GCN原理与实现

卷积公式：$f * g = F^{-1}(F(f) \cdot F(g))$

给定一个图信号x和一个卷积核g，$x * g = U(U^T x \odot U^T g) = U(U^T x \odot g_\theta) = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}X\Theta$

其中A为图的邻接矩阵，D为图的度数矩阵，

$\widetilde{D} = D + \gamma I$，$\widetilde{A} = A + \gamma I$，添加自环使得Laplace matrix 的特征值变小

1.下面是$\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$的计算

```
def gcn_norm(edge_index, edge_weight=None, num_nodes=None,
             add_self_loops=True, flow="source_to_target", dtype=None):

    fill_value = 1.

    num_nodes = maybe_num_nodes(edge_index, num_nodes)

    if add_self_loops:  #添加自环
        edge_index, edge_weight = add_remaining_self_loops(
            edge_index, edge_weight, fill_value, num_nodes)

    edge_weight = torch.ones((edge_index.size(1), ), dtype=dtype,
                                 device=edge_index.device)

    row, col = edge_index[0], edge_index[1]
    idx = col
    deg = scatter(edge_weight, idx, dim=0, dim_size=num_nodes, reduce='sum')
    deg_inv_sqrt = deg.pow_(-0.5)
    deg_inv_sqrt.masked_fill_(deg_inv_sqrt == float('inf'), 0)
    edge_weight = deg_inv_sqrt[row] * edge_weight * deg_inv_sqrt[col]

    return edge_index, edge_weight
```

代码解释

`edge_index, edge_weight = add_remaining_self_loops(edge_index, edge_weight, fill_value, num_nodes)`

$\widetilde{D} = D + \gamma I$，$\widetilde{A} = A + \gamma I$

`deg = scatter(edge_weight, idx, dim=0, dim_size=num_nodes, reduce='sum')`：根据edge_weight和idx=edge_index[1]得到度数矩阵，deg=D

`deg_inv_sqrt = deg.pow_(-0.5)`：deg_inv_sqrt=$D^{-0.5}$，

`deg_inv_sqrt.masked_fill_(deg_inv_sqrt == float('inf'), 0)`：由于D非对角元=0，其-0.5次幂=∞，需要转化为0，

`edge_weight = deg_inv_sqrt[row] * edge_weight * deg_inv_sqrt[col]`：输出归一化后的edge_index

2. PairNorm

> 对于每个节点，我们可以将其连接的所有邻居节点的特征向量的平均范数作为归一化因子，然后将该节点的特征向量除以该归一化因子，得到归一化后的节点特征向量。这样，就可以保证所有节点的特征向量范数大小相同，从而解决了特征放缩不平衡的问题。

$$\mathbf{x}_i^c = \mathbf{x}_i - \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

$$\mathbf{x}_i' = s \cdot \frac{\mathbf{x}_i^c}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\|\mathbf{x}_i^c\|_2^2}}$$

过于复杂，调库实现→`from torch_geometric.nn import PairNorm`

3.GCNConv的实现如下(删改自torch_geometric.nn.GCNConv)

```
class myGCNConv2(MessagePassing):
    def __init__(self, in_channels: int, out_channels: int,
                 add_self_loops: bool = True,bias: bool = True):
        super().__init__()

        self.in_channels = in_channels
        self.out_channels = out_channels
        self.add_self_loops = add_self_loops

        self.lin = Linear(in_channels, out_channels, bias=False,
                          weight_initializer='glorot')

        if bias:
            self.bias = Parameter(torch.Tensor(out_channels))
        else:
            self.register_parameter('bias', None)

        self.reset_parameters()

    def reset_parameters(self):
        super().reset_parameters()
        self.lin.reset_parameters()          #卷积层
        zeros(self.bias)                     #偏置层

    def forward(self, x: Tensor, edge_index: Adj,
                edge_weight: OptTensor = None) -> Tensor:

        edge_index, edge_weight = gcn_norm(  # yapf: disable
            edge_index, edge_weight, x.size(self.node_dim),
            self.add_self_loops, self.flow, x.dtype)

        x = self.lin(x)

        # propagate_type: (x: Tensor, edge_weight: OptTensor)
        out = self.propagate(edge_index, x=x, edge_weight=edge_weight,
                             size=None)

        if self.bias is not None:
            out = out + self.bias
```

```
        return out

    def message(self, x_j: Tensor, edge_weight: OptTensor) -> Tensor:
        return x_j if edge_weight is None else edge_weight.view(-1, 1) * x_j

    def message_and_aggregate(self, adj_t: SparseTensor, x: Tensor) -> Tensor:
        return spmm(adj_t, x, reduce=self.aggr)
```

代码解释

`x = self.lin(x)` 对应公式：x'=$X\Theta$

`out = self.propagate(edge_index, x=x, edge_weight=edge_weight,size=None)` 对应公式：

out=$A'x'=\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}X\Theta$

message&message_and_aggregate为MessagePassing.propagate的相关函数，

经测试，删除后，val acc下降，故予以保留

4.Net(GCN)的实现

```
class GCN(torch.nn.Module):
    def __init__(
        self,
        num_node_features: int,
        num_classes: int,
        hidden_dim: int = 16,
        dropout_rate: float = 0.5,
    ) -> None:
        super().__init__()
        self.dropout1 = torch.nn.Dropout(dropout_rate)
        self.conv1 = myGCNConv2(num_node_features,
hidden_dim,add_self_loops=True)
        self.relu = torch.nn.ReLU(inplace=True)
        self.dropout2 = torch.nn.Dropout(dropout_rate)
        self.conv2 = myGCNConv2(hidden_dim, num_classes,add_self_loops=True)
        self.pn=PairNorm()


    def forward(self, x: Tensor, edge_index: Tensor) -> torch.Tensor:
        x = self.pn(x)
        x = self.dropout1(x)
        x = self.conv1(x, edge_index)
        x = self.relu(x)
        x = self.dropout2(x)
        x = self.conv2(x, edge_index)
        return x
```

代码解释

`x = self.pn(x)` 对x作PairNorm处理，之后x^i~N(0,s^2)，各节点特征范数大小平衡，作用不明显

采用2层GCN卷积层，中间用relu激活，dropout避免过拟合

5.DropEdge realization

idea

1. 首先把有向图的边，转化为无向图的边，保存在single_edge_index中，实现时先用single_edge字典存储每条无向边(key-value 任意)1次，再把single_edge转化成无向图的边集索引(2-dim tensor array)

```
#single_edge_index
single_edge={}
for i in range(len(dataset.data.edge_index[0])):
    if(((dataset.data.edge_index[0][i],dataset.data.edge_index[1][i]) not in
single_edge.items()) and
        ((dataset.data.edge_index[1][i],dataset.data.edge_index[0][i]) not in
single_edge.items())):
        single_edge[dataset.data.edge_index[0][i]]=dataset.data.edge_index[1][i]

single_edge_index=[[],[]]

for key,value in single_edge.items():
    single_edge_index[0].append(key)
    single_edge_index[1].append(value)

single_edge_index=torch.tensor(single_edge_index)
```

2. 再把无向边集舍去dropout_rate比例的部分，之后转成有向边集索引

```
def drop_edge(single_edge_index, dropout_rate):
    # 计算需要丢弃的边数
    num_edges = single_edge_index.shape[1]
    num_drop = int(num_edges * dropout_rate)

    # 随机选择要丢弃的边
    remain_indices = torch.randperm(num_edges)[num_drop:]
    remain_single_edges = single_edge_index[:, remain_indices]
    reverse_edges =
torch.stack([remain_single_edges[1],remain_single_edges[0]],dim=0)
    remain_edges=torch.cat([remain_single_edges,reverse_edges],dim=1)

    return remain_edges
```

5. 其他细节

- net.train()和net.eval()是用于设置模型训练和评估的状态的方法。

net.train()用于设置模型为训练状态，它会启用一些特定于训练的层（如Dropout和BatchNorm），并关闭一些特定于评估的层（如Dropout）。此外，该方法还会使得每次训练进行反向传播更新参数。

net.eval()用于设置模型为评估状态，它会关闭一些特定于训练的层（如Dropout和BatchNorm），并启用一些特定于评估的层（如Dropout）。此外，该方法还会停止梯度的计算，该方法适用于在模型参数固定后进行推理。

节点分类

流程:

1. 图卷积GCN得到各节点的7维向量表示logits，train_set，val_set，test_set的划分采用掩码机制，只对mask对应的节点的out和label训练

```
    logits = model(data.x, drop_edge(single_edge_index,dp))[mask]
```

2. logits和真实标签y作交叉熵损失，用于训练
3. 和CNN分类一样，取最大值为标签，用于评估

```
    preds = logits.argmax(dim=1)
```

Note

值得一提的是，无论是train_set，valid_set，test_set，GCN的输入均为data.x，data.edge_index，而非data.x的子集，这是因为data.edge_index要求，输入节点特征向量按序；

链路预测

流程

1. GCN得到各节点的特征表示，z[0]~z[2707]
2. 对于边i=（edge_index[0][i]，edge_index[1][i]），logits[i]=两个端点的特征表示的内积，作为边i的正负性(正=有边)
3. when training，用logits和link_labels作binary_cross_entropy来训练，希望logits来接近0或1(负边或正边)

   最后根据logits>=0.5 or not 来得到图上所有边的估计
4. when validate/test，对logits作sigmoid变换，得到各条边为正的估计，再用AUC函数计算估计的准确度

PS：DropEdge只要对train_set，

Net

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(tg_data.num_features, 128)
        self.conv2 = GCNConv(128, 64)
        self.pn = PairNorm()

    def encode(self):
        x = self.pn(tg_data.x)
        x = self.conv1(x, tg_data.train_pos_edge_index)
        x = x.relu()
        return self.conv2(x, tg_data.train_pos_edge_index)

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=-1)      #xi,xj为
边的两个端点的向量表示,sum(xi·xj)得到边的值,
```

```
        return logits

    def decode_all(self, z):
        prob_adj = z @ z.t()
        return (prob_adj > 0).nonzero(as_tuple=False).t()                #得到整个
图上边的预测
```

解释

1. encode：2层GCN函数得到节点的特征表示
2. decode：对要预测的边集[pos_edge_index，neg_edge_index]，对两个端点的向量表示作内积，表示边的正负性
3. decode_all：特征矩阵z@z.t()，得到所有边的正负性矩阵，按>=0为正边，<0为负边，得到整个图的(正)边预测

优化与比较

优化方式

1. learning rate modification，对照组不做学习率优化
2. PairNorm
3. add_self_loops
4. dropEdge
5. 层数
6. 激活函数
7. dropout(默认)

Declare

在我们的对照组中，包含了上述所有优化方式，在每个实验组中去掉一个因素，来比较各因素的影响

各组说明：

1. group1：先不变学习率，让loss function快速下降到极值点附近，再缓慢调整学习率，逼近极值点

```
        if epoch%5==0 and epoch>=70:
            optimizer.param_groups[0]['lr']*=0.8
```

2. group2：不做PairNorm优化
3. group3：没有add_self_loops
4. group4：未作dropedge
5. group5：1层GCNGonv

```
        super().__init__()
        self.dropout1 = torch.nn.Dropout(dropout_rate)
        self.conv1 = myGCNConv2(num_node_features, num_classes, add_self_loops=True)
        self.relu = torch.nn.ReLU(inplace=True)
        self.lrelu = torch.nn.LeakyReLU()
        self.sigm = torch.nn.Sigmoid()
        self.dropout2 = torch.nn.Dropout(dropout_rate)
        self.conv2 = myGCNConv2(hidden_dim, num_classes,add_self_loops=True)
        self.pn=PairNorm()

    def forward(self, x: Tensor, edge_index: Tensor) -> torch.Tensor:
        x = self.pn(x)
        x = self.dropout1(x)
        x = self.conv1(x, edge_index)
        x = self.relu(x)
        # x = self.dropout2(x)
        # x = self.conv2(x, edge_index)      You, 1秒钟前 • Uncommitted changes
        return x
```

3层GCNConv

