深度学习 Lab3

PB20111686 黄瑞轩

1 实验目标

使用 pytorch 或者 tensorflow 编写图卷积神经网络模型GCN，并在图结构数据集 Cora 上完成节点分类和链路预测任务，研究自环、层数、DropEdge、PairNorm、激活函数等因素对节点分类和链路预测性能的影响。

2 实验过程和关键代码展示

注：为了展示核心功能，某些代码中一些无关紧要的部分在报告中被删去了。

2.1 数据集说明

本次实验的数据集为 Cora 数据集。该数据集是由 2708 篇机器学习论文作为节点、论文间引用关系作为有向边构成的图数据。

PyG 提供了专门载入、分割 Cora 数据集的接口，由于助教提供的数据集无法被 PyG 直接读取，所以 Cora 数据集使用 PyG 自动下载的版本。

```
import torch_geometric.transforms as T

# 读取数据集
dataset = Planetoid(root='./data/Cora', name='Cora',
transform=T.NormalizeFeatures())
data = dataset[0]

# 分成训练集、测试集和验证集
split = T.RandomNodeSplit(num_val=0.1, num_test=0.2)
data = split(data)
```

2.2 实验过程说明

首先，基于 MessagePassing 模型创建图卷积层。

```
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super().__init__(aggr='add')
        self.lin = Linear(
            in_channels, out_channels, bias=False)
        self.bias = Parameter(torch.Tensor(out_channels))
        self.reset_parameters()

    def reset_parameters(self):
        self.lin.reset_parameters()
```

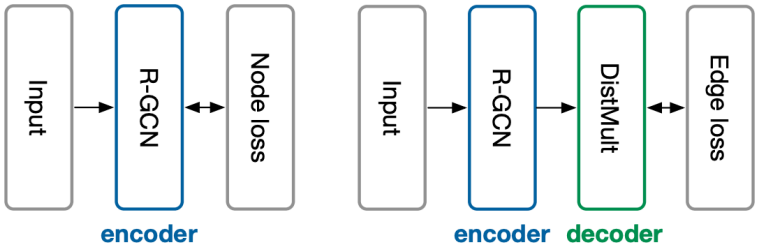
```
        self.bias.data.zero_()

    def forward(self, x, edge_index):
        # 设置自环，DropEdge，略
        x = self.lin(x)
        row, col = edge_index
        deg = degree(col, x.size(0), dtype=x.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        deg_inv_sqrt[deg_inv_sqrt == float('inf')] = 0
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        out = self.propagate(edge_index, x=x, norm=norm)
        return out + self.bias

    def message(self, x_j, norm):
        return norm.view(-1, 1) * x_j
```

然后即可构建 GCN 网络，如图所示是节点分类、链路预测分别对应的网络结构。



(a) Entity classification (b) Link prediction

```
class GCN(nn.Module):
    def __init__(self, input, hidden, output):
        super().__init__()
        self.conv1 = GCNConv(input, hidden)
        self.conv2 = GCNConv(hidden, output)

    def encode(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        return self.conv2(x, edge_index)

    def decode(self, z, edge_label_index):
        return (z[edge_label_index[0]] * z[edge_label_index[1]]).sum(dim=-1)

    def decode_all(self, z):
        prob_adj = z @ z.t()
        return (prob_adj > 0).nonzero(as_tuple=False).t()

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.encode(x, edge_index)
        return F.log_softmax(x, dim=1)
```

3 超参数的调节

本实验采用单一变量的对照法，即在一组训练结果不错的超参数上每次选择一个超参数进行调参，然后将每个超参数最好的取值组合在一起。

基准超参数设置如下所示：

自环	图卷积层数	DropEdge	PairNorm	激活函数
False	2	False	True	ReLU

节点分类任务评价指标选用准确率（Accuracy），链路预测任务评价指标选用 AUC。使用这一设置，两任务的指标（三次运行取平均值，下同）如下：

节点分类 Accuracy	链路预测 AUC
84.31%	0.889

3.1 自环的影响

在计算 $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ 时，其特征值在 $[0, 2]$ 内，多层叠加时容易发生数值不稳定现象。给图添加自环（$\tilde{A} = A + I$）理论上可以明显降低不稳定性。

在基准超参数设置基础上改成有自环的，即：

自环	图卷积层数	DropEdge	PairNorm	激活函数
True	2	False	True	ReLU

对比添加自环的指标：

自环	节点分类 Accuracy	链路预测 AUC
False	84.31%	0.889
True	**90.41%**	**0.905**

可以发现自环对效果提升较为明显，节点分类的 Accuracy 和链路预测的 AUC 均有提升，其中节点分类的 Accuracy 提升明显。所以在后续的基础超参数中将自环设置为 True。

3.2 图卷积层数的影响

基础超参数使用了两个图卷积层，下面将卷积层数增加。

自环	图卷积层数	DropEdge	PairNorm	激活函数
True	2	False	True	ReLU
True	3	False	True	ReLU
True	4	False	True	ReLU
True	5	False	True	ReLU

卷积层数增加之后，节点分类和链路预测指标：

图卷积层数	节点分类 Accuracy	链路预测 AUC
2	**90.41%**	**0.905**
3	86.35%	0.824
4	85.06%	0.735
5	83.39%	0.793

