lab4： 图卷积神经网络模型(GCN)

姓名：李晖茜
学号：SA22218131

一、实验要求

使用 pytorch 或者 tensorflow 的相关神经网络库，编写图卷积神经网络模型(GCN)，并在相应的图结构数据集上完成节点分类和链路预测任务，最后分析自环、层数、DropEdge、PairNorm、激活函数等因素对模型的分类和预测性能的影响。

二、 实验环境

pyg	2.2.0
torch	1.10.0+cu111
torch-cluster	1.5.9
torch-scatter	2.0.9
torch-sparse	0.6.12
torchvision	0.11.0+cu111

三、实验过程

3.1 数据

Cora、Citeseer、PPI。PyG内置了大量常用的基准数据集，使用PyG内置的Planetoid数据集和PPI数据集。
使用Cara数据集，利用神经网络进行节点分类任务为例，进行本次实验。Cora数据集包含 2708 篇科学出版物（节点），总共分为7类。引文网络由 5429 个引用链接（边）组成。数据集中的每个出版物都由一个 0/1 值的词向量描述，指示字典中相应词的缺失/存在。该词典由 1433 个独特的词组成，相对于一个one hot编码的词袋向量，此向量为节点的初始特征向量（data.x，维度为[2708,1433]）。训练数据为120个带类别标签的节点，测试数据为1000个未标记的节点。

```
# 1.加载数据，数据集划分 Cora or Citeseer
dataset = Planetoid('/data/lhq/code/gcn/dataset', name='Cora',
transform=T.NormalizeFeatures()) # 包括数据集的下载，若root路径存在数据集则直接加载数据集
data = dataset[0] #该数据集只有一个图len(dataset)
# Data(edge_index=[2, 10556], test_mask=[2708],
#         train_mask=[2708], val_mask=[2708], x=[2708, 1433], y=[2708])
```

3.2 网络搭建

节点分类任务是根据已知类别标签的节点和节点特征的映射，对未知类别标签节点进行类别标签标注。设计图神经网络GCN，使用两个`GCNConv`层，一个`ReLU`非线性层和一个`dropout`操作。第一个`GCNConv`层将1433维的特征向量嵌入（embedding）到低维空间中（hidden_channels=16），经过`ReLU`层激活，再经过`dropout`操作，输入第二个`GCNConv`层——将低维节点表征嵌入到类别空间中（num_classes=7）。

```
# 2.设计网络
from conv import GraphConvolution as GCNConv
class GCN(torch.nn.Module):
```

```
    # 初始化
    def __init__(self, hidden_channels):
        super(GCN, self).__init__()
        torch.manual_seed(12345)
        self.conv1 = GCNConv(dataset.num_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, dataset.num_classes)
    # 前向传播
    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        # 注意这里输出的是节点的特征，维度为[节点数,类别数]
        return x
```

其中`GCNConv`模块，实现如下：

```
class GraphConvolution(nn.Module):
    def __init__(self, input_dim, output_dim, use_bias=True):
        """图卷积：L*X*\theta
        Args:
        ----------
            input_dim: int
                节点输入特征的维度
            output_dim: int
                输出特征维度
            use_bias : bool, optional
                是否使用偏置
        """
        super(GraphConvolution, self).__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.use_bias = use_bias
        self.weight = nn.Parameter(torch.Tensor(input_dim, output_dim))
        if self.use_bias:
            self.bias = nn.Parameter(torch.Tensor(output_dim))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        init.kaiming_uniform_(self.weight)
        if self.use_bias:
            init.zeros_(self.bias)

    def forward(self, input_feature, adjacency):
        """邻接矩阵是稀疏矩阵，因此在计算时使用稀疏矩阵乘法

        Args:
        -------
            adjacency: torch.sparse.FloatTensor
                邻接矩阵
            input_feature: torch.Tensor
                输入特征
        """
        device = "cuda" if torch.cuda.is_available() else "cpu"
        support = torch.mm(input_feature, self.weight.to(device))
```

```
        output = torch.sparse.mm(adjacency, support)
        if self.use_bias:
            output += self.bias.to(device)
        return output

    def __repr__(self):
        return self.__class__.__name__ + ' (' + str(self.in_features) + ' -> ' +
str(self.out_features) + ')'
```

