

数据挖掘实验报告

学院: 软件学院

专业: 软件工程

|  |  |
| --- | --- |
| *作者:*  周治平 | *指导教师:*  陶乾 |
|  |  |
| *学号:*  202030483178 | *班级:*  20级软件工程1班 |

2020-11-6

# 基于图神经网络的分类算法实验

**摘要**

图神经网络（Graph Neural Networks, GNN）作为新的人工智能学习模型，可以将实际问题看作图数据中节点之间的连接和消息传播问题，对节点之间的依赖关系进行建模，挖掘传统神经网络无法分析的非欧几里得空间数据的潜在信息。在自然语言处理、计算机视觉、生物化学等领域中，图神经网络得到广泛的应用，并发挥着重要作用。

图卷积神经网络(Graph Convolutional Networks, GCN)是目前主流的图神经网络分支，分类任务则是机器学习中的常见任务。本次实验，我们将利用GCN算法完成分类任务，通过实验进一步体会理解图神经网络工作的原理、GCN的构建实现过程，以及如何将GCN应用于分类任务。

## **介绍**

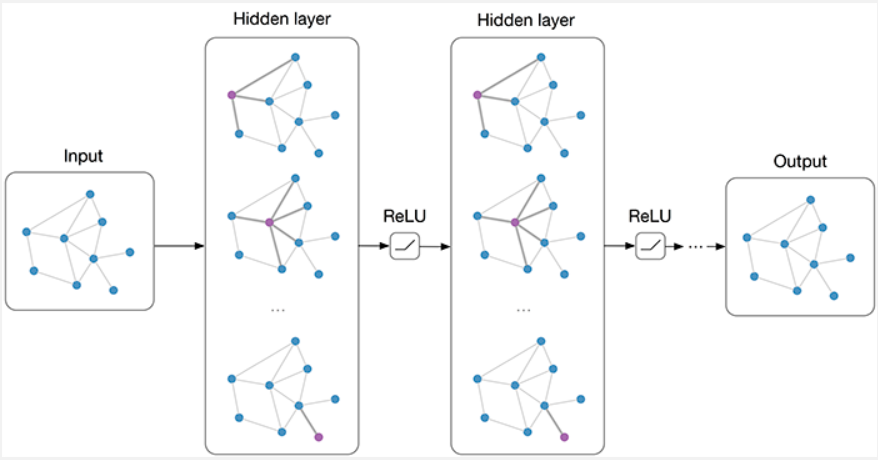
1. 理解GCN工作的原理
2. 熟悉Pytorch库的使用
3. 将理论进行实践，完成具体的机器学习分类任务

## **方法和理论**

1 GCN模型概述

为了对图数据进行处理和研究，提出了图卷积神经网络。图卷积神经网络（GCN），实际上跟CNN的作用一样，就是一个特征提取器，只不过它的对象是图数据。

GCN精妙地设计了一种从图数据中提取特征的方法，从而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），还可以得到图的嵌入表示（graph embedding）。