```
158 class GCN(torch.nn.Module):
159     def __init__(
160         self,
161         num_node_features: int,
162         num_classes: int,
163         hidden_dim: int = 16,
164         dropout_rate: float = 0.5,
165     ) -> None:
166         super().__init__()
167         self.dropout1 = torch.nn.Dropout(dropout_rate)
168         self.conv1 = myGCNConv2(num_node_features, hidden_dim,add_self_loops=True)
169         self.relu = torch.nn.ReLU(inplace=True)
170         self.dropout2 = torch.nn.Dropout(dropout_rate)
171         self.conv2 = myGCNConv2(hidden_dim, hidden_dim,add_self_loops=True)
172         self.conv3 = myGCNConv2(hidden_dim, num_classes,add_self_loops=True)
173         self.pn=PairNorm()
174
175     def forward(self, x: Tensor, edge_index: Tensor) -> torch.Tensor:
176         x = self.pn(x)
177         x = self.dropout1(x)
178         x = self.conv1(x, edge_index)
179         x = self.relu(x)
180         x = self.dropout2(x)
181         x = self.conv2(x, edge_index)
182         x = self.relu(x)
183         x = self.conv3(x,edge_index)
184         return x      You, 前天 • cp deephub
```

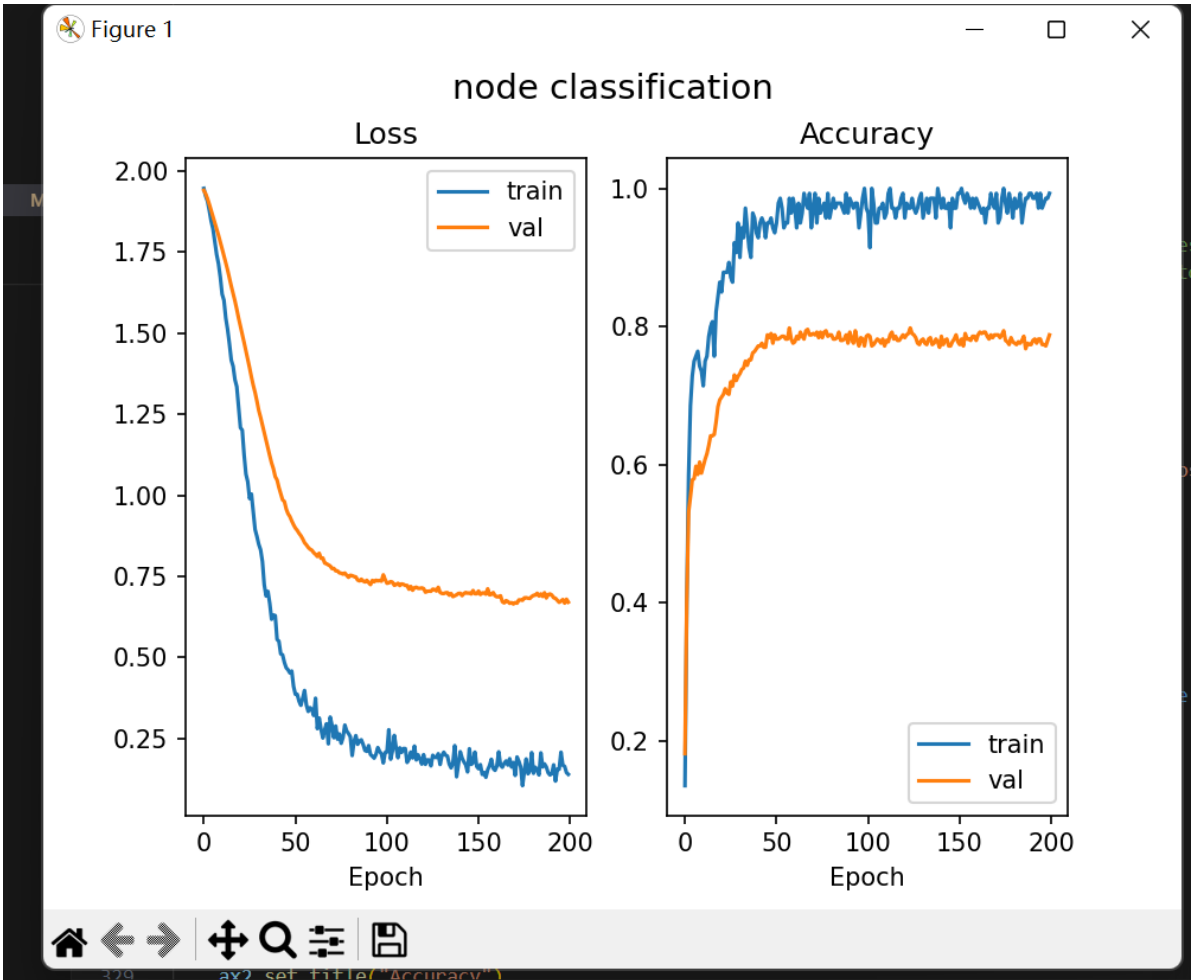
6. group6：采用sigmoid激活函数
7. group7：舍去dropout层

节点分类

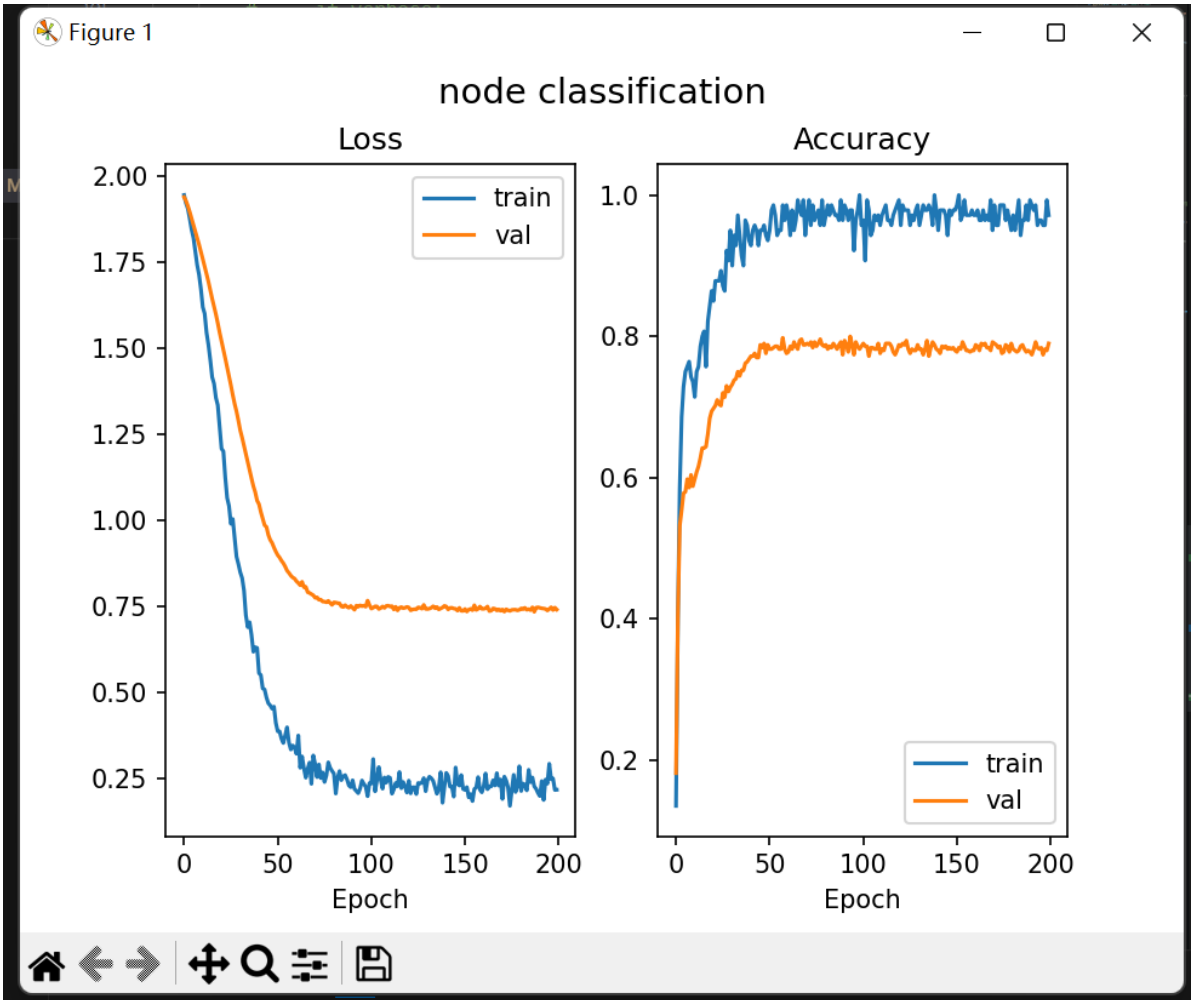
Group	Epoch 199: Train loss	Train acc	Val loss	Val acc	Test loss	Test acc
对照组	0.1399	0.9929	0.6707	0.7880	0.6330	0.8040
1	0.2167	0.9714	0.7406	0.7900	0.6959	0.8040
2	0.6860	0.8857	1.1047	0.7760	1.0864	0.7920
3	0.7554	0.8500	1.1721	0.7400	1.1647	0.7660

Group	Epoch 199: Train loss	Train acc	Val loss	Val acc	Test loss	Test acc
4	0.6670	0.8857	1.0837	0.7740	1.0608	0.8010
5.1层	0.8410	0.9714	1.4139	0.7580	1.3947	0.7660
5.3层	0.0872	0.9929	0.7809	0.7780	0.6767	0.7940
6.sigm	1.3313	0.8071	1.5431	0.7360	1.5386	0.7650
6.lrelu	0.2162	0.9714	0.7402	0.7920	0.6957	0.8060
7	0.1008	1.0000	0.7455	0.7780	0.6979	0.8000

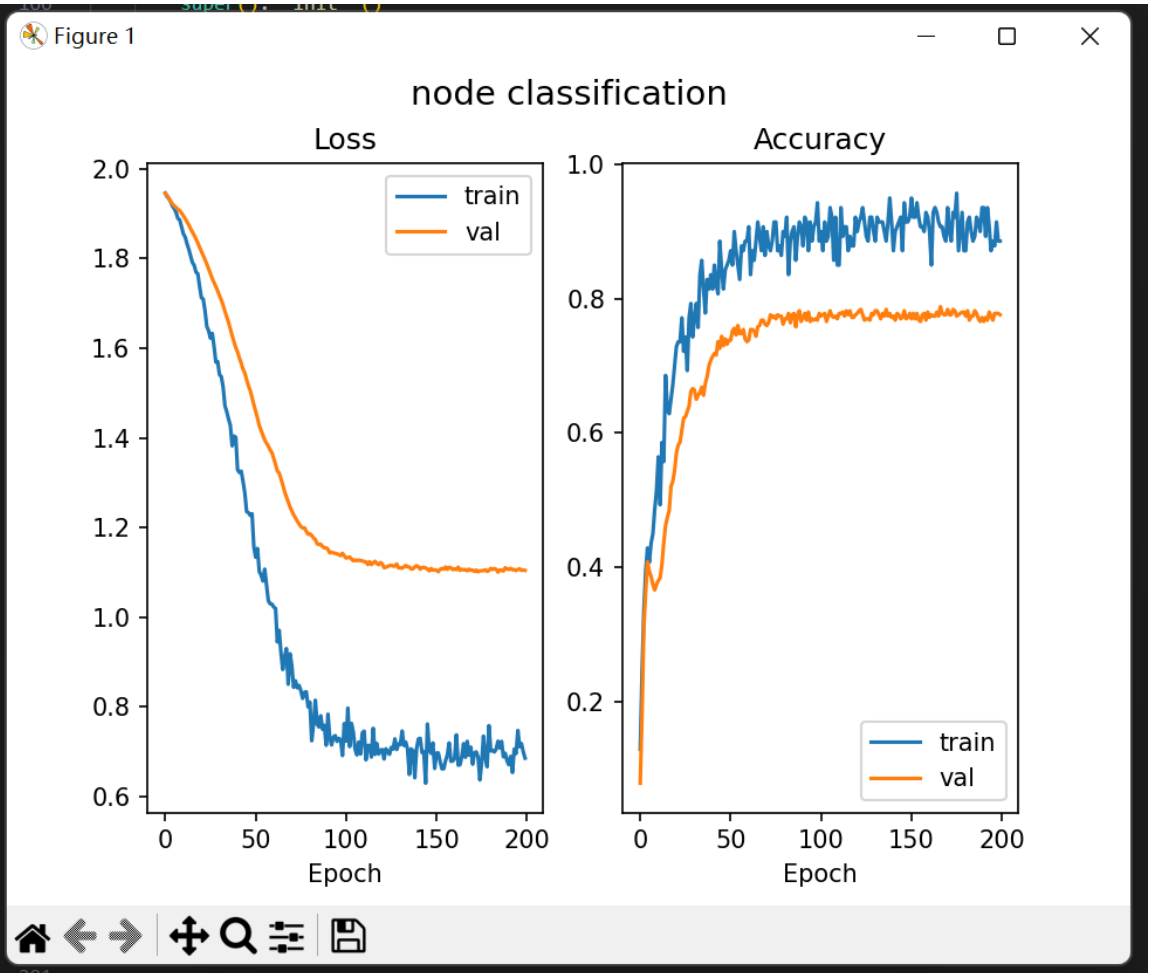
- 变动的学习率对训练效果不明显，应该是因为初始lr=0.01较小，而training_epoch=200，足以让lossfn到达极值点
- 可见 PairNorm，add_self_loops，Drop_Edge均能减少train loss，这是因为作了规范化处理，
- 1层的网络欠拟合，不能很好拟合分类函数；而3层的网络明显过拟合，TrainAcc不错，但Test_acc不如对照组
- sigmoid不能很好地拟合，可能会导致梯度消失；D，A，x，Θ，基本都为正，lrelu对负数的处理不明显
- 去除了dropout后accu达到了惊人的1.0000，说明网络结构足够复杂，test_acc较对照组低，说明dropout可以避免过拟合




