

# 图表示学习实验

## 一、图节点表示实验

### (1) karate club network 的 average degree(平均度)是多少(5 分)

给定空间中的一个完整网络的平均度可以通过以下公式计算：

$$\text{Average Degree} = \frac{2 \times \text{Number of Edges}}{\text{Number of Nodes}}$$

而 karate club network 是一个由 34 个节点和 78 条边组成的网络。因此，通过这些值代入上述公式，我们可以得到 karate club network 的平均度。

$$\text{Average Degree} = \frac{2 \times 78}{34} \approx 4.59 \approx 5$$

所以 karate club network 的平均度是约为 5。

Question 1: karate club network 的平均度是多少(5分)

```
def average_degree(num_edges, num_nodes):
    # TODO: Implement this function that takes number of edges
    # and number of nodes, and returns the average node degree of
    # the graph. Round the result to nearest integer (for example
    # 3.3 will be rounded to 3 and 3.7 will be rounded to 4)

    avg_degree = 0

    ##### Your code here #####
    avg_degree = round(2 * num_edges / num_nodes)
    #####

    return avg_degree

num_edges = G.number_of_edges()
num_nodes = G.number_of_nodes()
avg_degree = average_degree(num_edges, num_nodes)
print("Average degree of karate club network is {}".format(avg_degree))
```

Average degree of karate club network is 5

### (2) karate club 网络的平均聚类系数是多少 k? (5 分)

平均聚类系数是指网络中所有节点的聚类系数的平均值。聚类系数衡量了一个节点的邻居节点之间连接的密集程度。对于每个节点  $u$ ，其聚类系数  $C_u$  被定义为实际连接数量与可能连接数量的比值，也可以表述为：

$$C_u = \frac{|(v_1, v_2) \in \mathcal{E}: v_1, v_2 \in \mathcal{N}(u)|}{\binom{d_u}{2}}$$

这个方程中的分子统计了节点  $u$  的邻居节点之间的边的数量， $\mathcal{N}(u) = \{v \in \mathcal{V}: (u, v) \in \mathcal{E}\}$  来表示节点的邻居节点。分母计算了  $u$  的邻居节点中有多少对节点。

对应到包含结点集合  $\mathcal{V}$  的图来说，平均聚类系数可以计算为：

$$\text{Average Clustering Coefficient} = \frac{\sum_{v \in \mathcal{V}} C_v}{|\mathcal{V}|}$$

使用 NetworkX 库中的 `nx.average_clustering` 函数来计算给定图  $G$  的平均聚类系数。

Question 2: karate club 网络的平均聚类系数是多少k? (5分)

```
def average_clustering_coefficient(G):
    # TODO: Implement this function that takes a nx.Graph
    # and returns the average clustering coefficient. Round
    # the result to 2 decimal places (for example 3.333 will
    # be rounded to 3.33 and 3.7571 will be rounded to 3.76)

    avg_cluster_coef = 0

    ##### Your code here #####
    ## Note:
    ## 1: Please use the appropriate NetworkX clustering function
    avg_cluster_coef = nx.average_clustering(G)
    avg_cluster_coef = round(avg_cluster_coef, 2)
    #####

    return avg_cluster_coef

avg_cluster_coef = average_clustering_coefficient(G)
print("Average clustering coefficient of karate club network is {}".format(avg_cluster_coef))
```

Average clustering coefficient of karate club network is 0.57

(3) 在一次 PageRank 迭代之后, 节点 0(id 为 0 节点)的 PageRank 值是多少? (5 分)

PageRank 使用网络的链接结构来衡量图中节点的重要性。来自重要页面的“vote”更有价值。特殊的, 当一个重要程度为 $r_i$ 的页面 $i$ 具有 $d_i$ 个外部链接,那么每个链接将会得到 $\frac{r_i}{d_i}$ 的“vote”。因此一个页面 $j$ 的重要程度,记为 $r_j$ , 是他所有链接的总和 $r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$ , 其中 $d_i$ 是节点 $i$ 的出度。

PageRank 算法(由 Google 使用)输出一个概率分布, 表示随机浏览者点击链接到达任何特定页面的可能性。在每个时间步, 随机的冲浪者有两个选择:

- $\beta$  的概率随机跟随一个页面
- $1 - \beta$  的概率随机跳转到一个页面

因此, 一个特定页面的重要性可以计算为:  $r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$

Question 3: 在一次PageRank迭代之后, 节点0 (id为0的节点)的PageRank值是多少? (5 Points)

```
def one_iter_pagerank(G, beta, r0, node_id):
    # TODO: Implement this function that takes a nx.Graph, beta, r0 and node id.
    # The return value r1 is one iteration PageRank value for the input node.
    # Please round r1 to 2 decimal places.

    r1 = 0

    ##### Your code here #####
    # Note:
    # 1. You should not use nx.pagerank
    part_1 = sum([r0 / G.degree[neighbor] for neighbor in G.neighbors(node_id)])
    part_2 = (1 - beta) / G.number_of_nodes()
    r1 = beta * part_1 + part_2
    r1 = round(r1, 2)
    #####

    return r1

beta = 0.8
r0 = 1 / G.number_of_nodes()
node = 0
r1 = one_iter_pagerank(G, beta, r0, node)
print("The PageRank value for node 0 after one iteration is {}".format(r1))
```

