Assignment 05
Graph Convolution Network (GCN)

Due Date: 16th June 2022

Guidelines

- Submit all of your code, saved models, and results in a single zip file with the name FirstName_RollNumber_05.zip
- Submit a single zip file containing:
  - a) Code (task01.ipynb, task02.ipynb)
  - b) Report
- There should be a report.pdf detailing your experience and highlighting any interesting results.
- Email the instructor or TAs if there are any questions.
- **Don't resubmit the dataset provided.**

Overview

In this assignment, you will be training GCN for community detection and graph classification. The goals of this assignment are:

- Understand how to handle graph data.
- Understand how a model can be created, trained, validated, and saved in PyTorch Geometric.
- Understand how GCN can be used for community detection and graph classification

Datasets

In this assignment you will be using two datasets:

1) **Citeseer**
   Citeseer is a citation network where a node is a published paper and an edge represents a citation link. It has 3312 nodes and 4732 edges. Nodes belong to six different communities. Each paper/node feature in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3703 unique words.

2) **Proteins**
   Proteins dataset is a collection of 1,113 proteins that are classified as enzymes or non-enzymes. Nodes in each graph represent the amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart.

The dataset folder consists of two subfolders "citeseer" and "proteins". Read the readme.txt for dataset usage details.

Task 01: Semi-Supervised Community Detection using GCN (40 Marks)

In this task, you will perform semi-supervised community detection using GCN. For this task, you will be using the **Citeseer** dataset.

1. **Load Dataset:** Read the data files and convert them to a suitable format. Perform a 10/60/30 train/test/validation split.

2. **Initialize Network:** Write a function init_net() to initialize the network. You can use PyTorch.geometric built-in functions to create and initialize the network.

3. **Train Network:** Write a function train(epochs, train_x, train_y, val_x, val_y, loss_func, optimizer, learning rate) to train the initialized network. For both training and validation data record accuracy and loss at each epoch to plot accuracy and loss curves later.

4. **Save Network:** Save the trained network using either pickle or PyTorch for later use.

5. **Test Network:** Now write a function test() that uses a trained network to make predictions on a test set. The function should return loss and accuracy on test data.

6. **Visualize Results:** Plot the **"epoch vs accuracy"**, **"epoch vs loss"** for train and validation data, and confusion matrix for all the three splits separately. Also, show your best and worst predictions on the test dataset.

Task 02: Graph Classification using GCN (50 Marks)

In this task, you will perform graph classification using GCN. For this task, you will be using the **Proteins** dataset.

1) **Data Pre-processing:** The proteins dataset contains four .txt files. Read them using pandas and check their shapes. If you are not able to achieve the following shapes you are doing something wrong please recheck your reading method.

    - Shape of PROTEINS_graph_labels: (1113, 1)
    - Shape of PROTEINS_node_attributes: (43471, 1)
    - Shape of PROTEINS_graph_indicator: (43471, 1)
    - Shape of PROTEINS_A.txt: (162088, 2)

    Before moving forward, please make sure to subtract 1 from all the entries of PROTEINS_graph_labels and PROTEINS_A. We are doing so to avoid future errors. PROTEINS_graph_labels has graphs labeled as 1 and 2 however PyTorch needs them as 0 and 1. Similarly, PROTEINS_A has node numbers from 1 to 43472 but having them in the range 0 to 43471 can help us to avoid subtracting -1 from edge indexes when feeding data to GCN.

    Next, you need to extract graphs from the data. PROTEINS_graph_indicator rows are equal to the number of nodes and tell the mapping of nodes and graphs. The nodes belonging to one graph would have the same indicator values i.e. for example from index

0-10 the indicator is 1 and then from 11-15 the indicator is 2 it means that the first 10 nodes belong to graph 1 and the next 5 nodes are part of graph 2 and so on.

Once you know which nodes belong to one graph you need to extract their corresponding edges and node features from PROTEINS_A and PROTEINS_node_attributes respectively. Note that edges won't be equal to the number of nodes in the graph for most cases but node attributes will always be equal to the number of nodes in the graph.

Then you need to create a graph data object using **torch_geometric.data.Data** and store them in a list named "`Dataset`".

```
G1=torch_geometric.data.Data(x=torch.tensor(x,dtype=torch.float
),
edge_index=torch.tensor(edge,dtype=torch.long),y=torch.tensor(y
))
```

```
Dataset.append(G1)
```

Where, x=node features, edges=edge_index, y=graph label. Refer to PyTorch geometric documentation for more information.

You need to create data objects for each graph separately. Once you are done processing all the data you should have a dataset list of length 1113.

```
Dataset = [G1, G2, G3, …………, GN] where N = 1113
```

Except for the first graph you need to find the minimum node number from the edge index of each graph and subtract it from all the node numbers. You will be doing so because PyTorch Geometric expects that if there are 5 nodes in a graph then the edge_index must have node numbers from 0 to 4.

Note: Apart from the pre-processing procedure mentioned above you can use any other procedure as per your understanding as long as it fulfills the goal.

2) **Load Dataset:** Shuffle the `Dataset` list and perform a 60/20/20 train/test/validation split and then build an iterator around all three splits using Pytorch Geometric Dataloader.
3) **Initialize Network:** Write a function init_net() to initialize the network. You can use PyTorch.geometric built-in functions to create and initialize the network.
4) **Train Network:** Write a function train(epochs, train_x, train_y, val_x, val_y, loss_func, optimizer, learning rate) to train the initialized network. For both training and validation data record accuracy and loss at each epoch to plot accuracy and loss curves later.
5) **Save Network:** Save the trained network using either pickle or PyTorch for later use.

6) **Test Network:** Now write a function test() that uses a trained network to make predictions on a test set. The function should return loss and accuracy on test data.

7) **Visualize Results:** Plot the **“epoch vs accuracy”**, **“epoch vs loss”** for train and validation data, and confusion matrix for all the three splits separately. Also, show your best and worst predictions on the test dataset.

Report (10 Marks)

For both the tasks show the “epoch vs accuracy” and “epoch vs loss” graph of your best model. Report the train/validation/test accuracy and loss for all the experiments you did along with the model parameters. Also, add your best and worst predictions on the test dataset for both tasks.