中文词向量Word2Vec

胡昕韵 2019211420

2019211315班

总结: 使用python编程实现了两种实现中文词向量编码的方法：SVD分解，使用Negative Sampling的Skip-Gram（SGNS）方法。编码实现SGNS时采样了无框架手写，与使用Pytorch框架两种方法。均成功实现。

---

1 SVD词向量

1.1 设计思路

SVD程序流程较简单，首先进行数据预处理，再进行词频统计，最后进行SVD分解计算即可。

1.1.1 数据预处理

本次编程所用数据以使用BPE方法下得到的词表（未排除标点）进行了分词，内容例如：“人民网 2010年8月10日电： 记者 xxx ...”，每个词之间使用了空格隔开。由于词表统计时将连续的数字视为了同一个符号，数据预处理时也应进行相应的替换。

预处理的同时，从文本中获取词表，最终得到类型为`list(list(str))`的数据列表以及类型为`dict{str, int}`的按词频排序的词表。

1.1.2 词频统计

使用K=5的窗口进行词频统计。由于词频具有对称性，对于一个词，仅统计在它之后出现的词即可在转置相加后得到全部词频。

1.1.3 SVD分解

一个d*d的对称矩阵M可进行SVD分解：

$$M = U * S * V$$

其中S为对角矩阵，且元素按从大到小的顺序进行排列。取U的前i列，S的前i个元素，计算$U_{:i} * S_{:i}$即可得到降维后的词向量

1.2 算法详述

1.2.1 统计词频

若记词表长度为len，则伪代码如下

```
def get_frequency(data):
    初始化array为（len,len）大小的全0矩阵
    for sent in data:
        for window in sliding_window(K, (sent.strip()+ end_padding_token* end_padding_size)
                                     .split(' ')):
            if window[0] 在词表内:
                w_c_id = word_vocab[window[0]] # 第一个词的id
                for word in window[1:]:
                    if word in word_vocab: # 第一个词后面的词在词表内
                        array[w_c_id][word_vocab[word]] += 1
    array += array.transpose()
    return array
```

例如，若某句子为`['a'],['b'],['c'],['d'],['e']`，则首先在句子尾部先加上长为窗口长减2的padding，变为abcde<p><p><p>，滑动窗口从abcde至de<p><p><p>，对于窗口abcde，记它们分别对应的词表内的编号为ABCDE，则分别在M[A][B], M[A][C],..., M[A][E]处加1

统计完成后自身转置相加即可得词频统计结果

1.2.2 SVD分解

由于原词频矩阵较大，若采样`numpy`提供的svd分解函数在CPU上计算，耗时较长，故使用pytorch，选择`Numpy`矩阵转为张量`Tensor`在GPU上使用`torch.svd()`函数进行SVD分解。

SVD分解及降维核心代码如下：

```
def reduce_to_k_dim(M, k=2):
    u, sigma, v = torch.svd(M)
    u = u.to("cpu").numpy()
    sigma = sigma.to("cpu").numpy()
    notZero = len(sigma[sigma > 1e-5])
    sig_sum = np.sum(sigma)
    chosen_sum = np.sum(sigma[:k])
    print(sigma[:k])
    return np.matmul(u[:, :k], np.diag(sigma[:k]))
```

