词向量编程作业
汉语子词向量

分别基于 SVD 分解和基于 SGNS 两种方法构建汉语子词向量并评测。

1 SVD 实现

1.1 SVD 分解原理

Word2Vec 的思想即，让邻近的词的向量表示相似。使用词语共现性来构建词向量，可以达到这样的目的。对词频进行统计，即得到了共现矩阵。使用一个固定大小的窗口，统计中心词附近的词的频率，记录在矩阵 [中心词, 周围词] 内。

由此得到的 one-hot 编码矩阵存在稀疏问题，由此通过 SVD 矩阵分解实现降维。将巨大的共现矩阵进行 SVD 分解后，选取最重要的几个特征值，得到每个词的低维表示。

1.2 SVD 实现流程

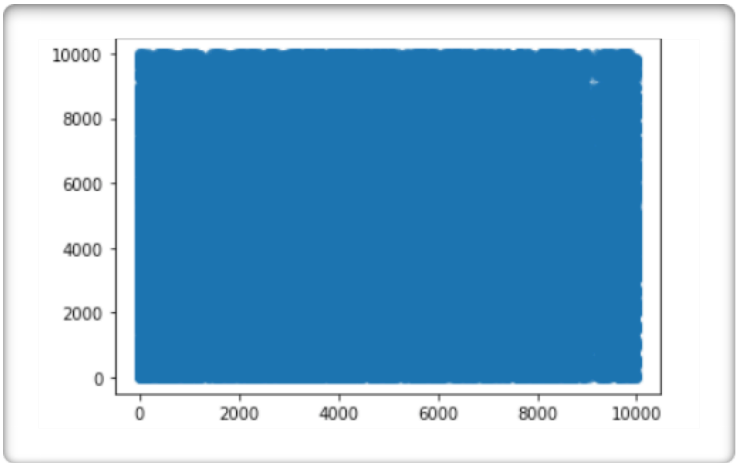
导入语料库，构建句子；对句子进行预处理，包括除去停用词和中文分词；对句子分词后，从词汇中取出 10000 个高频词，生成词表；对应词表构建矩阵；之后基于 SVD 分解，遍历所有句子，通过窗口 K = 5 滑动得到词频统计，完善词频矩阵的计数；基于 SVD 降维，获得子词向量 vec_sta；

经过检验，发现其零奇异值在 4500 取到，本实验选取了 3000 个奇异值，将维度减小到 3000；经过计算，选取的奇异值和为 71303.7，奇异值总和为 72804.7，二者的比值为 0.98；通过理论公式得到词空间向量为矩阵U 和 sigma 相乘，基于

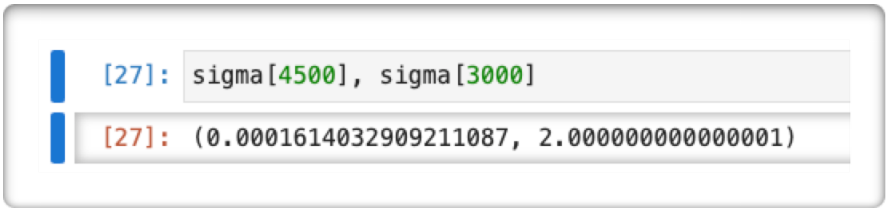
该向量计算 test.txt 中的同一行的两个子词的余弦相似度 sim_svd，并插入到文件行尾。更多详见 jupyter 文件，已经按顺序加入文字描述。

1.3 实验结果

下图是对词表词频进行统计得到的散点图，证明其可靠性（包含测试用词）。



下图是 simga 对角矩阵的计算结果截图。具体实验结果见 test.txt 结尾的第一列。



2 SGNS 实现

2.1 SGNS 原理

虽然独热向量很容易构建，但它们不是一个好的选择，主要原因是独热向量不能准确表达不同词之间的相似度，比如余弦相似度。

$$\frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \in [-1,1].$$

由于任意两个不同词的独热向量之间的余弦相似度为0，所以独热向量不能体现词之间的相似性。

Word2vec 将每个词映射到一个固定长度的向量，这些向量能更好地表达不同词之间的相似性和类比关系。Word2vec 工具包含跳元模型（skip-gram），其训练依赖于条件概率（跳元模型考虑了在给定中心词的情况下生成周围上下文词的条件概率），属于自监督模型。

$$P(w_o \mid w_c) = \frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)},$$

