2022 秋-深度学习-第四次课程实验-实验报告

李钊佚 SA22011035

报告目录：

快速复现代码：请参考 github 仓库 README.md：
https://github.com/Joeylee-rio/Courseworks-for-graduate-at-USTC/tree/master/deeplearn_lab4_gcn

报告正文：

一、实验简介

使用 Pytorch 或者 TensorFlow 的相关神经网络库，编写一个图卷积神经网络模型（Graph, Convolutional Network, GCN），并在相应的图结构数据集上完成节点分类(Node Classification)和链路预测(Link Prediction)任务，分析自环(Self—Loop)、层数(Layer_Num)、DropEdge、PiarNorm、激活函数(Activation)等因素对模型的分类和预测性能的影响。

二、Cora，Citeseer，PPI 数据集下载，预处理，数据结构处理

- 数据集下载：

```
Cora dataset : https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz \\
Citeseer dataset :
https://linqs-data.soe.ucsc.edu/public/lbc/citeseer.tgz\\
PPI dataset :  http://snap.stanford.edu/graphsage/ppi.zip \\
```

- 数据集解压：

```
mkdir datasets
# download 3 datasets into datasets dictionary
unzip ppi.zip
tar zxvf cora.tgz
tar zxvf citeseer.tgz
# for citeseer dataset, we need to preprocess it
python $home/deeplearn_lab4_gcn/src/preprocess_citeseer.py
# use the `.new` generated files to replace the original ones.
```

- 本次实验采用：python=3.8，pytorch=11.3，cuda=11.7，相应版本的 torch—sparse，torch—geometric 包（来处理使用 pair_norm 函数）
- 数据处理（数据结构）：
  - Cora 和 Citeseer 数据集：

    从所给数据集中提取节点特征（features）和标签（labels），链接（links）。

    使用 scipy.sparse 的 csr_matrix 来存储 features，对于 labels（字符串）采用 one—hot 编码，将节点需要重映射为 0,1,2,3,…(自然数序列)，对所有链接的 source 和 node 节点进行重新映射（自然数序列），建立 edges 数组（采用 numpy.array 存储）。

```
    idx_features_labels = np.genfromtxt("{}{}.content".format(path,
dataset),
                                        dtype=np.dtype(str))
    np.random.shuffle(idx_features_labels)
    features = sp.csr_matrix(idx_features_labels[:, 1:-1],
dtype=np.float32)
    labels = encode_onehot(idx_features_labels[:, -1])
    # build graph
    idx = np.array(idx_features_labels[:, 0], dtype=np.int32)
    idx_map = {j: i for i, j in enumerate(idx)}
    edges_unordered = np.genfromtxt("{}{}.cites".format(path, dataset),
                                    dtype=np.int32)

    temp1 = map(idx_map.get, edges_unordered.flatten())
```

```
        temp2 = list(temp1)
        x = list(edges_unordered.flatten())
        print(x[462])
        for i in range(len(temp2)):
            elem = temp2[i]
            try:
                elem = int(elem)
            except TypeError:
                print(i)

        edges = np.array(temp2, dtype=np.int32).reshape(edges_unordered.shape)
```

➢ 对于节点分类任务：

```
    elif task == 'nodecls':
```

采用 CSR 存储稀疏矩阵：

```
        adj = sp.coo_matrix((np.ones(edges.shape[0]), (edges[:, 0], edges[:,
1])),
                            shape=(labels.shape[0], labels.shape[0]),
                            dtype=np.float32)
```

建立对称矩阵：

```
        # build symmetric adjacency matrix
        adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
```

对节点特征归一化：

```
        features = normalize(features)
```

是否添加自环：

```
        if self_loop == True:
            adj = normalize(adj + sp.eye(adj.shape[0]))
        else:
            adj = normalize(adj)
```

划分 train || validation || set 节点集合后：

```
        features = torch.FloatTensor(np.array(features.todense()))
        labels = torch.LongTensor(np.where(labels)[1])
        adj = sparse_mx_to_torch_sparse_tensor(adj)

        idx_train = torch.LongTensor(idx_train)
        idx_val = torch.LongTensor(idx_val)
        idx_test = torch.LongTensor(idx_test)

        return adj, features, labels, idx_train, idx_val, idx_test
```

完成数据的处理部分。

➢ 对于链接预测任务：

```
    if task == 'linkpred':
```

划分 edge 的 train || validation || test 部分：

```
        edge_num = edges.shape[0]
```

```python
    shuffled_ids = np.random.permutation(edge_num)
    test_set_size = int(edge_num * 0.15)
    val_set_size = int(edge_num * 0.15)
    test_ids = shuffled_ids[ : test_set_size]
    val_ids = shuffled_ids[test_set_size : test_set_size + val_set_size]
    train_ids = shuffled_ids[test_set_size + val_set_size : ]

    train_pos_edges = torch.tensor(edges[train_ids], dtype=int)
    val_pos_edges = torch.tensor(edges[val_ids], dtype=int)
    test_pos_edges = torch.tensor(edges[test_ids], dtype=int)

    train_pos_edges = torch.transpose(train_pos_edges, 1, 0)
    # shape = [2, train_pos_edge_num]
    val_pos_edges = torch.transpose(val_pos_edges, 1, 0)
    test_pos_edges = torch.transpose(test_pos_edges, 1, 0)
```