该过程在一块显存大小为40G的A100 GPU上运行耗时约3分钟。

1.3 模型参数与运行结果

1.3.1 参数

选取窗口大小`K=5`，降维后的维度设定为`dim=300`

1.3.2 运行结果

SVD分解后，共有16869个非零奇异值（认为小于10^{-5}的奇异值为0），奇异值总和为2 636 794.5（$2.64 * 10^6$）

选取**300**个奇异值的和为773 311.375（$7.7 * 10^5$），占总和的**29.3277%**

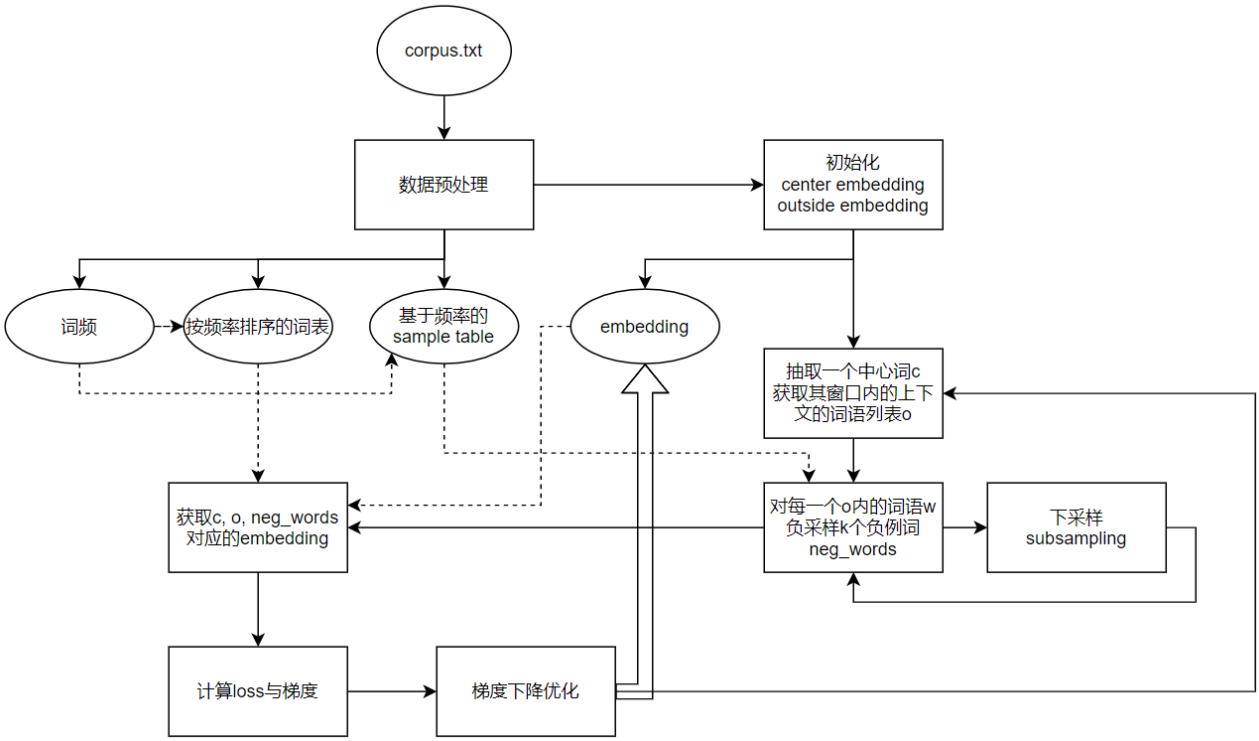
1.4 缺陷与改进

- 使用BPE分词后的词表进行分词时可能出现不符合不符合预期的情况。例如“一方面”一词虽然在词表中出现，但在语料中常出现为“另一方面”，而“另一”是一个更加高频的词语，导致分词结果均为“另一 方面”，使“一方面”一词在语料中消失。若使用更好的分词算法可能提高词向量的表现。
- 可能由于选取的词向量维度过小，无法体现全部特征或所选向量过大，对语料存在过拟合。应当结合语料实际大小与词表大小进行选择。

2 SGNS词向量

2.1 设计思路

无论是否使用pytorch框架实现，实现基于Skip-Gram与Negative Sampling的词向量表示均遵循以下流程。



以上流程图中方框代表一个过程，椭圆代表一个数据结构。

2.2 算法详述

2.2.1 无框架版本

2.2.1.1 负采样表构建

首先初始化一个足够大的列表（约等于语料的总词数）用于采样，记为`sample_table`

为减少高频词对词向量构建的负面影响，采用采样到词语w的概率为

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f(w_j)^{\frac{3}{4}}}$$

若有统计出的词频`freq`，则构建采样表的伪代码如下：

```
freq = freq ** 0.75
freq = np.cumsum(freq) * table_size
j = 0
for i in range(table_size):
    while i > freq_mat[j]:
        j += 1
        sample_table[i] = j
```

这样构建出一个采样表，在需要进行负采样时，只需要取一个随机整数i，访问`sample_table[i]`即可获取此次采样所得的词的编号

2.2.1.2 下采样

为进一步减少高频词对词向量构建的负面影响，对高频词进行概率丢弃，以以下概率丢弃高频词：

$$P_{discard}(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

其中超参数t我的取值为10^{-4}，伪代码如下

```
# 采样1个样本(下采样subsampling)
def subsampling(self):
    idx = sample_table[random.randint(0, len(self.sample_table) - 1)]
    p_discard = 1 - np.sqrt((1e-4 / frequency[idx]))
    if random.uniform(0, 1) < p_discard:
        return self.subsampling()
    return idx
```

如此采样，频率高于10^{-4}的词语均以一定的概率被丢弃，且频率越高，丢弃的概率越高。

2.2.1.3 Word Embedding

采用两个`numpy`矩阵存储词向量，一个矩阵是词作为某窗口中心词时的embedding，另一个矩阵是词作为某窗口的非中心词时的embedding，最终优化完毕后，采用两个矩阵的均值作为词向量的最终结果