（Figure. 1.GCN模型结构图）

2 GCN模型公式

简单来说：GCN=GC+NN

GCN节点分类的核心公式：



最核心的公式是：

* 为变性后的邻接矩阵；
* X为节点特征值的矩阵；
* W是一个简单的可学习的权重矩阵，用来改变embedding的维度

该公式是一个两层的GCN，即第一层为，而经过Relu之后作为一下层X输入。

计算公式：



所以GCN算法的核心思想就是对邻居及自身节点的特征值加权求和

而边的权重，与节点度 D 有关，度越大，权重越小。可以理解为一个节点向邻居散播信息，邻居越多，信息越被稀释。

假如 i 节点与 j 节点是邻居，则该边的权重为，如果邻居是自身，则。

以上只是单层的GCN，双层GCN以第一次GCN输出的 emb 作为第二次的特征值 feat 输入，相当于聚合了两层邻居的特征值。

尽管GCN用了很多谱图卷积的理论，但最后的落脚点、核心思想其实是邻居（包括自身）特征值的加权求和。

3.损失函数

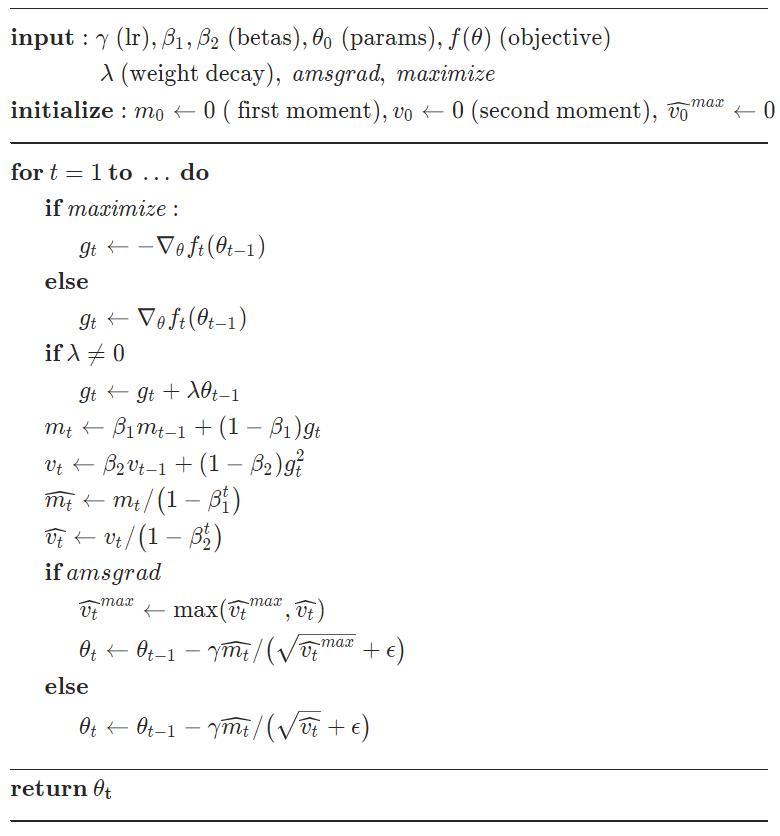
在本次实验中，GCN模型采用了NLLLOSS函数作为损失函数：



其中x是输入，y是输出，w是权重，N是批处理大小，默认是mean处理方法。本实验中使用的是mean处理方法。

4定义参数优化器

在本次实验中，GCN模型采用了Adam优化算法作为参数优化器，算法伪代码如下：



## **实验**

### 数据集

本次实验，我们将使用Cora数据集。

该数据集共2708个样本点，每个样本点都是一篇科学论文，所有样本点被分为7个类别，类别分别是1）基于案例；2）遗传算法；3）神经网络；4）概率方法；5）强化学习；6）规则学习；7）理论

每篇论文都由一个1433维的词向量表示，所以，每个样本点具有1433个特征。词向量的每个元素都对应一个词，且该元素只有0或1两种取值。取0表示该元素对应的词不在论文中，取1表示在论文中。所有的词来源于一个具有1433个词的字典。

每篇论文都至少引用了一篇其他论文，或者被其他论文引用，也就是样本点之间存在联系，没有任何一个样本点与其他样本点完全没联系。如果将样本点看作图中的点，则这是一个连通的图，不存在孤立点。

数据集主要文件有两个：cora.cites, cora.content。其中，cora.content包含了2708个样本的具体信息，每行代表一个论文样本，格式为

<论文id> <由01组成的1433维特征> <论文类别（label）>

28456 0 0 1 1 … 0 0 Theory

114514 1 0 1 0 … 0 0 Neural\_Networks

Cora.cites则包含了论文之间的引用关系（依赖关系），共5429行，每一行有两个论文编号，第一个编号为当前论文的编号，第二个编号为引用了当前论文的论文编号。

35 1033

35 103482  
35 103515

总的来说，如果将论文当作“图”的节点，则引用关系则为“图”的边，论文节点信息和引用关系共同构成了图数据。本次实验，我们将利用这些信息，对论文所属的类别进行预测，完成关于论文类别的分类任务。

