9 Global Vectors的实现

Glov的实现相比Skip-Gram要容易的多，主要是因为Glov没有使用大规模的负采样因而预处理非常方便。但模型训练前的统计过程十分痛苦，原文作者给出的目标函数也很危险，这一章将详细的介绍Glov的实现，你可以在https://github.com/Y-WJ/NLP-PLAYER/tree/master/Glovec获取代码以及相关文档。你也可以在同一个仓库中找到往期的实现和文档，如果这些工作使你受益，请务必恩赐一枚star。

9.1 文本的预处理

这里直接使用了Skip-Gram实现中的init-dictionary，关于这种处理方式你可以从第七章的第一小节获得详细的信息。

9.2 对全文做统计

为了在一次全文遍历中得到上一章介绍过的统计矩阵X,这里用指针i遍历文本的同时使用指针j指向i的后文，对所有遍历到的关联词对$i\,j$都会让X_{ij}和X_{ji}自增1。但矩阵X本身就是危险的，一个简单的估算可已说明这一点，如果矩阵使用32位的int，而embedding矩阵是同Skip-Gram相同的5W维，那么这个矩阵将占用：

$$\frac{50000\times 50000\times 32bit}{8bit/byte\times 1024^3byte/GB}\approx 9.32GB$$

但由于numpy的莫名其妙的存储方式，实际上并没有这么夸张的空间占用。注意到这个矩阵是对称阵而且非常稀疏，numpy可能在存储稀疏矩阵和处理较小的矩阵元素的数据类型上有些优化，以至于我能在PC上不采取任何措施地处理这么大的矩阵，但numpy和scipy貌似都没有提供构造稀疏矩阵的方法。如果你在预处理阶段收到内存错误的警告，你应该设置更小的matrix-size或者自己设计替代numpy矩阵的数据结构来节省空间。

9.3 训练样本的生成

Glov直接使用随机批量梯度下降，但直接在矩阵X上取样是不明智的，因为X是稀疏的，0元素会进入取样窗口而使得每批样本真正可用于优化的样本数量不等，而且计算$ln0$会引发危险，因此在使用X前剔除其中所有的0元素而产生紧密的矩阵是较好的策略。

算法中使用列表list-X而非矩阵来存储剔除0元素后的矩阵X,list-X中的每一项形如$[i,j,X_{ij}]$,前两个值确定了矩阵X中的一个坐标，第三个值确定了矩阵X在这个坐标位置的值。但列表list-X并非由X生成，而是在统计全文的过程中与X同步生成。X的所有元素被初始化为0，对遍历全文中的每一对$i\,j$，首先查找X_{ij}是否为0，若是说明$i\,j$和$j\,i$都不在list-X中出现，于是给list-X添上两项$[i,j,0]$，$[j,i,0]$。全文统计完成后，再次遍历list-X，对每一项在X中找到对应的频数，填入这一项的第三个值。在这个过程中，矩阵X的作用相当于一个巨大的哈希表。

在完成上面的过程后list-X实际上已经包含了学习器所需的全部信息，取样的策略是在整个list-X上随机地取batch-size个样本，得到形如$(batch-size,3)$的batch矩阵，batch的三列被分别装入三个向量w_i,w_j,X_ij以供填充学习器的占位符。

9.4 更好的损失函数

为了达到目标：

$$w_i^T w_j + b_i + b_j = \ln X_{ij}$$

原文中作者使用等式两边差值的平方作为优化目标：

$$J = f(\left(\frac{X_{ij}}{x_{\max}}\right)^a)(w_i^T w_j + b_i + b_j - \ln X_{ij})^2$$

这个损失的问题是越靠近最优点，损失的梯度越小，优化速度越慢，下面的计算可以说明这些：

$$\frac{\partial J}{\partial w_i} = f(x) \cdot 2(w_i^T w_j + b_i + b_j - \ln X_{ij}) \cdot w_j$$

考虑对同一组关联词的优化，越接近最优点，上式右部的中间一项越接近0，而其它两项都是常值，这个梯度也会趋于0值，不论怎么调整学习率，优化这个目标一定会在损失达到某一数量级后停止下降。