3.3 网络训练

选择Adam作为优化器，交叉熵(CrossEntropy)作为loss function，对网络进行训练。设置100个epoch，记录最佳测试结果。

```
if __name__ == '__main__':
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = GCN(hidden_channels=16).to(device) # 实例化模型
    data = data.to(device)
    # 选择优化器
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
    # 选择loss，二值交叉熵
    criterion = torch.nn.CrossEntropyLoss()
    best_test_acc = 0
    # 训练、验证和测试保存模型
    for epoch in range(1,101):
        loss = train(model,data,optimizer,criterion)
        val_acc, test_acc = test(model,data)
        if test_acc > best_test_acc:
            best_test_acc = test_acc
        print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}, Val: {val_acc:.4f}, Test:
{test_acc:.4f}')
    print(f'Test: {best_test_acc:.4f}')
```

由于数据集较小，没有分batch训练（直接作为1个batch），经过梯度置零，模型前向传播，计算loss，反向传播，优化器梯度下降，完成一个epoch的训练。

```
# 3.训练模型
def train(model,data,optimizer,criterion):
    model.train()
    optimizer.zero_grad()  # 梯度置零
    out = model(features, adj)  # 模型前向传播
    loss = criterion(out[data.train_mask],data.y[data.train_mask])  # 计算loss
    loss.backward()  # 反向传播
    optimizer.step()  # 优化器梯度下降
    return loss
```

3.5 测试性能

`model.eval()` 开启模型的测试模式，利用训练好模型中的各层权重矩阵聚合各层邻居节点的消息，预测目标结点的特征。

```
# 4.测试
@torch.no_grad()
def test(model,data):
    model.eval()
    out = model(features, adj)
    pred = out.argmax(dim=1)  # 使用最大概率的类别作为预测结果
    val_correct = pred[data.val_mask] == data.y[data.val_mask]  # 获取正确标记的节点
    val_acc = int(val_correct.sum()) / int(data.val_mask.sum()) # 计算正确率
    test_correct = pred[data.test_mask] == data.y[data.test_mask]  # 获取正确标记的
节点
    test_acc = int(test_correct.sum()) / int(data.test_mask.sum()) # 计算正确率
    return val_acc, test_acc
```

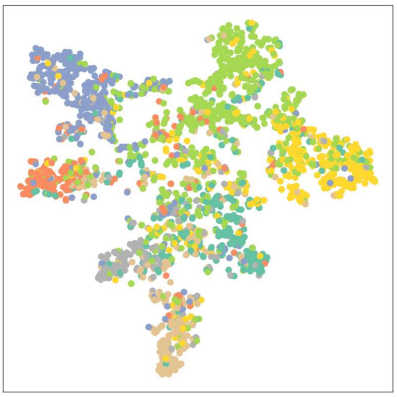
利用模型输出的节点特征降维进行可视化，用TSNE降维方法将节点特征降至2维，在坐标系中可视化。

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

#可视化
def visualize(out, color):
    z = TSNE(n_components=2).fit_transform(out.detach().cpu().numpy())
    plt.figure(figsize=(10,10))
    plt.xticks([])
    plt.yticks([])
    plt.scatter(z[:, 0], z[:, 1], s=70, c=color, cmap="Set2")
    plt.savefig('/data/lhq/code/gcn/show.png')
    plt.close()