若词表大小为l，设定的词向量维度为d，则将词向量初始化为在[-0.5/d, 0.5/d]范围内均匀分布的大小为l*d的随机数矩阵。

2.2.1.4 梯度计算

若记中心词为c，外词为w，负采样所得词为n，中心词的embedding矩阵为V，非中心词的embedding矩阵为U，则损失函数如下：

$$Loss = -\log\sigma(U_w \cdot V_c) - \sum_{k \in n} \log\sigma(-U_k \cdot V_c)$$

则梯度如下：

$$\begin{aligned}
\frac{\partial L}{\partial V_c} &= -U_w * (1 - \sigma(U_w \cdot V_c)) - \sum_{k \in n} U_k * [1 - \sigma(-U_k \cdot V_c)] \\
\frac{\partial L}{\partial U_w} &= -V_c * (1 - \sigma(U_w \cdot V_c)) \\
\frac{\partial L}{\partial U_k} &= V_c * (1 - \sigma(U_k \cdot V_c)) \quad for\ k \in n
\end{aligned}$$

代码实现见`SGNS.py`的`Word2Vec.negSamplingLossAndGradient()`，已详细注释

2.2.1.5 优化策略

采用SGD方法，每次迭代抽取`batch_size`个中心词，分别对每个batch下进行负采样与损失函数及梯度的计算，并将其累加得到一个batch的总loss与梯度，再使用梯度下降法优化两个词向量矩阵。对应代码实现位于`Word2Vec.sgd()`

2.2.2 有框架版本

Word Embedding的初始化策略与无框架相同，Loss计算也相同（Pytorch自动计算梯度，故梯度计算可忽略）

2.2.2.1 负采样

负采样采用的概率与无框架版本相同，但实现方法进行了简化。不再构建采用表，而是统计词频的0.75次方，组织成一个Tensor后，使用`torch.multinomial()`函数进行采样

```
neg_id = torch.multinomial(sample_freq, args.neg_sample_num * size, True)
```

统计词频与词表的工作在`init_vocab_and_sampling()`函数中完成

2.2.2.2 优化过程

每个batch随机选取了`batch_size`个中心词后，由于中心词可能位于行首或行末，导致窗口内其他词的总和可能小于batch_size*window*2，为后面的loss计算带来烦恼。为简化计算，记一个batch所采得的所有外词数量为size，并扩展中心词的长度。例如，若batch_size=3，采样所得的中心词编号分别为1, 2, 3， context的长度分别为[4, 2, 4]，则将中心词编号扩展为[1,1,1,1, 2,2, 3,3,3,3]，与context词一一对应，方便计算loss。

相关代码见`make_one_batch()`函数

计算出Loss后，使用Adam Optimizer进行优化。该方法可靠、有效，且不用手动调整学习率就能达到较好的优化效果。

2.3 模型参数

（最终提交的文件由2.3.2中的参数生成）

两个版本中，均取上下文窗口为2，即总窗口大小为5；对于一个词，负采样的个数设为3

2.3.1 无框架版本

初始词向量采用均匀分布的初始化，取值范围为$[-\frac{1}{batch_size}, \frac{1}{batch_size}]$

由于该版本运行速度较慢，词向量维度选取为35维，SGD的批次设为48，学习率设为0.1，训练了2000轮（由于运行较慢，虽然loss仍在继续下降，但没有尝试更多轮次）