另一个不突出但仍会时不时出现的问题是平方项有可能引发上溢。如果$i\ j$这一对关联词第一次参与优化，而且在这之前$w_i\ w_j$在同其它词配对的优化中被迭代的很大，更不巧的是$i\ j$没有什么联系以至X_{ij}是个接近0的值，求$(w_i^T w_j)^2$很可能引发上溢。当然如果某一对词在第一次参与优化时安全通过了，那么在以后的优化中这一对词也肯定比不会引发溢出了，因为优化总是向着使平方项内的值更小的方向进行。

算法中实际使用的损失函数是：

$$J = f(\left(\frac{X_{ij}}{x_{\max}}\right)^a)|w_i^T w_j + b_i + b_j - \ln X_{ij}|$$

用绝对值代替平方的好处是对同一关联词对，在非零点的梯度总是跟学习率相关的固定值，而不会随着优化过程越来越小：

$$\frac{\partial J}{\partial w_i} = \pm f(x) \cdot w_j$$

至于在零点不可导的问题同Skip-Gram中的情况相似，由于学习率是一个有精度的值，最终的优化结果会取决于学习率的数量级在零点两边震荡而不可能刚好优化到零点。

溢出的问题也能由此解决，只要在统计过程中X_ij没有上溢，后面的优化是一定不会产生上溢的。

9.5 损失的实现

每次训练样本生成得到了中心词w_i,上下文词w_j,统计频数X_ij,计算损失的过程如下：

1. 分别从embeddings-i,embeddings-j中选择对应的词向量填充w_i, w_j，以及从偏置向量中选择对应的偏置填充b_i,b_j:

$$w\_i = \begin{pmatrix} w_i^1 \\ w_i^2 \\ \vdots \\ w_i^{batch\_size} \end{pmatrix} \Rightarrow V\_i = \begin{pmatrix} V_i^1 \\ V_i^2 \\ \vdots \\ V_i^{batch\_size} \end{pmatrix}$$

$$w\_j = \begin{pmatrix} w_j^1 \\ w_j^2 \\ \vdots \\ w_j^{batch\_size} \end{pmatrix} \Rightarrow V\_j = \begin{pmatrix} V_j^1 \\ V_j^2 \\ \vdots \\ V_j^{batch\_size} \end{pmatrix}$$

$$\sum_{ij} w_i^T \cdot w_j = reduce\_sum(V\_i \otimes V\_j) = \begin{pmatrix} (V_i^1)^T \cdot V_j^1 \\ (V_i^2)^T \cdot V_j^2 \\ \vdots \\ (V_i^{batch\_size})^T \cdot V_j^{batch\_size} \end{pmatrix}$$

矩阵运算符⊗的意义是：

$$\otimes : (x, y) \cdot (x, y) \to (x, y)$$

2. 求绝对值项：

$$\sum_{ij} |w_i^T \cdot w_j + b_i + b_j - \ln X_{ij}| = abs(reduce\_sum(V\_i \otimes V\_j) + b\_i + b\_j - \ln(X\_ij))$$

$$= \begin{pmatrix} |(V_i^1)^T \cdot V_j^1 + b_i^1 + b_j^1 - \ln X_{ij}^1| \\ |(V_i^2)^T \cdot V_j^2 + b_i^2 + b_j^2 - \ln X_{ij}^2| \\ \cdots \\ |(V_i^{batch\_size})^T \cdot V_j^{batch\_size} + b_i^{batch\_size} + b_j^{batch\_size} - \ln X_{ij}^{batch\_size}| \end{pmatrix}$$

3. 添加衰减权重：

$$f(x) = \left(\left(\frac{X\_ij}{x_{\max}}\right)^a\right)^T = \begin{pmatrix} \left(\frac{X_{ij}^1}{x_{\max}}\right)^a & \left(\frac{X_{ij}^2}{x_{\max}}\right)^a & \cdots & \left(\frac{X_{ij}^{batch\_size}}{x_{\max}}\right)^a \end{pmatrix}$$

$$LOSS = \left(\left(\frac{X\_ij}{x_{\max}}\right)^a\right)^T \cdot abs(reduce\_sum(V\_i \otimes V\_j) + b\_i + b\_j - \ln(X\_ij))$$

