2021.10.7笔记

任务：

- 每个程序一个独立的环境，避免版本问题
- 全英文路径
- ad反编译的目标文件是可以选择的，根据需要来选择
- 先跑起来tf-gnn，来进一步了解tf、keras，感受一下

  跑程序的时候了解思路

  目标是工程
- 再去看codebert
- 第二个往后放，重点在code decoder
- anaconda的使用

关于beam search：

- 链接：https://zhuanlan.zhihu.com/p/82829880
- 一种搜索方法，拥有一个超参数beam size，比如为k，就取概率最大的k个候选，用于下一步的搜索，直到进行到最后。

关于Anaconda：

- 链接： Anaconda详细安装及使用教程（带图文） 代码帮-CSDN博客
- Anaconda指的是一个开源的Python发行版本，其包含了conda、Python等180多个科学包及其依赖项
- conda是一个环境管理的系统，可以类比于pip，可以为conda定义不同的源。

关于配置环境：

每个project都要有自己独立的配置环境，先用conda创建一个新的环境，对于缺少的库，现尝试用conda补全，conda源中没有的库，再尝试使用pip。

GNN

- GNN采用在每个节点上分别传播的方式进行学习，由此忽略了节点的顺序。
- GNN常通过邻居节点的加权求和来更新节点的隐藏状体。

深度学习

- 优化器的作用：调节w和b，使得loss尽可能地小就是optimizer的作用，常见的有Adam、Sgd、Rmsprop等。
- tf.placeholder()：每一个tensor值在graph上都是一个op，
- tensorflow程序通常被分为构建阶段和执行阶段；在构建阶段，op的执行步骤被描述成一个图，在执行阶段，使用会话（session）执行图中的op。feeds和fetches可以为任意的op赋值或者从其中获取数据。

- graph定义了计算方式，是一些加减乘除等运算的组合，类似于一个函数。它本身不会进行任何计算，也不保存任何中间计算结果。session用来运行一个graph，或者运行graph的一部分。它类似于一个执行者，给graph灌入输入数据，得到输出，并保存中间的计算结果。同时它也给graph分配计算资源（如内存、显卡等）。
- tensorflow提供了Variable Scope这种独特的机制来共享变量，主要涉及两个函数：

```
tf.get_variable(<name>, <shape>, <initializer>) 创建或返回给定名称的变量
tf.variable_scope(<scope_name>) 管理传给get_variable()的变量名称的作用域
```

tf-gnn-samples

- 用不同的模型完成了四项任务：
  - citation networks（引用网络）：

    The implementation illustrates how to handle the case of transductive graph learning on a single graph instance by masking out nodes that shouldn't be considered.
  - PPI：

    The implementation illustrates how to handle the case of inductive graph learning with node-level predictions.
  - QM9：

    The implementation illustrates how to handle the case of inductive graph learning with graph-level predictions.

    跑出理想的结果可能需要几天。
  - VarMisuse：

    The implementation illustrates how to handle the case of inductive graph learning with predictions based on node selection.

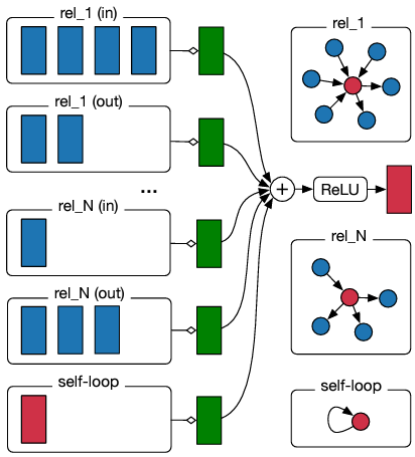
    并没有完全复现，也需要相对比较长的时间。
- 跑通情况：

	citation networks	**PPI**	**QM9**	**VarMisuse**
GGNN	√	√	√	13G .zip
RGCN			√	
RGAT			√	
RGIN			√	
GNN-Edge-MLP			√	
RGDCN			batch_size太大，内存不够	
GNN-FiLM			√	

- RGCN：适用于有大规模关系数据（relation的类型也有不同）的graph。

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

这个公式，对于一个node i，和以不同的关系r关联起来的node j，分关系类型地加在一起，再加上本身的信息$h_i^{(l)}$，就构成了新的node feature：$h_i^{(l+1)}$。



- 如果跑通的话数据结果应该都是可以复现的，自己跑了两次，从log文件看数据完全相同。
- 程序内容：
  - MODEL_NAME关键字：确定使用的模型，也就是下图之一。
    - `GGNN` : Gated Graph Neural Networks (Li et al., 2015).
    - `RGCN` : Relational Graph Convolutional Networks (Schlichtkrull et al., 2017).
    - `RGAT` : Relational Graph Attention Networks (Veličković et al., 2018).
    - `RGIN` : Relational Graph Isomorphism Networks (Xu et al., 2019).
    - `GNN-Edge-MLP` : Graph Neural Network with Edge MLPs - a variant of RGCN in which messages on edges are computed using full MLPs, not just a single layer applied to the source state.
    - `RGDCN` : Relational Graph Dynamic Convolution Networks - a new variant of RGCN in which the weights of convolutional layers are dynamically computed.
    - `GNN-FiLM` : Graph Neural Networks with Feature-wise Linear Modulation - a new extension of RGCN with FiLM layers.
  - TASK_NAME关键字：确定要完成的任务。

