深度学习 Lab4 实验报告

PB19151769　　马宇骁

1　实验要求

Given three datasets (cora, citeseer, ppi), implement the GCN algorithm for node classification and link prediction, and analyse the effect of self-loop, number of layers, DropEdge, PairNorm, activation function and other factors on performance。

2　实验原理

对于图表示 $G=(V,E)$。这里 V 是图中节点的集合，而 E 为边的集合，这里记图的节点数为 N。一个 G 中有 3 个比较重要的矩阵：

- 邻接矩阵 A：adjacency matrix，用来表示节点间的连接关系，这里我们假定是 0-1 矩阵；
- 度矩阵 D：degree matrix，每个节点的度指的是其连接的节点数，这是一个对角矩阵，其中对角线元素 $D_{ii}=\sum_j A_{jj}$；
- 特征矩阵 X：用于表示节点的特征，$X\in R^{N\times F}$，这里 F 是特征的维度；

2.1　GCN

GCN（Graph Convolutional Networks），图卷积神经网络是一个特征提取器让我们可以使用这些特征去对图数据进行节点分类（node classification）、图分类（graph classification）、边预测（link prediction），还可以顺便得到图的嵌入表示（graph embedding）. 用公式表达就是：

$$H^{(k+1)}=f\left(H^{(k)},A\right) \tag{1}$$

这里 k 指的是网络层数，$H^{(k)}$ 就是网络第 k 层的特征，其中 $H^{(0)}=X$.

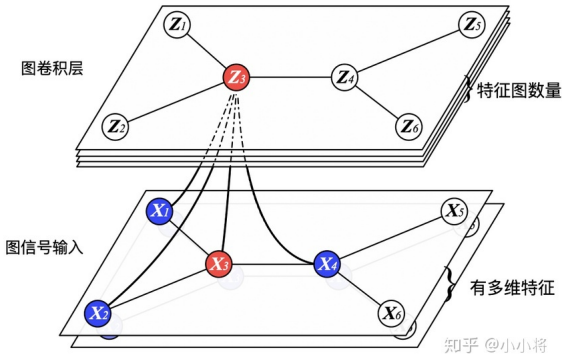

图 1: GCN

每个节点的新特征可以类似得到：对该节点的邻域节点特征进行变换，然后求和。采用加法规则时，对于度大的节点特征越来越大，而对于度小的节点却相反，这可能导致网络训练过程中梯度爆炸或者消失的问题。因此可以给图中每个节点增加自连接，实现上可以直接改变邻接矩阵，然后我们可以对邻接矩阵进行归一化，使得 A 的每行和值为 1，在实现上我们可以乘以度矩阵的逆矩阵：$\tilde{D}^{-1}\tilde{A}$，更进一步地，我们可以采用对称归一化来进行聚合操作，这就是图卷积方法：

$$H^{(k+1)} = f\left(H^{(k)}, A\right) = \sigma\left(\tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}H^{(k)}W^{(k)}\right) \tag{2}$$

这里：

$$\begin{aligned}
\left(\tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}H\right)_i &= \left(\tilde{D}^{-0.5}\tilde{A}\right)_i \tilde{D}^{-0.5}H \\
&= \left(\sum_k \tilde{D}_{ik}^{-0.5}\tilde{A}_i\right)\tilde{D}^{-0.5}H \\
&= \tilde{D}_{ii}^{-0.5}\sum_j \tilde{A}_{ij}\sum_k \tilde{D}_{jk}^{-0.5}H_j \\
&= \tilde{D}_{ii}^{-0.5}\sum_j \tilde{A}_{ij}\tilde{D}_{jj}^{-0.5}H_j \\
&= \sum_j \frac{1}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}}\tilde{A}_{ij}H_j
\end{aligned} \tag{3}$$

2.2 GraphSAGE

GCN 是一种在图中结合拓扑结构和顶点属性信息学习顶点的 embedding 表示的方法。然而 GCN 要求在一个确定的图中去学习顶点的 embedding，无法直接泛化到在训练过程没有出现过的顶点，即属于一种直推式 (transductive) 的学习。