### 实现

1.GCN层定义

1. class GraphConvolution(Module):
2. """
3. Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
4. """
5. def \_\_init\_\_(self, in\_features, out\_features, bias=True):
6. super(GraphConvolution, self).\_\_init\_\_()
7. *# 请补全代码*
8. *#输入序列*
9. self.in\_features=in\_features
10. *#输出序列*
11. self.out\_features=out\_features
12. *#初始化权重矩阵*
13. self.weight=Parameter(torch.FloatTensor(in\_features,out\_features))
14. *#判断是否需要偏置量*
15. if bias:
16. self.bias=Parameter(torch.FloatTensor(out\_features))
17. else:
18. self.reset\_parameters('bias',None)
19. self.reset\_parameters()
20. *#参数初始化*
21. def reset\_parameters(self):
22. stdv = 1. / math.sqrt(self.weight.size(1))
23. self.weight.data.uniform\_(-stdv, stdv)
24. if self.bias is not None:
25. self.bias.data.uniform\_(-stdv, stdv)
26. *#模型预测*
27. def forward(self, input, adj):
28. support=torch.mm(input,self.weight)*#矩阵乘法XW*
29. output=torch.spmm(adj,support)*#矩阵乘法AXW*
30. if self.bias is not None:
31. return output+self.bias*#输出加上偏置量*
32. else:
33. return output
34. *# 请补全代码*
35. def \_\_repr\_\_(self):
36. return self.\_\_class\_\_.\_\_name\_\_ + ' (' \
37. + str(self.in\_features) + ' -> ' \
38. + str(self.out\_features) + ')'

2.GCN两层模型定义

1. class GCN(nn.Module):
2. def \_\_init\_\_(self, nfeat, nhid, nclass, dropout):
3. super(GCN, self).\_\_init\_\_()
4. *#第一层GCN*
5. self.gc1 = GraphConvolution(nfeat, nhid)
6. *#第二层GCN*
7. self.gc2 = GraphConvolution(nhid, nclass)
8. *# 防止或拟合*
9. self.dropout = dropout
10. *#函数预测*
11. def forward(self, x, adj):
12. *#第一层预测*
13. x=F.relu(self.gc1(x,adj))
14. *#减少过拟合*
15. x=F.dropout(x,self.dropout,training=self.training)
16. *#第二层预测*
17. x=self.gc2(x,adj)
18. *#最后softmax*
19. return F.log\_softmax(x,dim=1)

3.数据导入以及预处理以及后处理

3.1数据导入

1. def load\_data(path="../data/cora/", dataset="cora"):
2. """Load citation network dataset (cora only for now)"""
3. print('Loading {} dataset...'.format(dataset))
4. #获取数据特征与标签
5. idx\_features\_labels = np.genfromtxt("{}{}.content".format(path, dataset),
6. dtype=np.dtype(str))
7. *#获取数据特征*
8. features = sp.csr\_matrix(idx\_features\_labels[:, 1:-1], dtype=np.float32)#给特征标上坐标
9. *#获取数据标签*
10. labels = encode\_onehot(idx\_features\_labels[:, -1])
11. # build graph
12. idx = np.array(idx\_features\_labels[:, 0], dtype=np.int32)
13. *#获取邻接矩阵*
14. idx\_map = {j: i for i, j in enumerate(idx)}
15. edges\_unordered = np.genfromtxt("{}{}.cites".format(path, dataset),
16. dtype=np.int32)
17. *#获取林街边*
18. edges = np.array(list(map(idx\_map.get, edges\_unordered.flatten())),
19. dtype=np.int32).reshape(edges\_unordered.shape)
20. *#获取tensor邻接矩阵*
21. adj = sp.coo\_matrix((np.ones(edges.shape[0]), (edges[:, 0], edges[:, 1])),
22. shape=(labels.shape[0], labels.shape[0]),
23. dtype=np.float32)
24. # build symmetric adjacency matrix 建立对称邻接矩阵
25. adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
26. features = normalize(features)#输入特征变形
27. adj = normalize(adj + sp.eye(adj.shape[0]))#sp.eye 邻接矩阵变形
28. #划分训练集，验证集，测试集
29. idx\_train = range(140)
30. idx\_val = range(200, 500)
31. idx\_test = range(500, 1500)
32. #数据特征、数据标签、邻接矩阵
33. features = torch.FloatTensor(np.array(features.todense()))
34. labels = torch.LongTensor(np.where(labels)[1])
35. adj = sparse\_mx\_to\_torch\_sparse\_tensor(adj)
36. #坐标划分
37. idx\_train = torch.LongTensor(idx\_train)
38. idx\_val = torch.LongTensor(idx\_val)
39. idx\_test = torch.LongTensor(idx\_test)
40. return adj, features, labels, idx\_train, idx\_val, idx\_test

