图神经网络-习题课1

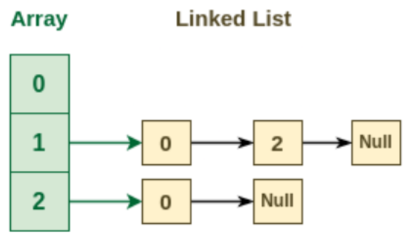
图机器学习的主要开发库-PyG

- PyG (Pytorch Geometric): 基于Pytorch的图机器学习库;
- 2017年开始开发，具有易用、功能全面、性能强的特点，现已是领域最流行的图机器学习开发框架;
- 安装：`pip install torch; pip install torch_geometric`
- 文档：https://pytorch-geometric.readthedocs.io/

- 其他库：Networkx(复杂网络分析、可视化), DGL(另一个图机器学习库)等，不再专门介绍。

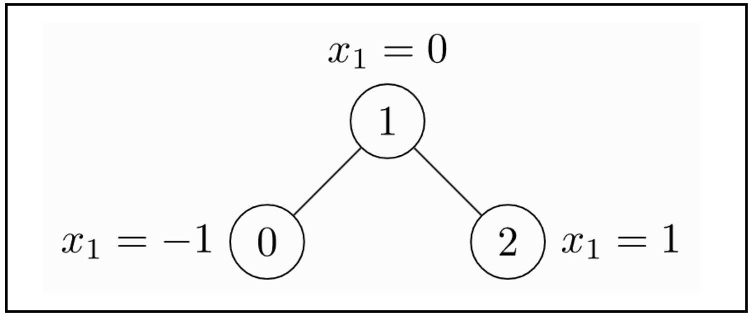
图数据的定义和存储

- 图数据包括两个部分：特征与结构；
- 特征一般包含顶点特征和边特征，可以用简单的tensor存储；
- 连边结构的存储方式：
  - 邻接矩阵(Adjacency Matrix): $A \in \{0,1\}^{N\times N}$;
    - 适合执行矩阵运算（邻接矩阵幂，Laplacian算子）；
    - N^2,浪费空间!
  - 邻接表(adjacency table):
    - 方便枚举每个点的邻居，适合图结构算法；
    - 通常以链表形式存储，对GPU非常不友好!
      - 在GPU上运行图几何算法时，一般采用CSR稀疏表示的邻接表；



图数据的定义和存储

- 连边结构的存储方式：
  - 边表(adjacency set):图机器学习最常用的数据结构！
  - 结构为$[(v_1,u_1),\ldots,(v_E,u_E)]$的矩阵，由所有的有向边$v \to u$组合得到；PyG中的edge_index;
  - 实际存取时考虑性能取其转置（GNN），形状为$2 \times E$;
    - 下图边表为$[(0,1),(1,0),(1,2),(2,1)]$




```
edge_index = torch.tensor([[0, 1, 1, 2],
                           [1, 0, 2, 1]], dtype=torch.long)
```

PyG的基本数据定义

- torch_geometric.data.Data：图数据的类
  - x：顶点特征，形状为$N \times d$;
  - edge_index：边表，形状为$2 \times E$;
    - 注意：无向边看成两条有向边，这里其实算$2E$;
  - edge_attr：边特征，形状为$E \times d'$（边表的另一个好处）；
- 转换函数：
  - torch_geometric.utils中的to_dense_adj和dense_to_sparse：邻接矩阵和边表的转换函数。
  - subgraph：从大图取子图的方法，会用到。

图数据的定义和存储

- **示例-边表到edge_index的转换；**

```python
edge_table = [(0, 1), (1, 0), (1, 2), (2, 1), (2, 3), (3, 2)]
edge_index = torch.tensor(edge_table).t()
print(edge_index)
```

✓ 0.0s

```
tensor([[0, 1, 1, 2, 2, 3],
        [1, 0, 2, 1, 3, 2]])
```

图数据的定义和存储

- **示例-edge_index和邻接矩阵的转换**

```
from torch_geometric.utils import dense_to_sparse, to_dense_adj

edge_index = torch.tensor([[0, 1, 1, 2, 2, 3], [1, 0, 2, 1, 3, 2]], dtype=torch.long)
adj = to_dense_adj(edge_index)
print(adj)

adj = torch.tensor([[0, 1, 0, 0], [1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0]])
print(dense_to_sparse(adj))
```

```
tensor([[[0., 1., 0., 0.],
         [1., 0., 1., 0.],
         [0., 1., 0., 1.],
         [0., 0., 1., 0.]]])
(tensor([[0, 1, 1, 2, 2, 3],
        [1, 0, 2, 1, 3, 2]]), tensor([1, 1, 1, 1, 1, 1]))
```

