

浙江大学

本科实验报告

课程名称： 自然语言处理

姓 名： 张溢弛

学 院： 计算机科学与技术学院

专 业： 软件工程 1801

学 号： 3180103772

指导教师： 汤斯亮

2021 年 6 月 7 日

浙江大学 实验报告

课程名称：自然语言处理 实验类型：综合型

实验项目名称：实验三：基于图神经网络的文本分类

学生姓名：张溢弛 专业：软件工程 1801 学号：3180103772

同组学生姓名：独立完成 指导老师：汤斯亮

实验地点：玉泉一舍 376 实验日期：2021 年 6 月 7 日

目录

一 项目介绍	IV
1.1 实验选题	IV
1.2 实验内容	IV
1.3 实验环境	IV
二 技术细节	IV
2.1 技术原理	IV
2.1.1 图神经网络	IV
2.1.2 图卷积网络	IV
2.1.3 基于谱分解的方法	V
2.1.4 谱分解方法的优缺点分析	VI
2.1.5 基于空间结构的方法	VI
2.2 实验细节与关键步骤	VII
2.2.1 实验环境配置	VII
2.2.2 代码文件结构	VII
2.2.3 模型文件结构	VIII
2.2.4 数据集	VIII
2.2.5 数据预处理	IX
2.2.6 模型的训练	IX
2.3 自己的 Contribution	X
三 实验结果	X
3.1 学习率与 epoch 数的探究	XI

	III
3.2 dropout 与隐层维度	XII
3.3 自己的 contribution	XII
四 总结思考	XIII

一 项目介绍

1.1 实验选题

基于图卷积神经网络的科学出版物分类实验

1.2 实验内容

使用 Mindspore 框架实现图卷积神经网络架构，并将其应用到科学出版物数据集中完成分类，同时我对 GCN 模型训练时的参数设定做出了一定的探究，探究了不同参数对模型训练的影响。

1.3 实验环境

- 操作系统：华为云 ModelArts 云平台
- 编程环境：Python3+Jupyter Notebook
- 使用的 Python 库：mindspore1.1.1

二 技术细节

2.1 技术原理

在开始实验之前我先深入学习了图神经网络相关的基本知识并做了一些笔记，摘录如下：

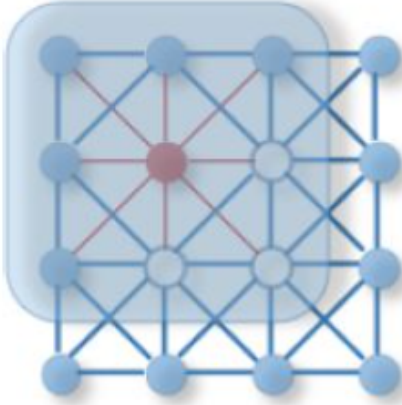
2.1.1 图神经网络

图神经网络是一类对图结构的数据进行处理的神经网络结构，常见的有图卷积神经网络、图循环神经网络、图时空神经网络和图自编码器等类型，相比于适合处理欧式数据结构的传统的神经网络架构如 CNN 和 RNN，图神经网络更加适合处理图结构的数据，并且可以有效地提取出图结构的邻域信息。

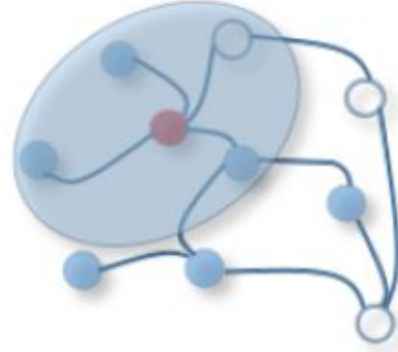
2.1.2 图卷积网络

卷积图神经网络的提出收到了卷积神经网络 CNN 的启发，CNN 中通过卷积层这样一种特殊的神经网络层，使用一系列卷积核来提取图像等结构中的局部特征，取得了非常好的效果，而图卷积神经网络就是将“卷积”操作推广到了图结构中的神经网络，而图作为一种非欧式数据结构，无法直接定义有效的卷积操作，这就衍生出了图卷积神经网络的两种

不同的发展方向，分别是基于谱分解的方法和基于空间结构的办法。



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.