3.2从数据集中获取数据标签

1. def encode\_onehot(labels):
2. classes = set(labels)
3. classes\_dict = {c: np.identity(len(classes))[i, :] for i, c in
4. enumerate(classes)}
5. labels\_onehot = np.array(list(map(classes\_dict.get, labels)),
6. dtype=np.int32)
7. return labels\_onehot

3.3矩阵数据归一化

1. def normalize(mx):*#归一化*
2. """Row-normalize sparse matrix"""
3. rowsum = np.array(mx.sum(1))
4. r\_inv = np.power(rowsum, -1).flatten()
5. r\_inv[np.isinf(r\_inv)] = 0.
6. r\_mat\_inv = sp.diags(r\_inv)*#有个处理矩阵的函数diags函数，diagonal是对矩阵进行对角化的意思*
7. mx = r\_mat\_inv.dot(mx)
8. return mx

3.4预测准确率计算函数

1. def accuracy(output, labels):
2. preds = output.max(1)[1].type\_as(labels)
3. correct = preds.eq(labels).double()
4. correct = correct.sum()
5. return correct / len(labels)

3.5scipy稀疏矩阵转换为torch稀疏张量

1. def sparse\_mx\_to\_torch\_sparse\_tensor(sparse\_mx):
2. """Convert a scipy sparse matrix to a torch sparse tensor. 将scipy稀疏矩阵转换为torch稀疏张量"""
3. sparse\_mx = sparse\_mx.tocoo().astype(np.float32)
4. indices = torch.from\_numpy(
5. np.vstack((sparse\_mx.row, sparse\_mx.col)).astype(np.int64))
6. values = torch.from\_numpy(sparse\_mx.data)
7. shape = torch.Size(sparse\_mx.shape)
8. return torch.sparse.FloatTensor(indices, values, shape)

4.模型训练

4.1训练参数初始化设置

1. parser = argparse.ArgumentParser()
2. parser.add\_argument('*--no-cuda', action='store\_true', default=False,*
3. help='Disables CUDA training.')
4. parser.add\_argument('--fastmode', action='store\_true', default=False,
5. help='Validate during training pass.')
6. parser.add\_argument('--seed', type=int, default=42, help='Random seed.')*#随机状态*
7. parser.add\_argument('--epochs', type=int, default=200,
8. help='Number of epochs to train.')*#训练论数*
9. parser.add\_argument('--lr', type=float, default=0.01,
10. help='Initial learning rate.')*#学习率*
11. parser.add\_argument('--weight\_decay', type=float, default=5e-4,
12. help='Weight decay (L2 loss on parameters).')*#初始化权重*
13. parser.add\_argument('--hidden', type=int, default=16,
14. help='Number of hidden units.')*#隐藏层层数*
15. parser.add\_argument('--dropout', type=float, default=0.5,
16. help='Dropout rate (1 - keep probability).')*#定义dropout函数*
17. args = parser.parse\_args()

4.2训练参数赋值以及数据导入

1. args = parser.parse\_args()
2. args.cuda = not args.no\_cuda and torch.cuda.is\_available()#判断是cpu还是gpu
3. np.random.seed(args.seed)
4. torch.manual\_seed(args.seed)
5. if args.cuda:
6. torch.cuda.manual\_seed(args.seed)
7. # Load data 加载数据
8. adj, features, labels, idx\_train, idx\_val, idx\_test = load\_data()
9. #设定数据对应设备
10. if args.cuda:
11. model.cuda()
12. features = features.cuda()
13. adj = adj.cuda()
14. labels = labels.cuda()
15. idx\_train = idx\_train.cuda()
16. idx\_val = idx\_val.cuda()
17. idx\_test = idx\_test.cuda()

4.3定义模型以及参数优化器

1. *#初始化模型*
2. model = GCN(nfeat=features.shape[1],
3. nhid=args.hidden,
4. nclass=labels.max().item() + 1,
5. dropout=args.dropout)
6. *#定义Adam参数优化算法*
7. optimizer = optim.Adam(model.parameters(),
8. lr=args.lr, weight\_decay=args.weight\_decay)

4.4初始化损失向量

1. *#绘图数据存储*
2. loss\_train\_his=[]
3. loss\_valid\_his=[]
4. acc\_train\_his=[]
5. acc\_valid\_his=[]

