ДОДАТОК А
Вихідний код програми

```
import math
import numpy as np
import networkx as nx
import torch
import torch.nn.functional as F
from torch_geometric.nn import ChebConv, GCNConv # noqa
from torch_geometric.utils.convert import to_networkx
from torch_geometric.data import Data
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score


#%%

# Params
score = accuracy_score # accuracy_score, recall_score, precision_score, f1_score
n = 50
is_gamma_dist = False
modelName = 'gcn'

innerDegree = 6
outerDegree = 1
class_end_rate = 0.2
featurePart_rate = 0.5


classes = 2

def connect_two_ranges(fr1, to1, fr2, to2):
  ar = []
  for i in range(fr1, to1):
    for j in range(fr2, to2):
      if i != j:
        ar.append([[i, j], [j, i]])

  array = np.array(ar, dtype=int)
  np.random.shuffle(array)
  return array

def get_edges(fr, to):
  return connect_two_ranges(fr, to, fr, to)

def part(a, p):
  return a[:int(math.sqrt(a.shape[0]) * p), :]

def generate_data_list():
  class_end = int(2*n* class_end_rate)
  featurePart = int((2 * n – class_end) * featurePart_rate)
  feature_rest = int(n * 2 – featurePart)

  data_list = []
  for i in range(10):
    edge_index = torch.tensor(np.concatenate((
      part(get_edges(0, class_end), innerDegree).reshape(-1, 2),
      part(get_edges(class_end, n * 2), innerDegree).reshape(-1, 2),
      part(connect_two_ranges(0, class_end, class_end, n * 2), outerDegree).reshape(-1, 2)
    ), axis=0))

    x = torch.cat((
```

```
      torch.from_numpy(np.random.gamma(5, 2, feature_rest)),
      torch.from_numpy(np.random.gamma(12, 2, featurePart))), dim=0).float() if is_gamma_dist else
torch.randn(
      n * 2, dtype=torch.float) + torch.cat((torch.zeros(feature_rest),
                    torch.ones(featurePart) * 2), dim=0)
    y = torch.cat((torch.ones(class_end, dtype=torch.long), torch.zeros((2*n-class_end), dtype=torch.long)), 0)
    # y = torch.ones(2*n, dtype=torch.long)

    data = Data(x=torch.reshape(x, (x.shape[0], 1)), edge_index=edge_index.t().contiguous(), y=y)
    data_list.append(data)
  return data_list

data_list = generate_data_list()
data = data_list[0]



import matplotlib.pyplot as plt
plt.figure(1,figsize=(14,12))

d = to_networkx(data, to_undirected=True)
node_labels = data.y[list(d.nodes)].numpy()
node_sizes = data.x[list(d.nodes)].numpy()
pos=nx.spring_layout(d)
nx.draw(d, pos=pos, cmap="seismic", node_color = node_labels, node_size=node_sizes*10 if is_gamma_dist
else (node_sizes+3)*20,linewidths=6)
# nx.draw_networkx_nodes(d, pos=nx.spring_layout(d), cmap="seismic", node_color = node_labels,
node_size=node_sizes*10 if is_gamma_dist else (node_sizes+3)*20,linewidths=6)

plt.show()

#%%

import matplotlib.pyplot as plt
plt.figure(1,figsize=(14,12))

d = to_networkx(data, to_undirected=True)
node_labels = data.y[list(d.nodes)].numpy()
node_sizes = data.x[list(d.nodes)].numpy()
# nx.draw_spring(d, cmap="seismic", node_color = node_labels, node_size=node_sizes*10 if is_gamma_dist
else (node_sizes+3)*20,linewidths=6)
nx.draw_networkx_nodes(d, pos=pos, cmap="seismic", node_color = node_labels, node_size=node_sizes*10 if
is_gamma_dist else (node_sizes+3)*20,linewidths=6)

plt.show()

#%%

class GCNNet(torch.nn.Module):
  def __init__(self):
    super().__init__()
    features = data.num_features
    self.conv1 = GCNConv(features, 6, cached=True, normalize=True)
    self.conv2 = GCNConv(6, classes, cached=True, normalize=True)
    # self.conv1 = ChebConv(features, 6, K=7)
    # self.conv2 = ChebConv(6, classes, K=7)

  def forward(self, data):
    x, edge_index = data.x, data.edge_index
    x = F.relu(self.conv1(x, edge_index))
    x = F.dropout(x, training=self.training)