(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of graph convolution operation takes the average value of node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

2.1.3 基于谱分解的方法

基于谱分解的方法有着非常坚实的数学背景和可解释性，这种方法是最早被提出的。基于谱分解的方法从图形信号处理的角度引入滤波器，并在傅立叶域中定义图的卷积运算。这里面用到了一种非常重要的图表示方式——拉普拉斯矩阵，图的拉普拉斯矩阵被定义为：

$$L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \quad (1)$$

其中 D 是图的度数对焦矩阵而 A 是图的邻接矩阵，这是一种重要的图的特征表示方式，且是一个实对称的半正定矩阵，因此可以被分解为 $L = U \Lambda U^T$ 的形式，并且矩阵 U 是 L 的特征值所构成的对角矩阵。关于谱分解的数学证明这里就不再赘述，仅作简要介绍。在此基础上，基于谱分解的图卷积网络将由节点特征表示组成的图信号输入 x 进行图傅立叶变换 $\mathcal{F}(x) = U^T x$ ，这种图傅立叶变换将输入的图信号映射到了以图的拉普拉斯矩阵的特征值为基的正交空间中。这样一来图上的卷积操作就很自然地被定义为：

$$\begin{aligned} x *_G g &= \mathcal{F}^{-1}(\mathcal{F}(x) \odot \mathcal{F}(g)) \\ &= U (U^T x \odot U^T g) \end{aligned} \quad (2)$$

这里的 g 表示卷积核，如果我们定义 $g_\theta = \text{diag}(U^T g)$ 那么这样一来图卷积可以简化成：

$$x *_G g_\theta = U g_\theta U^T x \quad (3)$$

所有的图卷积网络都遵循这种形式的卷积定义，区别在于卷积核的选取方式。

2011 年，David Hammond 等人提出可以用切比雪夫多项式的前 K 阶来逼近 $g_\theta(\Lambda)$ 从而提出了 ChebNet，2017 年的时候 Thomas Kipf 等人在 ChebNet 的基础上将层级卷积运算的 K 限制为 1，缓解了 ChebNet 中存在的过拟合问题，并对滤波器的参数进行了简化，最终提出了 GCN，而在此基础上又出现了可以挖掘不同节点之间隐式关系的自适应图卷积网络 (AGCN)，这种形式的图卷积网络会从图邻接矩阵中学习出“残差”拉普拉斯矩阵并将其添加到原始的拉普拉斯矩阵中，并通过超参数的调节实现图结构中的自适应，可以挖掘出图中的隐藏关系。

2.1.4 谱分解方法的优缺点分析

谱分解方法虽然将卷积操作引入了图结构中，是一种重大突破，然而基于谱分解的卷积定义都依赖于图的拉普拉斯矩阵的特征基向量，而这完全取决于图本身的结构，这意味着基于谱分解的方法存在一个严重的缺点，那就是模型必须针对不同结构的图进行针对性的训练，而不能直接使用到其他的图中，一言蔽之就是基于谱分解定义的图卷积的泛化能较差。

2.1.5 基于空间结构的方法

和谱分解方法相反，基于空间结构的方法直接在图上定义卷积运算，并针对空间上相邻的邻域 (即图中某个节点附近的若干节点组成的区域) 进行卷积运算，这种方法的主要挑战是如何针对不同大小的邻域来定义卷积并且保持卷积操作的局部不变性 (谱分解的方法通过对不同的图定义不同的卷积核解决了这个问题)。