```
问题  输出  调试控制台  终端  GITLENS: VISUAL FILE HISTORY

Epoch: 193 , Train loss: 0.1561 | Train acc: 0.9714 | Val loss: 0.6764 |   Val acc: 0.7820
Epoch: 194 , Train loss: 0.1555 | Train acc: 0.9929 | Val loss: 0.6701 |   Val acc: 0.7820
Epoch: 195 , Train loss: 0.2081 | Train acc: 0.9714 | Val loss: 0.6738 |   Val acc: 0.7740
Epoch: 196 , Train loss: 0.1680 | Train acc: 0.9786 | Val loss: 0.6787 |   Val acc: 0.7740
Epoch: 197 , Train loss: 0.1646 | Train acc: 0.9857 | Val loss: 0.6675 |   Val acc: 0.7720
Epoch: 198 , Train loss: 0.1451 | Train acc: 0.9857 | Val loss: 0.6797 |   Val acc: 0.7800
Epoch: 199 , Train loss: 0.1399 | Train acc: 0.9929 | Val loss: 0.6707 |   Val acc: 0.7880
对照组 Test loss: 0.6330 |  Test acc: 0.8040
```

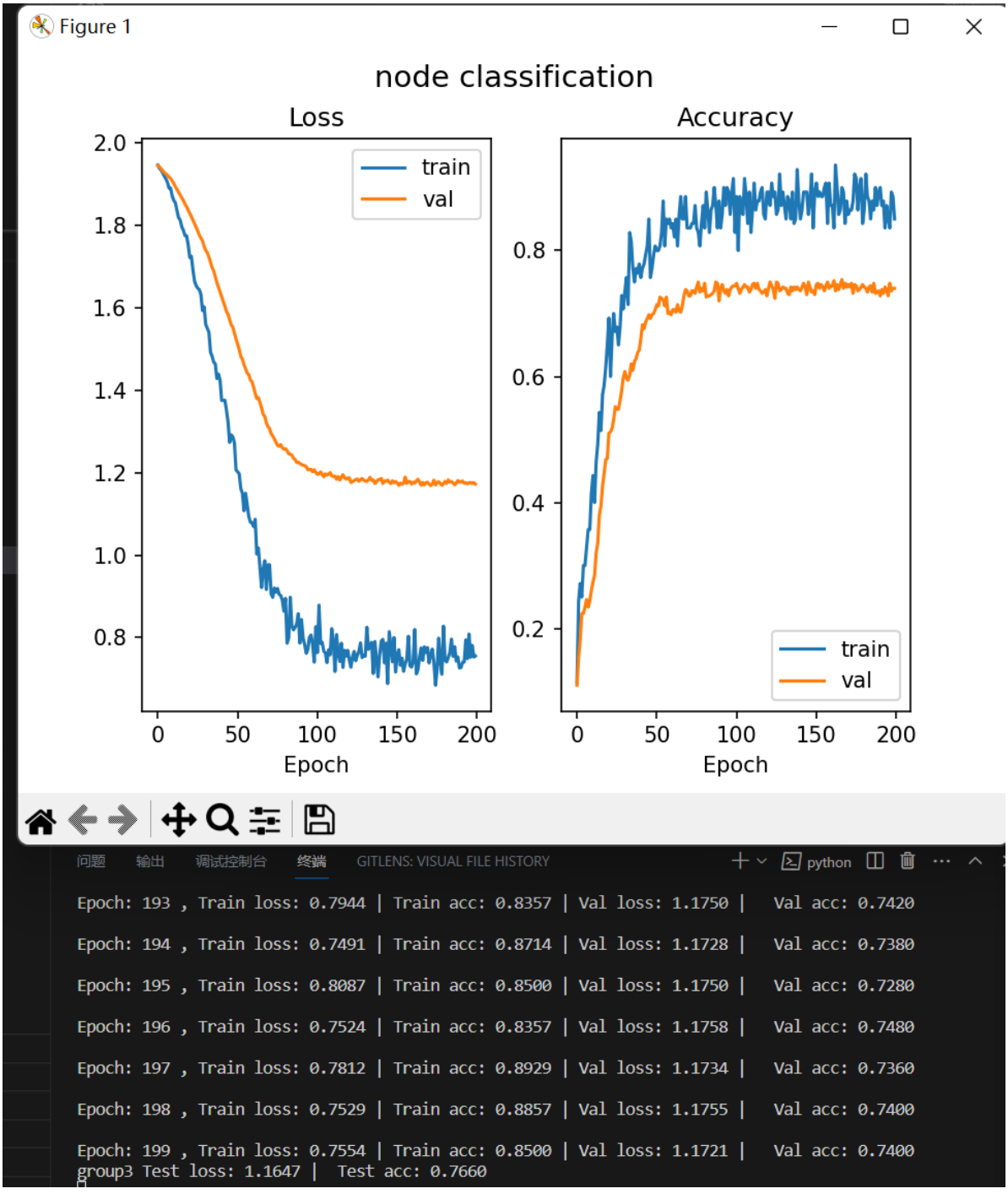



```
0.7840

Epoch: 196 , Train loss: 0.2472 | Train acc: 0.9571 | Val loss: 0.7473 |   Val acc:
0.7740

Epoch: 197 , Train loss: 0.2495 | Train acc: 0.9571 | Val loss: 0.7392 |   Val acc:
0.7820

Epoch: 198 , Train loss: 0.2161 | Train acc: 0.9929 | Val loss: 0.7462 |   Val acc:
0.7800

Epoch: 199 , Train loss: 0.2167 | Train acc: 0.9714 | Val loss: 0.7406 |   Val acc:
0.7900
group1 Test loss: 0.6959 |  Test acc: 0.8040
```



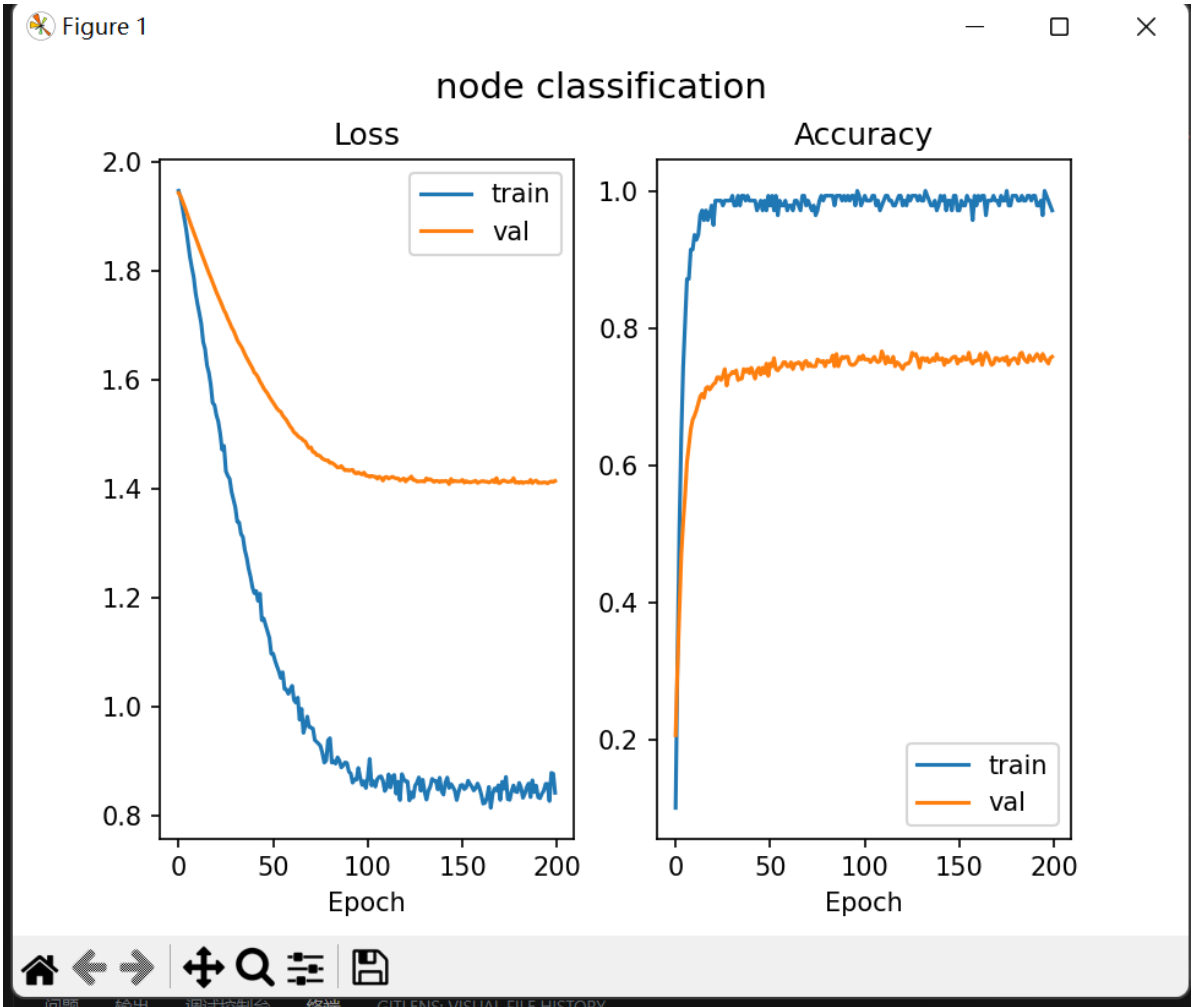
问题 输出 调试控制台 终端 GITLENS: VISUAL FILE HISTORY

