Assignment 3

- **Deadline: 11 Nov. 2022 23:59**
- Plagiarism will not be tolerated.
- Read carefully and adhere to formatting rules. As this is the third assignment, non-compliance to formatting rules will be strictly penalized.
- Your code must run on HPC. 0 marks for code that doesn't run.

Problem definition

The problem is of traffic prediction. Your input consists of two things: a road network graph (in the format of an adjacency matrix) and a feature matrix defining traffic at each node on the graph sampled every 5 minutes. One of the graphs has 3193 nodes and 7269 edges, the other has 207 nodes and 1535 edges. Edges are roads and nodes are junctions where the traffic volume is measured. The edge_weight in the adjacency matrix is the length of the road. You have three tasks:

1. Given the traffic at time t on some nodes of the graph, predict the traffic on nodes at time $t+1$.
2. Incorporate the traffic at time steps $\{t-p+1, t-p, \ldots, t-1, t\}$ to predict the traffic at time steps $\{t+1, \ldots, t+f\}$. Or, given a time instant you can look p steps into the past over the graph and have to predict f steps into the future. f may be 1.

Note: these are regression tasks. A regression task is one in which a continuous output has to be produced (as compared to discreet outputs in, for example, classification).

In machine learning, you are given features x of some "thing" and you have to predict a label y for it. For example, given an image's pixel values, predict whether it contains a dog or a cat.

In machine learning, the main task is that of generalisation. You should be able to figure out patterns from given data that are general enough to be applied on unseen data. So, all of the (x, y) pairs are not available. And at test time you'll be given some x^{test} without a corresponding y^{test} value which would need to predicted using your model.

In the context of your problem, the x of node u corresponds to the traffic at **neighboring nodes** and y is the traffic at node u itself.

So, you won't be shown the traffic feature of some nodes. The nodes whose information the model is allowed to see are given in `graph_splits.npz`. Extract this information using:

```
In [39]: import numpy as np
```

In [40]:
```
splits = np.load("graph_splits.npz")
train_node_ids = splits["train_node_ids"]
val_node_ids = splits["val_node_ids"]
test_node_ids = splits["test_node_ids"]
```

In [41]:
```
train_node_ids
```

Out[41]:
```
array([3034409143, 5283966324, 1460300834, ...,  441159658, 5529818201,
       1460311247])
```

Your model is not allowed to train on any data from val and test nodes

Task 1

30 Marks

Write a GNN model using Pytorch Geometric in python that looks at traffic data from nodes $vs = \{v : v \in \mathcal{N}(u)\}$ at time t (a row of `X.csv`) and predicts the traffic at node u at time $t+1$. To train this model, nodes u and v should belong to set of train nodes.

You can make use of the set of validation nodes to see how your model is generalising, or to do some hyperparameter tuning. Do not use the test nodes at all. If you have no use for validation nodes, you can merge them with your train nodes. Mention in the code exactly where you load data from train nodes and validation nodes.

Note, we'll be testing your code on time steps other than provided. So, it might be a good idea to generate validation data on the time axis as well.

Here's the interface definition:

`bash aiz218322_task1.sh train /path/to/train.csv /path/to/adj_mx.csv /path/to/graph_splits.npz`
`bash aiz218322_task1.sh test /path/to/test.csv /path/to/output_file.csv /path/to/task1.model`

The train code should train the model and save it in a file named `aiz218322_task1.model`. (Save the path to adj_mx and graph_splits.npz from the train call and use it during test.) The output file format should be the same as `train.csv` and `test.csv`. Each row of `test.csv` contains traffic at time instant t for all nodes, and corresponding row of `output_file.csv` contains predicted traffic at time instant $t+1$ for all nodes (train, val and test).

Of course, change `aiz218322` with your own roll number. We'll test on a hidden segment of the same dataset (for which you'll need to submit the trained model), as well as on a different dataset. The number of nodes may be upto 30000. Your code should train on GPUs in a reasonable amount of time (1 hour).

Save a model for each given dataset and include it in your submission.

