96 lines - 31 Removals

```
import json
import torch
import torch.nn as nn
import numpy as np
import torch_geometric.transforms as T
from datasets.load_datasets import get_dataset, get_dataloader

from models import GCN


def evaluate(dataloader, model, loss_fc):
    acc = []
    loss_list = []
    model.eval()
    with torch.no_grad():
        for data in dataloader:
            logit = model(data)
            loss = loss_fc(logit, data.y)
            prediction = torch.argmax(logit, -1)
            loss_list.append(loss.item())
            acc.append((prediction == data.y).numpy())
    return np.concatenate(acc, axis=0).mean(), np.average(loss_list)


if __name__ == '__main__':

    with open("configs.json") as config_file:
        configs = json.load(config_file)
        dataset_name = configs.get("dataset_name").get("graph")

    epochs = 5000
    pooling = {'mutagenicity': ['max', 'mean', 'sum'],
               'ba_2motifs': ['max'],
               'bbbp': ['max', 'mean', 'sum']}
    early_stop = 100
    loop = True




    if dataset_name == 'ba_2motifs':
        loop = False

    normalize = T.NormalizeFeatures()
    dataset = get_dataset(dataset_dir="./datasets", dataset_name=dataset_name)
    dataset.data.x = dataset.data.x.float()
    dataset.data = normalize(dataset.data)

    data_loader = get_dataloader(dataset, batch_size=32, random_split_flag=True,
                                 data_split_ratio=[0.8, 0.1, 0.1], seed=2)

    model = GCN(n_feat=dataset.num_node_features,
                n_hidden=20,
                n_class=dataset.num_classes,
                pooling=pooling[dataset_name],
                loop=loop)
```

93 lines + 28 Additions

```
import json
import torch
import torch.nn as nn
import numpy as np
import torch_geometric.transforms as T
from MalNet_Tiny import MalNetTiny

from torch_geometric.loader import DataLoader
from models import GCN
from tqdm import tqdm

def evaluate(dataloader, model, loss_fc):
    acc = []
    loss_list = []
    model.eval()
    with torch.no_grad():
        for data in dataloader:
            logit = model(data)
            loss = loss_fc(logit, data.y)
            prediction = torch.argmax(logit, -1)
            loss_list.append(loss.item())
            acc.append((prediction == data.y).numpy())
    return np.concatenate(acc, axis=0).mean(), np.average(loss_list)


if __name__ == '__main__':

    with open("configs.json") as config_file:
        configs = json.load(config_file)
        dataset_name = configs.get("dataset_name").get("graph")

    epochs = 5000
    pooling = {'malnet_tiny': ['max', 'mean', 'sum']}



    early_stop = 100
    transform = T.Compose([T.RemoveIsolatedNodes() ,T.AddSelfLoops(), T.AddLaplacianEigenvectorPE(5,attr_name='x'),T.AddRandomWalkPE(20,attr_name='x'),T.ToSparseTensor()])



    normalize = T.NormalizeFeatures()
    dataset = MalNetTiny(root='./datasets', transform=transform)
    for i,graph in enumerate(tqdm(dataset,total=3500)):
        dataset[i].x = torch.cat([graph.x.to("cpu") ,graph.random_walk_pe.to("cpu")],dim=1)
    data_loader = DataLoader(dataset, batch_size=32, shuffle=True)


    model = GCN(n_feat=dataset.num_node_features,
                n_hidden=20,
                n_class=dataset.num_classes,
                pooling=pooling[dataset_name][0],
                loop=True)  # Adjust as needed

```

```
    optimizer = torch.optim.Adam(model.parameters(), lr=5e-3)
    loss_fc = nn.CrossEntropyLoss()

    model_file = './src/' + dataset_name + '.pt'

    model.train()
    early_stop_count = 0
    best_acc = 0
    best_loss = 100
    for epoch in range(epochs):
        acc = []
        loss_list = []
        model.train()
        for i, data in enumerate(data_loader['train']):

            logit = model(data)
            loss = loss_fc(logit, data.y)
            optimizer.zero_grad()
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=2.0)
            optimizer.step()

            prediction = torch.argmax(logit, -1)
            loss_list.append(loss.item())
            acc.append((prediction == data.y).numpy())

        eval_acc, eval_loss = evaluate(dataloader=data_loader['eval'], model=model, loss_fc=loss_fc)
        print(epoch, eval_acc, eval_loss)

        is_best = (eval_acc > best_acc) or \
                  (eval_loss < best_loss and eval_acc >= best_acc)
        if is_best:
            early_stop_count = 0
        else:
            early_stop_count += 1
        if early_stop_count > early_stop:
            break
        if is_best:
            best_acc = eval_acc
            best_loss = eval_loss
            early_stop_count = 0
            model.save(model_file)

    model.load(model_file)
    model.eval()
    acc_test, acc_loss = evaluate(data_loader['test'], model, loss_fc)

    print(acc_test)
```

```
    optimizer = torch.optim.Adam(model.parameters(), lr=5e-3)
    loss_fc = nn.CrossEntropyLoss()

    model_file = './src/' + dataset_name + '.pt'

    model.train()
    early_stop_count = 0
    best_acc = 0
    best_loss = 100
    for epoch in range(epochs):
        acc = []
        loss_list = []
        model.train()
        for i, data in enumerate(data_loader):
            print(data)
            logit = model(data)
            loss = loss_fc(logit, data.y)
            optimizer.zero_grad()
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=2.0)
            optimizer.step()

            prediction = torch.argmax(logit, -1)
            loss_list.append(loss.item())
            acc.append((prediction == data.y).cpu().numpy())
        eval_acc, eval_loss = evaluate(dataloader=data_loader, model=model, loss_fc=loss_fc)
        print(epoch, eval_acc, eval_loss)

        is_best = (eval_acc > best_acc) or \
                  (eval_loss < best_loss and eval_acc >= best_acc)
        if is_best:
            early_stop_count = 0
        else:
            early_stop_count += 1
        if early_stop_count > early_stop:
            break
        if is_best:
            best_acc = eval_acc
            best_loss = eval_loss
            early_stop_count = 0
            torch.save(model.state_dict(), model_file)

    model.load_state_dict(torch.load(model_file))
    model.eval()
    acc_test, acc_loss = evaluate(dataloader=data_loader, model=model, loss_fc=loss_fc)

    print(acc_test)
```