由于这里我们给出的 edges 均为正样本，为了优化目标函数（BCEloss），我们应该采样相同数量的负样本 edges：

```python
    def negative_sample(pos_edges, nodes_num):
        '''
        pos_edges = [[src_1,...],
                     [dst_1,...]]
        '''
        neg_edges = negative_sampling(
            edge_index=pos_edges,
            num_nodes=nodes_num,
            num_neg_samples=pos_edges.shape[1],
            method='sparse'
        )
        edges = torch.cat((pos_edges, neg_edges), dim=-1)
        '''
        edges = [[src_1,src_2,...,src_m],
                 [dst_1,dst_2,...,dst_m]]
        shape = [2, 2*train_edge_num]
        '''
        edges_label = torch.cat((
            torch.ones(pos_edges.shape[1]),
            torch.zeros(neg_edges.shape[1])
        ),dim=0)
        # size = [2*train_edge_num]
        return edges, edges_label

    train_edges, train_label = negative_sample(train_pos_edges,
idx.shape[0])
    val_edges, val_label = negative_sample(val_pos_edges, idx.shape[0])
```

```
    test_edges, test_label = negative_sample(test_pos_edges, idx.shape[0])
```

使用 CSR 算法建立稀疏邻接矩阵：

```
    adj = sp.coo_matrix((np.ones(train_pos_edges.shape[1]),
(train_pos_edges[0], train_pos_edges[1])),
                    shape=(idx.shape[0], idx.shape[0]),
                    dtype=np.float32)
```

建立对称矩阵：

```
    # build symmetric adjacency matrix
    adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
```

对节点特征归一化：

```
    features = normalize(features)
```

是否添加自环（self_loop）：

```
    if self_loop == True:
        adj = normalize(adj + sp.eye(adj.shape[0]))
    else:
        adj = normalize(adj)
```

针对链接预测的数据处理完成，返回相应数据（节点特征，train || validation || test edge 集合，对称邻接矩阵等）。

- PPI 数据集 （与 Cora 和 Citeseer 的文件组织结构不同），我们提供提取特征，label，links，nodes 的代码，（之后根据任务类型建立邻接矩阵等的代码与上述数据集一致，不再额外提供）

valid_feats.npy文件保存节点的特征，shape为(56944, 50)(节点数目，特征维度)，值为0或1，且1的数目稀少

ppi-class_map.json为节点的label文件，shape为(121, 56944),每个节点的label为121维

ppi-G.json文件为节点和链接的描述信息，节点：{"test": true, "id": 56708, "val": false}, 表示节点id为56708的节点是否为test集或者val集，链接："links": [{"source": 0, "target": 372}, {"source": 0, "target": 1101}, 表示节点id为0的节点和为1101的节点之间有links,

ppi-walks.txt文件中为链接信息

296	1	1115
297	1	161
298	1	1394
299	1	1095
300	1	850
301	1	826
302	1	844
303	1	1547

ppi-id_map.json文件为节点id信息

其中，training，validation，testing 的 node 节点已经划分完毕（但是当运行链接预测任务时我们仍然自行划分 train || val || test 的 edge），我们处理数据的代码如下

```
    path =
"/data2/home/zhaoyi/labs/USTC-labs/deeplearn_lab4_gcn/datasets/ppi/"
    #print('Loading PPI dataset...')
    feature_file = path + "ppi-feats.npy"
    label_file = path + "ppi-class_map.json"
    edge_file = path + "ppi-walks.txt"
```

```
    graph_file = path + "ppi-G.json"

    #print('Uploading features ...')
    features = np.load(feature_file) # shape = (56944, 50)
    features = sp.csr_matrix(features, dtype=np.float32)
    #print('Uploading labels...')
    fr_label = open(label_file, "r")
    label_dict = json.load(fr_label)
    proc_label_dict = dict()
    for key in label_dict:
        proc_label_dict[int(key)] = list(label_dict[key])
    _labels = sorted(proc_label_dict.items(), key=lambda d: d[0])
    labels = list()
    for item in _labels:
        _, x = item
        labels.append(x)
    labels = np.array(labels, dtype=np.int32)
    print('Uploading graph...')
    fr_graph = open(graph_file, "r")
    graph_dict = json.load(fr_graph)
    nodes = graph_dict["nodes"]
    links = graph_dict["links"]
    #print('Generating edges')
    edges = [[links[i]["source"], links[i]["target"]] for i in range(len(links))]
    edges = np.array(edges, dtype=np.int32)
    #print('Generating nodes')
    idx = list()
    idx_train = list()
    idx_val = list()
    idx_test = list()
    for i in range(len(nodes)):
        idx.append(nodes[i]["id"])
        if nodes[i]["test"] == True:
            idx_test.append(nodes[i]["id"])
        elif nodes[i]["val"] == True:
            idx_val.append(nodes[i]["id"])
        else:
            idx_train.append(nodes[i]["id"])
    idx = np.array(idx, dtype=np.int32)
```

三、基于 PyTorch 手作实现 GCN

- 图卷积层的实现

  GCN 层的核心更新如下（通过前乘邻接矩阵和归一化矩阵实现节点信息的聚合）

  (semi—supervised classification with graph convolutional networks，ICLR'2017)

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right).$$

```
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features,
out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' (' \
               + str(self.in_features) + ' -> ' \
               + str(self.out_features) + ')'
```

- 图卷积神经网络的实现

基于前文图卷积层的实现，GCN 的搭建如下：

```
import torch
```

```python
import torch.nn as nn
import torch.nn.functional as F
from layers import GraphConvolution
from torch_geometric.nn import PairNorm

class GCN(nn.Module):
    def __init__(self, in_channels, hid_channels, out_channels, dropout,
                 layer_num=2, activation='relu', drop_edge=False, pair_norm=False):
        super(GCN, self).__init__()

        self.gc_inp = GraphConvolution(in_channels, hid_channels)
        self.gc_hids = nn.ModuleList([GraphConvolution(hid_channels, hid_channels) for _ in range(layer_num-2)])
        self.gc_out = GraphConvolution(hid_channels, out_channels)
        if activation == 'relu':
            self.activate = F.relu
        elif activation == 'sigmoid':
            self.activate = torch.sigmoid
        elif activation == 'tanh':
            self.activate =  torch.tanh


        self.pair_norm = pair_norm
        if pair_norm == True:
            self.norm = PairNorm()


        self.dropout = nn.Dropout(dropout)

        # for ppi dataset
        self.linear_out = nn.Linear(out_channels, 121)

    def forward(self, x, adj, task='nodecls', edges=None, ppi=False):
        x = self.gc_inp(x, adj)
        x = self.activate(x)

        for gc_layer in self.gc_hids:
            x = self.dropout(x)
            x = gc_layer(x, adj)

            if self.pair_norm:
                x = self.norm(x)