GraphSAGE 则是一种能够利用顶点的属性信息高效产生未知顶点 embedding 的一种归纳式 (inductive) 学习的框架。其核心思想是通过学习一个对邻居顶点进行聚合表示的函数来产生目标顶点的 embedding 向量。

GraphSAGE 是 Graph SAmple and aggreGatE 的缩写，其运行流程如上图所示，可以分为三个步骤:

1. 对图中每个顶点邻居顶点进行采样
2. 根据聚合函数聚合邻居顶点蕴含的信息
3. 得到图中各顶点的向量表示供下游任务使用

伪代码如下：

$$\begin{aligned}
&\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V} \\
&\text{for } k = 1 \ldots K \text{ do} \\
&\quad \text{for } v \in \mathcal{V} \text{ do} \\
&\quad\quad \mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k\left(\left\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\right\}\right) \\
&\quad\quad \mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}\left(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k\right)\right) \\
&\quad \text{end} \\
&\quad \mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \left\|\mathbf{h}_v^k\right\|_2, \forall v \in \mathcal{V} \\
&\text{end} \\
&\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}
\end{aligned} \tag{4}$$

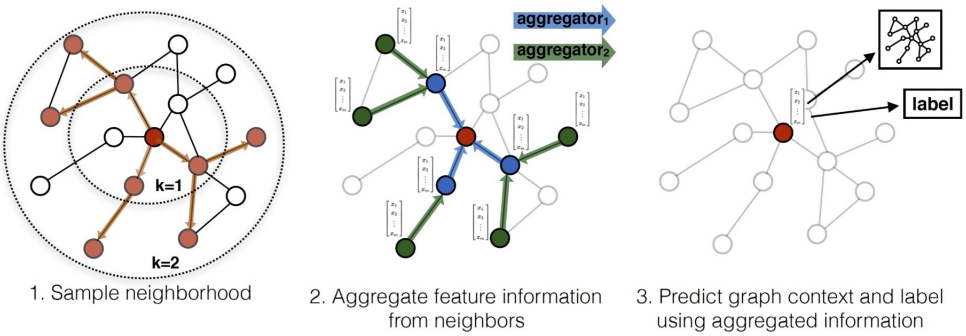

Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

图 2: GraphSAGE

3 实现及分析

利用助教提供的 demo 的代码，在 node 中构建 GCN 模型，并在后面的比较中使用 PairNorm 进一步对于模型进行优化，在 link 中的预训练处理数据时考虑加入或者不加入自环的连接方式，并在 GraphSAGE 中加入 DropEdge 继续优化模型。

在这些过程中使用不同的三种激活函数横向对比模型训练：ReLu，Leaky_ReLu，ELU.

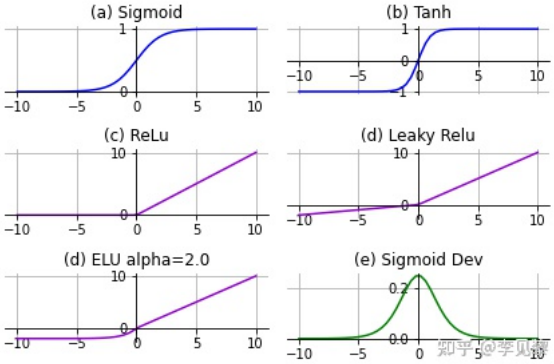

图 3: 激活函数

1. Relu

   ReLU 激活函数的提出就是为了解决梯度消失问题。ReLU 的梯度只可以取两个值：0 或 1，当输入小于 0 时，梯度为 0；当输入大于 0 时，梯度为 1 好处就是：ReLU 的梯度的连乘不会收敛到 0，连乘的结果也只可以取两个值：0 或 1 。如果值为 1，梯度保持值不变进行前向传播；如果值为 0 , 梯度从该位置停止前向传播。