可以看到，随着卷积层数的增加，节点分类的准确率有所下降，而链路预测的 AUC 得分也呈现总体下降的趋势。这表明增加卷积层数可能会导致过度拟合，从而降低模型的泛化能力。在后续的基础超参数中，仍将 图卷积层数设置为 2。

3.3 DropEdge 的影响

为了允许图神经网络的深度加深，研究者们提出了 DropEdge 的训练策略。这种策略会在训练时随机丢弃一些边（丢弃的比例以参数 `drop_rate` 指定），以缓解更深的图卷积网络中出现的过度平滑的问题。

```python
def drop_edge(edge_index, drop_rate=0.5):
    num_keep = int(edge_index.shape[1] * (1 - drop_rate))
    temp = [True] * num_keep +
        [False] * (edge_index.shape[1] - num_keep)
    random.shuffle(temp)
    return edge_index[:, temp]
```

首先测试 DropEdge 在基础参数上的效果，然后再对比在不同深度 GCN 上的效果，综合判断 DropOut 参数的设置。

3.3.1 在基础参数上的效果

使用的配置如下，记 drop_rate $= p$:

自环	图卷积层数	DropEdge	PairNorm	激活函数
True	2	True, $p = 0.1$	True	ReLU
True	2	True, $p = 0.2$	True	ReLU
True	2	True, $p = 0.5$	True	ReLU

节点分类和链路预测指标:

DropEdge	节点分类 Accuracy	链路预测 AUC
False	90.41%	0.905
True, $p = 0.1$	88.37%	**0.920**
True, $p = 0.2$	**90.59%**	0.914
True, $p = 0.5$	86.35%	0.901

可以看到，添加了 DropEdge 策略后，节点分类的 Accuracy 和链路预测的 AUC 均有提升。权衡 Accuracy 和 AUC，后续的 DropEdge 都取 $p = 0.2$。

3.3.2 不同 GCN 深度上的效果

使用的配置如下:

自环	图卷积层数	DropEdge	PairNorm	激活函数
True	3	True, $p = 0.2$	True	ReLU
True	4	True, $p = 0.2$	True	ReLU
True	5	True, $p = 0.2$	True	ReLU

节点分类和链路预测指标:

DropEdge	图卷积层数	节点分类 Accuracy	链路预测 AUC
True, $p = 0.2$	2	**90.59%**	**0.914**
True, $p = 0.2$	3	85.61%	0.743
True, $p = 0.2$	4	81.73%	0.851
True, $p = 0.2$	5	81.70%	0.747

和无 DropEdge 的对比:

图卷积层数	节点分类 Accuracy	链路预测 AUC
2	**90.41%**	**0.905**
3	86.35%	0.824
4	85.06%	0.735
5	83.39%	0.793

可以看到，DropEdge 策略在一定程度上延缓了卷积层数增多时的 Accuracy 下降，但是整体而言效果不佳，所以仅采纳 DropEdge $p = 0.2$，不采纳卷积层数的继续增加。

3.4 PairNorm 的影响

PairNorm 是为了解决过平滑而提出的正则化方法，因为图卷积本质上是一种局部平滑，这意味着进行卷积层如果比较深，最终输出的顶点特征趋于相同。PairNorm 的引入理论上可以减缓这个问题。

在基准超参数设置基础上改成不使用 PairNorm 的，即:

自环	图卷积层数	DropEdge	PairNorm	激活函数
True	2	True, $p = 0.2$	False	ReLU

训练、运行任务后，对比指标:

PairNorm	节点分类 Accuracy	链路预测 AUC
True	**90.59%**	**0.914**
False	85.60%	0.911

可以看到，当不使用 PairNorm 时，节点分类的 Accuracy 和链路预测 AUC 都下降了，这表明 PairNorm 对于缓解过度平滑问题的效果是显著的，能够提高节点分类和链路预测的性能。

3.5 激活函数的影响

看几个不同的激活函数上问题的表现：

自环	图卷积层数	DropEdge	PairNorm	激活函数
True	2	True, $p = 0.2$	True	Tanh
True	2	True, $p = 0.2$	True	Sigmoid

训练、运行任务后，对比指标：

激活函数	节点分类 Accuracy	链路预测 AUC
ReLU	**90.59%**	**0.914**
Sigmoid	70.66%	0.738
Tanh	87.45%	0.908

可以看到，当使用 Sigmoid 激活函数时，节点分类和链路预测的指标都显著下降，这表明 Sigmoid 激活函数对于图神经网络的性能不太适用。相比之下，使用 Tanh 激活函数的模型在节点分类和链路预测任务上都是稍微劣于 ReLU，因此对于这两个任务，使用 ReLU 激活函数是最好的选择。

4 最终参数及测试

经过调参，最终选定的参数如下：

Epoch 大小：600

网络深度：2 层卷积层

DropEdge 设置：$p = 0.2$

正则化设置：使用 PairNorm

学习率设置：0.01，节点分类任务上 weight_decay $= 5 \times 10^{-4}$，链路预测上固定为 0.01

激活函数：ReLU

引入自环：是

```
..........................................................
Test Accuracy: 0.8911439114391144
..........................................................
Test AUC: 0.8930648221827753
```

测试集上指标：

节点分类 Accuracy	链路预测 AUC
89.11%	0.893