9.6 损失估计与参数调整策略

1. 估算初始损失。在生成list-X的过程中算法同时统计了所有出现过的关联词对的次数，以及list—X的总长度，并据此估算出list-X中的每一项的第三个值的均值，也就是X_ij的均值。在使用同SG相同的整个训练集的情况下，这个值在7左右。平方项中的其它值均以0为均值初始化，据此可以计算出初始化后的理论估计损失：

$$LOSS\_AFTER\_INIT = \left(\frac{7}{x_{\max}}\right)^a \ln 7 \approx 0.2648$$

但在实际运行中，这个值保持在0.34左右，这应该同Skip-Gram中的情况一样归功于tensorflow的伪随机数发生器。在使用算法提供的默认参数运行时，平均损失应该降至0.2648以下才能认为学习器发挥了效果（比全0矩阵强），这个值不是固定的，这与SG中的情况不同，根据数据集和参数的不同，这个值需要被重新计算，这个算法没有计算这个参考值。

实际上由于绝对值函数的梯度优势，这个算法能很快将损失压至学习率的数量级。如果不停的更换更小的学习率继续训练，理论上损失是可以被压到任意小的数量级的。

2. 增加context-window会大幅增长list-X而且过大的上下文窗口会显著降低学习质量，这是这一类模型的通病。直观地看，上下文窗口能包括中心词所在的句子应该是不错的，算法中这个值同Skip-Gram一样被设置为6。

3. 训练轮数，批次数量，list-X长度，这三个参数互相关联。首先是取样的覆盖率，随机批量取样会导致一些样本在整个训练中从未被更新，再调整参数时你应当考虑下面的式子的值停留在尽可能小的数量级：

$$p = (1 - \frac{batch\_size}{length\_list\_X})^{train\_step\_num}$$

假设list-X中的样本被等概的选取，下面的式子可以估算每对关联词被优化的次数：

$$n = \frac{batch\_size \times train\_step\_num}{length\_list\_X}$$

4. 原文中提到的两个用于压制低频样本的玄学超参数直接使用了建议值，实际上我想不出任何关于如何调节这两个超参的理论依据。

5. 这个算法在统计全文阶段开销巨大，但由于没有使用大规模的负采样，训练速度比Skip-Gram快得多。如果Skip-gram使用总和为64个的正负采样，在矩阵计算的阶段的计算量将是Glov的64倍。总的来说，主要的运算量集中在训练阶段，因此Glov的这种实现要比第七章的Skip-Gram实现快不少。增加NCE负采样数量带来的开销是显而易见的，这样做的收益还暂时无法的得到验证。词向量的训练过程是无监督的，不存在一个标准的测试集可供对这些词向量打分，检验一份词向量最好的办法是将它们运用在下游的任务中，根据下游任务的标准来评估词向量。但我现在还没有能力完成这些下游任务，因此这里我没有办法比较Word2vec，Skip-Gram,Glov这三种实现的实际训练效果。