```

```
            x = self.activate(x)

        x = self.dropout(x)
        x = self.gc_out(x, adj)
```

- 节点分类任务：

```
        if task == 'nodecls':
            if ppi == False:
                # x.shape = [node_num, label_class_num]
                return F.log_softmax(x, dim=1)
            else:
                x = self.linear_out(x)
                # x.shape = [node_num, label_dim]
                return x
```

对于 Cora，Citeseer 数据集，由于 label 是 one-hot 形式，GCN 只需输出每个节点属于各个 label 的概率值，通过计算交叉熵即可计算损失函数，进而反向传播优化网络参数，我们采用 Acc 评价指标。

对于 PPI 数据集，由于 label 不是 one-hot 形式（多标签分类），我们需要让 GCN 针对每个节点输出一个多维向量（label_num），我们采用 BCEloss 来计算损失，进而反向传播更新参数，我们采用 F1 评价指标。

- 链路预测任务：

```

        elif task == 'linkpred':
            # x.shape = [node_num, hid_channels]
            assert edges != None
            src = x[edges[0]] # shape = [src_num, hid_channels]
            dst = x[edges[1]] # shape = [node_num, hid_channels]
            inner_prods = (src * dst).sum(dim=-1) # shape =[src_num]
            return inner_prods
```

对于链路预测任务，对三个数据集，我们都是输出最终需要预测链路的 src 节点和 dst 节点的特征的内积，采用 BCE loss 来计算 loss 函数，反向传播计算梯度更新参数，我们采用 AUC 指标评价。

四、实验结果和参数分析

- 全局结果（搜索参数空间之后最好的结果）：

Datasets	Node Classification (AUC)	Link Prediction (Acc, F1)
Cora	80.70%	80.39% (Acc)
Citeseer	64.70%	80.57% (Acc)
PPI	45.56%	91.39% (F1)

Best performances on Cora, Citeseer and PPI datasets with our implementations.

其中，对应的超参数为：

Hyper-parameters : (self_loop, pair_norm, drop_edge, layer_num, activate, hidden)
dataset="Cora", task="Node Classification", (True, True, True, 2, relu\tanh, 16)
dataset="Cora", task="Link Prediction", (True, True, True, 4, tanh, 16)
dataset="Citeseer", task="Node Classification", (True, True, True, 2, relu, 16)
dataset="Citeseer", task="Link Prediction", (True, True, True, 2, relu, 16)
dataset="PPI", task="Node Classification", (True, True, True, 4, relu, 256)
dataset="PPI", task="Link Prediction", (True, True, True, 4, relu, 256)

下面我们详细的分析各个参数的影响：

- 自环的影响
- PairNorm 的影响
- EdgeDrop 的影响

Self-Loop	Cora-NC	Cora-LP	Citeseer-NC	Citeseer-LP
True	80.70% (AUC)	80.39% (Acc)	64.70%(AUC)	80.57% (Acc)
False	76.10% (AUC)	73.41% (Acc)	45.90%(AUC)	70.08% (Acc)

Impact of `self-loop`.

Pair-Norm	PPI-NC	PPI-LP	Cora-NC	Cora-LP
True	45.56%(AUC)	91.39% (F1)	80.70% (AUC)	80.39% (Acc)
False	43.45%(AUC)	58.74% (F1)	80.40% (AUC)	54.52% (Acc)

Impact of `pair-norm`.

DropEdge	Cora-NC	Cora-LP	Citeseer-NC	Citeseer-LP	PPI-NC	PPI-LP
True	80.70% (AUC)	80.39% (Acc)	64.70%(AUC)	80.57% (Acc)	45.56%(AUC)	91.39% (F1)
False	80.52%(AUC)	80.10% (Acc)	64.47%(AUC)	80.40% (Acc)	45.52%(AUC)	91.22% (F1)

Impact of `drop-edge`.

分析：

- ➢ 自环（self-loop）：添加自环对于不同数据集的不同任务都有较为显著的性能提升。
- ➢ Pair-Norm：添加 Pair-Norm 对于深层网络（PPI->4 层）有更为显著的性能提升，因为 Pair-Norm 可以有效缓解 over-smoothing 现象，同时观察到添加 Pair-Norm 对于 Link-Prediction 任务的影响相较于 Node-Classification 更大。
- ➢ DropEdge：在我的实现中：添加 0.3 的 DropEdge 会 consistenly 的提升三个数据集的两个任务上的模型性能，因为 DropEdge 可以一定程度上缓解 oversmoothing 问题。

- 图卷积层数的影响

  我们在 PPI 数据集上探究了图卷积层数对于模型预测性能的影响：（见下图）

- 激活函数种类的影响

我们在 Citeseer 数据集上探究了激活函数种类对于模型预测性能的影响：（见下图）

Layer Num	PPI-NC	PPI-LP
2	42.10%(AUC)	70.59% (F1)
3	44.59%(AUC)	87.69% (F1)
4	45.56%(AUC)	91.39% (F1)

Impact of `layer-num`.

Acti Func	Citeseer-NC	Citeseer-LP
relu	64.70%(AUC)	80.57% (Acc)
sigmoid	36.10%(AUC)	59.71% (Acc)
tanh	64.20%(AUC)	62.21% (Acc)

Impact of `activation function`.