2. Leaky ReLU

   ReLU 尽管稀疏性可以提升计算高效性，但同样也可能阻碍训练过程。通常，激活函数的输入值有一偏置项 (bias)，假设 bias 变得太小，以至于输入激活函数的值总是负的，那么反向传播过程经过该处的梯度恒为 0, 对应的权重和偏置参数此次无法得到更新。如果对于所有的样本输入，该激活函数的输入都是负的，那么该神经元再也无法学习，称为神经元”死亡“问题。

Leaky ReLU 的提出就是为了解决神经元”死亡“问题，Leaky ReLU 与 ReLU 很相似，仅在输入小于 0 的部分有差别，ReLU 输入小于 0 的部分值都为 0，而 LeakyReLU 输入小于 0 的部分，值为负，且有微小的梯度使用 Leaky ReLU 的好处就是：在反向传播过程中，对于 Leaky ReLU 激活函数输入小于零的部分，也可以计算得到梯度 (而不是像 ReLU 一样值为 0)，这样就避免了梯度方向锯齿问题。

3. ELU(Exponential Linear Unit)

理想的激活函数应满足两个条件：

1. 输出的分布是零均值的，可以加快训练速度。

2. 激活函数是单侧饱和的，可以更好的收敛。

LeakyReLU 和 PReLU 满足第 1 个条件，不满足第 2 个条件；而 ReLU 满足第 2 个条件，不满足第 1 个条件。两个条件都满足的激活函数为 ELU(Exponential Linear Unit)

$$f(x)=\begin{cases} x, & x>0 \\ \alpha\left(e^{x}-1\right), & x\leq 0 \end{cases}$$

3.1 node

(由于 ppi 的形式和 cora、citeseer 不同，是 20 个图，因此在前两个的基础上再使用 simple_dataloader 继续打乱之后再训练（训练函数 train2 稍有不同）)

PairNorm 是一种规范化的方法来处理过平滑问题。图神经网络（GNN）的性能随着层数的增加而逐渐降低。这种衰减部分归因于过度平滑，其中重复的图形卷积最终使节点嵌入难以区分。PairNorm 的关键思想是确保总的成对特征距离在各层之间保持恒定，使得相隔较远的节点对具有较少的相似特征，从而防止特征在整个群集中混合。

PairNorm 能被直接的运用，并且不会引入额外的参数，它被简单地运用在每一层（除了最后一层）的输出特征上，由简单的操作组成，主要是中心化和缩放，它们和输入的大小成线性关系。对于一些分类任务，运用浅层 GNN 就已经足够了，因此 PairNorm 虽然可以阻止性能随着层数增加显著下降，但是不一定能绝对提高性能。但是在现实世界中，有的分类任务，比如有一部分节点特征缺失，此时可能需要更广泛的邻域，也就是需要更深的层数，节点的特征才能被有效的恢复出来，这种情景下，深层 GNN 对分类任务更有效，这时运用 PairNorm 能显著超越其他模型，PairNorm 的优势也就体现出来了。

PariNorm 能被表示为两步，中心化和放缩：

$$\begin{aligned} \tilde{\mathbf{x}}_i^c &= \tilde{\mathbf{x}}_i - \tfrac{1}{n}\sum_{i=1}^{n}\tilde{\mathbf{x}}_i \\ \dot{\mathbf{x}}_i &= s\cdot\frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|\tilde{\mathbf{x}}_i^c\right\|_2^2}} = s\sqrt{n}\cdot\frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\left\|\tilde{\mathbf{X}}^c\right\|_F^2}} \end{aligned} \tag{5}$$

在实验中，初试 cora 训练，用交叉熵作为损失计算方式，relu 作为激活函数，层数设置为 2，隐藏神经元为 64，不使用 pairnorm 时，见图4。

考虑通过训练过程比较不同的层数对于模型结果的影响，对于 cora 数据集用交叉熵作为损失计算方式，relu 作为激活函数，隐藏神经元为 64，不使用 pairnorm，层数设置为 1-5 五种，进行训练，绘制 lossval 图如图5. 可以看出此时，2 层的结果明显优于其他的设置。再比较不同的隐藏神经元数量对于模型的影响，从 8 到 64 每隔 8 个训练一次，结果如图6. 比较显著的可以判断选择 56 作为隐藏神经元数相对最好。