David Duvenaud 等人在 2015 年的时候就提出了针对不同度数的节点使用不同度数的权重矩阵 $W_t^{|N_v|}$ (N_v 表示节点 v 的邻居节点构成的集合)，这样一来图节点的嵌入表示的更新方式就变成了：

$$\begin{aligned} x &= h_v^{t-1} + \sum_{i=1}^{|N_v|} h_i^{t-1} \\ h_v^t &= \sigma(x W_t^{|N_v|}) \end{aligned} \quad (4)$$

这种模型现将节点自己和所有邻居节点的嵌入表示相加再使用一个权重矩阵进行投影得到新的嵌入表示，这种模型的主要缺陷在于，在大规模的图结构中节点的度数往往有很多不同取值，这就会导致模型的参数过多而难以应用。

而 GraphSAGE 结构是一种通用的归纳推理框架，通过采样和聚合相邻节点的特征来

生成节点的嵌入表示，其传播过程可以描述为：

$$\begin{aligned} h_{N_v}^t &= \text{AGGREGATE}(h_u^{t-1}, u \in N_v) \\ h_v^t &= \sigma(W^t[h_v^{t-1} || h_{N_v}^t]) \end{aligned} \quad (5)$$

即现将邻居节点的嵌入表示进行一定的聚合 (通过聚合函数 AGGREGATE)，然后和当前节点的嵌入进行组合并投影到嵌入空间中生成新的节点嵌入，和其他的基于空间结构的模型不同的是，GraphSAGE 并不会使用所有的相邻节点，而是均匀采样固定数量的邻居节点，而聚合函数通常有均值聚合，LSTM 聚合和最大池化聚合等多种选择，分别对应不同的特征提取需求而选用。我们可以将 GraphSAGE 中的聚合操作理解为是对 CNN 中池化操作的一种推广。

2.2 实验细节与关键步骤

本次实验的细节和关键的代码主要分为如下几个部分，其中实验结果将在第三部分具体给出。

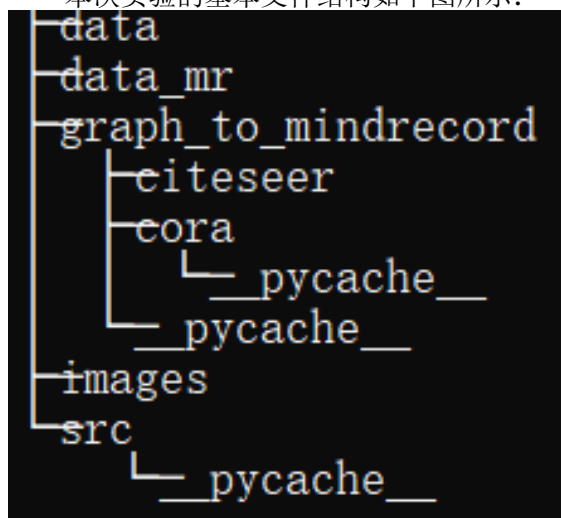
2.2.1 实验环境配置

注册华为云平台的账号，由于代金券还没有到账因此自费充值来使用 ModelArts 平台，按照教程中创建好 OBS 桶并在 ModelArts 中配置好实验环境之后开始实验，其中数据集需要通过 moxing 库从 OBS 桶中复制到云平台当前的工作目录下。

同时本次实验使用了新版的实验手册，因此

2.2.2 代码文件结构

本次实验的基本文件结构如下图所示：



其中 graph_to_mindrecord 目录下面包含了数据集预处理所需的一系列代码，用途是将数据集转化成 mindrecord 格式用于模型的训练，而 src 目录下包含了 GCN 模型的相关

代码。





















2.2.3 模型文件结构

本实验中的核心代码在 `src` 目录下，其中 `dataset.py` 是数据集处理的模块，`gcn.py` 中实现了图卷积网络模型，而 `metrics.py` 中包含了模型训练过程中的一些评价指标，比如损失函数和精确度。