- 图卷积层数 Layer Num：对于比较复杂（较大规模图）的数据集（e.g.，PPI）增加图卷积层数可以有效提升模型预测性能，但是在较为简单的数据集（Cora，Citeseer）上，增加图卷积层数并不会对模型预测性能有明显提升。
- 激活函数 Activation Function：明显可以得到的是：sigmoid 函数作为激活函数时，模型的预测性能会收到较大影响，往往 relu 函数作为激活函数时，模型的预测性能比较好且稳定。

五、参考资料


```
@1: Mu Li's tutorial:
https://www.bilibili.com/video/BV1iT4y1d7zP/?spm_id_from=333.337.search-card.all.click&vd_source=0b94491685a644f4e70b2ffc09079337
@2: Google Research's distill blog:
https://distill.pub/2021/gnn-intro/
@3: pygcn tutorial:
https://www.bilibili.com/video/BV1Y64y1B7Qc/?spm_id_from=333.337.search-card.all.click&vd_source=0b94491685a644f4e70b2ffc09079337
@4: pygcn github (official implementation of GCN in pytorch):
https://github.com/tkipf/pygcn
@5: GCN original paper: (Semi-Supervised Classification with Graph Convolutional Networks, ICLR'17, Thomas N.Kipf, Max Welling)
https://arxiv.org/abs/1609.02907
@6: a blog around GCN:
https://ai.plainenglish.io/graph-convolutional-networks-gcn-baf337d5cb6b?gi=a61c544a76c5
@7: how to use GCN to deal with link prediction task?
https://blog.csdn.net/Cyril_KI/article/details/125956540
@8: EdgeDrop paper: https://arxiv.org/abs/1907.10903
@9: PairNorm paper: https://arxiv.org/abs/1909.12223
@10: an example of processing PPI dataset:
https://blog.csdn.net/KPer_Yang/article/details/128810698?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-0-128810698-blog-112979175.pc_relevant_multi_platform_whitelistv3&spm=1001.2101.3001.4242.1&utm_relevant_index=3
```