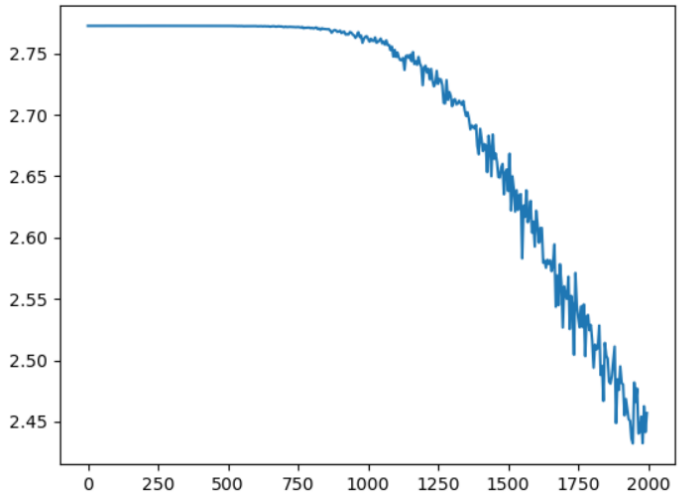
2.3.2 有框架版本

初始词向量采用均匀分布的初始化，取值范围为$[-\frac{1}{batch_size}, \frac{1}{batch_size}]$

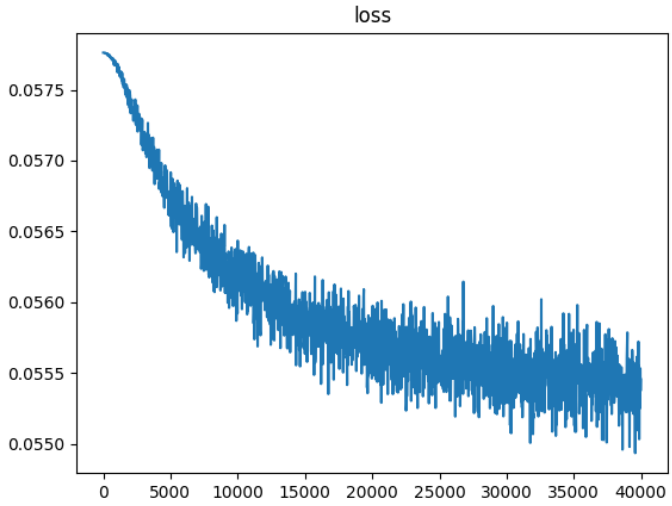
词向量维度取100维，由于使用Adam方法优化，无需调整学习率。以训练批次为48的batch size训练了40 000 轮。

2.4 运行结果

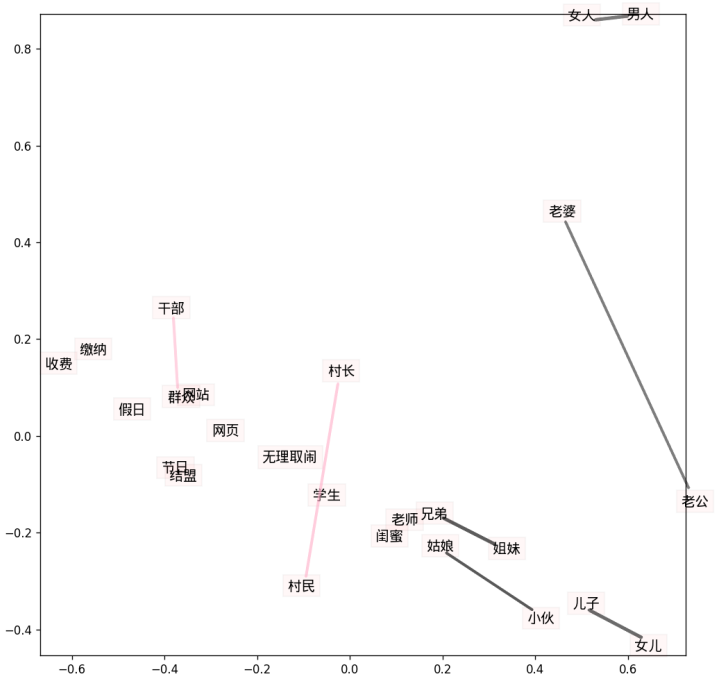
由于无框架版本在CPU上运行，运行速度较慢，单纯的SGD方法也难以调至合适的学习率。仅运行了2000个epoch以观察方法确实正确有效。2000个epoch训练的loss变化图像如下：（该Loss计算方式与下图loss有所区别，该loss为加和，而下图loss为一个batch的均值，不可比较）



使用Pytorch框架编写的版本，随epoch增加，loss的变化图像如下：



训练结束后，为了可视化词向量的效果，采用了TruancatedSVD降维进行主成分分析，绘制了部分词在向量空间中的分布，如下图：



由上图可见：词向量生成较为成功，词向量可以一定程度上表现词语的相似度以及一些词汇间隐形的关联，例如图中标出的黑色线段链接的词语，为男女的相同角色的称谓，它们的词向量相对位置基本相似。（若希望生成其他词的位置，请更改`words_for_visualization.txt`文件内容并重新运行程序。在一张40G的A100的GPU上进行训练，不进行下采样以batch size=48训练40 000个epoch需要约2G显存，5分钟左右的时间）

2.5 缺陷与改进

- 语料分词不够完善，可能导致SGNS得到的词向量不够准确，并受到许多干扰。
- 语料不够丰富数据量较少，部分参与评测的词汇在语料中出现次数很少，影响了词向量的生成效果。