7.1 分布式训练所需的图数据预处理

(English Version)

DGL要求预处理图数据以进行分布式训练，这包括两个步骤：1)将一张图划分为多张子图(分区)，2)为节点和边分配新的ID。 DGL提供了一个API以执行这两个步骤。该API支持随机划分和一个基于 Metis 的划分。Metis划分的好处在于， 它可以用最少的边分割以生成分区，从而减少了用于分布式训练和推理的网络通信。DGL使用最新版本的Metis， 并针对真实世界中具有幂律分布的图进行了优化。在图划分后，API以易于在训练期间加载的格式构造划分结果。

Note: 图划分API当前在一台机器上运行。 因此如果一张图很大，用户将需要一台大内存的机器来对图进行划分。 未来DGL将支持分布式图划分。

默认情况下，为了在分布式训练/推理期间定位节点/边，API将新ID分配给输入图的节点和边。 分配ID后，该API会相应地打乱所有节点数据和边数据。在训练期间，用户只需使用新的节点和边的ID。 与此同时，用户仍然可以通过 `g.ndata['orig_id']` 和 `g.edata['orig_id']` 获取原始ID。 其中 `g` 是 `DistGraph` 对象（详细解释，请参见:ref:*guide-distributed-apis*）。

DGL将图划分结果存储在输出目录中的多个文件中。输出目录里始终包含一个名为xxx.json的JSON文件，其中xxx是提供给划分API的图的名称。 JSON文件包含所有划分的配置。如果该API没有为节点和边分配新ID，它将生成两个额外的NumPy文件：*node_map.npy* 和 *edge_map.npy*。 它们存储节点和边ID与分区ID之间的映射。对于具有十亿级数量节点和边的图，两个文件中的NumPy数组会很大， 这是因为图中的每个节点和边都对应一个条目。在每个分区的文件夹内，有3个文件以DGL格式存储分区数据。 *graph.dgl* 存储分区的图结构以及节点和边上的一些元数据。*node_feats.dgl* 和 *edge_feats.dgl* 存储属于该分区的节点和边的所有特征。

```
data_root_dir/
    |-- xxx.json            # JSON中的分区配置文件
    |-- node_map.npy        # 存储在NumPy数组中的每个节点的分区ID（可选）
    |-- edge_map.npy        # 存储在NumPy数组中的每个边的分区ID（可选）
    |-- part0/              # 分区0的数据
        |-- node_feats.dgl  # 以二进制格式存储的节点特征
        |-- edge_feats.dgl  # 以二进制格式存储的边特征
        |-- graph.dgl       # 以二进制格式存储的子图结构
    |-- part1/              # 分区1的数据
        |-- node_feats.dgl
        |-- edge_feats.dgl
        |-- graph.dgl
```

负载均衡

在对图进行划分时，默认情况下，Metis仅平衡每个子图中的节点数。根据当前的任务情况，这可能带来非最优的配置。 例如，在半监督节点分类的场景里，训练器会对局部分区中带标签节点的子集进行计算。 一个仅平衡图中节点(带标签和未带标签)的划分可能会导致计算负载不平衡。为了在每个分区中获得平衡的工作负载， 划分API通过在 `dgl.distributed.partition_graph()` 中指定 `balance_ntypes` 在每个节点类型中的节点数上实现分区间的平衡。用户可以利用这一点将训练集、验证集和测试集中的节点看作不同类型的节点。

以下示例将训练集内和训练集外的节点看作两种类型的节点：

```
dgl.distributed.partition_graph(g, 'graph_name', 4, '/tmp/test',
balance_ntypes=g.ndata['train_mask'])
```

除了平衡节点的类型之外， `dgl.distributed.partition_graph()` 还允许通过指定 `balance_edges` 来平衡每个类型节点在子图中的入度。这平衡了不同类型节点的连边数量。

Note: 传给 `dgl.distributed.partition_graph()` 的图名称是一个重要的参数。 `dgl.distributed.DistGraph` 使用该名称来识别一个分布式的图。一个有效的图名称应该仅包含字母和下划线。