2.2.4 数据集

本次实验室用的是 Cora 和 CiteSeer 数据集，这是两个图神经网络中非常常见的数据集，Cora 数据集包含 2708 个科学出版物，分为七个类别。引用网络由 5429 个链接组成。数据集中的每个出版物都用一个 0/1 值的词向量描述，0/1 指示词向量中是否出现字典中相应的词。该词典包含 1433 个独特的单词。CiteSeer 数据集包含 3312 种科学出版物，分为六类。引用网络由 4732 个链接组成。数据集中的每个出版物都用一个 0/1 值的词向量描述，0/1 指示词向量中是否出现字典中相应的词。该词典包含 3703 个独特的单词。

而得益于华为提供的实验手册，本次实验中使用了已经处理好的数据集 (这里的截图不全):

 ind.citeseer.allx	2021/4/9 10:13	ALLX 文件	582 KB
 ind.citeseer.ally	2021/4/9 10:13	ALLY 文件	55 KB
 ind.citeseer.graph	2021/4/9 10:13	GRAPH 文件	61 KB
 ind.citeseer.test.index	2021/4/9 10:13	INDEX 文件	5 KB
 ind.citeseer.tx	2021/4/9 10:13	TX 文件	255 KB
 ind.citeseer.ty	2021/4/9 10:13	TY 文件	24 KB
 ind.citeseer.x	2021/4/9 10:13	X 文件	31 KB
 ind.citeseer.y	2021/4/9 10:13	Y 文件	3 KB
 ind.cora.allx	2021/4/9 10:13	ALLX 文件	252 KB
 ind.cora.ally	2021/4/9 10:13	ALLY 文件	47 KB
 ind.cora.graph	2021/4/9 10:13	GRAPH 文件	59 KB
 ind.cora.test.index	2021/4/9 10:13	INDEX 文件	5 KB
 ind.cora.tx	2021/4/9 10:13	TX 文件	145 KB
 ind.cora.ty	2021/4/9 10:13	TY 文件	28 KB
 ind.cora.x	2021/4/9 10:13	X 文件	22 KB
 ind.cora.y	2021/4/9 10:13	Y 文件	4 KB
 ind.pubmed.allx	2021/4/9 10:13	ALLX 文件	7,401 KB
 ind.pubmed.ally	2021/4/9 10:13	ALLY 文件	220 KB
 ind.pubmed.graph	2021/4/9 10:13	GRAPH 文件	461 KB
 ind.pubmed.test.index	2021/4/9 10:13	INDEX 文件	6 KB

并且处理后的数据集包括了:

- `x`, 已标记的训练实例的特征向量
- `y`, 已标记的训练实例的 one-hot 标签
- `allx`, 标记的和未标记的训练实例 (`x` 的超集) 的特征向量

- graph, 一个 dict, 格式为 index: [index_of_neighbor_nodes]
- tx 和 ty, 测试实例的特征向量以及 one-hot 标签
- test.index, graph 中测试实例的索引
- ally, 是 allx 中实例的标签。

2.2.5 数据预处理

虽然实验已经提供了处理后的数据集，但是在正式的实验过程中仍然需要对数据集进行一定的格式转换，将其转化成 mindrecord 格式的数据，实验中得到的预处理结果如下：

转换数据格式

```
In [6]: print("==== Graph To Mindrecord ====")
run(cfg)
```

```
==== Graph To Mindrecord =====
Init writer ...
exec task 0, parallel: False ...
Node task is 0
transformed 512 record...
transformed 1024 record...
transformed 1536 record...
transformed 2048 record...
transformed 2560 record...
Processed 2708 lines for nodes.
transformed 2708 record...
exec task 0, parallel: False ...
Edge task is 0
transformed 512 record...
transformed 1024 record...
transformed 1536 record...
transformed 2048 record...
transformed 2560 record...
transformed 3072 record...
transformed 3584 record
```