附录 关键代码

Train.py

```
from __future__ import division
from __future__ import print_function

import time
import argparse
import numpy as np

import torch
import torch.nn.functional as F
import torch.optim as optim
from sklearn.metrics import roc_auc_score, f1_score

from utils import load_data, accuracy, load_ppi_data
from models import GCN

# Training settings
parser = argparse.ArgumentParser()
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='Disables CUDA training.')
parser.add_argument('--fastmode', action='store_true', default=False,
                    help='Validate during training pass.')
parser.add_argument('--seed', type=int, default=42, help='Random seed.')
parser.add_argument('--epochs', type=int, default=200,
                    help='Number of epochs to train.')
parser.add_argument('--lr', type=float, default=0.01,
                    help='Initial learning rate.')
parser.add_argument('--weight_decay', type=float, default=5e-4,
                    help='Weight decay (L2 loss on parameters).')
parser.add_argument('--hidden', type=int, default=16,
                    help='Number of hidden units.')
parser.add_argument('--dropout', type=float, default=0.5,
                    help='Dropout rate (1 - keep probability).')
'''
self-loop, pairnorm. dropedge, layer_num, activate
'''
parser.add_argument('--drop_edge', type=float, default=0.,
                    help='DropEdge rate (1 - keep probability).')
parser.add_argument('--pair_norm', type=bool, default=False,
                    help='Wether to use PairNorm or not')
parser.add_argument('--self_loop', type=bool, default=False,
                    help='Whether to use Self-Loop or not')
```

```
parser.add_argument('--layer_num', type=int, default=2,
                    help='How many GC-layers are going to be used')
parser.add_argument('--activate', type=str, default='relu',
                    help='Which kind of activation function is going to be used')
parser.add_argument('--dataset', type=str, default='citeseer',
                    help='Select which dataset to conduct experiment')
parser.add_argument('--task', type=str, default='nodecls',
                    help='nodecls(Node classification) or linkpred(Link
Prediction)')


args = parser.parse_args()
print(args.self_loop)
args.cuda = not args.no_cuda and torch.cuda.is_available()

np.random.seed(args.seed)
torch.manual_seed(args.seed)
if args.cuda:
    torch.cuda.manual_seed(args.seed)

# Load data
if args.task == 'nodecls':
    if args.dataset == 'cora' or args.dataset == 'citeseer':
        adj, features, labels, idx_train, idx_val, idx_test = load_data(
                                                    dataset=args.dataset,
                                                    task=args.task,
                                                 self_loop=args.self_loop)
        # Model and optimizer
        model = GCN(in_channels=features.shape[1],
                    hid_channels=args.hidden,
                    out_channels=labels.max().item() + 1,
                    dropout=args.dropout,
                    layer_num=args.layer_num,
                    activation=args.activate,
                    drop_edge=args.drop_edge,
                    pair_norm=args.pair_norm)

    elif args.dataset == 'ppi':
        adj, features, labels, idx_train, idx_val, idx_test = load_ppi_data(
                                                    task=args.task,
                                                    self_loop=args.self_loop)
        # Model and optimizer
        model = GCN(in_channels=features.shape[1],
                    hid_channels=args.hidden,
                    out_channels=args.hidden,
```

```
                dropout=args.dropout,
                layer_num=args.layer_num,
                activation=args.activate,
                drop_edge=args.drop_edge,
                pair_norm=args.pair_norm)

    if args.cuda:
        model.cuda()
        features = features.cuda()
        adj = adj.cuda()
        labels = labels.cuda()
        idx_train = idx_train.cuda()
        idx_val = idx_val.cuda()
        idx_test = idx_test.cuda()

elif args.task == 'linkpred':
    if args.dataset == 'cora' or args.dataset == 'citeseer':
        adj, features, train_edges, val_edges, test_edges, \
                train_label, val_label, test_label = load_data(
                                                dataset=args.dataset,
                                                task=args.task,
                                                self_loop=args.self_loop)
    elif args.dataset == 'ppi':
        adj, features, train_edges, val_edges, test_edges, \
                train_label, val_label, test_label = load_ppi_data(
                                                task=args.task,
                                                self_loop=args.self_loop)
    '''
    train_edges = list [[src_pos_1,...,src_pos_m, src_neg_1,...,src_neg_m],
                  [dst_pos_1,...,dst_pos_m, dst_neg_1,...,dst_neg_m]]
    train_label = torch.tensor([1, 1, 1,...,1, 0, ..., 0], dtype=long)
    '''
    # Model and optimizer
    model = GCN(in_channels=features.shape[1],
            hid_channels=args.hidden,
            out_channels=args.hidden,
            dropout=args.dropout,
            layer_num=args.layer_num,
            activation=args.activate,
            drop_edge=args.drop_edge,
            pair_norm=args.pair_norm)

    if args.cuda:
        model.cuda()
```

```
        features = features.cuda()
        adj = adj.cuda()
        train_label = train_label.cuda()
        val_label = val_label.cuda()
        test_label = test_label.cuda()

else:
    raise Exception('task({}) is supposed to belong to \{"nodecls",
"linkpred"\}.'.format(task))

optimizer = optim.Adam(model.parameters(),
                       lr=args.lr, weight_decay=args.weight_decay)

if args.task == 'nodecls':
    if args.dataset != 'ppi':
        criterion = F.nll_loss
    else:
        criterion = torch.nn.BCEWithLogitsLoss()
elif args.task == 'linkpred':
    criterion = torch.nn.BCEWithLogitsLoss()

val_performances = list()
test_performances = list()

def train(epoch, task='nodecls'):
    t = time.time()
    model.train()
    optimizer.zero_grad()

    if task == 'nodecls':
        if args.dataset != 'ppi':
            output = model(features, adj)
            loss_train = criterion(output[idx_train], labels[idx_train])
        else:
            output = model(x=features, adj=adj, ppi=True)
            loss_train = criterion(output[idx_train], labels[idx_train].float())

        if args.dataset != 'ppi':
            acc_train = accuracy(output[idx_train], labels[idx_train])
        else:
            preds = (output[idx_train] > 0).float().cpu()
            #print(labels[idx_train].shape, preds.shape)
            f1_train = f1_score(labels[idx_train].cpu(), preds, average='micro')
```

```
    elif task == 'linkpred':
        output = model(features, adj, 'linkpred', train_edges)
        loss_train = criterion(output, train_label)
        logits = torch.sigmoid(output)
        auc_train = roc_auc_score(train_label.cpu().numpy(),
logits.detach().cpu().numpy())

    loss_train.backward()
    optimizer.step()


    model.eval()
    if task == 'nodecls':
        if args.dataset != 'ppi':
            output = model(features, adj)
            loss_val = criterion(output[idx_val], labels[idx_val])
        else:
            output = model(x=features, adj=adj, ppi=True)
            loss_val = criterion(output[idx_val], labels[idx_val].float())

        if args.dataset != 'ppi':
            acc_val = accuracy(output[idx_val], labels[idx_val])
        else:
            preds = (output[idx_val] > 0).float().cpu()
            f1_val = f1_score(labels[idx_val].cpu(), preds, average='micro')



        if args.dataset != 'ppi':
            loss_test = criterion(output[idx_test], labels[idx_test])
            acc_test = accuracy(output[idx_test], labels[idx_test])
        else:
            loss_test = criterion(output[idx_test], labels[idx_test].float())
            preds = (output[idx_test] > 0).float().cpu()
            f1_test = f1_score(labels[idx_test].cpu(), preds, average='micro')

        if args.dataset != 'ppi':
            '''
            print('Epoch: {:04d}'.format(epoch+1),
                'loss_train: {:.4f}'.format(loss_train.item()),
                'acc_train: {:.4f}'.format(acc_train.item()),
                'loss_val: {:.4f}'.format(loss_val.item()),
                'acc_val: {:.4f}'.format(acc_val.item()),
                'loss_val: {:.4f}'.format(loss_test.item()),
                'acc_val: {:.4f}'.format(acc_test.item()),
```

```
            'time: {:.4f}s'.format(time.time() - t))
        '''
        val_performances.append(acc_val.item())
        test_performances.append(acc_test.item())
    else:
        '''
        print('Epoch: {:04d}'.format(epoch+1),
            'loss_train: {:.4f}'.format(loss_train.item()),
            'f1_train: {:.4f}'.format(f1_train),
            'loss_val: {:.4f}'.format(loss_val.item()),
            'f1_val: {:.4f}'.format(f1_val),
            'loss_val: {:.4f}'.format(loss_test.item()),
            'f1_test: {:.4f}'.format(f1_test),
            'time: {:.4f}s'.format(time.time() - t))
        '''
        val_performances.append(f1_val.item())
        test_performances.append(f1_test.item())

elif task == 'linkpred':
    output = model(features, adj, 'linkpred', val_edges)
    loss_val = criterion(output, val_label)
    logits = torch.sigmoid(output)
    auc_val = roc_auc_score(val_label.cpu().numpy(),
logits.detach().cpu().numpy())

    output = model(features, adj, 'linkpred', test_edges)
    loss_test = criterion(output, test_label)
    logits = torch.sigmoid(output)
    auc_test = roc_auc_score(test_label.cpu().numpy(),
logits.detach().cpu().numpy())
    '''
    print('Epoch: {:04d}'.format(epoch+1),
        'loss_train: {:.4f}'.format(loss_train.item()),
        'auc_train: {:.4f}'.format(auc_train),
        'loss_val: {:.4f}'.format(loss_val.item()),
        'auc_val: {:.4f}'.format(auc_val),
        'loss_test: {:.4f}'.format(loss_test.item()),
        'auc_test: {:.4f}'.format(auc_test),
        'time: {:.4f}s'.format(time.time() - t))
    '''
    val_performances.append(auc_val)
    test_performances.append(auc_test)
```

```
def test(task='nodecls'):
    if task == 'nodecls':
        model.eval()
        output = model(features, adj)
        loss_test = F.nll_loss(output[idx_test], labels[idx_test])
        acc_test = accuracy(output[idx_test], labels[idx_test])
        '''
        print("Test set results:",
            "loss= {:.4f}".format(loss_test.item()),
            "accuracy= {:.4f}".format(acc_test.item()))
        '''
    elif task == 'linkpred':
        model.eval()
        with torch.no_grad():
            output = model(features, adj, 'linkpred', test_edges)
            loss_test = criterion(output, test_label)
            logits = torch.sigmoid(output)
        auc_test = roc_auc_score(test_label.cpu().numpy(),
logits.detach().cpu().numpy())
        '''
        print("Test set results:",
            "loss= {:.4f}".format(loss_test.item()),
            "auc score= {:.4f}".format(auc_test))
        '''
def output_best(val_performances, test_performances, task='nodecls'):
    val_performances = np.array(val_performances)
    max_id = np.argmax(val_performances)
    if task == 'linkpred':
        print("Test set results (with best validation performance):",
            "auc score= {:.4f}".format(test_performances[max_id]))
        pass
    else:
        if args.dataset != 'ppi':
            print("Test set results (with best validation performance):",
                "acc = {:.4f}".format(test_performances[max_id]))
        else:
            print("Test set results (with best validation performance):",
                "f1_score = {:.4f}".format(test_performances[max_id]))




# Train model
t_total = time.time()
```

```
for epoch in range(args.epochs):
    train(epoch, args.task)
#print("Optimization Finished!")
#print("Total time elapsed: {:.4f}s".format(time.time() - t_total))

# Testing
#test(args.task)
print('dataset:',args.dataset,' --- task:',args.task,' ---
self_loop:',args.self_loop,' --- layer_num:',args.layer_num,' ---
pair_norm:',args.pair_norm,' --- activate:',args.activate,' ---
hidden:',args.hidden)

output_best(val_performances, test_performances,args.task)
print('-----------------------------------------')
```