Marking Scheme and Metrics

We'll be using MAE (mean absolute error) and you will be awarded marks as:

$$\text{marks} = \frac{\text{best MAE}}{\text{your MAE}} \times 30$$

Best MAE refers to the lowest MAE score achieved by anyone in the class. Lower MAE is better.

Task 2

50 Marks

Extend your GNN model from task 1 to now take in past p timestep information for neighboring nodes vs to produce future f step prediction for node u. That is take p consecutive rows of `X.csv` and produce next f rows of data. You might need to use a time series model to incorporate temporal information.

You are free to read up work being done on modeling spatio-temporal data using graph neural networks to inspire your work. Your code however must be your own original code. Make sure to cite the work in your report.

$1 \leq p \leq 12, 1 \leq f \leq 12.$

Interface description:

```
bash aiz218322_task2.sh train p f /path/to/train.csv /path/to/adj_mx.csv /path/to/graph_splits.npz
bash aiz218322_task2.sh test p f /path/to/test.npz /path/to/output_file.npz /path/to/task2.model
```

The output file is now in `npz` format, because the output will now not be a 2D table. `test.npz` is also not a 2D table, it is a list of windows. We are providing example files for $p = 12$ and $f = 12$:

```
test_data = np.load('test.npz')
test_data['x'].shape # should be (17, 12, 3193)
# n_windows = 17, p = 12, n_nodes = 3193
# note the key is 'x'
```

```
example_output = np.load('output_file.npz')
example_output['y'].shape # should be (17, 12, 3193)
# n_windows = 17, f = 12, n_nodes = 3193
# note the key is 'y'
```

This should save a model named `aiz218322_task2.model`. You would need to submit one model for each dataset provided.

Marking Scheme and Metrics

We'll be using MAE (mean absolute error) and you will be awarded marks as:

$$\text{marks} = \frac{\text{best MAE}}{\text{your MAE}} \times 50$$

Datasets and model naming

You are being given two datasets **d1** and **d2**. In the following two tasks you shall train a GNN model for **both these datasets** and provide a saved model for both of them (titled d1_aiz218322_task1.model, d2_aiz218322_task1.model, d1_aiz218322_task2.model, d2_aiz218322_task2.model).

Note: do not get confused by different names, the model should save files as aiz218322_task1.model and aiz218322_task2.model when run by us. But pretrained model files of these names should exist in your root folder.

Environment setup

You need to write the code in python. To ensure equality among students, you will be provided with a `environment.yml` file with which you need to create a conda environment using the following commands:

```
module load apps/anaconda/3EnvCreation
conda env create -f environment.yml
module purge
```

Make sure you're connected to the internet.

Then, anytime you want to use the environment in interactive mode on HPC, you can do:

```
module load apps/anaconda/3
conda activate gnn
module purge
```

For activating environment when running jobs in non-interactive mode, do [1]:

```
module load apps/anaconda/3
source activate gnn
module purge
```

Folder structure and things to submit

1. Submit an aiz218322.sh file to moodle. It will clone github repository and cd into it. No need to load any modules, as the environment would already be loaded.
2. Inside this folder should be all your code, with folder structure as required and the two shell script files aiz218322_task1.sh and aiz218322_task2.sh
3. Also, inside this same folder should be the four models d1_aiz218322_task1.model, d2_aiz218322_task1.model, and so on.

General ideas

Idea 1: Using a dense adjacency matrix representation is usually memory heavy. Use sparse adjacency matrices instead.

```
In [42]: import torch
         from torch_geometric.utils import dense_to_sparse
```

```
In [43]: adj_mat = torch.tensor(
             [
                 [0, 0, 0, 0, 4],
                 [0, 0, 0, 5, 0],
                 [0, 0, 0, 0,14],
                 [0,16, 0, 0, 0],
                 [0,21, 0, 0, 4],
             ]
         )
         edge_index, edge_weight = dense_to_sparse(adj_mat)
```

```
In [44]: edge_index

Out[44]: tensor([[0, 1, 2, 3, 4, 4],
                 [4, 3, 4, 1, 1, 4]])
```