    x = self.conv2(x, edge_index)
```

```
    x = F.log_softmax(x, dim=1)
    return x

class ChebNet(torch.nn.Module):
  def __init__(self):
    super().__init__()
    features = data.num_features
    # self.conv1 = GCNConv(features, 6, cached=True, normalize=True)
    # self.conv2 = GCNConv(6, classes, cached=True, normalize=True)
    self.conv1 = ChebConv(features, 6, K=7)
    self.conv2 = ChebConv(6, classes, K=7)

  def forward(self, data):
    x, edge_index = data.x, data.edge_index
    x = F.relu(self.conv1(x, edge_index))
    x = F.dropout(x, training=self.training)
    x = self.conv2(x, edge_index)
    x = F.log_softmax(x, dim=1)
    return x

class LinearNet(torch.nn.Module):
  def __init__(self):
    super().__init__()
    self.layer = torch.nn.Linear(1, 2)
    # self.activation = torch.nn.LogSoftmax()

  def forward(self, data):
    x, edge_index = data.x, data.edge_index
    x = self.layer(x)
    x = F.log_softmax(x, dim=1)
    return x

modelDict = {'gcn' : GCNNet, 'cheb': ChebNet, 'reg': LinearNet}

#%%

def initModel(modelName):
  return modelDict[modelName]()

model = initModel(modelName)

# device = torch.device('cpu')
# model, data = Net().to(device), data.to(device)

def initOptimizer():
  if modelName == 'reg':
    return torch.optim.Adam([
    dict(params=model.layer.parameters(), weight_decay=0) # 0 or 5e-4
  ], lr=0.01)
  else:
    return torch.optim.Adam([
      dict(params=model.conv1.parameters(), weight_decay=5e-4),
      dict(params=model.conv2.parameters(), weight_decay=0)
    ], lr=0.001) # Only perform weight-decay on first convolution.

optimizer = initOptimizer()


#%%

train_data = data_list[:4]
test_data = data_list[4:]
```

```
def train(model, optimizer):
  model.train()

  total_loss = 0
  for data in train_data:
    # data = data.to(device)
    optimizer.zero_grad()
    out = model(data)
    # print(out)
    loss = F.nll_loss(out, data.y)
    loss.backward()
    optimizer.step()
    total_loss += loss.item()
  return total_loss


#%%


score = accuracy_score # accuracy_score, recall_score, precision_score, f1_score
@torch.no_grad()
def test(model):
  model.eval()

  train_accs, test_accs, recalls, prs, f1s = [], [], [], [], []

  for data in train_data:
    logits = model(data)
    pred = logits.max(1)[1]
    train_acc = score(data.y, pred)
    train_accs.append(train_acc)

  for data in test_data:
    logits = model(data)
    pred = logits.max(1)[1]
    test_accs.append(accuracy_score(data.y, pred))
    recalls.append(recall_score(data.y, pred))
    prs.append(precision_score(data.y, pred))
    f1s.append(f1_score(data.y, pred))


  return sum(train_accs) / len(train_accs), \
      sum(test_accs) / len(test_accs), \
      sum(recalls) / len(recalls), \
      sum(prs) / len(prs), \
      sum(f1s) / len(f1s)


#%%

plot_X, plot_Y = [], []
plot_Xt, plot_Yt = [], []
for epoch in range(1, 50):
  train(model, optimizer)
  train_acc, test_acc, recall, pr, f1 = test(model)
  plot_X.append(epoch)
  plot_Xt.append(epoch)
  plot_Y.append(train_acc)
  plot_Yt.append(test_acc)
  print(f'Epoch: {epoch:03d}, Train: {train_acc:.4f}, '
     f'Test: {test_acc:.4f}, r: {recall:.4f}, p: {pr:.4f}, f1: {f1:.4f}')
```

```
plt.xlabel('Epoch')
plt.ylim(bottom=0, top=1)
plt.plot(plot_X, plot_Y, label='Train')
plt.plot(plot_Xt, plot_Yt, label='Test')
plt.legend()
plt.savefig('learning-result.png', dpi=150)
```

ДОДАТОК Б
Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	78 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Осьмачко В. Є.			Дослідження спектральної конволюції графових нейронних мереж
Перевірив		Шевченко О. Ю.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІздм-20-2	122
ХНУРЕ кафедра ШІ	