**学习周报**

1. 学习内容与进度
2. 理解并总结HeteGNN的论文源码思路

原论文主要通过author-paper、paper-paper、paper-venue三种图组成的异质图来获取三类强有力的实体嵌入，并将这些嵌入用于后续的节点分类和链接预测任务当中。

|  |  |
| --- | --- |
| HetGNN.py | 论文的核心模块  【搭建数据生成器】：  首先利用args参数获取模型需要的数据生成器，包含两类数据：  一类是三类实体与属性相关的嵌入表示，即特征数据。主要包括paper的属性嵌入（预训练嵌入、摘要、标题、对应的venue、对应的所有author）、author的属性嵌入（预训练嵌入、发表的3篇paper）、venue的属性嵌入（预训练嵌入、对应的5篇paper）；  二类是模型构建需要的其他数据。 主要包括用于训练集的author、paper、venue的节点编号列表，从随机游走数据中获取的每类结点当中每个结点所对应的10个paper节点邻居、  10个author节点邻居、3个venue节点邻居列表。  【创建模型】：  利用获取的两类数据创建heteGNN模型  【模型训练】：  共进行n次迭代，每次迭代的过程为：   1. 首先从随机游走数据当中采样9种类型的三元组数据，包括 (a,a,a),(a,p,p),(a,v,v),(p,a,a), (p,p,p), (p,v,v), (v,a,a), (v,p,p),(v,v,v)； 2. 以9类三元组样本中的最短样本长度为基准，结合batch\_size来得到进行批处理的次数，每进行一次批处理，模型就训练一次；每一次批处理就是分9次从每类中单独抽取batch\_size个三元组并将这类的三元组列表馈入到HetGNN模型当中，就能得到这类三元组中batch\_size个中心节点的嵌入矩阵、batch\_size个正样本节点的嵌入矩阵、batch\_size个负样本节点的嵌入矩阵，一次批处理结束就能得到中心节点、正样本和负样本节点的（9，batch\_size, embed\_dim)嵌入矩阵。 3. 将9类结果进行堆叠，分别得到中心结点、正样本和负样本结点的（9 \* batch\_size, 1，embed\_dim)、   （9 \* batch\_size, embed\_dim，1）、（9 \* batch\_size, embed\_dim，1）的嵌入矩阵，最后求解交叉熵损失 |
| data\_generator.py | 进行数据的相关处理，提供数据服务。  除了主模块中的两类数据，还包括进行随机游走、计算随机游走采样率和采样三元组的功能。  【基于重启的随机游走】  即het\_random\_walk函数，其目的是为了获取每类节点所对应的每个节点的长为100的随机游走序列，每个游走序列分别包含45个paper类节点、45个author类节点、10个venue类节点。  其具体过程根据节点类别有细微的差异。以author节点为例说明：如果此author节点有paper节点邻居，则可以对其进行游走，不然无法进行游走；具体的游走中以一定概率来选择寻找邻居结点还是回退到起始节点重新开始；如果选择寻找邻居结点其依赖于游走序列的前一个节点，如果前一个节点是author节点，则下一个节点从其所有的paper节点中随机产生，如果前一个节点是paper节点，则下一个节点从其邻居中的author、venue节点中随机产生，如果前一个节点是venue节点，则下一个节点从其所有的paper邻居节点中随机产生；游走过程中必须满足每类节点的个数上限以及总序列长度为100的要求。  最后将产生的游走序列存储到het\_neigh\_train.txt文件的。  【计算游走采样率和采样三元组】：在代码实现上是非常相似的，它们都是基于het\_random\_walk.txt这个文件并且都采用了滑动窗口算法， 不过是先计算采样率后面根据采样率进行采样。het\_random\_walk.txt包含了多条从author节点出发的长为30的随机游走序列。计算采样率时，从每条序列的每个节点的左右各window\_size大小的窗口内寻找邻居节点，根据当前结点和邻居结点的类型进行计数，最后得到了（a，a）、（a,p）、（a,v）、（p,a)、（p，p），（p，v）、（v,a）、（v,p）、  （v,v）共9种类型的个数，最后结合batch\_size计算采样率。采样三元组类似，以一定概率选择邻居结点作为正样本，然后选择随机的同种类型节点作为负样本。 |
| tools.py | 模型类，主要包括三类节点的内容聚合模块、同一类型的聚合模块、不同类型节点的聚合，采用了双向的LSTM |
| args.py | 参数选项 |
| DeepWalk.py | 先从文本语料库txt文件中将多条序列读取成二维数组，最后利用wordVec来生成语料库中节点的嵌入表示并持久化为文件 |
| node\_classification\_model.py | 已有多个节点的嵌入表示，利用sklearn的kmeans库通过嵌入数据来进行节点的分类 |
| link\_prediction.py | 已有多个link的特征嵌入表示，利用LogisticRegression  来进行模型训练，模型的target是链接是否存在，最后在测试集上的链接嵌入上给出预测 |

