PyTorch Geometric基本介绍

1. 基本数据结构

*PyTorch Geometric*中设计了一种新的表示图数据的存储结构，也是PyTorch Geometric中实现的各种方法的基本数据形式。

1.1 符号定义

在PyTorch Geometric中，一个图被定义为$\mathcal{G}=(X,(I,E))$，其中$X\in\mathbb{R}^{N\times F}$表示节点的特征矩阵，$N$为节点的个数，$F$为每个节点的特征数；用$(I,E)$这种元组形式表示图的稀疏邻接矩阵，$I\in\mathbb{N}^{2\times E}$为边的索引，$E\in\mathbb{R}^{E\times D}$为$D$维的边特征。

1.2API接口

用于模型的图（graph）数据包括对象（nodes）及成对对象之间的关系（edges）组成。用于PyTorch Geometric中的每个图都是一个torch_geometric.data.Data类型的实例，其属性有：

- `data.x`：节点特征矩阵，形状为`[num_nodes, num_node_features]`
- `data.edge_index`：COO格式的图的边关系，形状为`[2, num_edges]`，类型为`torch.long`
- `data.edge_attr`：边特征矩阵，形状为`[num_edges, num_edge_features]`
- `data.y`：针对训练的目标可能具有不同的形状
- `data.pos`：节点的位置矩阵，形状为`[num_nodes, num_dimensions]`

`Data`对象不是必须有上面所有的这些属性，也不是只能有这些属性。比如，我们可以通过`data.face`进行扩展，用一个张量（tensor）来保存一个3D网格的三元链接关系，形状为`[3, num_faces]`，类型为`torch.long`。

PyTorch Geometric已经实现了基于这种图数据结构的常用操作。

2.消息传递网络

将卷积操作推广到不规则数据通常表示为邻域聚合（*neighborhood aggregation*）或消息传递（*message passing*）。在PyTorch Geometric中将各种图神经网络中的邻域聚合方法统一到一种消息传递网络的架构中。

2.1符号定义

$$\mathbf{x}_i^{(k)} = \gamma^{(k)}\left(\mathbf{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \phi^{(k)}\left(x_i^{(k-1)}, x_j^{(k-1)}, \mathbf{e}_{i,j}\right)\right)$$

其中，□表示可微的置换不变的函数，如`sum`，`mean`或者`max`；γ和ϕ表示可微函数，如MLPs。

□函数要求置换不变的原因是，在图结构中，节点的多个邻居节点之间是无序的，不应该因为邻居节点的序列顺序不同而产生不同的结果或者相差很大的结果。γ和ϕ要求可微的原因是便于在传播过程中计算微分以便使用梯度下降等方式对网络进行优化。上式表达了一层消息传递网络从输入到输出的对图数据的某种邻域聚合操作。

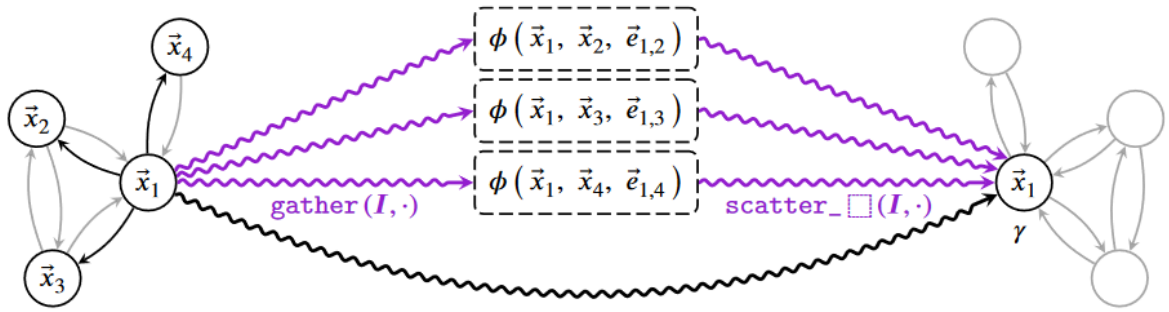

Figure 1: Computation scheme of a GNN layer by leveraging gather and scatter methods based on edge indices I, hence alternating between node parallel space and edge parallel space.

