Semantic Proximity Search on Heterogeneous Graph by Proximity Embedding代码解析

该文章按照关系是否具有对称性将图分为了两类，symmetric和asymmetric。

对于每一类来说，其中又都具有两个模块，一个是用java语言实现的数据生成模块[1]，该模块的目的是利用图的节点信息与边的信息，来生成路径采样，以及生成输入LSTM中的特征向量x；另一个是用python语言实现的ProxEmbed模块[2]。

对[1]而言，依次分析它的各个文件：

1、**Config.java**与**javaParams.properties**：这两个文件用来设置一些运行时参数。例如：节点文件的路径、边文件的路径、节点特征的存储路径、类型与类型id的映射文件的存储路径、截断类型名称、子路径的存储路径、子路径的最长长度、每个节点的采样次数、每条路径的采样长度等。

2、**Main.java**：数据生成模块的入口。在该入口中，定义了数据处理的顺序。首先进行randomWalkSampling路径采样；再生成输入LSTM的特征向量x；最后从randomWalkSampling中生成子路径。

3、**Node.java**：定义了Node类。对于Node类，有属性：id、type、typeId、in\_nodes、out\_nodes、in\_ids、out\_ids、neighbours，以及具有其对应的属性的操作方法。

4、**ReadWholeGraph.java**：定义了读取node与edge文件的方法，将这些信息全部读入Node类型的变量中。返回的是节点id与节点的映射。

5、**RandomWalkSampling.java**：首先调用ReadWholeGraph用来获取Node信息，而后利用这些信息进行采样。对每个节点的采样为SAMPLING\_TIMES\_PER\_NODE次，每条路径的采样长度为SAMPLING\_LENGTH\_PER\_PATH长。对每条路径的具体采样是步骤（采样函数的传入参数是采样的开始节点，采样长度，节点id到节点的映射）：首先统计当前节点的邻居节点的类型以及各个类型所包含的节点的节点id，再在邻居的所有类型中随机选择一个类型，再在该类型的所有的邻居节点中选择一个节点作为采样的节点，直到采样到了没有下一个节点（因为是有向图，所以有可能没有下一个节点）或者是采样到了所设置的采样长度就停止采样，返回采样的节点的列表。如果所采样的路径的长度小于所设置的最小的路径的长度，则舍弃本次采样；否则将路径的节点转化为对应的节点id，写入保存随机采样路径的文件中。

6、**GenerateEntitiesFeatureByTypes.java**：利用Node的信息，生成LSTM的特征向量。该特征向量是当前节点的类型的one-hot向量。其中one-hot向量中的非零值为config文件中所设置的FEATURE\_TYPE\_VALUE值。

7、**GenerateEntitiesFeaturesByGraph.java**：利用Node信息，生成LSTM的特征向量。该特征向量是由四部分组成：1、当前节点的类型的one-hot向量；2、log(节点的入度+1)；3、邻居类型的分布，类似于词袋模型的统计方法；4、邻居类型的熵，value=now.in\_nodes同一类型的节点的数量/in\_nodes的总数(注：value的数量等于邻居的不同的类型的数量)，sum=-value\*log(value)，sum即是作为第四个特征。设K为图中所有的节点的类型的个数，那么x的维度为(K+1+K+1)。

8、**GenerateSubPathsFromSamplings.java**：利用**RandomWalkSampling.java**生成的路径文件，对这些路径进行子采样。子采样的目的是：1、按照所定的TRUNCATED\_TYPE\_NAME，来进行截断，保证子采样之后的路径的第一个节点与最后一个节点的类型都是TRUNCATED\_TYPE\_NAME中的类型；2、去掉原本采样中可能存在的环。

对theano不熟悉，python的代码就不一一说明了（有的计算细节看着太费时间了），接下来就总结一下python所写的训练模块都干了些什么：

1. 首先是加载从java数据准备模块中准备好的数据，一个是特征向量x（在代码中被称为wordsEmbeddings），一个是子路径的采样（在该python程序中，将子路径处理为了一个map，key值是节点对类似于“p-q”，value值是p到q的子路径的列表（二维列表，可能有很多个子路径））。
2. 再是从给定的训练数据集中读取训练数据。训练数据即是文章中所说的训练三元组。
3. 然后是使用theano来定义模型（计算图），使用之前的数据来训练模型，训练是通过训练次数来定义计算结束，每过saveFreq次就将模型存储一次。

从它python程序中，我们可以看出一个问题：就是它对不同的关系的嵌入是通过加载不同的数据集来实现的。即是说对于不同的关系，需要进行多次嵌入才行。还有就是它的三元组是如何生成的在文中并没有提及到，在它的源代码中也并没有生成三元组的代码。

在文章Semantic proximity search on graphs with metagraph-based learning中，提到了“the desired class of proximity can be substituted by a set of training examples”，即是说将特定的语义类由一系列的训练数据所代替。

（代码的分析放在了github上，dbgroup-uestc/[dengliwei](https://github.com/dbgroup-uestc/dengliwei)/[Interactive Paths Embedding for Semantic Proximity Search on Heterogeneous Graphs](https://github.com/dbgroup-uestc/dengliwei/tree/master/Interactive%20Paths%20Embedding%20for%20Semantic%20Proximity%20Search%20on%20Heterogeneous%20Graphs)/[相关文章](https://github.com/dbgroup-uestc/dengliwei/tree/master/Interactive%20Paths%20Embedding%20for%20Semantic%20Proximity%20Search%20on%20Heterogeneous%20Graphs/%E7%9B%B8%E5%85%B3%E6%96%87%E7%AB%A0)/Proximity Embedding）