1. 针对毕设论文症状-草药问题域编写HeteGCN代码

本来以为是只用将症状-草药对应的三种图组合成异质图然后按照HetGNN论文一样的方法写即可，但后发现这里的基线是HetGCN，只是利用了HetGNN组合异质图和用注意力机制聚合不同邻居节点的思想。目前刚理解了计算过程，矩阵公式还没推出

1. 修改符号图SGR的代码

本周花了四五天修改代码，重新发现了不少问题：

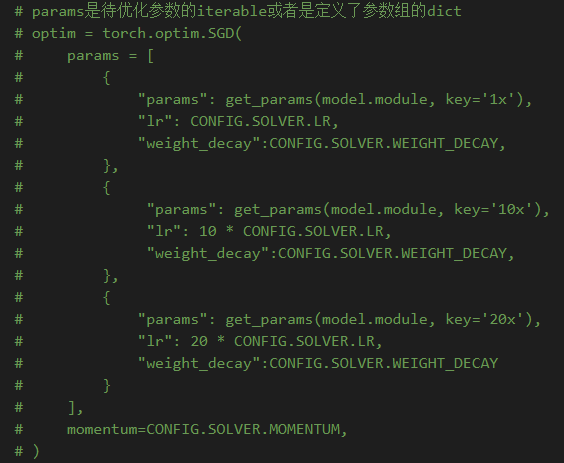
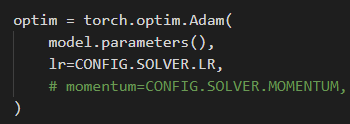
1. 语义分割算法架构的了解

|  |  |  |
| --- | --- | --- |
| 学长说的 | 我之前理解的 | 调查结果 |
| 是不断下采样然后不断上采样至原图大小，最后与label图片进行损失计算 | 我之前参考的代码  是只有连续下采样的过程，最后是将label的尺寸改到与下采样后图片的大小再进行损失计算 | 学习了Unet网络结构和代码，Unet就是不断下采样和上采样  找到了deeplab v2的论文，其方法是下采样8倍，然后label也resize为1/8, 这说明我的方法也是可以的  应该是上采样方法会更好，因为包含更多的信息 |