2.2.6 模型的训练

在设置好训练参数之后就可以进行模型的训练，按照实验指导书中给出的样例代码里提供的参数，模型的训练结果如下图所示：

```

print("===== Starting Training =====")
train_eval(cfg)
# 此时的参数是
"""
learning_rate = 0.01
epochs = 200
hidden1 = 16
dropout = 0.5
weight_decay = 5e-4
early_stopping = 10
"""

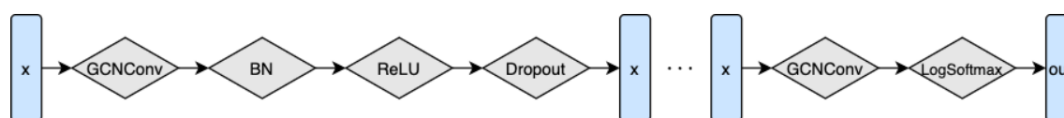
===== Starting Training =====
Epoch: 0000 train_loss= 1.95344 train_acc= 0.20000 val_loss= 1.94884 val_acc= 0.32600 time= 36.86902
Epoch: 0010 train_loss= 1.86379 train_acc= 0.82143 val_loss= 1.90587 val_acc= 0.49600 time= 0.00332
Epoch: 0020 train_loss= 1.75816 train_acc= 0.87857 val_loss= 1.86139 val_acc= 0.52800 time= 0.00288
Epoch: 0030 train_loss= 1.60205 train_acc= 0.87143 val_loss= 1.80741 val_acc= 0.55800 time= 0.00283
Epoch: 0040 train_loss= 1.45929 train_acc= 0.88571 val_loss= 1.74227 val_acc= 0.57800 time= 0.00278
Epoch: 0050 train_loss= 1.30949 train_acc= 0.95000 val_loss= 1.67165 val_acc= 0.65400 time= 0.00282
Epoch: 0060 train_loss= 1.18356 train_acc= 0.95000 val_loss= 1.59673 val_acc= 0.72000 time= 0.00280
Epoch: 0070 train_loss= 1.05597 train_acc= 0.96429 val_loss= 1.52379 val_acc= 0.74800 time= 0.00282
Epoch: 0080 train_loss= 0.97079 train_acc= 0.97143 val_loss= 1.46297 val_acc= 0.76000 time= 0.00281
Epoch: 0090 train_loss= 0.88234 train_acc= 0.97143 val_loss= 1.39307 val_acc= 0.77800 time= 0.00281
Epoch: 0100 train_loss= 0.81958 train_acc= 0.98571 val_loss= 1.33636 val_acc= 0.77600 time= 0.00284
Epoch: 0110 train_loss= 0.77699 train_acc= 0.96429 val_loss= 1.28415 val_acc= 0.78000 time= 0.00285
Epoch: 0120 train_loss= 0.69423 train_acc= 0.98571 val_loss= 1.24872 val_acc= 0.78600 time= 0.00282
Epoch: 0130 train_loss= 0.67243 train_acc= 0.98571 val_loss= 1.22011 val_acc= 0.78200 time= 0.00281
Epoch: 0140 train_loss= 0.63718 train_acc= 0.97857 val_loss= 1.17916 val_acc= 0.78200 time= 0.00281
Epoch: 0150 train_loss= 0.61189 train_acc= 0.97143 val_loss= 1.15654 val_acc= 0.78400 time= 0.00282
Epoch: 0160 train_loss= 0.56969 train_acc= 0.99286 val_loss= 1.12524 val_acc= 0.78800 time= 0.00282
Epoch: 0170 train_loss= 0.55138 train_acc= 0.98571 val_loss= 1.11112 val_acc= 0.78600 time= 0.00279
Epoch: 0180 train_loss= 0.55572 train_acc= 0.99286 val_loss= 1.09941 val_acc= 0.78400 time= 0.00280
Epoch: 0190 train_loss= 0.52402 train_acc= 0.98571 val_loss= 1.07182 val_acc= 0.78400 time= 0.00280
Test set results: loss= 1.00606 accuracy= 0.81900 time= 0.73015