```
Epoch: 193 , Train loss: 0.7062 | Train acc: 0.9071 | Val loss: 1.1064 |   Val acc: 0.7800
Epoch: 194 , Train loss: 0.6962 | Train acc: 0.8714 | Val loss: 1.1043 |   Val acc: 0.7740
Epoch: 195 , Train loss: 0.7477 | Train acc: 0.8857 | Val loss: 1.1068 |   Val acc: 0.7680
Epoch: 196 , Train loss: 0.7113 | Train acc: 0.8786 | Val loss: 1.1082 |   Val acc: 0.7780
Epoch: 197 , Train loss: 0.7182 | Train acc: 0.9143 | Val loss: 1.1047 |   Val acc: 0.7780
Epoch: 198 , Train loss: 0.6999 | Train acc: 0.8857 | Val loss: 1.1056 |   Val acc: 0.7780
Epoch: 199 , Train loss: 0.6860 | Train acc: 0.8857 | Val loss: 1.1047 |   Val acc: 0.7760
group2 Test loss: 1.0864 |  Test acc: 0.7920
```




```
问题  输出  调试控制台  终端  GITLENS: VISUAL FILE HISTORY                                    python

Epoch: 193 , Train loss: 0.7944 | Train acc: 0.8357 | Val loss: 1.1750 |   Val acc: 0.7420
Epoch: 194 , Train loss: 0.7491 | Train acc: 0.8714 | Val loss: 1.1728 |   Val acc: 0.7380
Epoch: 195 , Train loss: 0.8087 | Train acc: 0.8500 | Val loss: 1.1750 |   Val acc: 0.7280
Epoch: 196 , Train loss: 0.7524 | Train acc: 0.8357 | Val loss: 1.1758 |   Val acc: 0.7480
Epoch: 197 , Train loss: 0.7812 | Train acc: 0.8929 | Val loss: 1.1734 |   Val acc: 0.7360
Epoch: 198 , Train loss: 0.7529 | Train acc: 0.8857 | Val loss: 1.1755 |   Val acc: 0.7400
Epoch: 199 , Train loss: 0.7554 | Train acc: 0.8500 | Val loss: 1.1721 |   Val acc: 0.7400
group3 Test loss: 1.1647 |  Test acc: 0.7660
```

```
Epoch: 193 , Train loss: 0.6684 | Train acc: 0.9286 | Val loss: 1.0840 |   Val acc: 0.7740
Epoch: 194 , Train loss: 0.6815 | Train acc: 0.9214 | Val loss: 1.0839 |   Val acc: 0.7740
Epoch: 195 , Train loss: 0.6730 | Train acc: 0.9071 | Val loss: 1.0839 |   Val acc: 0.7740
Epoch: 196 , Train loss: 0.6789 | Train acc: 0.8929 | Val loss: 1.0838 |   Val acc: 0.7740
Epoch: 197 , Train loss: 0.6638 | Train acc: 0.9214 | Val loss: 1.0838 |   Val acc: 0.7740
Epoch: 198 , Train loss: 0.6898 | Train acc: 0.9071 | Val loss: 1.0838 |   Val acc: 0.7740
Epoch: 199 , Train loss: 0.6670 | Train acc: 0.8857 | Val loss: 1.0837 |   Val acc: 0.7740
group4 Test loss: 1.0608 |  Test acc: 0.8010
```




```
问题  输出  调试控制台  终端  GITLENS: VISUAL FILE HISTORY

Epoch: 193 , Train loss: 0.8449 | Train acc: 0.9857 | Val loss: 1.4114 |   Val acc: 0.7520

Epoch: 194 , Train loss: 0.8574 | Train acc: 0.9643 | Val loss: 1.4110 |   Val acc: 0.7620

Epoch: 195 , Train loss: 0.8575 | Train acc: 1.0000 | Val loss: 1.4095 |   Val acc: 0.7560

Epoch: 196 , Train loss: 0.8259 | Train acc: 0.9929 | Val loss: 1.4117 |   Val acc: 0.7520

Epoch: 197 , Train loss: 0.8776 | Train acc: 0.9857 | Val loss: 1.4131 |   Val acc: 0.7480

Epoch: 198 , Train loss: 0.8761 | Train acc: 0.9786 | Val loss: 1.4120 |   Val acc: 0.7560

Epoch: 199 , Train loss: 0.8410 | Train acc: 0.9714 | Val loss: 1.4139 |   Val acc: 0.7580
group5 Test loss: 1.3947 |  Test acc: 0.7660
```

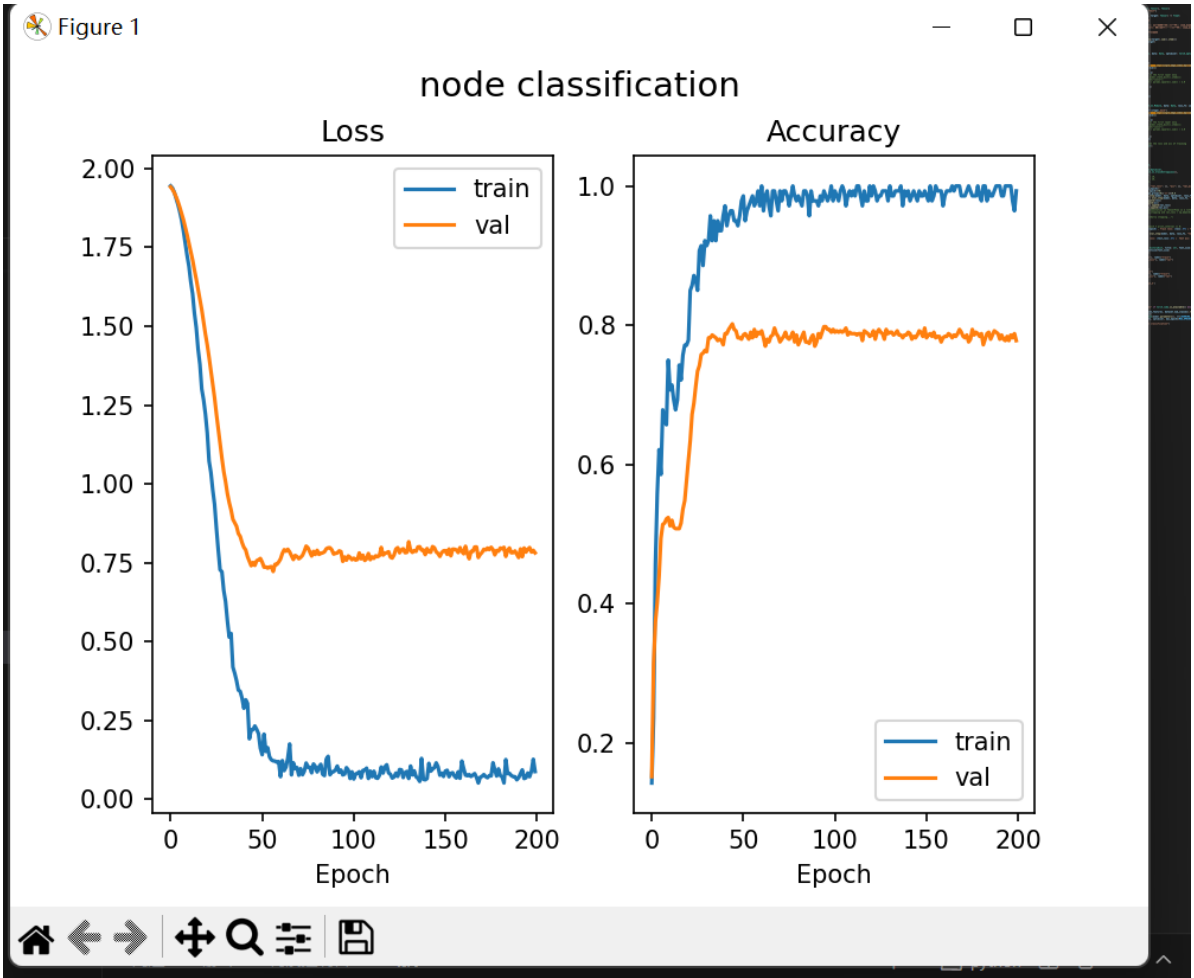
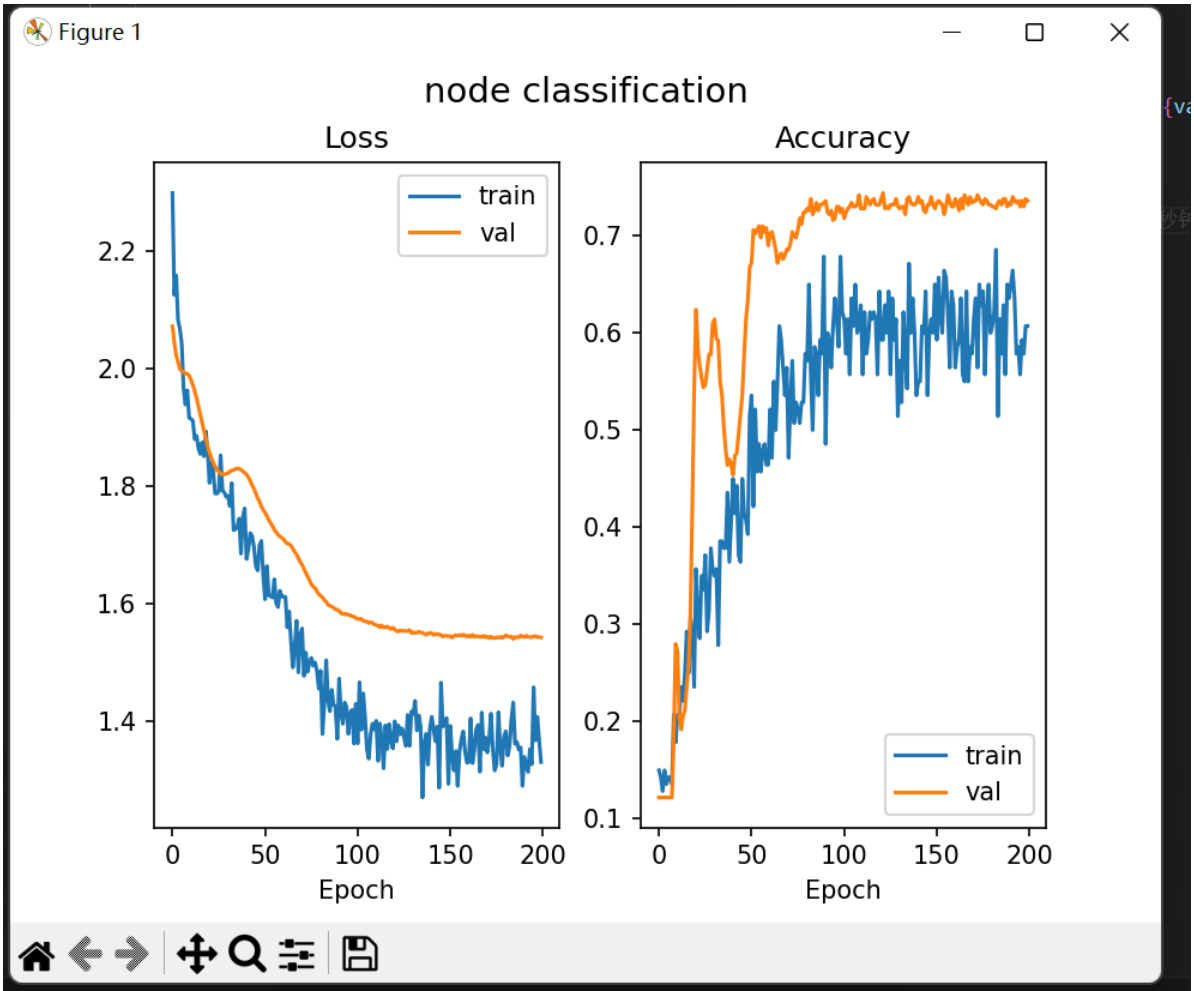


Figure 1
node classification
Loss
Accuracy
train
val
Epoch
Epoch
Epoch: 193 , Train loss: 0.0649 | Train acc: 1.0000 | Val loss: 0.7962 | Val acc: 0.7780
Epoch: 194 , Train loss: 0.0675 | Train acc: 1.0000 | Val loss: 0.7833 | Val acc: 0.7840
Epoch: 195 , Train loss: 0.0827 | Train acc: 1.0000 | Val loss: 0.7919 | Val acc: 0.7780
Epoch: 196 , Train loss: 0.0708 | Train acc: 1.0000 | Val loss: 0.7978 | Val acc: 0.7860
Epoch: 197 , Train loss: 0.0887 | Train acc: 0.9786 | Val loss: 0.7835 | Val acc: 0.7820
Epoch: 198 , Train loss: 0.1262 | Train acc: 0.9643 | Val loss: 0.7889 | Val acc: 0.7880
Epoch: 199 , Train loss: 0.0872 | Train acc: 0.9929 | Val loss: 0.7809 | Val acc: 0.7780
group5 3 Test loss: 0.6767 | Test acc: 0.7940



问题 输出 调试控制台 终端 GITLENS: VISUAL FILE HISTORY