layers.py

```
import math

import torch

from torch.nn.parameter import Parameter
from torch.nn.modules.module import Module


class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

```
    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' (' \
               + str(self.in_features) + ' -> ' \
               + str(self.out_features) + ')'
```

Models.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from layers import GraphConvolution
from torch_geometric.nn import PairNorm

class GCN(nn.Module):
    def __init__(self, in_channels, hid_channels, out_channels, dropout,
                 layer_num=2, activation='relu', drop_edge=False,
pair_norm=False):
        super(GCN, self).__init__()

        self.gc_inp = GraphConvolution(in_channels, hid_channels)
        self.gc_hids = nn.ModuleList([GraphConvolution(hid_channels,
hid_channels) for _ in range(layer_num-2)])
        self.gc_out = GraphConvolution(hid_channels, out_channels)
        if activation == 'relu':
            self.activate = F.relu
        elif activation == 'sigmoid':
            self.activate = torch.sigmoid
        elif activation == 'tanh':
            self.activate =  torch.tanh


        self.pair_norm = pair_norm
        if pair_norm == True:
            self.norm = PairNorm()
```

```python
        self.dropout = nn.Dropout(dropout)

        # for ppi dataset
        self.linear_out = nn.Linear(out_channels, 121)

    def forward(self, x, adj, task='nodecls', edges=None, ppi=False):
        x = self.gc_inp(x, adj)
        x = self.activate(x)

        for gc_layer in self.gc_hids:
            x = self.dropout(x)
            x = gc_layer(x, adj)

            if self.pair_norm:
                x = self.norm(x)

            x = self.activate(x)

        x = self.dropout(x)
        x = self.gc_out(x, adj)

        if task == 'nodecls':
            if ppi == False:
                # x.shape = [node_num, label_class_num]
                return F.log_softmax(x, dim=1)
            else:
                x = self.linear_out(x)
                # x.shape = [node_num, label_dim]
                return x
        elif task == 'linkpred':
            # x.shape = [node_num, hid_channels]
            assert edges != None
            src = x[edges[0]] # shape = [src_num, hid_channels]
            dst = x[edges[1]] # shape = [node_num, hid_channels]
            inner_prods = (src * dst).sum(dim=-1) # shape =[src_num]
            return inner_prods
```