model.eval()
out = model(features, adj)
visualize(out, color=data.y.cpu().numpy())
```

四、实验结果

4.1 节点分类实验结果

使用以上默认参数进行训练测试，测试最佳`Test: 0.5235`，可视化结果如下，左图分类前，右图分类后：



4.2 节点分类参数分析

分析自环、层数、DropEdge、PairNorm、激活函数等因素对模型的分类和预测性能的影响。

- 将`GCNConv`替换为`pytorch_geometric`库中的`GCNConv`层，便于以下操作；
- 通过参数`add_self_loops`设置是否添加自环；
- 设置`n_layers`，调整网络层数；
- 设置`drop_edge`的drop比例去掉部分`edge_index`；
- 在每层 GCN 后都紧跟一层`PairNorm`层；
- 激活函数可以选择`relu, tanh, sigmoid`。

重新设计网络，添加上述参数：

```
class GCN(torch.nn.Module):
    # 初始化
    def __init__(self, hidden_channels, n_layers, act: str = 'relu',
add_self_loops: bool = True,\
                    pair_norm: bool = True, dropout: float = .0, drop_edge:
float =.0):
        super(GCN, self).__init__()
        self.dropout = dropout
        self.drop_edge = drop_edge
        self.pair_norm = pair_norm
        self.act = activations[act] if isinstance(act, str) else act
        self.conv_list = torch.nn.ModuleList()
        for i in range(n_layers):
            in_c, out_c = hidden_channels, hidden_channels
            if i == 0:
                in_c = dataset.num_features
            elif i == n_layers - 1:
                out_c = dataset.num_classes
            self.conv_list.append(GCNConv(in_c, out_c,
add_self_loops=add_self_loops))
    def forward(self, x, edge_index):
        edge_index, _ = dropout_adj(edge_index, p=self.drop_edge)
        for i, conv in enumerate(self.conv_list):
            x = conv(x, edge_index)
            if self.pair_norm:
                x = PairNorm()(x)
            if i < len(self.conv_list) - 1:
                x = self.act(x)
                x = F.dropout(x, p=self.dropout, training=self.training)
        return x
```

设置不同参数：

```
model = GCN(hidden_channels=16, n_layers=2, act='relu', add_self_loops=True, \
                pair_norm=True, dropout=.0, drop_edge=.0).to(device) # 实例化模型
```

固定其他，`n_layers`设为2，3，4，结果分别为 0.4354，0.4004，0.3614；
固定其他，`act`设为'relu'，'sigmoid'，'tanh'，结果分别为 0.4234，0.3353，0.4344；
固定其他，`add_self_loops`设为True，False，结果分别为 0.4374，0.4184；
固定其他，`pair_norm`设为True，False，结果分别为 0.4314，0.5045；
固定其他，`dropout`设为.0，.1，.2，.5，结果分别为 0.5165，0.5205，0.5185，0.5115；
固定其他，`drop_edge`设为.0，.1，.2，.5，结果分别为 0.5205，0.5225，0.5305，0.5365。

4.3 Citeseer和PPI节点分类

按照如下设置，在Citeseer和PPI数据集上训练测试。

```
model = GCN(hidden_channels=16, n_layers=2, act='tanh', add_self_loops=True, \
                pair_norm=False, dropout=.1, drop_edge=.5).to(device) # 实例化模型
```

在Citeseer数据集上进行训练测试，将`Planetoid`的`name`参数改为`'Citeseer'`，结果为Test: 0.6810。在PPI数据集上进行训练测试，加载数据集如下，结果为Test: 0.7160。