```
Epoch: 193 , Train loss: 1.3533 | Train acc: 0.5786 | Val loss: 1.5439 |   Val acc: 0.7340
Epoch: 194 , Train loss: 1.3272 | Train acc: 0.5857 | Val loss: 1.5435 |   Val acc: 0.7360
Epoch: 195 , Train loss: 1.4584 | Train acc: 0.5571 | Val loss: 1.5449 |   Val acc: 0.7300
Epoch: 196 , Train loss: 1.3681 | Train acc: 0.5929 | Val loss: 1.5454 |   Val acc: 0.7360
Epoch: 197 , Train loss: 1.4082 | Train acc: 0.5786 | Val loss: 1.5438 |   Val acc: 0.7300
Epoch: 198 , Train loss: 1.3681 | Train acc: 0.6071 | Val loss: 1.5439 |   Val acc: 0.7380
Epoch: 199 , Train loss: 1.3313 | Train acc: 0.6071 | Val loss: 1.5431 |   Val acc: 0.7360
group6_sigm Test loss: 1.5386 |  Test acc: 0.7650
```

```
Epoch: 193 , Train loss: 0.2456 | Train acc: 0.9571 | Val loss: 0.7419 |   Val acc: 0.7820
Epoch: 194 , Train loss: 0.2339 | Train acc: 0.9643 | Val loss: 0.7375 |   Val acc: 0.7820
Epoch: 195 , Train loss: 0.2915 | Train acc: 0.9643 | Val loss: 0.7418 |   Val acc: 0.7860
Epoch: 196 , Train loss: 0.2470 | Train acc: 0.9571 | Val loss: 0.7469 |   Val acc: 0.7740
Epoch: 197 , Train loss: 0.2489 | Train acc: 0.9571 | Val loss: 0.7388 |   Val acc: 0.7820
Epoch: 198 , Train loss: 0.2155 | Train acc: 1.0000 | Val loss: 0.7457 |   Val acc: 0.7780
Epoch: 199 , Train loss: 0.2162 | Train acc: 0.9714 | Val loss: 0.7402 |   Val acc: 0.7920
group6_lrelu Test loss: 0.6957 |  Test acc: 0.8060
```

```
Epoch: 193 , Train loss: 0.1034 | Train acc: 1.0000 | Val loss: 0.7418 |   Val acc: 0.7740

Epoch: 194 , Train loss: 0.1091 | Train acc: 1.0000 | Val loss: 0.7398 |   Val acc: 0.7700

Epoch: 195 , Train loss: 0.1055 | Train acc: 1.0000 | Val loss: 0.7461 |   Val acc: 0.7700

Epoch: 196 , Train loss: 0.1046 | Train acc: 1.0000 | Val loss: 0.7404 |   Val acc: 0.7780

Epoch: 197 , Train loss: 0.1077 | Train acc: 1.0000 | Val loss: 0.7452 |   Val acc: 0.7780

Epoch: 198 , Train loss: 0.1076 | Train acc: 1.0000 | Val loss: 0.7468 |   Val acc: 0.7740

Epoch: 199 , Train loss: 0.1008 | Train acc: 1.0000 | Val loss: 0.7455 |   Val acc: 0.7780
group7_lrelu Test loss: 0.6979 |  Test acc: 0.8000
```

链路预测

PS：一开始采用torch_geometric.utils.train_test_split_edges划分，训练过程中，验证集连续若干个周期不变化到结束；后来采用torch_geometric.transforms.RandomLinkSplit，训练正常

对照组，不采用学习率调整，采用PairNorm，add_self_loops，DropEdge，2层神经网络，relu激活，采用dropout.

各训练组参数参考“各组说明”

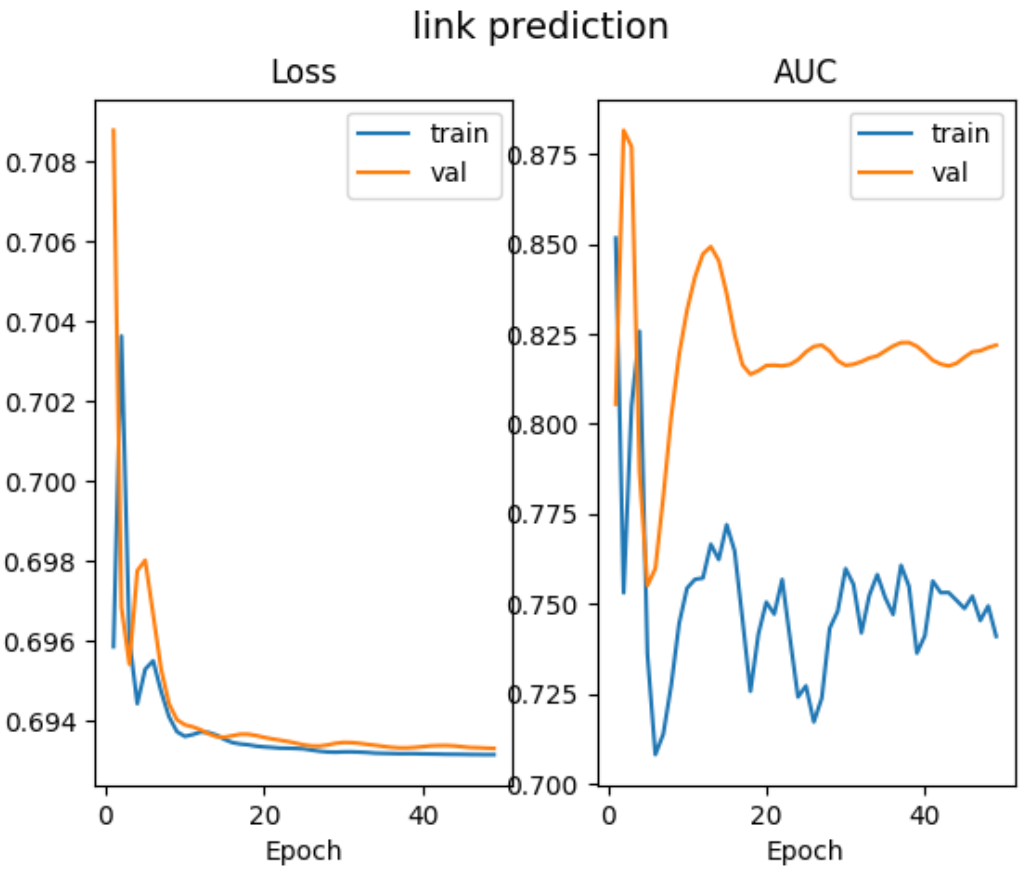
优化方式

1. 作learning rate modification，对照组不做学习率优化

由于作学习率优化后auc明显提升，因此2~7组均默认作学习率优化