由此，选择用交叉熵作为损失计算方式，relu 作为激活函数，隐藏神经元为 56，不使用 pairnorm，层数设置为 2，再对 citeseer 数据集训练结果如图7. 训练结果为：

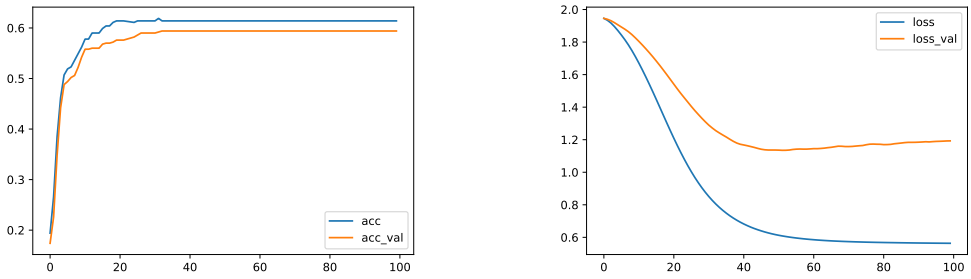

图 4: cora 的训练信息

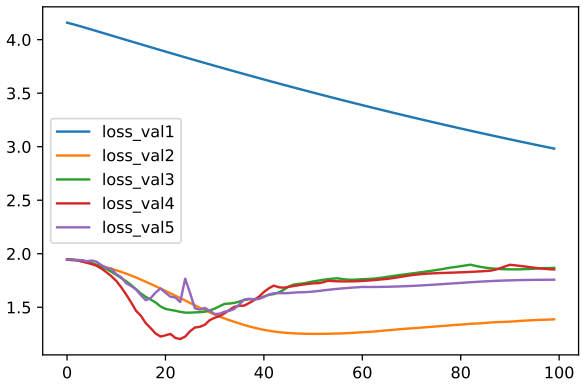

图 5: cora num_layers

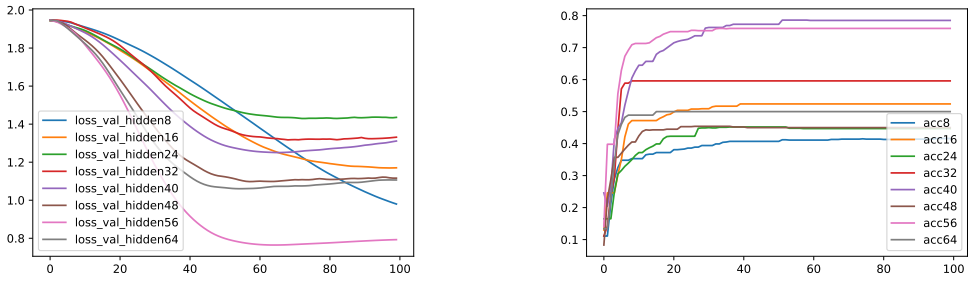

图 6: cora hidden_dim

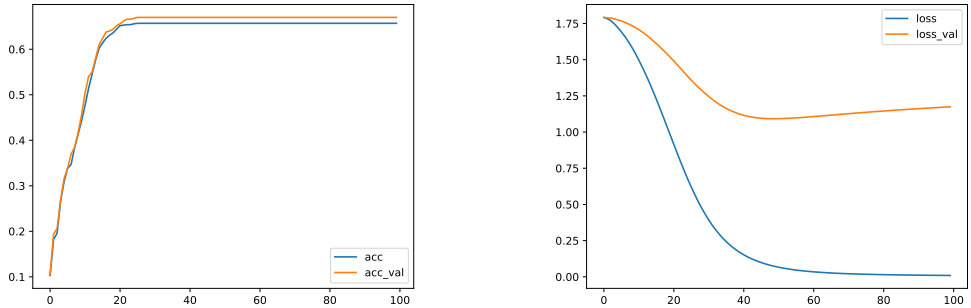

图 7: citeseer GCN

(tensor(0.6740), tensor(1.0768))