The PageRank value for node 0 after one iteration is 0.13

(4) 获取 karate club network 的边列表并将其转换为 torch.LongTensor.pos\_edge\_index tensor 的 torch.sum 值是多少? (10 分)

Question 5: 获取karate club network的边列表并将其转换为 torch.LongTensor. pos\_edge\_index tensor的 torch.sum 值是多少? (10 分)

```
def graph_to_edge_list(G):
    # TODO: Implement the function that returns the edge list of
    # an nx.Graph. The returned edge_list should be a list of tuples
    # where each tuple is a tuple representing an edge connected
    # by two nodes.

    edge_list = []

    ##### Your code here #####
    edge_list = [edge for edge in G.edges()]
    #####

    return edge_list

def edge_list_to_tensor(edge_list):
    # TODO: Implement the function that transforms the edge_list to
    # tensor. The input edge_list is a list of tuples and the resulting
    # tensor should have the shape [2, len(edge_list)].

    edge_index = torch.tensor([])

    ##### Your code here #####
    edge_index = torch.tensor(edge_list).t()
    #####

    return edge_index

pos_edge_list = graph_to_edge_list(G)
pos_edge_index = edge_list_to_tensor(pos_edge_list)
print("The pos_edge_index tensor has shape {}".format(pos_edge_index.shape))
print("The pos_edge_index tensor has sum value {}".format(torch.sum(pos_edge_index)))
```

The pos\_edge\_index tensor has shape torch.Size([2, 78])  
The pos\_edge\_index tensor has sum value 235

实现两个函数: graph\_to\_edge\_list 和 edge\_list\_to\_tensor。然后使用这些函数获取 karate club 网络的边列表, 并将其转换为 torch.LongTensor 格式的张量。

a) graph\_to\_edge\_list 函数将给定图 G 转换为一个边列表。这个边列表是由图中所有的边组成的, 每个边表示为一个包含两个节点的元组。此函数使用 NetworkX 库的 edges() 方法获取图 G 中所有的边, 并存储在一个列表中并返回。

b) edge\_list\_to\_tensor 函数将给定的边列表转换为一个 PyTorch 张量。这个函数

首先创建一个空的张量 `edge_index`，然后使用 PyTorch 提供的 `torch.tensor()` 方法将边列表转换为张量。而后使用 `.T` 方法来转置张量，使得每一列代表一个边，然后每一行代表边的两个节点。最后返回这个转换后的张量。

**(5) 请实现以下对负边进行采样的函数。然后回答哪些边(edge\_1 到 edge\_5)是 karate club network 中的负边?(10 分)**

“负”边是指图中不存在的边/链接。“负”一词是从链路预测中的“负抽样”借来的。这与边的权值无关。函数 `sample_negative_edges` 用于从图中采样负边：

1. 首先通过调用 `graph_to_edge_list` 函数获取 karate club 网络边列表 `pos_edge_list`;
2. 接着使用两个嵌套的循环遍历所有节点对，对于每一对节点 (`node1, node2`):
  - 如果节点 1 大于等于节点 2，或者边 (`node1, node2`) 已经存在于正边列表中，则跳过该边；
  - 否则将其添加到负边列表 `neg_edge_list` 中；
3. 最后，使用 `random.sample` 方法从负边列表中随机抽样出指定数量的负边。

接下来，对于给定的五个边 (`edge_1, edge_2, edge_3, edge_4, edge_5`)，我们将它们转换成规范形式，即保证第一个节点的编号小于第二个节点的编号。然后，我们检查它们是否在正边列表中。如果不在正边列表中，则它们就是负边，输出为 "Yes"；否则，输出为 "No"。

Question 6: 请实现以下对负边进行采样的函数。然后回答哪些边(edge\_1到edge\_5)是karate club network中的负边?(10分)