4.5模型训练

1. def train(epoch):
2. #定义初始时间
3. t=time.time()
4. #将模型设置为训练模式
5. model.train()
6. #梯度归0
7. optimizer.zero\_grad()
8. #模型预测
9. output=model(features,adj)
10. #使用NLLLOSS作为损失函数
11. loss\_train=F.nll\_loss(output[idx\_train],labels[idx\_train])
12. #预测准确率
13. acc\_train=accuracy(output[idx\_train],labels[idx\_train])
14. #梯度归0
15. loss\_train.backward()
16. #参数优化
17. optimizer.step()
18. #判断模型是否需要验证集
19. if not args.fastmode:
20. model.eval()
21. output=model(features,adj)
22. #验证集损失
23. loss\_val=F.nll\_loss(output[idx\_val],labels[idx\_val])
24. #验证集准确率
25. acc\_val=accuracy(output[idx\_val],labels[idx\_val])
26. print('Epoch: {:04d}'.format(epoch+1),
27. 'loss\_train: {:.4f}'.format(loss\_train.item()),
28. 'acc\_train: {:.4f}'.format(acc\_train.item()),
29. 'loss\_val: {:.4f}'.format(loss\_val.item()),
30. 'acc\_val: {:.4f}'.format(acc\_val.item()),
31. 'time: {:.4f}'.format(time.time()-t))
32. loss\_train\_his.append(loss\_train.item())
33. loss\_valid\_his.append(loss\_val.item())
34. acc\_train\_his.append(acc\_train)
35. acc\_valid\_his.append(acc\_val)

4.6模型预测

1. def test():
2. *#将模型设置为测试模式*
3. model.eval()
4. *#模型预测*
5. output = model(features, adj)
6. *#计算损失*
7. loss\_test = F.nll\_loss(output[idx\_test], labels[idx\_test])
8. *#计算准确率*
9. acc\_test = accuracy(output[idx\_test], labels[idx\_test])
10. print("Test set results:",
11. "loss= {:.4f}".format(loss\_test.item()),
12. "accuracy= {:.4f}".format(acc\_test.item()))

4.7训练启动

1. # Train model
2. t\_total = time.time()#记录初试时间
3. for epoch in range(args.epochs):
4. train(epoch)
5. print("Optimization Finished!")
6. print("Total time elapsed: {:.4f}s".format(time.time() - t\_total))
7. # Testing
8. test()

4.8训练结果绘图

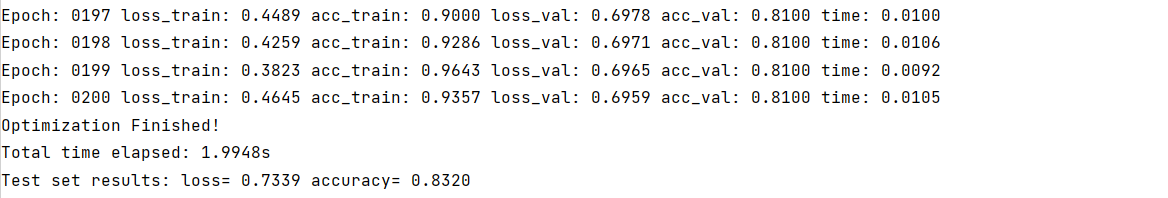
1. *#绘图loss*
2. iteration = np.arange(1, args.epochs+1, step = 1)
3. fig, ax = plt.subplots(figsize = (6,4))
4. ax.set\_title('Train—Valid')
5. ax.set\_xlabel('iteration')
6. ax.set\_ylabel('loss')
7. plt.plot(iteration, loss\_train\_his, 'b', label='Train')
8. plt.plot(iteration, loss\_valid\_his, 'r', label='Valid')
9. plt.legend()
10. plt.show(block=True)
11. *#绘图acc*
12. iteration = np.arange(1, args.epochs+1, step = 1)
13. fig, ax = plt.subplots(figsize = (6,4))
14. ax.set\_title('Train—Valid')
15. ax.set\_xlabel('iteration')
16. ax.set\_ylabel('accuracy')
17. plt.plot(iteration, acc\_train\_his, 'b', label='Train')
18. plt.plot(iteration, acc\_valid\_his, 'r', label='Valid')
19. plt.legend()
20. plt.show(block=True)