6. 算法的其它组件同Skip-Gram的实现中相同。

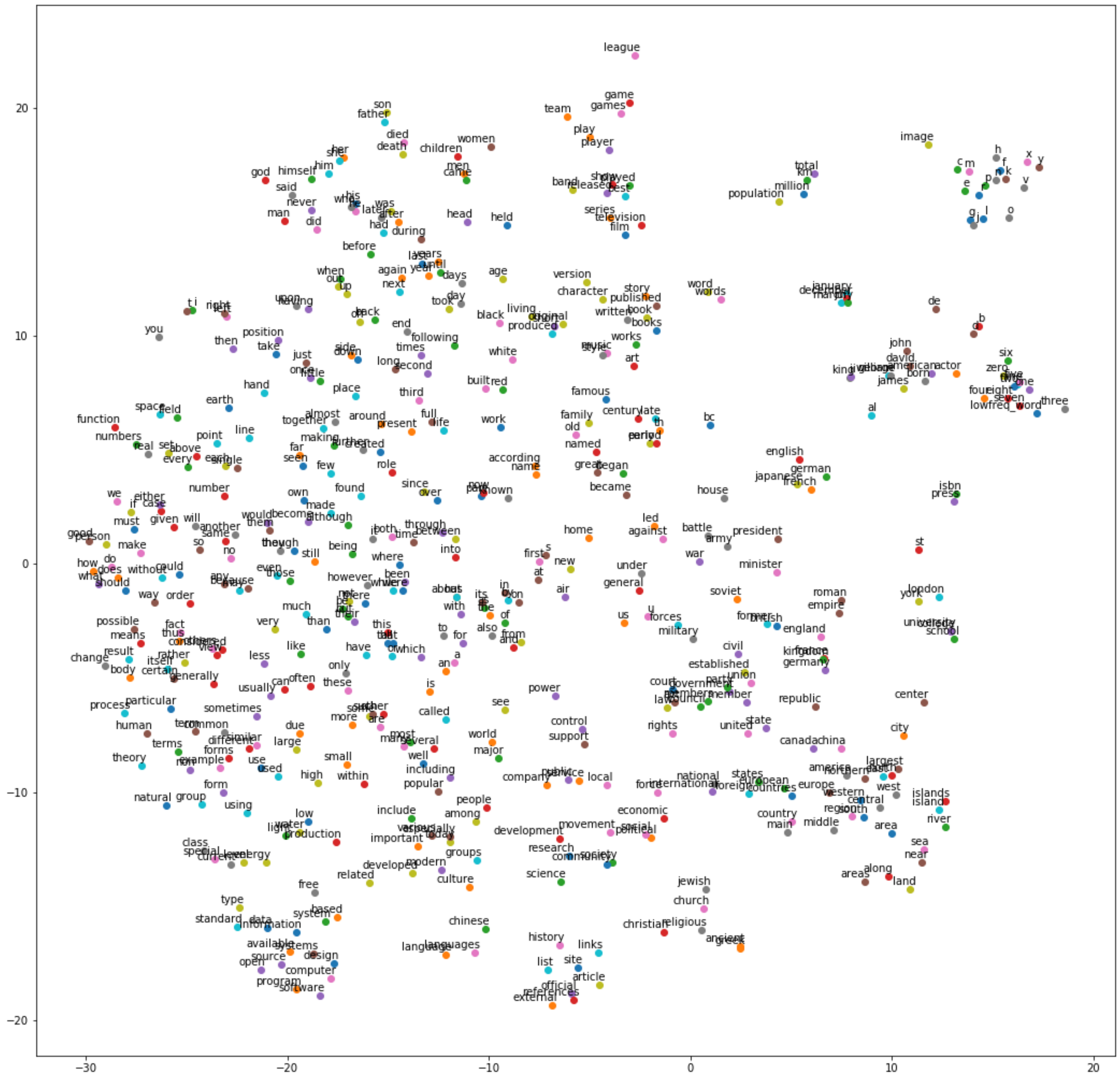
20
10
0
−10
−20
−30
−20
−10
0
10
20
league
game
games
team
play
player
son
father
died
death
women
children
god
himself
said
man
never
did
head
held
population
million
image
television
film
series
before
during
when
again
next
age
version
character
story
published
book
books
writer
works
art
word
words
de
john
david
james
king
american
actor
born
six
zero
three
four
seven
eight
al
you
then
position
take
just
hand
earth
function
numbers
space
field
point
line
almost
together
around
present
full
life
work
family
old
named
famous
english
german
japanese
french
house
isbn
press
we
either
number
own
found
since
over
known
became
began
battle
army
president
league
good
person
make
how
what
should
way
order
possible
fact
means
result
change
body
process
particular
human
terms
theory
natural
group
using
form
class
special
energy
free
type
standard
data
information
system
based
available
systems
design
source
open
computer
program
software
very
less
like
have
only
these
usually
often
see
power
called
control
support
world
major
well
including
popular
small
high
within
people
among
include
important
development
economic
research
community
society
science
groups
modern
culture
related
chinese
history
links
list
site
article
official
references
external
languages
jewish
church
religious
christian
ancient
greek
sea
near
along
land
areas
area
river
islands
island
region
south
west
central
europe
country
middle
main
city
center
republic
state
united
rights
established
civil
union
military
forces
us
england
france
germany
london
york
university
school
college
roman
empire
soviet
minister
war
general
under
against
led
home
new
first
air
public
local
company
international
national
foreign
countries
states
european

图 6: 100W轮