“负”边是指图中不存在的边/链接。“负”一词是从链路预测中的“负抽样”借来的。这与边的权值无关。

例如，给定一条边(src, dst)，那将检查该边(src, dst)和(dst, src)都不是图中的边，如果这两种都成立，那么它就是一条负边。

```
import random

def sample_negative_edges(G, num_neg_samples):
    neg_edge_list = []
    # ===== Your code here =====
    pos_edge_list = graph_to_edge_list(G)
    for node1 in G.nodes():
        for node2 in G.nodes():
            if node1 >= node2 or (node1, node2) in pos_edge_list:
                continue
            neg_edge_list.append((node1, node2))
    neg_edge_list = random.sample(neg_edge_list, num_neg_samples)
    # ===== Your code here =====
    return neg_edge_list

# Sample 10 negative edges
neg_edge_list = sample_negative_edges(G, len(pos_edge_list))
# Transform the negative edge list to tensor
neg_edge_index = edge_list_to_tensor(neg_edge_list)
print("The neg_edge_index tensor has shape {}".format(neg_edge_index.shape))
# Which of following edges can be negative ones?
edge_1 = (7, 1)
edge_2 = (1, 15)
edge_3 = (33, 22)
edge_4 = (0, 4)
edge_5 = (4, 2)
# ===== Your code here =====
pos_edge_list = graph_to_edge_list(G)
for edge in [edge_1, edge_2, edge_3, edge_4, edge_5]:
    edge = (edge[1], edge[0]) if edge[0] > edge[1] else edge
    output = "Yes" if edge not in pos_edge_list else "No"
    print(output)
# ===== Your code here =====
```

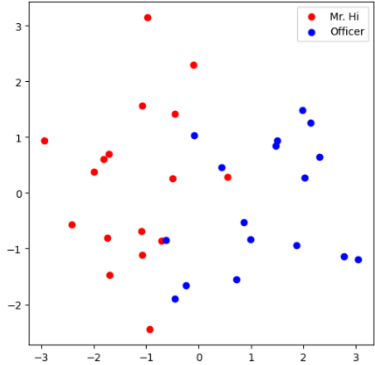
```
[10]:
-- the neg edge index tensor has shape torch.Size([10, 2])
No
Yes
No
No
Yes
```

**(6) 你能得到的最好表示是什么?请在实验报告上记录最佳损失和准确性。(20 分)**

训练 500 个 epochs，训练结果如下：

Epoch: 500, Loss: 0.016031766310334206, Acc: 1.0

可视化结果如下：



## 二、LSTM 搭建与训练

### (1) ENZYMES 数据集中的类和特征的数量是多少?(5 分)

使用 `torch_geometric.datasets` 的 `num_classes` 和 `num_features` 方法查看数据集中的类和特征的数量如下：

```
Question 1: ENZYMES数据集中的类和特征的数量是多少?(5分)

def get_num_classes(pyg_dataset):
    """ Implement a function that takes a pyg dataset object
    and returns the number of classes for that dataset.

    num_classes = 0

    ===== Your code here =====
    """
    # TODO: Implement a function that takes a pyg dataset object
    # and returns the number of classes for that dataset.
    num_classes = pyg_dataset.num_classes
    return num_classes

def get_num_features(pyg_dataset):
    """ Implement a function that takes a pyg dataset object
    and returns the number of features for that dataset.

    num_features = 0

    ===== Your code here =====
    """
    # TODO: Implement a function that takes a pyg dataset object
    # and returns the number of features for that dataset.
    num_features = pyg_dataset.num_features
    return num_features

if __name__ == '__main__':
    dataset = ENZYMES()
    num_classes = get_num_classes(dataset)
    num_features = get_num_features(dataset)
    print("ENZYMES dataset has {} classes".format(num_classes))
    print("ENZYMES dataset has {} features".format(num_features))

# Output:
# ENZYMES dataset has 6 classes
# ENZYMES dataset has 4 features
```

### (2) 在 ENZYMES 数据集中索引为 100 的图的标签是什么?(5 分)

每个 PyG dataset 都存储一个 `torch_geometric.data.Data` 列表中的 objects，其中每个 `torch_geometric.data.Data` object 都表示一个图，我们能轻松通过 dataset 的 index 找到其中的 Data object：

```
Question 2: 在ENZYMES数据集中索引为100的图的标签是什么?(5分)

def get_graph_label(pyg_dataset, idx):
    """ Implement a function that takes a PyG dataset object,
    the index of a graph within the dataset, and returns the class/label
    of the graph (as an integer).

    label = 1

    ===== Your code here =====
    """
    # TODO: Implement a function that takes a PyG dataset object,
    # the index of a graph within the dataset, and returns the class/label
    # of the graph (as an integer).
    label = pyg_dataset[idx].y[0]
    return label

if __name__ == '__main__':
    dataset = ENZYMES()
    graph_0 = pyg_dataset[0]
    idx = 100
    label = get_graph_label(pyg_dataset, idx)
    print("Graph with index {} has label {}".format(idx, label))

# Output:
# Graph with index 100 has label 4
```