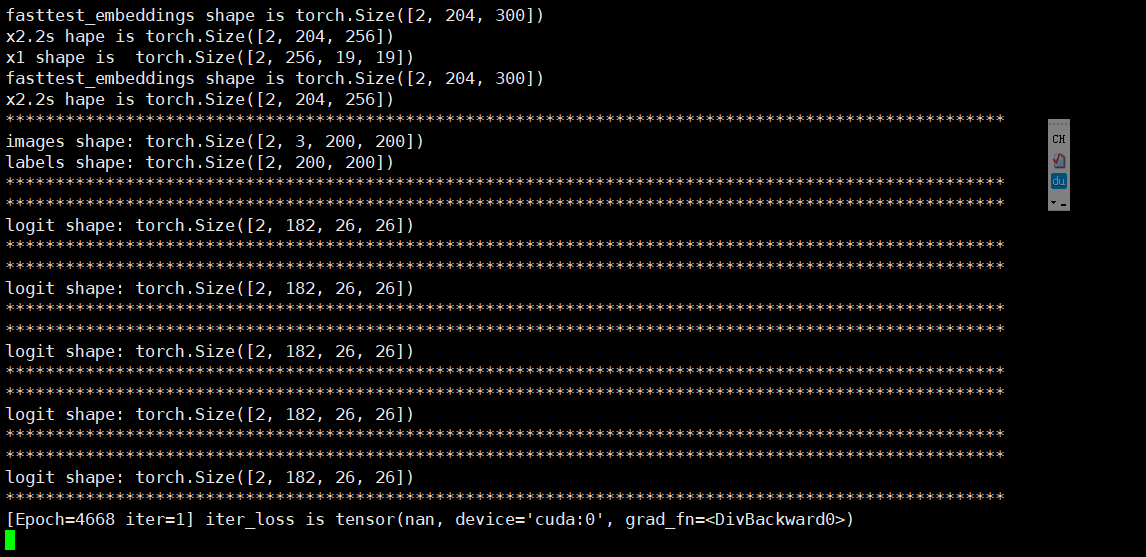
1. 代码bug

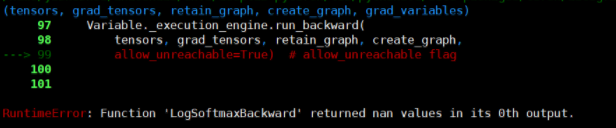
【现象1】：发现代码不管训练多少次，在测试集上的得分是完全一模一样的，后发现使用随机初始化的模型也是得到一样的结果

【分析1】：我的训练过程中模型的参数根本就没有更新，相当于没有进行训练；后来发现是我参照的代码只将named\_module中的相关参数加入了SGD优化器中，而我通过register\_parameter注册的参数不在里面，也得不到训练

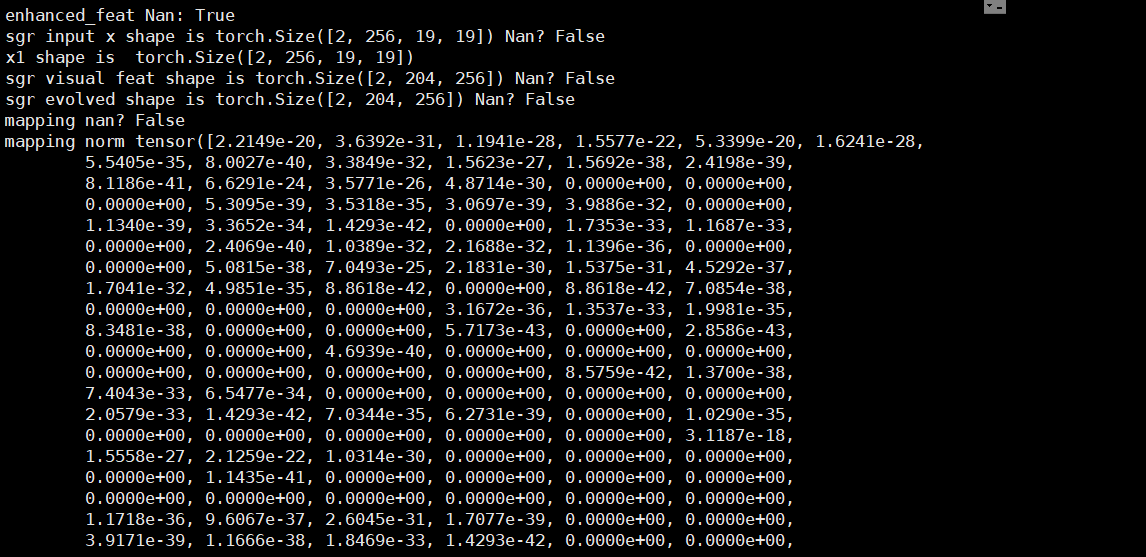
【现象2】：从模型的第一次训练开始损失都是nan，训练了5000次仍然是nan；加上了with torch.autograd.set\_detect\_anomaly(True)报各种各样的nan错误，修改学习率后发现可以运行一段时间，后面的数据就报同样的错误；极少学习率能完整跑完5000次，但是同样的学习率再跑一遍却又报错