5.模型预测结果

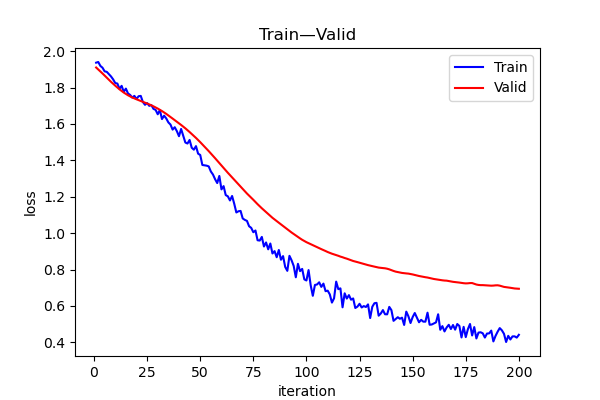
TABLE I

**模型拟合结果**

|  |  |
| --- | --- |
| **总用时长** | **1.9948s** |
| **训练误差** | **0.7379** |
| **训练准确率** | **0.8320** |



（Figure. 2.模型预测结果）



（Figure. 3.模型训练与验证的损失随迭代次数变化图）



（Figure. 3.模型训练与验证的准确率随迭代次数变化图）

6实验拓展

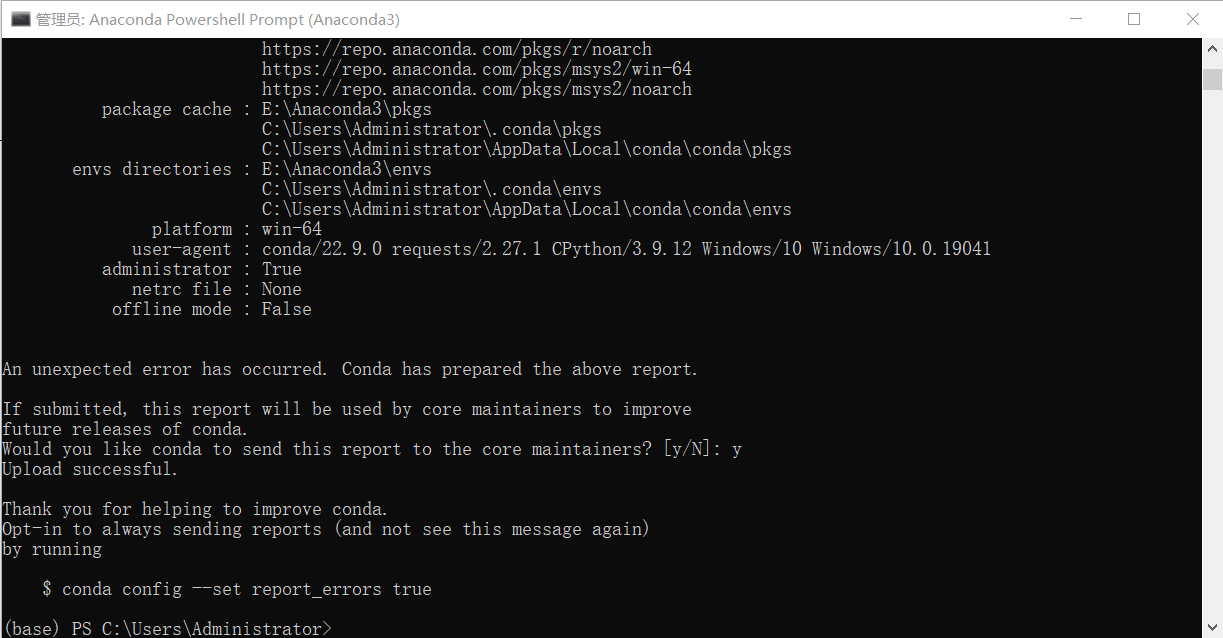
6.1实验介绍

神经架构搜索（Neural Architecture Search ，NAS）能帮助我们搜寻合适的网络去完成特定任务，甚至搜索出来的网络模型，其不输于目前SOTA的方法。而我们人工设计的模型，可能需要耗费大量精力的同时，效果也不一定很好。那我们能否利用架构搜索算法，寻找出更强大的GCN呢（比如改变GCN的信息聚合方式、改变激活函数，或者更改超参数的设置等等）

