(D=1|w,c,\theta) + \sum_{(w,c)\in \hat{D}} log(1-P(D=1|w,c,\theta)) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} log\frac{1}{1+exp(-u_w^T v_c)} + \sum_{(w,c)\in \hat{D}} log\frac{1}{1+exp(u_w^T v_c)} \\
&= \arg\min_{\theta} - \sum_{(w,c)\in D} log\frac{1}{1+exp(-u_w^T v_c)} - \sum_{(w,c)\in \hat{D}} log\frac{1}{1+exp(u_w^T v_c)}
\end{aligned} \tag{2}$$

所以对于 Sikp-gram 模型来说，新的目标函数就是

$$J = -\sum_{j=0,j\neq m}^{2m} log\sigma(v_c \cdot u_{c-m+j}) - \sum_{k=1}^{K} log\sigma(-v_c \cdot u_k)$$

对于 CBOW 模型来说，新的目标函数就是

$$J = -log\sigma(\hat{v} \cdot u_c) - \sum_{k=1}^{K} log\sigma(-\hat{v} \cdot u_k)$$

这就是负采样，简单的说，就是增加正样本的概率，减少负样本的概率. 通过负采样可以提高模型训练的效率.

最后一个问题，怎么进行负采样?

定义 f_w 是单词 w 在语料中的频数 (或者频率，本质一样)，那么这个单词被采样的概率是 $\frac{f(w)^{0.75}}{\sum_{u\in V} f(u)^{0.75}}$

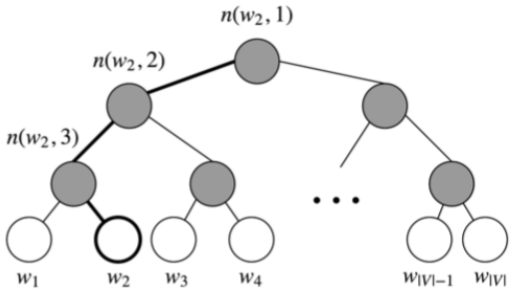
没有 0.75 次幂就是普通的一元模型分布，加上 0.75 的好处是，让低频词的采样概率提高.

3.4 层级 softmax

还有一种优化的方案是层级的 softmax，对于最原始的方案，计算 $P(o|c)$ 的时间复杂度是 $O(|V|)$，层级 softmax，是一种树的结构. 字典中的每一个单词，都是树的叶子结点，定义 $L(w)$ 是从根节点到单词 w 节点上的节点数.

定义 $P(w|w_i)$ 如下:

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma([n(w,j+1) = ch(n(w,j))] \cdot v_{n(w,j)}^T v_{w_i})$$



$$[x] = \begin{cases} 1 \text{ if } x \text{ is true} \\ -1 \text{ otherwise} \end{cases}$$

.

这样就只需要计算路径上的节点，而不是整个字典，把时间复杂度降低到了 $O(logN)$

所以在这里就不存在输出矩阵 U 了，优化的方案就是让 $P(w|w_i)$ 最大，也就是让 $-logP(w|w_i)$ 最小. 这里构造树的方案是通过哈夫曼树进行构造.

经验表明,Hierarchical Softmax 对低频词效果较好；Negative Sampling 对高频词效果较好，向量维度较低时效果更好.

`http://qiancy.com/2016/08/17/word2vec-hierarchical-softmax/`

4 Glove

5 评价指标