```

2.3 自己的 Contribution

在运行给定代码的基础上，我还自己进行了一系列的学习和尝试，包括：

- 学习了斯坦福 CS224W 中关于 GNN 的相关内容，并做了一些简单的整理（也就是上面的 2.1）
- 通过调节训练参数并比较 GCN 训练后的预测准确率，探究了影响图神经网络训练效果的各种因素，并做出总结，这一部分主要体现在实验报告的第三部分。
- 同时在学习了图神经网络基本知识和完成 lab3 的基本内容的情况下，完成了 CS224W 课程中公开的 lab2，并实现了一个如下架构的图卷积神经网络，并在 OGB 数据集上进行了训练和测试：



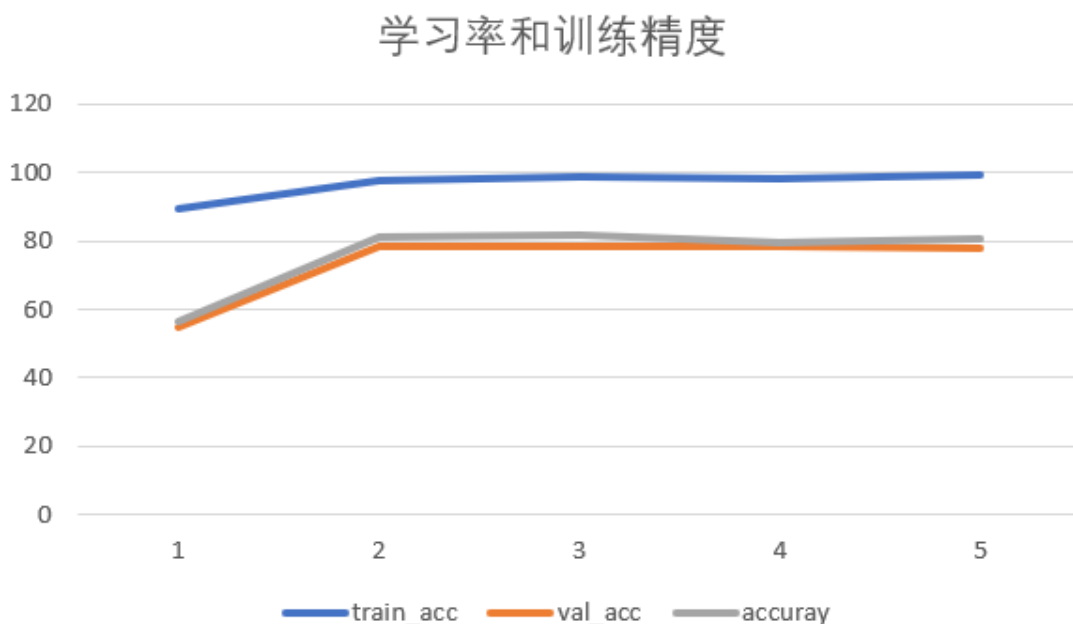
三 实验结果

在运行给定的 baseline 代码的基础上，我还自己进行了一系列的参数调节和训练，统计得到的训练结果和如下：

实验编号	learning_rate	epoch	hidden	dropout	weight_decay	early_stopping	train_acc	val_acc	accuracy
0	0.01	200	16	0.5	5.00E-04	10	98.57	78.4	81.9
1	0.01	100	16	0.5	5.00E-04	10	97.86	78	81.1
2	0.01	100	16	0.75	5.00E-04	10	94.28	77.6	80.5
3	0.01	100	16	0.01	5.00E-04	10	98.5	77.8	80.8
4	0.1	200	16	0.5	5.00E-04	10	99.29	78	80.5
5	0.1	200	16	0.5	5.00E-04	10	99.29	77	79.5
6	0.05	200	16	0.5	5.00E-04	10	98.5	78.6	79.7
7	0.005	200	16	0.5	5.00E-04	10	97.86	78.2	81
8	0.001	200	16	0.5	5.00E-04	10	89.2	54.8	56.2
9	0.001	500	16	0.5	5.00E-04	10	95	73	74.8
10	0.005	500	16	0.5	5.00E-04	10	99.286	79.2	81.4
11	0.01	500	16	0.5	5.00E-04	20	100	78.6	81.4
12	0.01	200	16	0.5	1.00E-03	20	97.86	78.2	80.7
13	0.01	200	32	0.5	1.00E-03	20	99.3	78.6	81.9
14	0.01	200	64	0.5	1.00E-03	20	100	77.6	81.4
15	0.01	200	4	0.5	1.00E-03	20	70.7	42.6	44.1
16	0.01	200	32	0	1.00E-03	20	100	79	80.2
17	0.01	200	128	0.5	1.00E-03	20	99.28	80.2	82.3