图数据的定义和存储

- **示例-寻找节点的邻居**

```
edge_index = torch.tensor([[0, 1, 1, 2, 2, 3], [1, 0, 2, 1, 3, 2]], dtype=torch.long)
node_id = 1
neighbors = edge_index[1, data.edge_index[0] == node_id]
print(neighbors)
```

✓ 0.0s

```
tensor([0, 2])
```

PyG的应用

- **示例-计算图中所有节点的度数**

```
def count_degrees(edge_index):
    num_nodes = edge_index.max().item() + 1
    degrees = torch.zeros(num_nodes)
    for e in range(edge_index.shape[1]):
        src, dst = edge_index[:, e]
        degrees[src] += 1
        degrees[dst] += 1
    return degrees / 2
```

PyG的应用

- **示例-取子图的subgraph方法：注意节点被重排序！**

```
import torch_geometric

edge_index = torch.tensor([[0, 1, 1, 2, 2, 3], [1, 0, 2, 1, 3, 2]], dtype=torch.long)
data = torch_geometric.data.Data(edge_index=edge_index)
subgraph = data.subgraph([0, 1, 2])
print(subgraph.edge_index)

subgraph = data.subgraph([1, 2]) # nodes are relabeled!
print(subgraph.edge_index)
```

✓ 0.0s

```
tensor([[0, 1, 1, 2],
        [1, 0, 2, 1]])
tensor([[0, 1],
        [1, 0]])
```

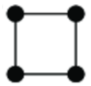
PyG的应用

- 示例-计算图中三角形graphlet的数目（暴力法）
- 思路：枚举所有的三元子图-计数所有含有3条边的子图;
- 计算graphlet，一定注意同构导致的重复计数!

```python
def count_triangles_simple(edge_index):
    # find num_nodes from edge_index
    num_nodes = edge_index.max().item() + 1
    data = torch_geometric.data.Data(edge_index=edge_index)
    possible_subgraph_indices = list(combinations(range(num_nodes), 3)) # enumerate all possible subgraphs
    num_triangles = 0
    for ind in possible_subgraph_indices:
        subgraph = data.subgraph(ind)
        if subgraph.edge_index.shape[1] == 6:
            num_triangles += 1

    return num_triangles
```

Graphlet的其余讨论

- 其他的graphlet怎么枚举?
- 任意阶graphlet、任意大小图上的枚举涉及到图同构匹配问题，没有通用的解决方案;
- 对小型graphlet可以通过枚举+筛选的方法:
  - 如枚举  可以通过类似计算三角形的方法，**筛选所有含有4个2度节点的子图**得到， 

 可以通过**筛选所有含3度顶点和3条边的4阶子图**得到(见作业);
  - 这类方法有进一步延申:Ahmed, Nesreen K., et al. "Efficient graphlet counting for large networks." 2015 IEEE international conference on data mining. IEEE, 2015.
- graphlet计算昂贵，但组成的特征具有独特的表达能力，在图领域的理论和模型设计应用广泛。