在拓展实验中，我们将运行一个针对GNN的AutoML框架[2]，寻找合适的GCN结构，查看搜索出来的网络是否比人工设计的GCN更好呢？

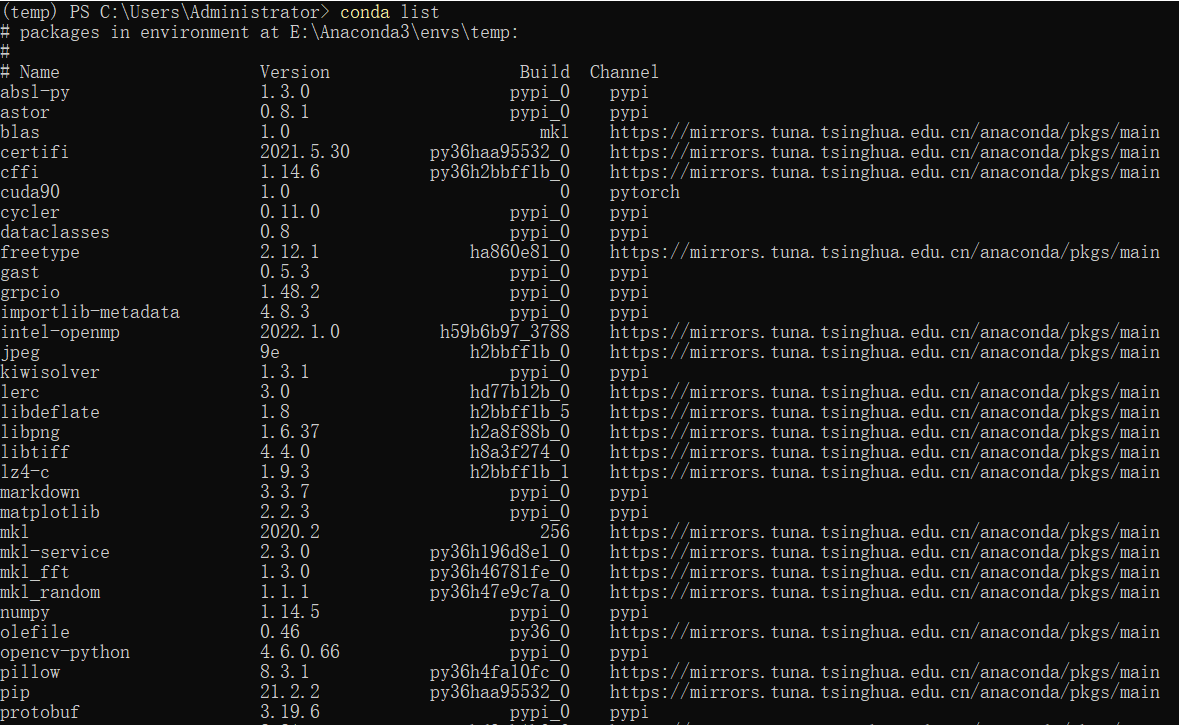
6.2实验部署

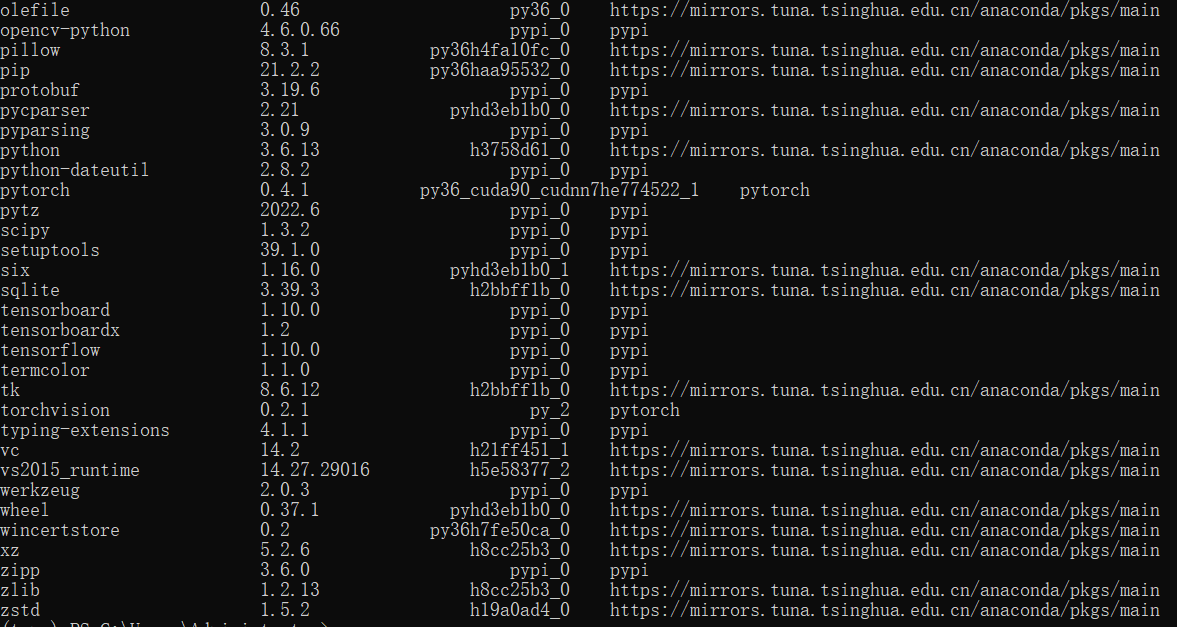
1. conda create –n temp python=3.7



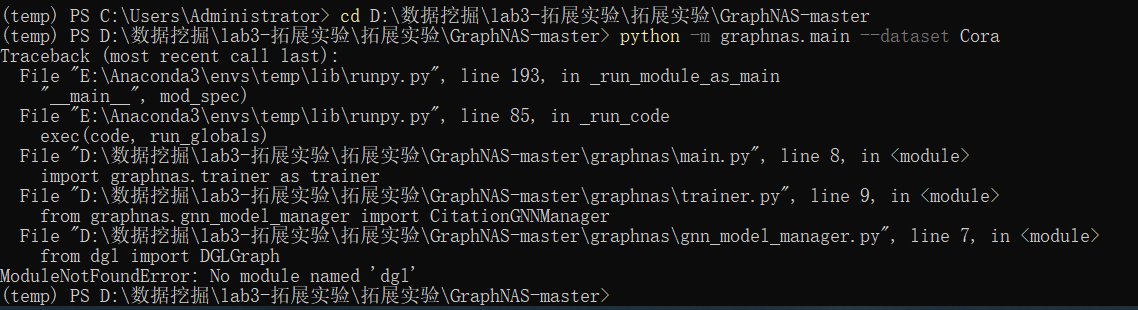
1. conda install pytorch==1.7.1 torchvision==0.8.2 torchaudio==0.7.2 cudatoolkit=11.0 -c pytorch
2. 进入拓展实验安装包目录，对三个文件分别进行 pip install xxx.whl操作
3. pip install torch-geometric==1.4.3
4. conda install -c dglteam dgl-cuda11.0==0.7.1

最后可得到所有安装环境的列表





1. 重新进入GraphNAS文件夹，执行以下命令进行网络搜索python -m graphnas.main --dataset Cora



由于dgl的版本的不适配，导致实验失败了，但是在网上找的各种dgl安装适配方法都没有效果，因此在本次拓展实验中只能以失败告终了。