```
    if epoch%5==0:
        optimizer.param_groups[0]['lr']*=0.5
```

2. 删除PairNorm
3. 删除add_self_loops
4. 删除dropEdge
5. 层数：1层+3层
6. 激活函数：relu compared with sigmoid
7. 删除dropout(默认保留)



结果对比

组号	Train loss	Epoch(at 50) Train acc	Val loss	Val acc	Test loss	Test acc
对照组	0.6932	0.7410	0.6933	0.8219	0.6934	0.8443
1(真实的对照组)	0.6932	0.8429	0.6932	0.8801	0.6933	0.9186
2	0.6968	0.5011	0.6971	0.4858	0.6972	0.4928
3	0.6932	0.5631	0.6932	0.5180	0.6933	0.4758
4	0.6932	0.8160	0.6934	0.8589	0.6935	0.8954
5_1layer	0.6931	0.5197	0.6932	0.6077	0.6932	0.6514
5_3layer	0.6932	0.5344	0.6932	0.6813	0.6932	0.6985
6_sig	0.7834	0.8836	0.7612	0/7279	0.7570	0.8305
7_no_dropout	0.6932	0.8324	0.6933	0.8608	0.6933	0.8955

分析

- 由group1，学习率递减对训练效果有明显提升，表明初始学习率可能较大，只要在lossfunction到达极值点附近(衰减前训练足够长的周期），再进行学习率衰减，就是有益无害的
- 由group2，舍去PairNorm后val_auc降到极点，表明PairNorm至关重要，也可能是同时Dropout0.5导致的

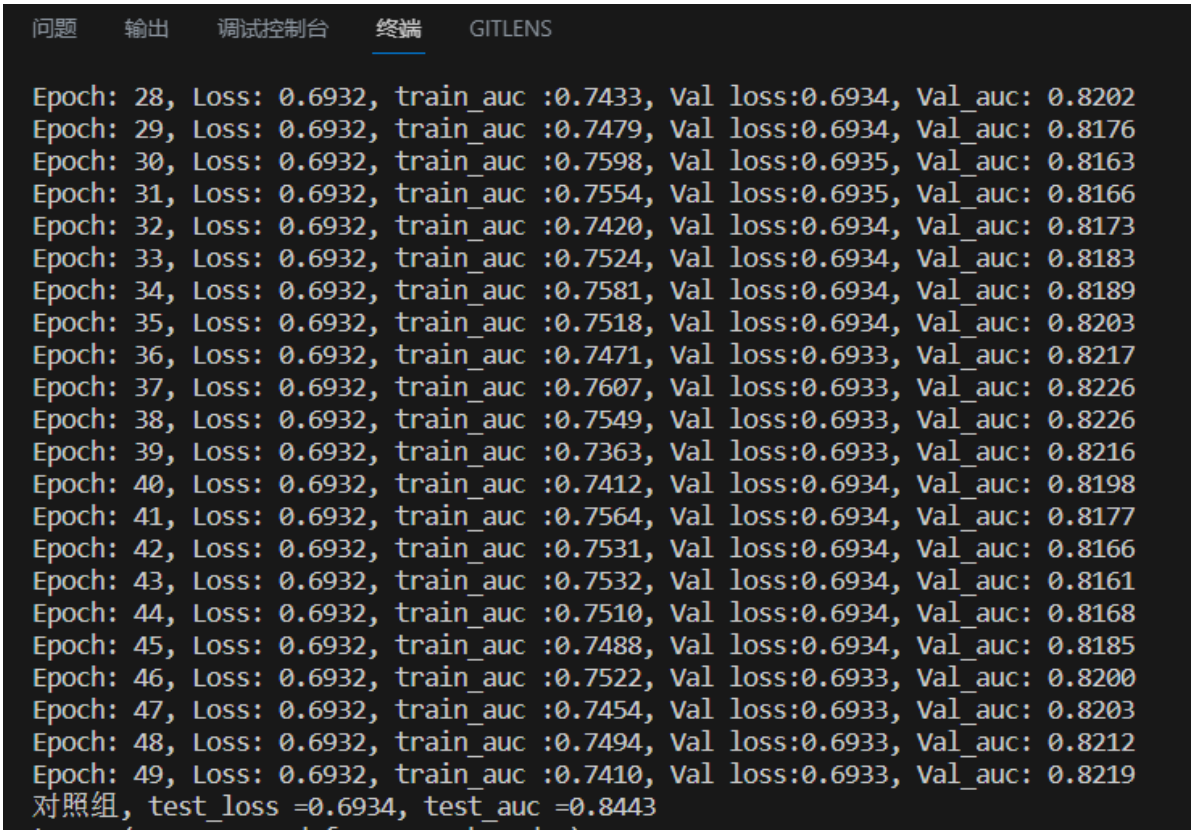
- 由group3，可见添加自环的重要性，这是因为添加自环可以：

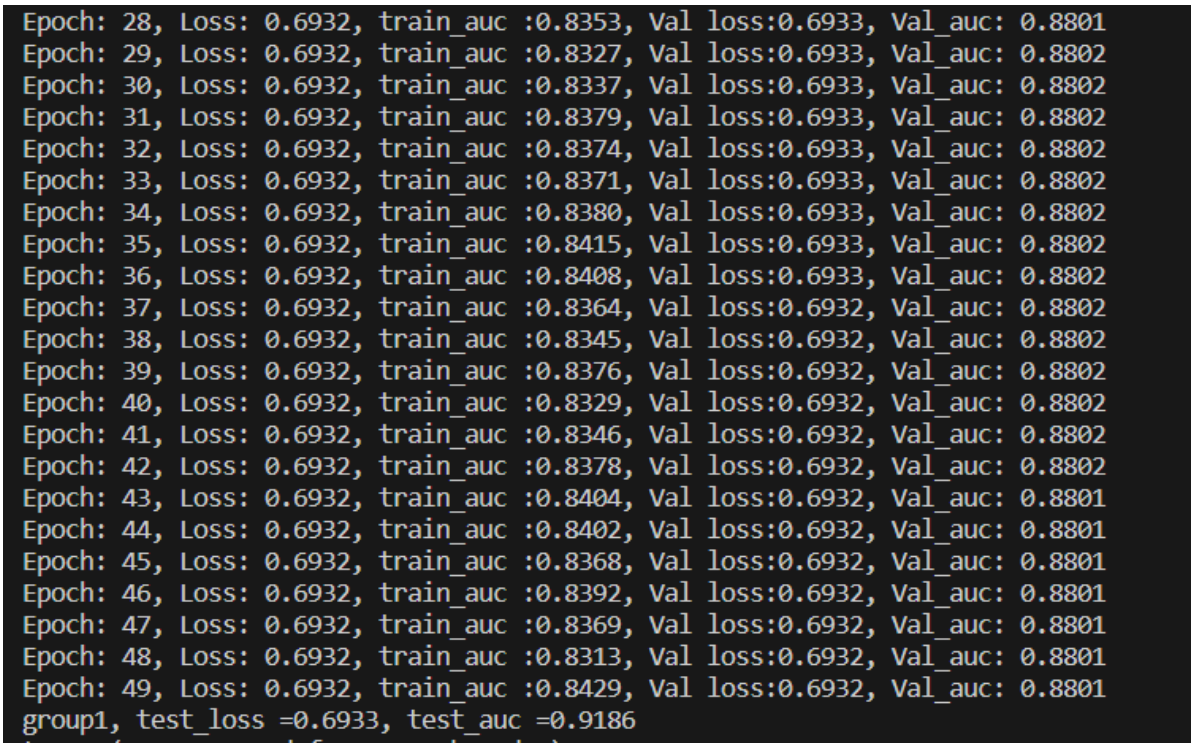
  让节点本身的特征对自己产生影响，提高节点特征的影响力；

  将节点的特征与其邻居的特征进行融合，使模型更好地捕捉节点的局部信息，提高泛化能力；

  还可以让提高节点的度数基数，减小不同Edge_index下的变化幅度，进而提升稳定性，

- 由group4，DropEdge能提高模型的泛化能力
- 由group5，1层的GCN不足以拟合Link Prediction Model，3层的GCN在实验中未能取得更好的效果(学习率和GCN_channel数可能有待调整，可能需要训练更长的周期)
- 由group6，sigmoid不太适合Link Prediction，可能是因为梯度消失
- 由group7，可知dropout能提升模型的泛化能力



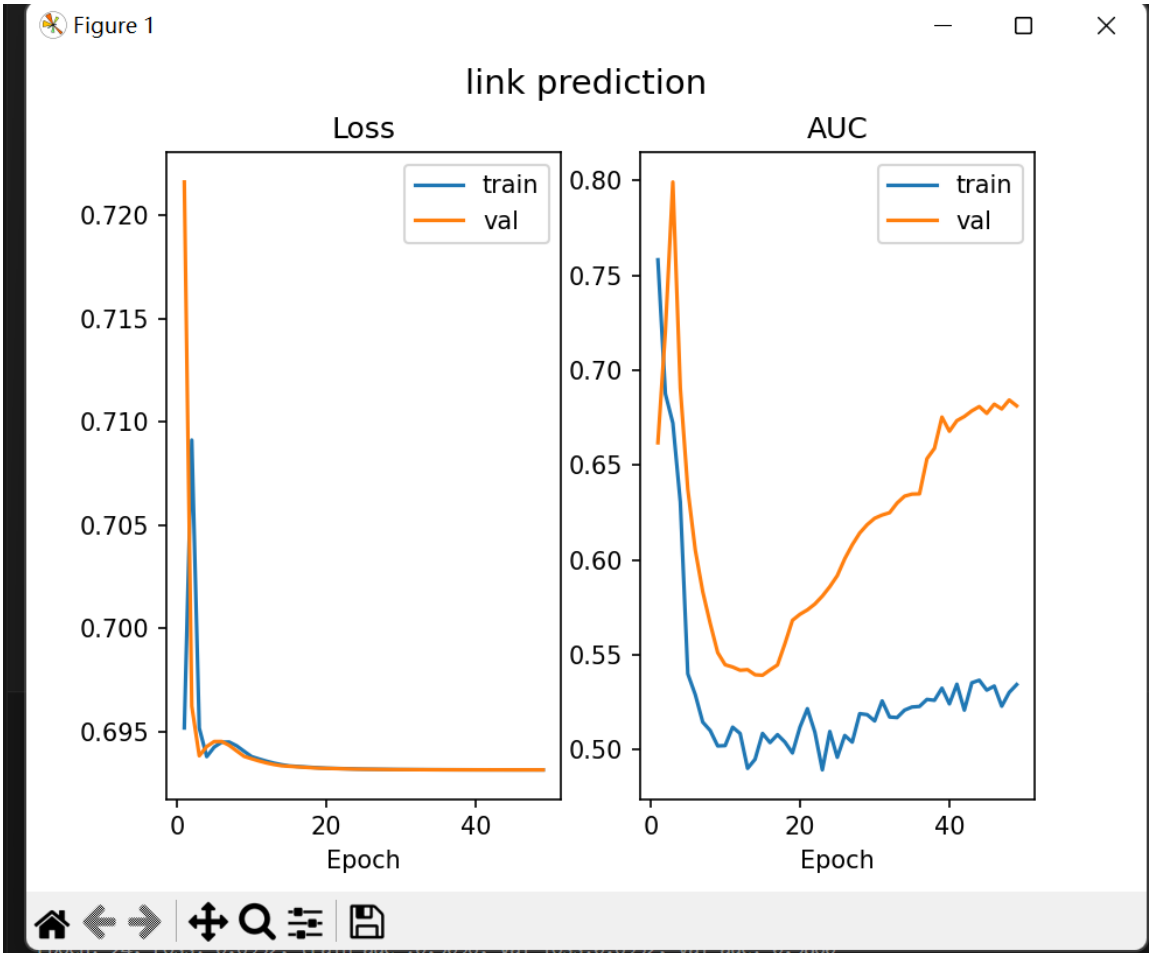