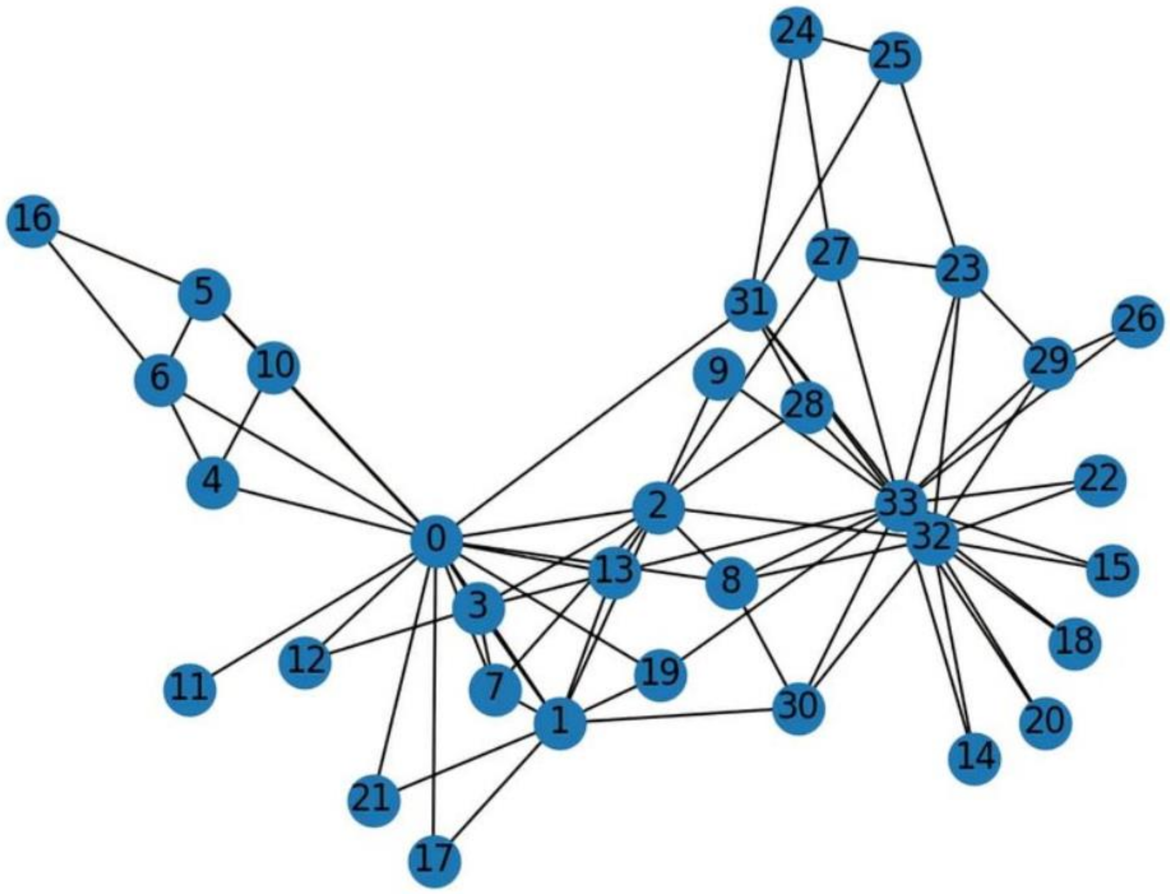
典型数据集

- torch_geometric.datasets：大量公开图数据集的集合
  - KarateClub：空手道俱乐部，经典的小规模社交数据集；34个节点，78条边；
  - Planetoid：经典的图节点、边级别任务的论文引用图数据集，包含Cora，Citeseer，Pubmed三个。统计数据为：

Name	#nodes	#edges	#features	#classes
Cora	2,708	10,556	1,433	7
CiteSeer	3,327	9,104	3,703	6
PubMed	19,717	88,648	500	3

典型数据集

- **KarateClub数据集**



典型数据集

- **示例-加载KarateClub和Cora数据集**

```
from torch_geometric.datasets import KarateClub, Planetoid

karate = KarateClub()[0]
cora = Planetoid(root='./datasets/Cora', name='Cora')[0]

print(karate)
print(cora)
```

✓ 0.0s

```
Data(x=[34, 34], edge_index=[2, 156], y=[34], train_mask=[34])
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
```

Weisfeiler-Lehman Kernel (Test)

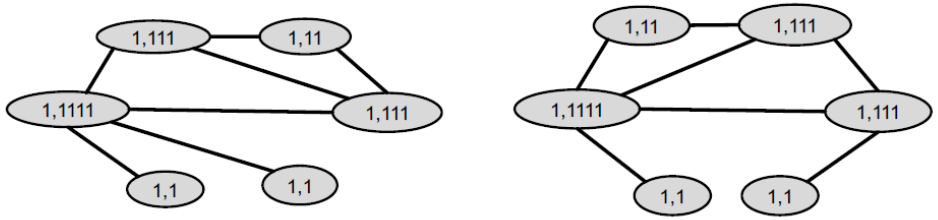
- **回顾：Color Refinement**

$$c^{(k+1)}(v) = \mathrm{HASH}\left(\left\{c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right\}\right)$$