跳元模型的主要思想是使用 softmax 运算来计算基于给定的中心词生成上下文字的条件概率。

然而，连续词袋模型的梯度计算包含求和。为了降低上述计算复杂度，实验采用近似训练方法，即负采样。负采样添加从预定义分布中采样的负样本。通过修正后的如下近似条件概率，可以化简训练过程的遍历长度。

$$P(w^{(t+j)} \mid w^{(t)}) = P(D = 1 \mid w^{(t)}, w^{(t+j)}) \prod_{k=1,\ w_k \sim P(w)}^{K} P(D = 0 \mid w^{(t)}, w_k)\,.$$

最后取出预先定义的 embedding 网络的层，即为所求的词向量。

2.2 SGNS 实现流程

中文预处理部分与 SVD 相同，在此不再赘述；之后一步将句子根据词表 token 取值替代，换成 one-hot 形式便于生成训练数据；构造数据生成器 DataLoader，包括中心词和上下文词的提取、定义负采样取噪声词、定义 pytorch 批处理函数、构造数据集；

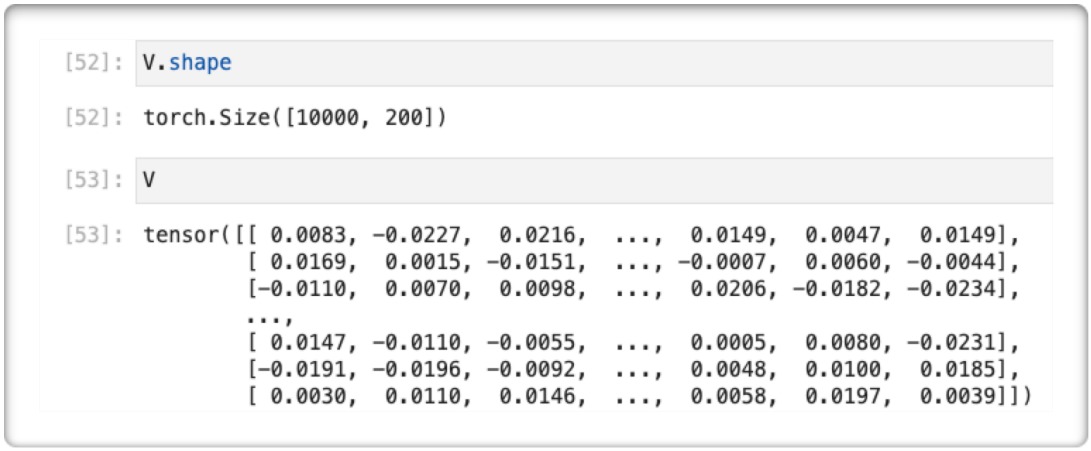
再一步，按要求设定训练批次大小 256，窗口 K 值为 5，负采样个数为 3，实例化数据迭代器；定义前向传播、定义二元交叉熵损失函数；

模型初始化，模型为两个词表长度的嵌入层，将词表中的所有单词分别作为中心词和上下文词使用。字向量维度 embed_size 设为 200；

训练函数定义，并训练；调参学习率为 0.002，训练轮数为 5。得到结果。更多详见 jupyter 文件，已经按顺序加入文字描述。

2.3 实验结果

下图是模型层，也即词向量的结果截图。具体实验结果见 test.txt 结尾的第二列。



```
[52]: V.shape

[52]: torch.Size([10000, 200])

[53]: V

[53]: tensor([[ 0.0083, -0.0227,  0.0216,  ...,  0.0149,  0.0047,  0.0149],
              [ 0.0169,  0.0015, -0.0151,  ..., -0.0007,  0.0060, -0.0044],
              [-0.0110,  0.0070,  0.0098,  ...,  0.0206, -0.0182, -0.0234],
              ...,
              [ 0.0147, -0.0110, -0.0055,  ...,  0.0005,  0.0080, -0.0231],
              [-0.0191, -0.0196, -0.0092,  ...,  0.0048,  0.0100,  0.0185],
              [ 0.0030,  0.0110,  0.0146,  ...,  0.0058,  0.0197,  0.0039]])
```