18	0.01	200	128	0.5	1.00E-03	50	95.57	78.6	80.5
19	0.01	500	64	0.5	1.00E-03	50	98.578	78.4	80.3

对上面的数据进行一定的可视化，我们可以分析出一些结论。

3.1 学习率与 epoch 数的探究



从上图中我们可以分析得到，模型训练的精确度随着学习率的变大总体呈现先增后减的态势，如果学习率过小可能在有限的 epoch 中无法完成足够多的梯度下降，而学习率过大则可能引起训练时的波动反而使得训练效果变差，而我们从表格中又可以看出，学习率合理的情况下，epoch 数的增加一定程度上可以提高训练的效果，所以在训练的时候应该综合考虑 epoch 数和学习率。

3.2 dropout 与隐层维度

dropout 是一种非常常见的防止过拟合的神经网络训练策略，在我做的实验中，组别 1, 2, 3 对 dropout 做出了一定的探究，发现 dropout 对准确率影响不是很大，可能是因为这一组模型的其他参数的选取比较合理使得模型在 dropout 上的差异化表现并不大。

而隐层的维度也是影响模型性能的一个非常重要的指标，一般来说隐层的维度设定的比较小的时候会学习到更稠密的特征表示，但也可能会因为维度过低而丢失大量的有效信息，而维度比较大的时候会学到更稀疏的表示，但是计算的复杂度大大提高了，并且过于稀疏的表示也不能很好地表示一系列特征，本次实验中的 12-15 和 18 这几组数据对隐层维度的问题做出了探究，最终发现 128 维的时候学习到的效果是对照组中最好的。

3.3 自己的 contribution

我在实验本身的基础上完成了 CS224W 中的 lab2 的 GCN 模型，其训练的过程截图如下：

```
for epoch in range(1, 1 + args["epochs"]):
    loss = train(model, data, train_idx, optimizer, loss_fn)
    result = test(model, data, split_idx, evaluator)
    train_acc, valid_acc, test_acc = result
    if valid_acc > best_valid_acc:
        best_valid_acc = valid_acc
        best_model = copy.deepcopy(model)
    print(f'Epoch: {epoch:02d}, '
          f'Loss: {loss:.4f}, '
          f'Train: {100 * train_acc:.2f}%, '
          f'Valid: {100 * valid_acc:.2f}%, '
          f'Test: {100 * test_acc:.2f}%')
```