Example of color refinement given two graphs

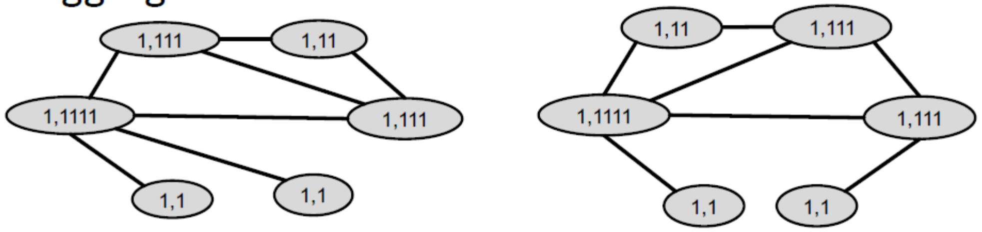
- Assign initial colors

- Aggregate neighboring colors

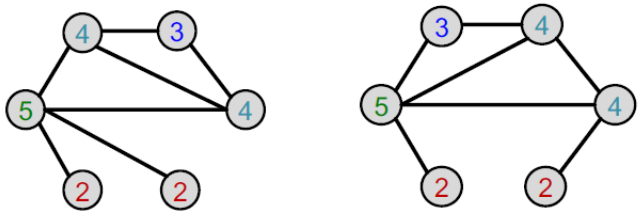


Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors

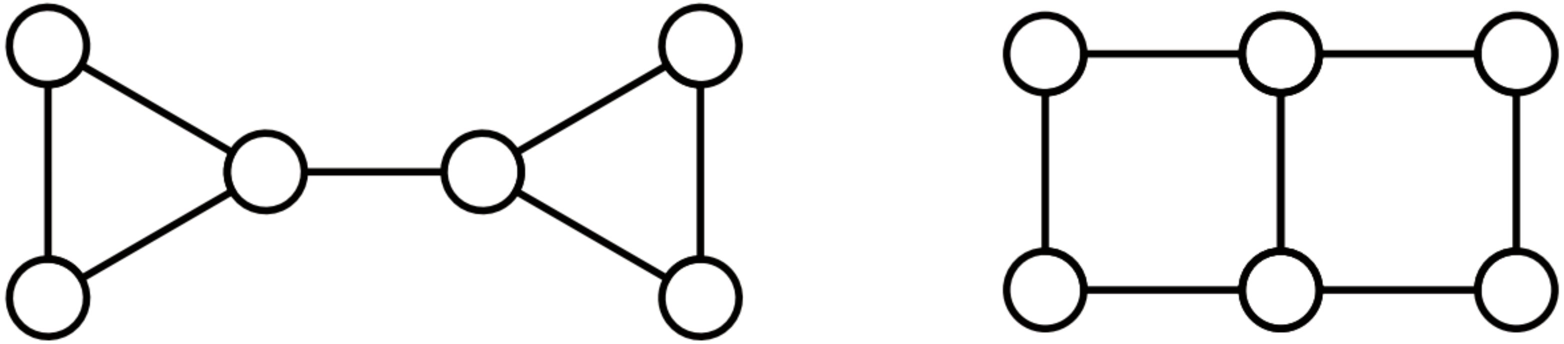


Hash table

1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Weisfeiler-Lehman Kernel (Test)

- Color Refinement过程不能区分所有图!



- Weisfeiler-Lehman test是后续GNN的理论模型，对我们研究GNN理论和提升GNN的结构表达能力非常重要。

作业1：图数据经典指标

1：On `KarateClub` dataset, compute the following features using `pytorch`，`pytorch_geometric` and `numpy` **only**:

(1) degree, clustering coefficient for every node; (4 points)

(2) Jaccard’s coefficient and Katz index for every node pair; (4 points)

(3) counting of all connected 3,4-node graphlets (G1-G8, P19); (1 points)

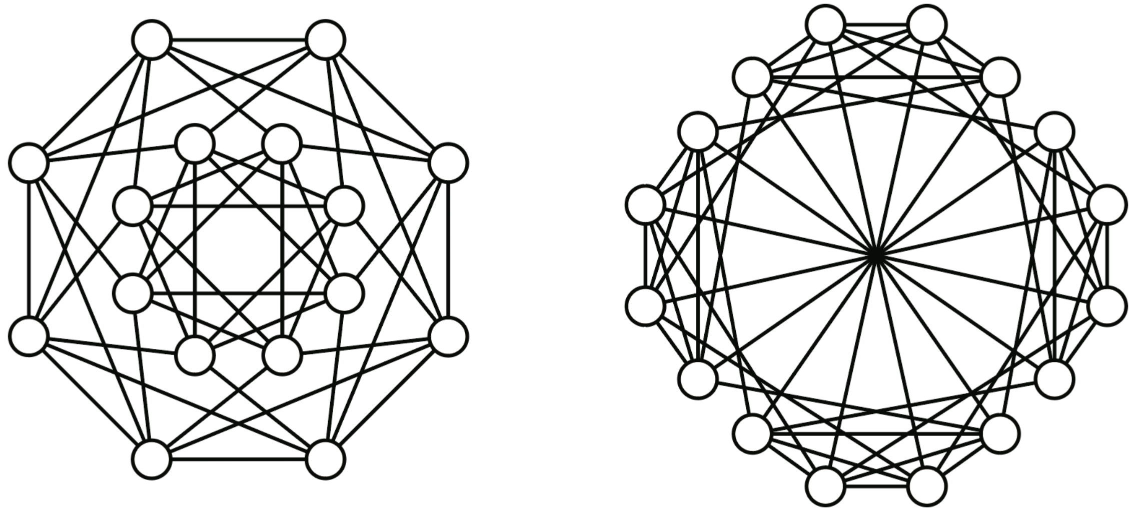
2：Prove that $N_{triangles} = \frac{trace(A^3)}{6}$. (1 points)

提交python代码和简单的报告描述思路，教学网提交。

作业1：可选

A: Count all 3,4-node connected graphlets on Cora (3k+ nodes). Can your code complete the task in a reasonable time?

B: For the following two graphs:



(1) According to the definition, explain why Weisfeiler-Lehman kernel can not distinguish between these two graphs;

(2) Find a way to distinguish between these two graphs from the content of this assignment.