对于 PPI 数据集，simple_dataloader 之后随机取下标作为训练、测试、验证集，将 20 个图分别用上述的参数设置的模型进行训练，取每次训练结果的最大 acc 和最小 lossval，对 20 组数据平均处理，结果为：

(tensor(0.6570), tensor(1.0921))

(tensor(0.7271), tensor(178.2713))

进一步分析：

还是使用 cora 数据集，由于 PairNorm 在深层才有作用，选 2,4,8,16 层再次训练（此时其他参数暂时选择之前上述的参数），结果如图8.

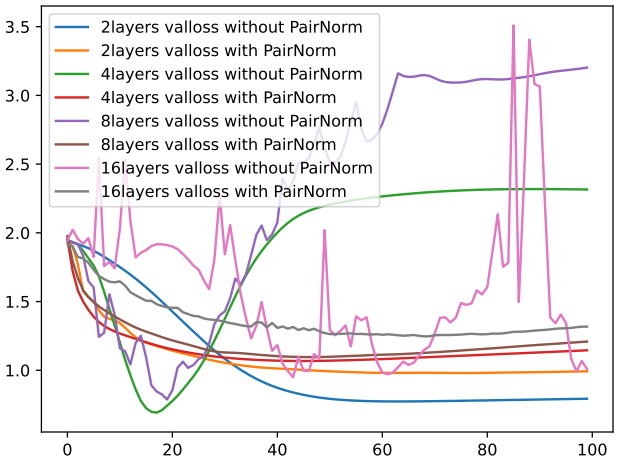

图 8: cora PairNorm

可以看出，对比相同层数的不用 PairNorm 的模型，使用 PairNorm 确实可以在深度加深的时候提高模型都性能。但是在我的 GCN 中，就算只使用 2 层的效果还是很好的。再比较不同的激活函数的影响（2 层且不使用 PairNorm，其他参数不变），再次对 cora 数据集训练，结果见图9.

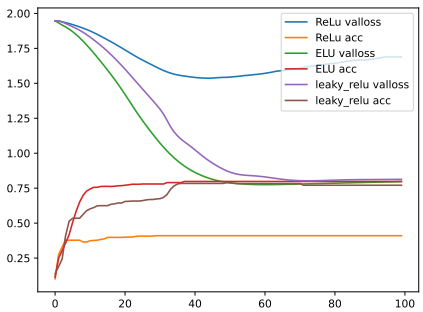

图 9: cora 激活函数影响

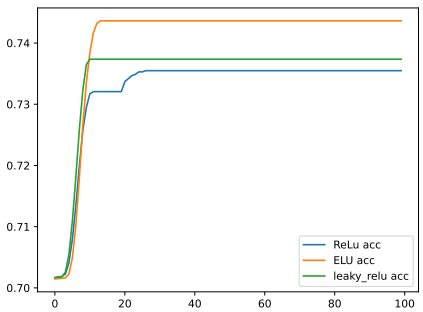

图 10: ppi 激活函数影响

对这个数据集发现 elu 比 relu 明显好一些，leaky relu 的效果和 elu 相似，由于 citeseer 和 cora 数据集的形式一样，用 ppi 的一个图再验证一下，结果如图10. ELU 比较明显地合适一些。

3.2 link

根据 demo 的算法，修改并整理为自己需要的函数模型，并在 GraphSAGE 中嵌入 DropEdge，初始化为不使用。

过拟合以及过平滑是影响节点分类准确度的两个主要阻碍。过拟合会弱化模型在小数据上的泛化能力，在深层网络中过平滑会使得输入的特征与输出的表示相隔离从而会妨碍模型的训练。

DropEdge 方法可以通过这种方法减弱过拟合以及过平滑所带来的问题。这种方法的核心是在每次训练中从输入的图中随机去掉一些边，就像是一个数据增强器以及消息传递的减弱器，然后，本文从理论上分析了 DropEdge 减弱过平滑的收敛速度以及减弱信息的损失。通过实验证明 DropEdge 方法可以增强浅层以及深层模型的能力。