```
Epoch: 26, Loss: 0.6933, train_auc :0.5755, Val loss:0.6933, Val_auc: 0.5140
Epoch: 27, Loss: 0.6932, train_auc :0.5606, Val loss:0.6933, Val_auc: 0.5138
Epoch: 28, Loss: 0.6932, train_auc :0.5592, Val loss:0.6933, Val_auc: 0.5139
Epoch: 29, Loss: 0.6932, train_auc :0.5630, Val loss:0.6933, Val_auc: 0.5142
Epoch: 30, Loss: 0.6932, train_auc :0.5702, Val loss:0.6933, Val_auc: 0.5150
Epoch: 31, Loss: 0.6932, train_auc :0.5680, Val loss:0.6933, Val_auc: 0.5148
Epoch: 32, Loss: 0.6932, train_auc :0.5689, Val loss:0.6933, Val_auc: 0.5152
Epoch: 33, Loss: 0.6932, train_auc :0.5587, Val loss:0.6932, Val_auc: 0.5166
Epoch: 34, Loss: 0.6932, train_auc :0.5487, Val loss:0.6932, Val_auc: 0.5166
Epoch: 35, Loss: 0.6932, train_auc :0.5684, Val loss:0.6932, Val_auc: 0.5174
Epoch: 36, Loss: 0.6932, train_auc :0.5543, Val loss:0.6932, Val_auc: 0.5173
Epoch: 37, Loss: 0.6932, train_auc :0.5528, Val loss:0.6932, Val_auc: 0.5176
Epoch: 38, Loss: 0.6932, train_auc :0.5736, Val loss:0.6932, Val_auc: 0.5176
Epoch: 39, Loss: 0.6932, train_auc :0.5843, Val loss:0.6932, Val_auc: 0.5176
Epoch: 40, Loss: 0.6932, train_auc :0.5689, Val loss:0.6932, Val_auc: 0.5177
Epoch: 41, Loss: 0.6932, train_auc :0.5648, Val loss:0.6932, Val_auc: 0.5177
Epoch: 42, Loss: 0.6932, train_auc :0.5630, Val loss:0.6932, Val_auc: 0.5177
Epoch: 43, Loss: 0.6932, train_auc :0.5669, Val loss:0.6932, Val_auc: 0.5177
Epoch: 44, Loss: 0.6932, train_auc :0.5717, Val loss:0.6932, Val_auc: 0.5177
Epoch: 45, Loss: 0.6932, train_auc :0.5730, Val loss:0.6932, Val_auc: 0.5177
Epoch: 46, Loss: 0.6932, train_auc :0.5763, Val loss:0.6932, Val_auc: 0.5179
Epoch: 47, Loss: 0.6932, train_auc :0.5628, Val loss:0.6932, Val_auc: 0.5180
Epoch: 48, Loss: 0.6932, train_auc :0.5757, Val loss:0.6932, Val_auc: 0.5180
Epoch: 49, Loss: 0.6932, train_auc :0.5631, Val loss:0.6932, Val_auc: 0.5180
group3, test_loss =0.6933, test_auc =0.4758
```

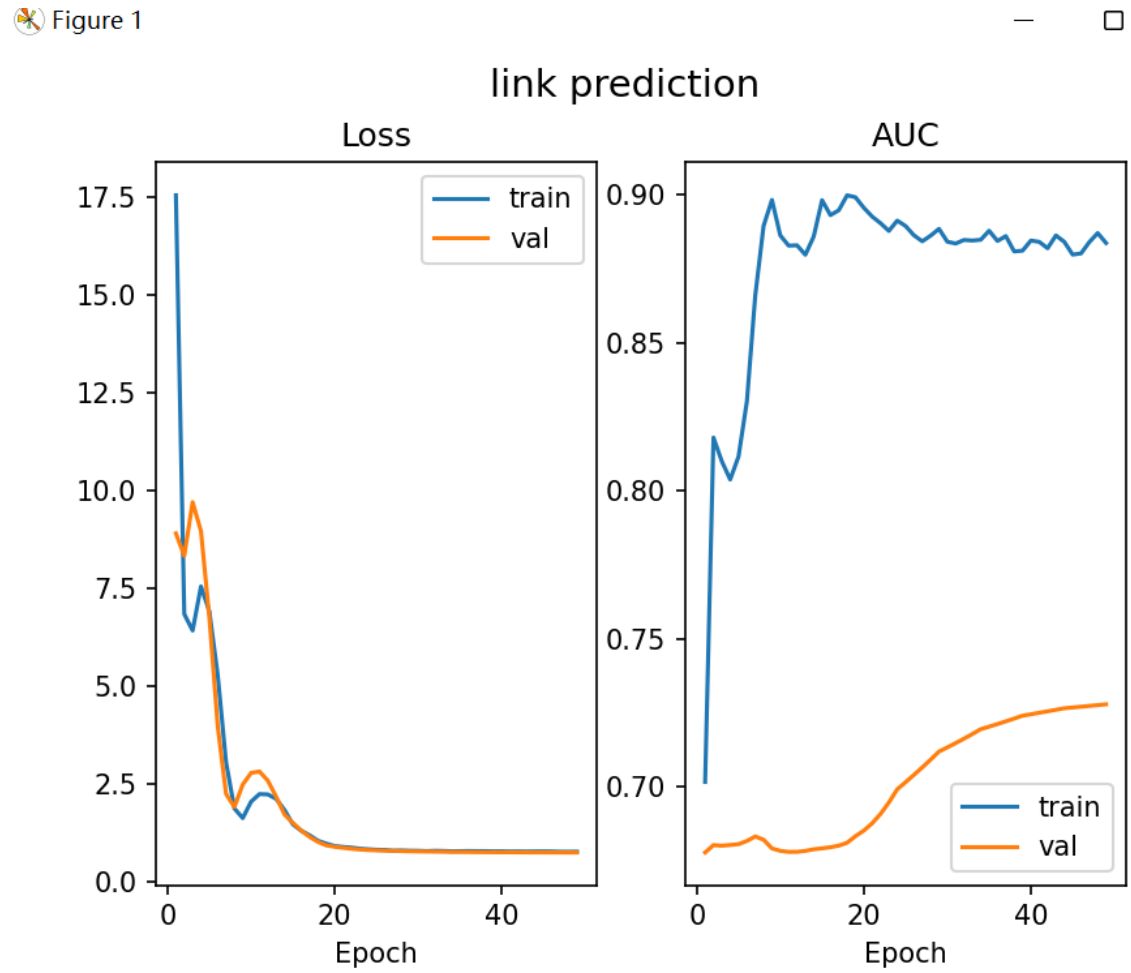
```
Epoch: 17, Loss: 0.6932, train_auc :0.7730, Val loss:0.6936, Val_auc: 0.8440
Epoch: 18, Loss: 0.6932, train_auc :0.7725, Val loss:0.6936, Val_auc: 0.8451
Epoch: 19, Loss: 0.6932, train_auc :0.7764, Val loss:0.6936, Val_auc: 0.8466
Epoch: 20, Loss: 0.6932, train_auc :0.7815, Val loss:0.6936, Val_auc: 0.8475
Epoch: 21, Loss: 0.6932, train_auc :0.7828, Val loss:0.6935, Val_auc: 0.8486
Epoch: 22, Loss: 0.6932, train_auc :0.7858, Val loss:0.6935, Val_auc: 0.8497
Epoch: 23, Loss: 0.6932, train_auc :0.7861, Val loss:0.6935, Val_auc: 0.8509
Epoch: 24, Loss: 0.6932, train_auc :0.7934, Val loss:0.6935, Val_auc: 0.8522
Epoch: 25, Loss: 0.6932, train_auc :0.7949, Val loss:0.6935, Val_auc: 0.8529
Epoch: 26, Loss: 0.6932, train_auc :0.7986, Val loss:0.6935, Val_auc: 0.8536
Epoch: 27, Loss: 0.6932, train_auc :0.8030, Val loss:0.6935, Val_auc: 0.8543
Epoch: 28, Loss: 0.6932, train_auc :0.8015, Val loss:0.6934, Val_auc: 0.8550
Epoch: 29, Loss: 0.6932, train_auc :0.8063, Val loss:0.6934, Val_auc: 0.8557
Epoch: 30, Loss: 0.6932, train_auc :0.8102, Val loss:0.6934, Val_auc: 0.8560
Epoch: 31, Loss: 0.6932, train_auc :0.8097, Val loss:0.6934, Val_auc: 0.8564
Epoch: 32, Loss: 0.6932, train_auc :0.8097, Val loss:0.6934, Val_auc: 0.8568
Epoch: 33, Loss: 0.6932, train_auc :0.8099, Val loss:0.6934, Val_auc: 0.8571
Epoch: 34, Loss: 0.6932, train_auc :0.8118, Val loss:0.6934, Val_auc: 0.8574
Epoch: 35, Loss: 0.6932, train_auc :0.8131, Val loss:0.6934, Val_auc: 0.8576
Epoch: 36, Loss: 0.6932, train_auc :0.8124, Val loss:0.6934, Val_auc: 0.8578
Epoch: 37, Loss: 0.6932, train_auc :0.8131, Val loss:0.6934, Val_auc: 0.8580
Epoch: 38, Loss: 0.6932, train_auc :0.8198, Val loss:0.6934, Val_auc: 0.8581
Epoch: 39, Loss: 0.6932, train_auc :0.8187, Val loss:0.6934, Val_auc: 0.8583
Epoch: 40, Loss: 0.6932, train_auc :0.8181, Val loss:0.6934, Val_auc: 0.8584
Epoch: 41, Loss: 0.6932, train_auc :0.8184, Val loss:0.6934, Val_auc: 0.8584
Epoch: 42, Loss: 0.6932, train_auc :0.8198, Val loss:0.6934, Val_auc: 0.8585
Epoch: 43, Loss: 0.6932, train_auc :0.8206, Val loss:0.6934, Val_auc: 0.8586
Epoch: 44, Loss: 0.6932, train_auc :0.8178, Val loss:0.6934, Val_auc: 0.8587
Epoch: 45, Loss: 0.6932, train_auc :0.8192, Val loss:0.6934, Val_auc: 0.8587
Epoch: 46, Loss: 0.6932, train_auc :0.8216, Val loss:0.6934, Val_auc: 0.8588
Epoch: 47, Loss: 0.6932, train_auc :0.8213, Val loss:0.6934, Val_auc: 0.8588
Epoch: 48, Loss: 0.6932, train_auc :0.8236, Val loss:0.6934, Val_auc: 0.8588
Epoch: 49, Loss: 0.6932, train_auc :0.8160, Val loss:0.6934, Val_auc: 0.8589
group4, test_loss =0.6935, test_auc =0.8954
```