utils.py

```python
import numpy as np
```

```python
import scipy.sparse as sp
import torch
from torch_geometric.utils import negative_sampling
import json

def encode_onehot(labels):
    classes = set(labels)
    classes_dict = {c: np.identity(len(classes))[i, :] for i, c in
                    enumerate(classes)}
    labels_onehot = np.array(list(map(classes_dict.get, labels)),
                             dtype=np.int32)
    return labels_onehot


def load_data(dataset, task, self_loop):

    if dataset == 'cora':
        path =
"/data2/home/zhaoyi/labs/USTC-labs/deeplearn_lab4_gcn/datasets/cora/"
        dataset = "cora"
    elif dataset == 'citeseer':
        #path = "../datasets/citeseer_new/"
        path =
"/data2/home/zhaoyi/labs/USTC-labs/deeplearn_lab4_gcn/datasets/citeseer_new/
"
        dataset = "citeseer"

    #print('Loading {} dataset...'.format(dataset))

    idx_features_labels = np.genfromtxt("{}{}.content".format(path, dataset),
                                        dtype=np.dtype(str))
    np.random.shuffle(idx_features_labels)
    features = sp.csr_matrix(idx_features_labels[:, 1:-1], dtype=np.float32)
    labels = encode_onehot(idx_features_labels[:, -1])
    # build graph
    idx = np.array(idx_features_labels[:, 0], dtype=np.int32)
    idx_map = {j: i for i, j in enumerate(idx)}
    edges_unordered = np.genfromtxt("{}{}.cites".format(path, dataset),
                                    dtype=np.int32)

    temp1 = map(idx_map.get, edges_unordered.flatten())
    temp2 = list(temp1)
    x = list(edges_unordered.flatten())
    print(x[462])
    for i in range(len(temp2)):
```

```
        elem = temp2[i]
        try:
            elem = int(elem)
        except TypeError:
            print(i)

    edges = np.array(temp2, dtype=np.int32).reshape(edges_unordered.shape)
    '''
    edges = np.array(list(map(idx_map.get, edges_unordered.flatten())),
                     dtype=np.int32).reshape(edges_unordered.shape)
    '''
    #print('You are currently running {} task on {} dataset...'.format(task,
dataset))
    if task == 'linkpred':
        edge_num = edges.shape[0]
        shuffled_ids = np.random.permutation(edge_num)
        test_set_size = int(edge_num * 0.15)
        val_set_size = int(edge_num * 0.15)
        test_ids = shuffled_ids[ : test_set_size]
        val_ids = shuffled_ids[test_set_size : test_set_size + val_set_size]
        train_ids = shuffled_ids[test_set_size + val_set_size : ]

        train_pos_edges = torch.tensor(edges[train_ids], dtype=int)
        val_pos_edges = torch.tensor(edges[val_ids], dtype=int)
        test_pos_edges = torch.tensor(edges[test_ids], dtype=int)

        train_pos_edges = torch.transpose(train_pos_edges, 1, 0)
        # shape = [2, train_pos_edge_num]
        val_pos_edges = torch.transpose(val_pos_edges, 1, 0)
        test_pos_edges = torch.transpose(test_pos_edges, 1, 0)

        def negative_sample(pos_edges, nodes_num):
            '''
            pos_edges = [[src_1,...],
                         [dst_1,...]]
            '''
            neg_edges = negative_sampling(
                edge_index=pos_edges,
                num_nodes=nodes_num,
                num_neg_samples=pos_edges.shape[1],
                method='sparse'
            )
            edges = torch.cat((pos_edges, neg_edges), dim=-1)
            '''
```

```
        edges = [[src_1,src_2,...,src_m],
                 [dst_1,dst_2,...,dst_m]]
        shape = [2, 2*train_edge_num]
        '''
        edges_label = torch.cat((
            torch.ones(pos_edges.shape[1]),
            torch.zeros(neg_edges.shape[1])
        ),dim=0)
        # size = [2*train_edge_num]
        return edges, edges_label

    train_edges, train_label = negative_sample(train_pos_edges,
idx.shape[0])
    val_edges, val_label = negative_sample(val_pos_edges, idx.shape[0])
    test_edges, test_label = negative_sample(test_pos_edges, idx.shape[0])

    adj = sp.coo_matrix((np.ones(train_pos_edges.shape[1]),
(train_pos_edges[0], train_pos_edges[1])),
                    shape=(idx.shape[0], idx.shape[0]),
                    dtype=np.float32)

    # build symmetric adjacency matrix
    adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)

    features = normalize(features)

    if self_loop == True:
        adj = normalize(adj + sp.eye(adj.shape[0]))
    else:
        adj = normalize(adj)

    features = torch.FloatTensor(np.array(features.todense()))
    adj = sparse_mx_to_torch_sparse_tensor(adj)

    train_edges = train_edges.tolist()
    val_edges = val_edges.tolist()
    test_edges = test_edges.tolist()
    train_label = train_label.type(torch.float)
    val_label = val_label.type(torch.float)
    test_label = test_label.type(torch.float)

    return adj, features, train_edges, val_edges, test_edges, \
            train_label, val_label, test_label
```

```
    elif task == 'nodecls':
        adj = sp.coo_matrix((np.ones(edges.shape[0]), (edges[:, 0], edges[:,
1])),
                            shape=(labels.shape[0], labels.shape[0]),
                            dtype=np.float32)

        # build symmetric adjacency matrix
        adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)

        features = normalize(features)

        if self_loop == True:
            adj = normalize(adj + sp.eye(adj.shape[0]))
        else:
            adj = normalize(adj)

        # split train || val || test
        idx_train = range(140)
        idx_val = range(200, 500)
        idx_test = range(500, 1500)

        features = torch.FloatTensor(np.array(features.todense()))
        labels = torch.LongTensor(np.where(labels)[1])
        adj = sparse_mx_to_torch_sparse_tensor(adj)

        idx_train = torch.LongTensor(idx_train)
        idx_val = torch.LongTensor(idx_val)
        idx_test = torch.LongTensor(idx_test)

        return adj, features, labels, idx_train, idx_val, idx_test

    else:
        raise Exception("hyper-parameter `task` belongs to \{'nodecls',
'linkpred'\}.")


def normalize(mx):
    """Row-normalize sparse matrix"""
    rowsum = np.array(mx.sum(1))
    r_inv = np.power(rowsum, -1).flatten()