在每次训练过程中，DropEdge 奖罚随机丢弃了一些边，使邻接矩阵 A 中的 Vp 个非零元素为零，其中 V 是边数，v 是丢弃率，新产生的邻接矩阵为 A_{drop}:

$$\boldsymbol{A}_{\text{drop}} = \boldsymbol{A} - \boldsymbol{A}', \tag{6}$$

使用 dgl.add_self_loop() 时，为图中的每个节点添加自循环并返回一个新图。

由此，对于 cora 数据集先在不添加自环，不使用 DropEdge，使用 relu 激活函数，2 层时，从 16 到 64 每隔 8 个训练一次，对于隐藏神经元的训练结果见图11, acc 为：

[0.8977300599716987, 0.9117045888457133, 0.9217340131623278, 0.9267437838323487, 0.9325891152489836, 0.9402106870914849, 0.9371927854271019]

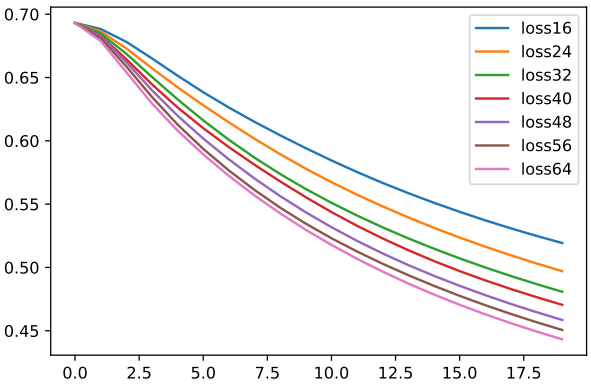

图 11: cora GraphSAGE hidden_dim

56 还是好，再比较层数的区别（隐藏为 56，其他参数如上不变），从 1 到 5 的层数的训练结果见图12, acc 为：

[0.9317652343837741, 0.9341510747737024, 0.9344480132970957, 0.9169075267851127, 0.9087837200422274]

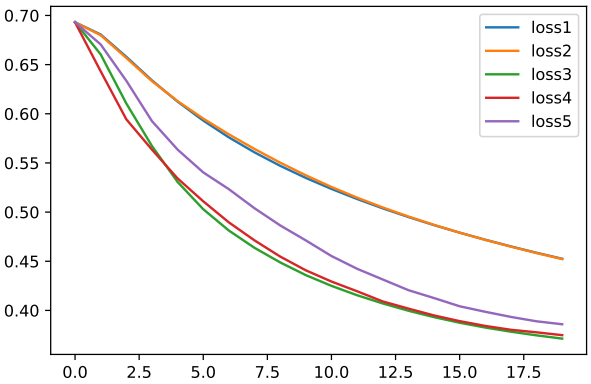

图 12: cora GraphSAGE num_layers

从结果上，3 层的效果相对较好。在 cora 数据集上在比较有无 DropEdge 的区别 (层数此时为 3，其余参数不变)。对于使用 DropEdge 时，p 的值从 0.6 开始每隔 0.05 取一个一直到 1，将训练结果绘制如图13. 此时，acc 为：

[0.8331048269356034, 0.8451059050785024, 0.8503807192111588, 0.8613324049325036, 0.8493897261966262, 0.8611500190921137, 0.8688376271871701, 0.8509081107791829, 0.8379668021832394, 0.8599079086273892]