## **结论**

由实验结果可知，使用GCN模型对数据进行训练时，随着迭代次数增加，模型的预测损失整体越来越小，预测准确率整体越来越高，但训练集的损失以及准确率的波动较大，而验证集的训练损失与准确率波动较小，整体而言GCN模型在论文分类中的应用是较为合适的。

在本次实验中，主要是实现了基于GCN的分类算法。

在GCN分类算法中，其主要步骤是先导入数据，提取数据特征和数据标签，但在GCN中，因为其主要面向互相具有依赖关系的数据集，因此其还需提取一个依赖矩阵，也就是邻接矩阵，保存着数据之间的依赖关系，然后经过一系列转化处理以及归一化之后，划分训练集与验证集和测试集，最终确定模型所需要的数据，然后定义GCN模型，在本次试验中GCN主要由两层构成，第一层输出经过Relu之后作为第二层的输入，然后定义NLLLOSS损失函数，然后使用Adam优化算法进行训练，进行参数优化，最后训练结果输出就完成了。

对于本次实验而言，主要有代码参考，与LSTM相比，GCN模型的公式较为容易理解，也较为简单，而GCN稍难的一点是需要提取依赖邻接矩阵。虽然在公式上比较好理解，但更多是了解如何去进行代码实现，这与在课本上仅学习到的公式是相差很大的。此外还学习到了许多机器学习的库，让自己对这些库中的函数有了更进一步的了解与运用。总体而言本次实验的收获是挺大的。