    r_inv[np.isinf(r_inv)] = 0.
    r_mat_inv = sp.diags(r_inv)
    mx = r_mat_inv.dot(mx)
    return mx
```

```python
def accuracy(output, labels):

    preds = output.max(1)[1].type_as(labels)

    correct = preds.eq(labels).double()
    correct = correct.sum()
    return correct / len(labels)



def sparse_mx_to_torch_sparse_tensor(sparse_mx):
    """Convert a scipy sparse matrix to a torch sparse tensor."""
    sparse_mx = sparse_mx.tocoo().astype(np.float32)
    indices = torch.from_numpy(
        np.vstack((sparse_mx.row, sparse_mx.col)).astype(np.int64))
    values = torch.from_numpy(sparse_mx.data)
    shape = torch.Size(sparse_mx.shape)
    return torch.sparse.FloatTensor(indices, values, shape)


def load_ppi_data(task='nodecls', self_loop=True):
    path =
"/data2/home/zhaoyi/labs/USTC-labs/deeplearn_lab4_gcn/datasets/ppi/"
    #print('Loading PPI dataset...')
    feature_file = path + "ppi-feats.npy"
    label_file = path + "ppi-class_map.json"
    edge_file = path + "ppi-walks.txt"
    graph_file = path + "ppi-G.json"

    #print('Uploading features ...')
    features = np.load(feature_file) # shape = (56944, 50)
    features = sp.csr_matrix(features, dtype=np.float32)
    #print('Uploading labels...')
    fr_label = open(label_file, "r")
    label_dict = json.load(fr_label)
    proc_label_dict = dict()
    for key in label_dict:
        proc_label_dict[int(key)] = list(label_dict[key])
    _labels = sorted(proc_label_dict.items(), key=lambda d: d[0])
    labels = list()
    for item in _labels:
        _, x = item
        labels.append(x)
    labels = np.array(labels, dtype=np.int32)
    print('Uploading graph...')
```

```
fr_graph = open(graph_file, "r")
graph_dict = json.load(fr_graph)
nodes = graph_dict["nodes"]
links = graph_dict["links"]
#print('Generating edges')
edges = [[links[i]["source"], links[i]["target"]] for i in range(len(links))]
edges = np.array(edges, dtype=np.int32)
#print('Generating nodes')
idx = list()
idx_train = list()
idx_val = list()
idx_test = list()
for i in range(len(nodes)):
    idx.append(nodes[i]["id"])
    if nodes[i]["test"] == True:
        idx_test.append(nodes[i]["id"])
    elif nodes[i]["val"] == True:
        idx_val.append(nodes[i]["id"])
    else:
        idx_train.append(nodes[i]["id"])
idx = np.array(idx, dtype=np.int32)

#print('You are currently running {} task on PPI dataset...'.format(task))
if task == 'linkpred':
    edge_num = edges.shape[0]
    shuffled_ids = np.random.permutation(edge_num)
    test_set_size = int(edge_num * 0.15)
    val_set_size = int(edge_num * 0.15)
    test_ids = shuffled_ids[ : test_set_size]
    val_ids = shuffled_ids[test_set_size : test_set_size + val_set_size]
    train_ids = shuffled_ids[test_set_size + val_set_size : ]

    train_pos_edges = torch.tensor(edges[train_ids], dtype=int)
    val_pos_edges = torch.tensor(edges[val_ids], dtype=int)
    test_pos_edges = torch.tensor(edges[test_ids], dtype=int)

    train_pos_edges = torch.transpose(train_pos_edges, 1, 0)
    # shape = [2, train_pos_edge_num]
    val_pos_edges = torch.transpose(val_pos_edges, 1, 0)
    test_pos_edges = torch.transpose(test_pos_edges, 1, 0)

    def negative_sample(pos_edges, nodes_num):
        '''
        pos_edges = [[src_1,...],
```

```
                    [dst_1,...]]
          '''
          neg_edges = negative_sampling(
             edge_index=pos_edges,
             num_nodes=nodes_num,
             num_neg_samples=pos_edges.shape[1],
             method='sparse'
          )
          edges = torch.cat((pos_edges, neg_edges), dim=-1)
          '''
          edges = [[src_1,src_2,...,src_m],
                 [dst_1,dst_2,...,dst_m]]
          shape = [2, 2*train_edge_num]
          '''
          edges_label = torch.cat((
             torch.ones(pos_edges.shape[1]),
             torch.zeros(neg_edges.shape[1])
          ),dim=0)
          # size = [2*train_edge_num]
          return edges, edges_label

       train_edges, train_label = negative_sample(train_pos_edges,
idx.shape[0])
       val_edges, val_label = negative_sample(val_pos_edges, idx.shape[0])
       test_edges, test_label = negative_sample(test_pos_edges, idx.shape[0])

       adj = sp.coo_matrix((np.ones(train_pos_edges.shape[1]),
(train_pos_edges[0], train_pos_edges[1])),
                    shape=(idx.shape[0], idx.shape[0]),
                    dtype=np.float32)

       # build symmetric adjacency matrix
       adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)

       features = normalize(features)

       if self_loop == True:
          adj = normalize(adj + sp.eye(adj.shape[0]))
       else:
          adj = normalize(adj)

       features = torch.FloatTensor(np.array(features.todense()))
       adj = sparse_mx_to_torch_sparse_tensor(adj)
```

```
        train_edges = train_edges.tolist()
        val_edges = val_edges.tolist()
        test_edges = test_edges.tolist()
        train_label = train_label.type(torch.float)
        val_label = val_label.type(torch.float)
        test_label = test_label.type(torch.float)

        return adj, features, train_edges, val_edges, test_edges, \
                    train_label, val_label, test_label

    elif task == 'nodecls':
        adj = sp.coo_matrix((np.ones(edges.shape[0]), (edges[:, 0], edges[:,
1])),
                        shape=(labels.shape[0], labels.shape[0]),
                        dtype=np.float32)

        # build symmetric adjacency matrix
        adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)

        features = normalize(features)

        if self_loop == True:
            adj = normalize(adj + sp.eye(adj.shape[0]))
        else:
            adj = normalize(adj)


        features = torch.FloatTensor(np.array(features.todense()))
        labels = torch.LongTensor(labels)
        adj = sparse_mx_to_torch_sparse_tensor(adj)

        idx_train = torch.LongTensor(idx_train)
        idx_val = torch.LongTensor(idx_val)
        idx_test = torch.LongTensor(idx_test)

        return adj, features, labels, idx_train, idx_val, idx_test

    else:
        raise Exception("hyper-parameter `task` belongs to \{'nodecls',
'linkpred'\}.")
```