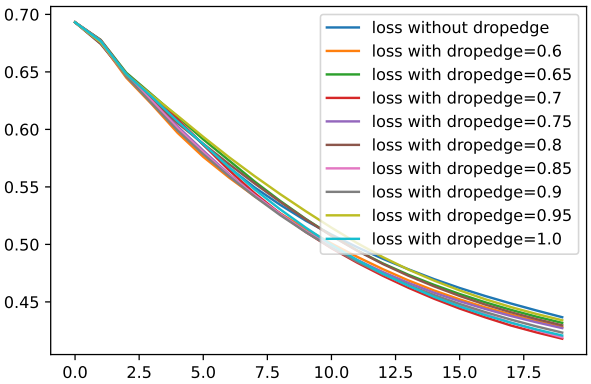

图 13: cora GraphSAGE DropEdge

0.7,0.8,0.85 的效果相对明显好一点，选 0.85（它最大，loss 第二好），比较不同激活函数,loss 训练曲线见图14。acc 为：

[0.8595287617079581, 0.852263875474495, 0.8616455155993802]

ELU 激活时的 loss 收敛快效果好，但 leaky_relu 的预测结果好，用 ELU 看有无自环的区别，结果如图15。acc 为：

[0.8354888704207004, 0.8626404500772348]

对于 citeseer 数据集，参数设定为隐藏神经元 56，层数为 3，激活函数使用 ELU，使用 DropEdge，p 设置为 0.85。对于有无自环的结果模型拟合见图16. acc 为：

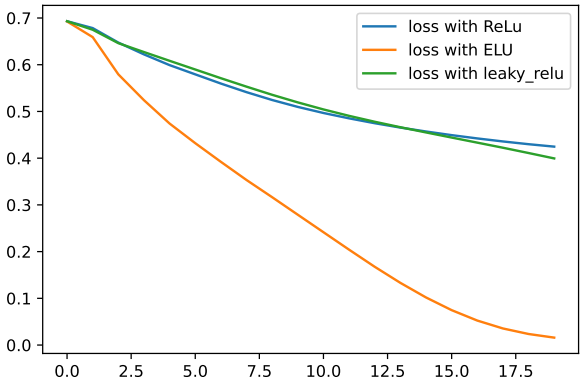

图 14: cora GraphSAGE 激活函数

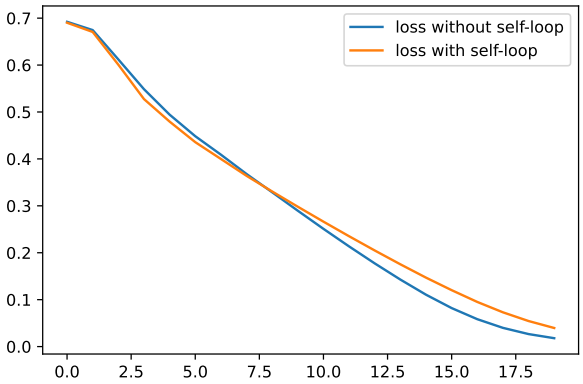

图 15: cora GraphSAGE 自环

[0.8602632210463906, 0.8880500309518897]

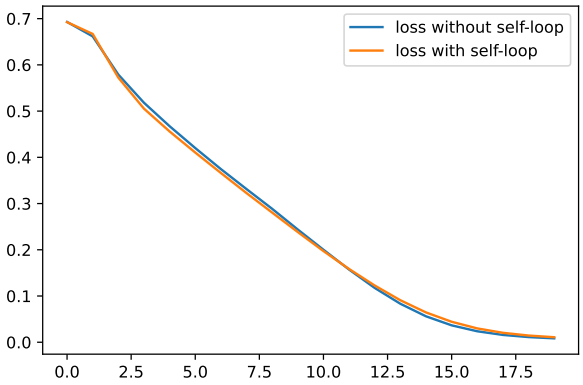

图 16: citeseer GraphSAGE 自环

对于 ppi 数据集，依旧采用随机打乱，将 20 张图一个一个学习模型，展示最终的平均值 acc 和 loss 为：
无自环：

tensor(0.6152, dtype=torch.float64) tensor(0.6691)

有自环：

tensor(0.6192, dtype=torch.float64) tensor(0.6646)

4 一点小总结

由于模型数据不是很大，因此在 CPU 和 GPU 上跑从速度上差别不大。但由于 ppi 的图如果循环建模读取放入 GPU 显存中，可能会有些困难（比如在 node 训练时一半显存爆了），所以最终剩余的学习都使用 CPU 训练。