```
from torch_geometric.datasets import PPI

# 1.加载数据，数据集划分
dataset = PPI("/data/lhq/code/gcn/dataset/PPI")
data = dataset[0] #该数据集只有一个图len(dataset)
```

4.4 Cora和Citeseer链路预测

网络中的链路预测(Link Prediction)是指如何通过已知的网络节点以及网络结构等信息预测网络中尚未产生连边的两个节点之间产生链接的可能性。这种预测既包含了对未知链接的预测，也包含了对未来链接的预测。
利用PyG的工具函数`train_test_split_edges(data,val_ratio=0.05, test_ratio=0.1)`将数据划分成：训练集、验证集、测试集三个部分。该函数返回测试集和验证集的正/负链路样本索引列表，以及训练集的负链路mask矩阵。

- 链路负采样

负样本与正样本相对，比如数据集中两个节点间存在链接，那么为一个正样本，两个节点间在已知数据集中不存在链接，那么构成一个负样本。根据数据集中的正样本，可以反向获取到大量的负样本，从中随机采集一部分负样本用于训练。
利用`train_test_split_edges()`函数获取了测试集和验证集的正/负链路样本集合，存储在`data['val_pos_edge_index']`、`data['test_pos_edge_index']`、`data['val_neg_edge_index']`、`data['test_neg_edge_index']`中。在训练过程中，需要利用`negative_sampling`函数仅基于训练集进行负样本采样（也就是"第二次"负采样）。

```
# 负采样代码
from torch_geometric.utils import negative_sampling
neg_edge_index = negative_sampling(
    edge_index=data.train_pos_edge_index,
    num_nodes=data.num_nodes,
    num_neg_samples=data.train_pos_edge_index.size(1))
```

- 设计网络，利用`torch_geometric.nn`的GCNConv模块，设计如下网络：

```
# 2.设计网络
class Net(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Net, self).__init__()
        self.conv1 = GCNConv(in_channels, 128)
        self.conv2 = GCNConv(128, out_channels)

    def encode(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = x.relu()
```

```
        return self.conv2(x, edge_index)

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)  #
[2,E]
        return (z[edge_index[0]] * z[edge_index[1]]).sum(dim=-1)  # *: element-
wise乘法

    def decode_all(self, z):
        prob_adj = z @ z.t()  # @: 矩阵乘法，自动执行适合的矩阵乘法函数
        return (prob_adj > 0).nonzero(as_tuple=False).t()

    def forward(self, x, pos_edge_index, neg_edge_index):
        return self.decode(self.encode(x, pos_edge_index), pos_edge_index,
neg_edge_index)
```

用于做边预测的神经网络主要由两部分组成：其一是编码（encode），其二是解码（decode），边两端节点的表征生成边为真的几率（odds）。预测是将模型训练得到的两个节点特征计算存在连边的概率，得到概率矩阵。其中，`decode`函数预测样本节点链路存在的概率，`decode_all`函数用于推理阶段预测所有节点间链路存在的概率。
同样优化器选择Adam，损失函数选择二值交叉熵，训练函数为：

```
# 3.训练模型
def train(data, model, optimizer, criterion):
    model.train()

    # 链路负采样
    neg_edge_index = negative_sampling(  # 训练集负采样，每个epoch负采样样本可能不同
        edge_index=data.train_pos_edge_index,
        num_nodes=data.num_nodes,
        num_neg_samples=data.train_pos_edge_index.size(1))

    optimizer.zero_grad()
    # link_logits = model(data.x, data.train_pos_edge_index, neg_edge_index)
    z = model.encode(data.x, data.train_pos_edge_index)
    link_logits = model.decode(z, data.train_pos_edge_index, neg_edge_index)
    link_labels = get_link_labels(data.train_pos_edge_index,
neg_edge_index).to(data.x.device)  # 训练集中正样本标签
    loss = criterion(link_logits, link_labels)
    loss.backward()
    optimizer.step()

    return loss
```

最后用`sigmoid()`计算链路存在的概率，判断某个节点和其他多个节点是否存在链接。
在Cora数据集上结果为Test: 0.8547；在Citeseer数据集上结果为Test: 0.8970。

五、实验总结

根据此次实验，学习了解了图神经网络，学习使用`torch_geometric`库以及自己编写`GCNConv`模块。了解使用Cora、CIteceer和PPI数据集，初步完成了图神经网络相关的实验和应用。