```
Epoch: 21, Loss: 0.6931, train_auc :0.5163, Val loss:0.6932, Val_auc: 0.6084
Epoch: 22, Loss: 0.6931, train_auc :0.5186, Val loss:0.6932, Val_auc: 0.6082
Epoch: 23, Loss: 0.6931, train_auc :0.5175, Val loss:0.6932, Val_auc: 0.6081
Epoch: 24, Loss: 0.6931, train_auc :0.5178, Val loss:0.6932, Val_auc: 0.6082
Epoch: 25, Loss: 0.6931, train_auc :0.5186, Val loss:0.6932, Val_auc: 0.6080
Epoch: 26, Loss: 0.6931, train_auc :0.5204, Val loss:0.6932, Val_auc: 0.6080
Epoch: 27, Loss: 0.6931, train_auc :0.5175, Val loss:0.6932, Val_auc: 0.6078
Epoch: 28, Loss: 0.6931, train_auc :0.5180, Val loss:0.6932, Val_auc: 0.6079
Epoch: 29, Loss: 0.6931, train_auc :0.5178, Val loss:0.6932, Val_auc: 0.6077
Epoch: 30, Loss: 0.6931, train_auc :0.5203, Val loss:0.6932, Val_auc: 0.6077
Epoch: 31, Loss: 0.6931, train_auc :0.5189, Val loss:0.6932, Val_auc: 0.6078
Epoch: 32, Loss: 0.6931, train_auc :0.5155, Val loss:0.6932, Val_auc: 0.6078
Epoch: 33, Loss: 0.6931, train_auc :0.5187, Val loss:0.6932, Val_auc: 0.6078
Epoch: 34, Loss: 0.6931, train_auc :0.5201, Val loss:0.6932, Val_auc: 0.6078
Epoch: 35, Loss: 0.6931, train_auc :0.5162, Val loss:0.6932, Val_auc: 0.6078
Epoch: 36, Loss: 0.6931, train_auc :0.5209, Val loss:0.6932, Val_auc: 0.6078
Epoch: 37, Loss: 0.6931, train_auc :0.5180, Val loss:0.6932, Val_auc: 0.6076
Epoch: 38, Loss: 0.6931, train_auc :0.5192, Val loss:0.6932, Val_auc: 0.6076
Epoch: 39, Loss: 0.6931, train_auc :0.5185, Val loss:0.6932, Val_auc: 0.6076
Epoch: 40, Loss: 0.6931, train_auc :0.5211, Val loss:0.6932, Val_auc: 0.6076
Epoch: 41, Loss: 0.6931, train_auc :0.5186, Val loss:0.6932, Val_auc: 0.6076
Epoch: 42, Loss: 0.6931, train_auc :0.5224, Val loss:0.6932, Val_auc: 0.6077
Epoch: 43, Loss: 0.6931, train_auc :0.5178, Val loss:0.6932, Val_auc: 0.6077
Epoch: 44, Loss: 0.6931, train_auc :0.5168, Val loss:0.6932, Val_auc: 0.6077
Epoch: 45, Loss: 0.6931, train_auc :0.5184, Val loss:0.6932, Val_auc: 0.6077
Epoch: 46, Loss: 0.6931, train_auc :0.5165, Val loss:0.6932, Val_auc: 0.6077
Epoch: 47, Loss: 0.6931, train_auc :0.5175, Val loss:0.6932, Val_auc: 0.6077
Epoch: 48, Loss: 0.6931, train_auc :0.5188, Val loss:0.6932, Val_auc: 0.6077
Epoch: 49, Loss: 0.6931, train_auc :0.5197, Val loss:0.6932, Val_auc: 0.6077
group4, test loss =0.6932, test auc =0.6514
```

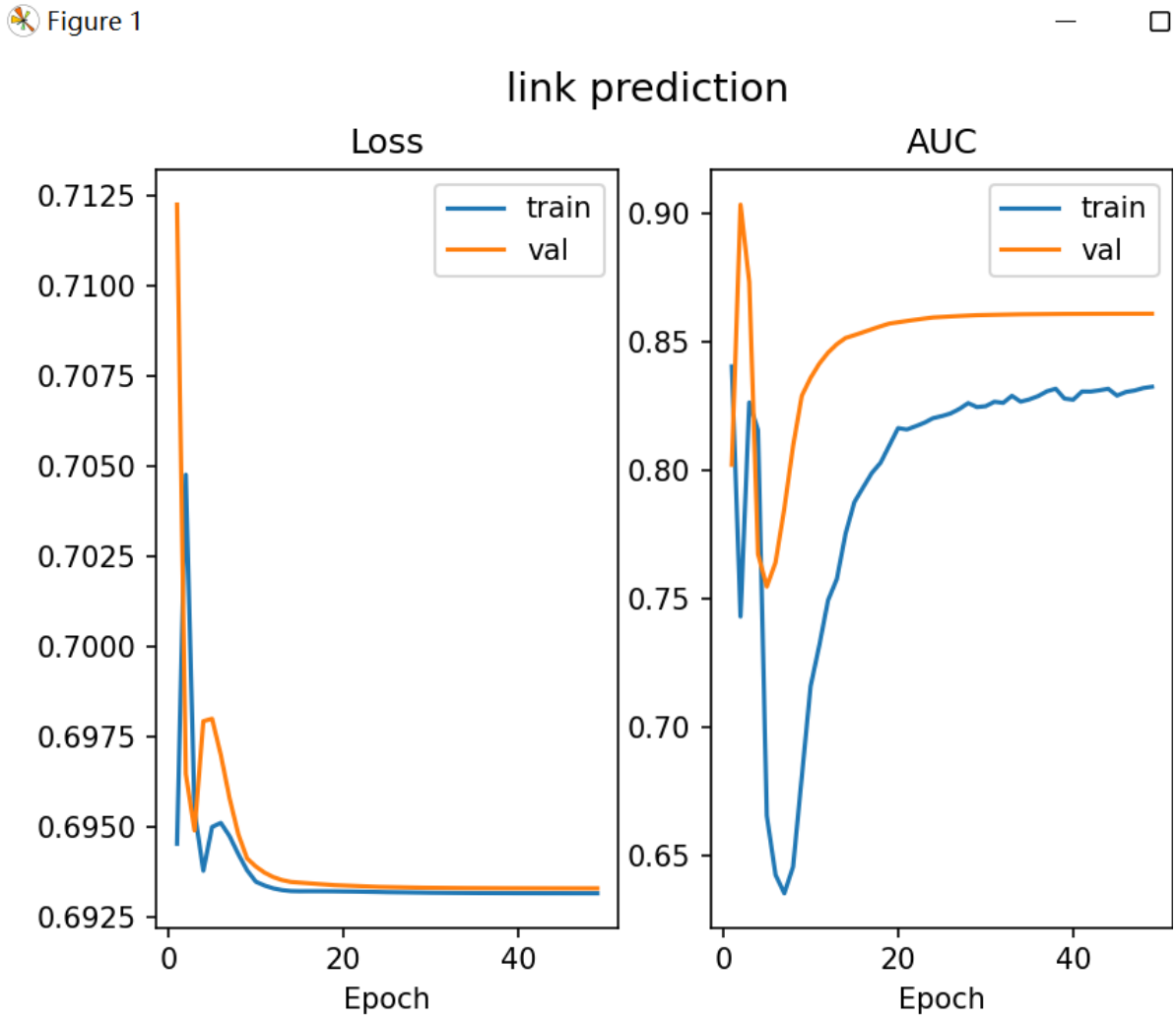
(这个是group5_1layer，更改未保存)




```
Epoch: 25, Loss: 0.6932, train_auc :0.4960, Val loss:0.6932, Val_auc: 0.5918
Epoch: 26, Loss: 0.6932, train_auc :0.5075, Val loss:0.6932, Val_auc: 0.6007
Epoch: 27, Loss: 0.6932, train_auc :0.5040, Val loss:0.6932, Val_auc: 0.6080
Epoch: 28, Loss: 0.6932, train_auc :0.5190, Val loss:0.6932, Val_auc: 0.6142
Epoch: 29, Loss: 0.6932, train_auc :0.5185, Val loss:0.6932, Val_auc: 0.6187
Epoch: 30, Loss: 0.6932, train_auc :0.5152, Val loss:0.6932, Val_auc: 0.6220
Epoch: 31, Loss: 0.6932, train_auc :0.5257, Val loss:0.6932, Val_auc: 0.6237
Epoch: 32, Loss: 0.6932, train_auc :0.5173, Val loss:0.6932, Val_auc: 0.6250
Epoch: 33, Loss: 0.6932, train_auc :0.5170, Val loss:0.6932, Val_auc: 0.6301
Epoch: 34, Loss: 0.6932, train_auc :0.5209, Val loss:0.6932, Val_auc: 0.6337
Epoch: 35, Loss: 0.6932, train_auc :0.5225, Val loss:0.6932, Val_auc: 0.6348
Epoch: 36, Loss: 0.6932, train_auc :0.5228, Val loss:0.6932, Val_auc: 0.6349
Epoch: 37, Loss: 0.6932, train_auc :0.5265, Val loss:0.6932, Val_auc: 0.6535
Epoch: 38, Loss: 0.6932, train_auc :0.5261, Val loss:0.6932, Val_auc: 0.6589
Epoch: 39, Loss: 0.6932, train_auc :0.5325, Val loss:0.6932, Val_auc: 0.6754
Epoch: 40, Loss: 0.6932, train_auc :0.5242, Val loss:0.6932, Val_auc: 0.6679
Epoch: 41, Loss: 0.6932, train_auc :0.5345, Val loss:0.6932, Val_auc: 0.6735
Epoch: 42, Loss: 0.6932, train_auc :0.5209, Val loss:0.6932, Val_auc: 0.6757
Epoch: 43, Loss: 0.6932, train_auc :0.5353, Val loss:0.6932, Val_auc: 0.6786
Epoch: 44, Loss: 0.6932, train_auc :0.5367, Val loss:0.6932, Val_auc: 0.6809
Epoch: 45, Loss: 0.6932, train_auc :0.5314, Val loss:0.6932, Val_auc: 0.6773
Epoch: 46, Loss: 0.6932, train_auc :0.5336, Val loss:0.6932, Val_auc: 0.6822
Epoch: 47, Loss: 0.6932, train_auc :0.5229, Val loss:0.6932, Val_auc: 0.6796
Epoch: 48, Loss: 0.6932, train_auc :0.5302, Val loss:0.6932, Val_auc: 0.6844
Epoch: 49, Loss: 0.6932, train_auc :0.5344, Val loss:0.6932, Val_auc: 0.6813
group5_3, test_loss =0.6932, test_auc =0.6985
```




```
Epoch: 31, Loss: 0.7982, train_auc :0.8834, Val loss:0.7864, Val_auc: 0.7149
Epoch: 32, Loss: 0.8051, train_auc :0.8846, Val loss:0.7833, Val_auc: 0.7164
Epoch: 33, Loss: 0.8017, train_auc :0.8844, Val loss:0.7801, Val_auc: 0.7179
Epoch: 34, Loss: 0.7938, train_auc :0.8847, Val loss:0.7769, Val_auc: 0.7196
Epoch: 35, Loss: 0.7962, train_auc :0.8877, Val loss:0.7753, Val_auc: 0.7204
Epoch: 36, Loss: 0.7997, train_auc :0.8843, Val loss:0.7736, Val_auc: 0.7213
Epoch: 37, Loss: 0.7970, train_auc :0.8859, Val loss:0.7719, Val_auc: 0.7222
Epoch: 38, Loss: 0.7975, train_auc :0.8808, Val loss:0.7701, Val_auc: 0.7231
Epoch: 39, Loss: 0.7917, train_auc :0.8810, Val loss:0.7683, Val_auc: 0.7241
Epoch: 40, Loss: 0.7924, train_auc :0.8844, Val loss:0.7674, Val_auc: 0.7246
Epoch: 41, Loss: 0.7879, train_auc :0.8840, Val loss:0.7665, Val_auc: 0.7251
Epoch: 42, Loss: 0.7895, train_auc :0.8818, Val loss:0.7655, Val_auc: 0.7256
Epoch: 43, Loss: 0.7867, train_auc :0.8861, Val loss:0.7646, Val_auc: 0.7261
Epoch: 44, Loss: 0.7893, train_auc :0.8840, Val loss:0.7636, Val_auc: 0.7266
Epoch: 45, Loss: 0.7901, train_auc :0.8797, Val loss:0.7631, Val_auc: 0.7269
Epoch: 46, Loss: 0.7877, train_auc :0.8800, Val loss:0.7626, Val_auc: 0.7271
Epoch: 47, Loss: 0.7821, train_auc :0.8839, Val loss:0.7622, Val_auc: 0.7274
Epoch: 48, Loss: 0.7806, train_auc :0.8869, Val loss:0.7617, Val_auc: 0.7277
Epoch: 49, Loss: 0.7834, train_auc :0.8836, Val loss:0.7612, Val_auc: 0.7279
group6, test loss =0.7570, test auc =0.8305
```




```
Epoch: 34, Loss: 0.6932, train_auc :0.8266, Val loss:0.6933, Val_auc: 0.8606
Epoch: 35, Loss: 0.6932, train_auc :0.8275, Val loss:0.6933, Val_auc: 0.8606
Epoch: 36, Loss: 0.6932, train_auc :0.8288, Val loss:0.6933, Val_auc: 0.8607
Epoch: 37, Loss: 0.6932, train_auc :0.8307, Val loss:0.6933, Val_auc: 0.8607
Epoch: 38, Loss: 0.6932, train_auc :0.8317, Val loss:0.6933, Val_auc: 0.8607
Epoch: 39, Loss: 0.6932, train_auc :0.8278, Val loss:0.6933, Val_auc: 0.8607
Epoch: 40, Loss: 0.6932, train_auc :0.8274, Val loss:0.6933, Val_auc: 0.8608
Epoch: 41, Loss: 0.6932, train_auc :0.8306, Val loss:0.6933, Val_auc: 0.8608
Epoch: 42, Loss: 0.6932, train_auc :0.8306, Val loss:0.6933, Val_auc: 0.8608
Epoch: 43, Loss: 0.6932, train_auc :0.8311, Val loss:0.6933, Val_auc: 0.8608
Epoch: 44, Loss: 0.6932, train_auc :0.8316, Val loss:0.6933, Val_auc: 0.8608
Epoch: 45, Loss: 0.6932, train_auc :0.8290, Val loss:0.6933, Val_auc: 0.8608
Epoch: 46, Loss: 0.6932, train_auc :0.8304, Val loss:0.6933, Val_auc: 0.8608
Epoch: 47, Loss: 0.6932, train_auc :0.8310, Val loss:0.6933, Val_auc: 0.8608
Epoch: 48, Loss: 0.6932, train_auc :0.8319, Val loss:0.6933, Val_auc: 0.8608
Epoch: 49, Loss: 0.6932, train_auc :0.8324, Val loss:0.6933, Val_auc: 0.8608
group7, test_loss =0.6933, test_auc =0.8955
```