【分析2】：原先以为是coco的知识图数据有错误，随机性太强，导致一开始模型的预测结果与标签损失非常大，但是调整了很多次参数初始化方式、学习率后任然不行；查阅大量资料后发现可能是nn.CrossEntropyLoss的两个参数矩阵中的预测结果为NAN，然后一步一步打印输出结果，发现是在SGR层的语义-本地模块的映射矩阵计算中，在softmax后还加了一步不必要的归一化操作，导致出现了矩阵元素 e / 0.0这样的情况，因为softmax的结果不一定含有0，所以会有一些数据不报错。

注：损失函数两个参数矩只要任意一个含有一个nan就会导致出现计算结果为nan

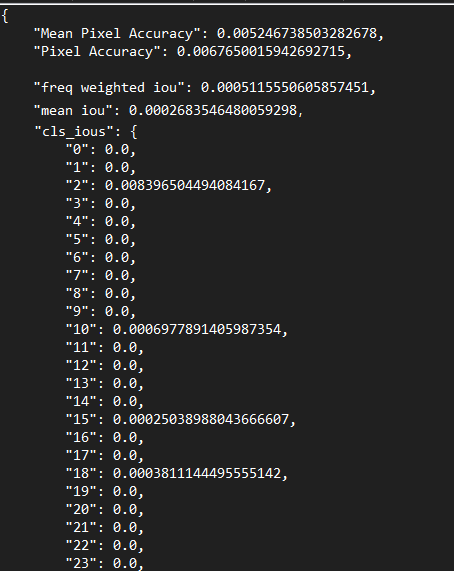


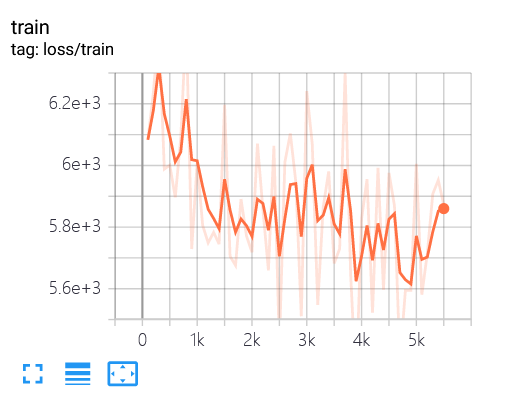
【现象3】：每个epoch使用数据集中的两个batch，并且每个batch都使用backward，疑惑这里为什么不需要加上retain\_graph

【现象4】：无法使用pytorch自带的Imagenet预训练的resnet模型，因为跟自定义的模型名称和结构不一致，只能使用caffe在deeplabv1-coco上的预训练参数

【目前实验结果】：

昨晚才改出来核心bug，目前只跑了5000 epoch，但是预测得分仍然很低, 只是比之前的稍微好一点





4.其他杂知识

（1）稀疏矩阵转化为ndarray使用sp.toarray，而转化为matrix必须使用sp.todense，前者可以直接用【idx1】【idx2】的方式索引，后者必须使用【idx1，idx2】的方式；

（2）反卷积和转置卷积的概念

（3）加上空洞卷积后的图像尺寸计算

等等

1. 学习问题
2. 毕设这周因为改SGR代码的bug耽搁的比较久，需要加快点进度
3. 学习计划和安排
4. 完成HetGCN代码
5. 对其他的方法再进行细调
6. 设计毕设展示系统的原型