### (3) 索引为 200 的图有多少条边?(5 分)

对于无向图而言，我们只需要考虑边索引矩阵中的一半。因此，我们只计算满足 `edges[0, :] <= edges[1, :]` 条件的边，并将其数量作为图的边数。

```
Question 3: 索引为200的图有多少条边?(5分)

def get_graph_num_edges(pyg_dataset, idx):
    """ Implement a function that takes a PyG dataset object,
    the index of a graph within the dataset, and returns the number of
    edges in the graph (as an integer). You should not count an edge
    twice if the graph is undirected. For example, in an undirected
    graph G, if two nodes v and u are connected by an edge, this edge
    should only be counted once.

    num_edges = 0

    ===== Your code here =====
    """
    # TODO: Implement a function that takes a PyG dataset object,
    # the index of a graph within the dataset, and returns the number of
    # edges in the graph (as an integer). You should not count an edge
    # twice if the graph is undirected. For example, in an undirected
    # graph G, if two nodes v and u are connected by an edge, this edge
    # should only be counted once.
    num_edges = pyg_dataset[idx].edge_index
    num_edges = num_edges[0, :].shape[0]
    return num_edges

if __name__ == '__main__':
    dataset = ENZYMES()
    idx = 200
    num_edges = get_graph_num_edges(pyg_dataset, idx)
    print("Graph with index {} has {} edges".format(idx, num_edges))

# Output:
# Graph with index 200 has 54 edges
```

#### (4) 在 ogbn-arxiv 图中有多少特征?(5 分)

同样使用 `torch_geometric.datasets` 的 `num_features` 方法查看数据集中特征数量如下:

```
Question 4: 在ogbn-arxiv图中有多少特征?(5分)

def graph_num_features(data):
    """Implement a function that takes a PyG data object,
    and returns the number of features in the graph (as an integer).

    num_features = 0

    ===== your code here =====
    as (a line of code)
    num_features = data.num_features
    =====

    return num_features

if 'IS_GRADSCOPE_ENV' not in os.environ:
    num_features = graph_num_features(data)
    print('The graph has {} features'.format(num_features))

[10] The graph has 128 features Python
```

#### (5) 你的 best\_model 验证和测试精度是多少?(20 分)

将输出结果与对应标签在验证和测试集上计算正确率如下:

```
Question 5: 你的best_model验证和测试精度是多少?(20分)

运行下面的单元格以查看最佳模型的结果,并将模型的预测保存到一个名为ogbn-arxiv_node.csv的文件中。在实验报告上报告结果。

if 'IS_GRADSCOPE_ENV' not in os.environ:
    best_result = test(best_model, data, split_idx, evaluator, save_model_results=True)
    train_acc, valid_acc, test_acc = best_result
    print('Best model: ')
    print(f'Train: {100 * train_acc:.2f}%, ')
    print(f'Valid: {100 * valid_acc:.2f}%, ')
    print(f'Test: {100 * test_acc:.2f}%',)

[10] ✓ 0.2s Python

Saving Model Predictions
Best model: Train: 73.71%, Valid: 71.88% Test: 71.00%
```

#### (6) 你的 best\_model 验证和测试 ROC-AUC 分数是多少?(20 分)

```
Question 6: 你的best_model验证和测试ROC-AUC分数是多少?(20分)

运行下面的单元格以查看最佳模型的结果,并将模型的预测保存在名为ogbg-molhiv_graph_valid/test.csv的文件中。在实验报告上报告结果。

if 'IS_GRADSCOPE_ENV' not in os.environ:
    train_auc = eval(best_model, device, train_loader, evaluator)[dataset.eval_metric]
    valid_auc = eval(best_model, device, valid_loader, evaluator, save_model_results=True, save_file="valid")[dataset.eval_metric]
    test_auc = eval(best_model, device, test_loader, evaluator, save_model_results=True, save_file="test")[dataset.eval_metric]

    print('Best model: ')
    print(f'Train: {100 * train_auc:.2f}%, ')
    print(f'Valid: {100 * valid_auc:.2f}%, ')
    print(f'Test: {100 * test_auc:.2f}%',)

[10] ✓ 9.2s Python

Iteration: 100% 1029/1029 [00:07<00:00, 142.25k/s]
Iteration: 100% 128/128 [00:00<00:00, 137.82k/s]

Saving Model Predictions
Iteration: 100% 128/128 [00:00<00:00, 148.00k/s]

Saving Model Predictions
Best model: Train: 84.06%, Valid: 79.56% Test: 75.25%
```

#### (7) 在 Pytorch Geometric 中使用另外两个全局池化层进行实验。

修改 `self.pool` 为 `global_max_pool` 和 `global_add_pool` 再进行测试,结果无明显差异,具体准确率如下:

池化层	Train	Valid	Test
global_mean_pool	84.06%	79.56%	75.25%
global_max_pool	85.55%	78.59%	76.10%
global_add_pool	82.30%	80.14%	75.18%

详细训练过程见附件.ipynb 文件