/Users/randomstar/opt/anaconda3/envs/cs224n/lib/python3.7/site-packages/ipyk
for log_softmax has been deprecated. Change the call to include dim=X as an

```
Epoch: 01, Loss: 4.2904, Train: 17.50%, Valid: 25.29% Test: 22.96%
Epoch: 02, Loss: 2.3894, Train: 27.10%, Valid: 26.98% Test: 32.24%
Epoch: 03, Loss: 1.9281, Train: 25.23%, Valid: 23.34% Test: 28.74%
Epoch: 04, Loss: 1.7637, Train: 29.60%, Valid: 24.86% Test: 28.58%
Epoch: 05, Loss: 1.6562, Train: 30.19%, Valid: 24.88% Test: 23.86%
Epoch: 06, Loss: 1.5586, Train: 33.43%, Valid: 28.83% Test: 27.78%
Epoch: 07, Loss: 1.4938, Train: 36.04%, Valid: 32.13% Test: 32.44%
Epoch: 08, Loss: 1.4428, Train: 38.58%, Valid: 35.03% Test: 39.99%
Epoch: 09, Loss: 1.3992, Train: 38.81%, Valid: 32.22% Test: 36.37%
Epoch: 10, Loss: 1.3617, Train: 38.69%, Valid: 31.58% Test: 36.03%
Epoch: 11, Loss: 1.3359, Train: 39.86%, Valid: 33.24% Test: 37.50%
Epoch: 12, Loss: 1.3110, Train: 42.11%, Valid: 36.62% Test: 41.77%
Epoch: 13, Loss: 1.2886, Train: 45.46%, Valid: 41.23% Test: 46.96%
Epoch: 14, Loss: 1.2628, Train: 47.63%, Valid: 44.01% Test: 49.58%
Epoch: 15, Loss: 1.2472, Train: 49.99%, Valid: 46.62% Test: 51.38%
Epoch: 16, Loss: 1.2295, Train: 51.22%, Valid: 47.99% Test: 52.58%
Epoch: 17, Loss: 1.2162, Train: 52.97%, Valid: 51.57% Test: 55.19%
```

完成了之后我按照实验中的指导在 OGB 数据集上测试了模型的效果，得到的结果是：

```

train_acc = eval(best_model, device, train_loader, evaluator)[dataset.eval_metric]
valid_acc = eval(best_model, device, valid_loader, evaluator)[dataset.eval_metric]
test_acc = eval(best_model, device, test_loader, evaluator)[dataset.eval_metric]

print(f'Best model: '
      f'Train: {100 * train_acc:.2f}%, '
      f'Valid: {100 * valid_acc:.2f}%, '
      f'Test: {100 * test_acc:.2f}%')

```

```
HBox(children=(HTML(value=' Iteration'), FloatProgress(value=0.0, max=1029.0), HTML(value='')))
```

```
/Users/randomstar/opt/anaconda3/envs/cs224n/lib/python3.7/site-packages/ipykernel_launcher.py:76: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
```

```
HBox(children=(HTML(value=' Iteration'), FloatProgress(value=0.0, max=129.0), HTML(value='')))
```

```
/Users/randomstar/opt/anaconda3/envs/cs224n/lib/python3.7/site-packages/ipykernel_launcher.py:76: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
```

```
HBox(children=(HTML(value=' Iteration'), FloatProgress(value=0.0, max=129.0), HTML(value='')))
```

```
/Users/randomstar/opt/anaconda3/envs/cs224n/lib/python3.7/site-packages/ipykernel_launcher.py:76: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
```

```
Best model: Train: 80.03%, Valid: 79.11% Test: 71.44%
```

当然这和我使用 CPU 训练为了缩短时间减少了 epoch 数有关，总的来看这里的 GCN 模型的效果还是不错的。

四 总结思考

本次实验中我完成了图卷积网络模型的相关实验，深入了解了图神经网络和图卷积网络架构的特性，并在训练模型的过程中尝试探究了不同的参数对于模型性能的影响，并完成了同类型课程中的 GCN 模型实验，也算是对课堂所学知识和 lab 本身的一个补充。

同时我之前曾经尝试过旧版本的图神经网络实验，发现旧版本的实验中存在着很多问题和 bug，比如和华为云中现有的版本 1.1.1 有诸多 api 不匹配，而如果在本地搭建 1.0.0 环境运行代码则会出现一些算子不支持 CPU 运行的情况，当时心态有点崩，索性我将这些问题反馈给了华为云的客服和老师，最终新版本的实验做的比较顺利，这也算是我的一点 contribution 吧。

参考文献

[1] <http://web.stanford.edu/class/cs224n/index.html>

[2] <http://web.stanford.edu/class/cs224w/>