名词解释：

- transductive learning：与inductive learning（归纳式学习）相对应，在训练时已经用到了测试集数据（unlabelled）的信息，这样可以从特征分布上学到额外的信息，但是只要有新的样本来，模型就得重新训练。
- python相关：
  - 函数后面的“ -> ”，是为函数添加元数据，描述函数的返回类型；可以在参数后加上 ： 和数据类型来指定参数的数据类型，加等号表示初始值，比如：

    ```
    def fun(num_timesteps: int = 1)->int
    ```

  - raise关键字：用来抛出异常，后续的代码将无法执行。
  - with关键字：是一种上下文管理协议，目的在于去掉try、except、finally等关键字去掉，进化流程。
  - numpy中，[:, a]，可以理解为在矩阵中取第a列的元素。
- MAE：平均绝对误差（mean absolute error），观测值与真实值的误差绝对值的平均值。

$$\mathrm{MAE}(X,h) = \frac{1}{m}\sum_{i=1}^{m}|h(x^{(i)} - y^{(i)}|$$

问题

1. 在执行 python train.py GGNN QM9 的时候为什么会出现train loss高于valid loss，是否是正常的?

   （这里是loss，从最开始的随机参数训练，每个batch的loss自然是比较大的，而valid loss，是用训练后的模型验证得到的loss，自然就会小一些）

```
== Epoch 1
?[K Train: loss: 1.60242 || MAEs: 0:1.08391 | Error Ratios: 0:16.29600 || graphs/sec: 475.40 | nodes/sec: 8571 | edges/sec: 26305
?[K Valid: loss: 0.26455 || MAEs: 0:0.55612 | Error Ratios: 0:8.36103 || graphs/sec: 1361.03 | nodes/sec: 24575 | edges/sec: 75404
  (Best epoch so far, target metric decreased to 0.26455 from inf. Saving to 'trained_models\QM9_GGNN_2021-10-09-14-59-47_6132_best_model.pickle')
== Epoch 2
?[K Train: loss: 0.25348 || MAEs: 0:0.54556 | Error Ratios: 0:8.20226 || graphs/sec: 489.94 | nodes/sec: 8833 | edges/sec: 27110
?[K Valid: loss: 0.24218 || MAEs: 0:0.53175 | Error Ratios: 0:7.99459 || graphs/sec: 1492.31 | nodes/sec: 26945 | edges/sec: 82678
  (Best epoch so far, target metric decreased to 0.24218 from 0.26455. Saving to 'trained_models\QM9_GGNN_2021-10-09-14-59-47_6132_best_model.pickle')
== Epoch 3
Traceback (most recent call last):5 (has 2769 graphs). Loss so far: 0.2408
```

2. 看了一些资料，还是不能理解with variable_scope() 的用法：

```
with tf.variable_scope("graph_model"):  #? 这里的作用?
    self.__placeholders['num_graphs'] = \
        tf.placeholder(dtype=tf.int64, shape=[], name='num_graphs')
    self.__placeholders['graph_layer_input_dropout_keep_prob'] = \
        tf.placeholder_with_default(1.0, shape=[], name='graph_layer_input_dropout_keep_prob')
```

3. sparse_graph_model()中对于一些参数的运用没有理解？比如graph_residual_connection_every_num_layers

   174行：

```
cur_node_representations = self.__ops['projected_node_features']     #这里应该是 n*128 的tensor
last_residual_representations = tf.zeros_like(cur_node_representations) #构造了一个shape与cur_node_representations相同，值
for layer_idx in range(self.params['graph_num_layers']):  #这里确定了有多少层，每层做相同的操作
    with tf.variable_scope('gnn_layer_%i' % layer_idx):
        cur_node_representations = \
            tf.nn.dropout(cur_node_representations, rate=1.0 - self.__placeholders['graph_layer_input_dropout_keep_prob
        if layer_idx % self.params['graph_residual_connection_every_num_layers'] == 0:  #? 这一步又是什么目的?
            t = cur_node_representations
            if layer_idx > 0:
                cur_node_representations += last_residual_representations
                cur_node_representations /= 2
            last_residual_representations = t         #这上面可以理解为对cur_node_representations做了一定的变换操作
        cur_node_representations = \
            self._apply_gnn_layer( # 这里调用了ggnn_model.py,然后调用了ggnn.py
                cur_node_representations,  #相当于是node_embedding
                self.__ops['adjacency_lists'], #邻接表
                self.__ops['type_to_num_incoming_edges'],  #每个type的edge（Tensor类型，是很多edge的集合）
                self.params['graph_num_timesteps_per_layer'])  #把现在的 node_embedding 输入进去，然后得到一个经过 gnn 和 gai
```

   194行：参数用处；待确定：这里设置的Dense就是单纯地乘了一个W矩阵，这里再使用一个activation。

```
            if self.params['graph_inter_layer_norm']: #对node 的Tensor进行一个标准化
                cur_node_representations = tf.contrib.layers.layer_norm(cur_node_representations)
            if layer_idx % self.params['graph_dense_between_every_num_gnn_layers'] == 0:
                cur_node_representations = \
                    tf.keras.layers.Dense(units=h_dim,
                                          use_bias=False,
                                          activation=activation_fn,
                                          name="Dense",
                                          )(cur_node_representations)   #这里再经过一个Dense(没理解错的话应该是只有一个W)
```