> Fast Graph Representation Learning with PyTorch Geometric

2.2API接口

PyTorch Geometric提供了`torch_geometric.nn.MessagePassing`的基类，它能够通过自动处理消息传播来创建这种类型的消息传递图神经网络。用户只需要定义函数ϕ，如`message()`和函数γ，如`update()`，以及需要使用的聚合方案，如`aggr='add'`，`aggr='mean'`或者`aggr='max'`。

- 指定聚合函数和消息流向 - `torch_geometric.nn.MessagePassing(aggr="add", flow="source_to_target")`：定义了要使用的聚合方案（`"add"`，`"mean"` 或者`"max"`）和消息传播方向（是 `"source_to_target"` 还是`"target_to_source"`）。
- 传播消息 - `torch_geometric.nn.MessagePassing.propagate(edge_index, size=None, **kwargs)`：初始化时调用这个函数开始传播消息。输入边的索引和所有其他的用于构造消息和更新节点嵌入的数据。值得注意的是，这个`propagate()`方法不仅可以在形状为 `[N, N]` 的对称邻接矩阵中交换消息，还可以在一般的稀疏分配矩阵中交换消息，例如二分图，形状为`[N, M]`的话只需要`size=(N, M)`作为一个额外的参数就可以。如果设置为`None`，那么分配的矩阵会被假设为对称的。对于具有两组独立节点和索引的二分图，并且每组保存自己的信息，这种拆分可以通过在传递过程中将信息作为一个元组来标记。例如 `x=(x_row, x_col)`，以此表明不同集合中的节点关系。
- 创建消息 - `torch_geometric.nn.MessagePassing.message()`：以函数ϕ的方式构造消息，如果`flow="source_to_target"`，那么对每条边$(j,i) \in \mathcal{E}$做此操作，即将消息从节点j传播到节点i；如果`flow="target_to_source"`，那么对每条边$(i,j) \in \mathcal{E}$做此操作，即将消息从节点i传播到节点j。任何用于构造消息的输入都可以作为参数传递给`propagate()`函数。另外，通过将`_i`或者`_j`加在变量名后面可以将特征映射到相应的节点i和j，例如`x_i`和`x_j`。
- 更新嵌入表示 - `torch_geometric.nn.MessagePassing.update()`：以函数 的方式对每一个节点 进行更新节点嵌入的操作。将聚合操作后的输出结果作为第一个参数，任何需要在初始化时传递给`propagate()`函数的参数作为输入。

MessagePassing类是实现各种图神经网络模型的基础。

从一个图神经层到另一个图神经进行的消息传递操作可以分为三个计算层次：

- 第一层计算ϕ函数操作，这属于创建消息的过程。输入为上一层的节点特征和边关系，可以指定消息的流向。（边关系以索引的形式进行查找）利用了图结构数据的边关系。
- 第二层计算□函数操作，这属于邻域聚合的过程。经过第一层的操作，已经按边关系建立了节点间的消息，$j \in \mathcal{N}(i)$表示节点i的（一阶）邻

居节点，即限定了消息只在指定节点的邻域范围内传递。□的`add`，`mean`和`max`都不会因为邻居节点的顺序排列问题而产生不同的结果。利用了图结构数据中的结构信息。

- 第三层计算γ函数操作，这属于更新特征的过程。经过邻域上的消息传递之后，将邻域上的信息聚合到目标节点i，然后更新节点i的特征，作为这一层的输出。

3.常用基准数据集

PyTorch Geometric包含了大量的常用基准数据集，例如所有的Planetoid数据集（Cora，Citeseer，Pubmed），来自http://graphkernels.cs.tu-dortmund.de/的所有经典的图数据集，QM7和QM9数据集，还有少数的3D mesh/点云数据集，像FAUST，ModelNet10/40和ShapeNet。初始化一个数据集非常简单直接。数据集初始化的时候会自动下载原始数据并将它们处理成之前描述的`Data`格式。

对于下载并处理好的数据集，可以进行查看相关属性如图数，节点数，边数，特征数，类别数等，还可以进行数据集分割，打乱顺序等经常使用的操作。

4.小批量数据训练

人工神经网络通常以分批方式进行训练。PyTorch Geometric通过创建稀疏块对角邻接矩阵（由`edge_index`和`edge_attr`定义）并在节点维度上连接特征和目标矩阵，以达到在小型批量数据集上实现并行化的目的。PyTorch Geometric已经实现了一个自己的`torch_geometric.data.DataLoader`类，它已经处理了连接的过程。`torch_geometric.data.Batch`继承自`torch_geometric.data.Data`并包含一个附加属性`batch`。`batch`是批处理中所有图的所有节点的图标识符的列向量。

5.数据转换

转换（Transforms）是torchvision中转换图像和进行数据增强的常用方法。PyTorch Geometric也包含自己的转换，它以`Data`对象作为输入并返回一个新的转换后的`Data`对象。可以使用`torch_geometric.transforms.Compose`将转换链接在一起，并在将已

处理的数据集保存到磁盘之前（`pre_transform`）或者访问数据集中的图之前应用变换操作。例如，用户可以通过转换操作从点云生成最邻近图从而将点云数据集转换为图数据集。