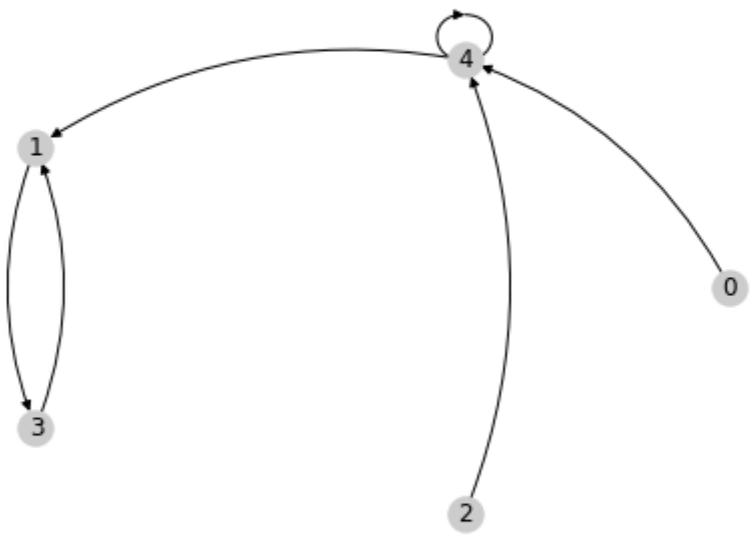
```
In [45]: edge_weight

Out[45]: tensor([ 4,  5, 14, 16, 21,  4])
```

This is the **coo format** [2]) [3] of sparse matrix representation.

Idea 2: Use networkx for graph (or subgraph) visualisation to understand what's going on in a node's neighborhood

```
In [46]: import networkx as nx
         edge_list = [k for k in zip(edge_index[0].numpy().tolist(),
                                     edge_index[1].numpy().tolist())]
         G = nx.from_edgelist(edge_list,create_using=nx.DiGraph)
         nx.draw(G,
                 with_labels=True,
                 node_color='#ccc',
                 arrows=True,
                 connectionstyle='arc3,rad=0.2',
                 pos=nx.circular_layout(G))
```



Idea 3: This is more of a general ML framework for those who are unaware (also refer this).

Steps to follow:

1. Figure out what's the train data.
2. Write the model code, use the available layers and non-linearities in the pipeline.
3. Decide on your loss function.
4. Decide on your optimizer.
5. Write the train step: this includes a forward propagation of data through your model to get a prediction, a comparison of how good the prediction is via loss calculation, back-propagation of gradients w.r.t loss, and the optimization step.
6. Do the train step n_epoch number of times, or until you reach desired level of score.

In [47]:
```
# example of optimizer pipeline
# minimizing f(x) = x ** 2
import torch.optim as optim

def loss_fn(x):
    return x

def f(x):
    return x**2

x = torch.nn.Parameter(torch.randn(1)*10)
print(x.detach().numpy(),f(x).detach().numpy())
opt = optim.SGD([x],lr=1e-3)
for epoch in range(1000):
    opt.zero_grad()
    y = f(x)
    loss = loss_fn(y)
    loss.backward()
    opt.step()
print(x.detach().numpy(),f(x).detach().numpy())
```
```
[4.518917] [20.420612]
[0.6103452] [0.37252125]
```

Idea 4: Use GPUs.

Idea 5: Use cross-validation

A look at the dataset

```
In [48]: import pandas as pd
         df=pd.read_csv('X.csv',index_col=0)
         df.head()
```

Out[48]:

	288416374	288416379	288416380	288416386	288416399	314622896	314622918	314
2016-10-31 16:02:00	0	1	1	0	0	0	0	
2016-10-31 16:07:00	0	1	3	0	0	1	8	
2016-10-31 16:12:00	0	4	5	2	0	0	7	
2016-10-31 16:17:00	6	5	3	7	1	4	12	
2016-10-31 16:22:00	2	5	5	4	0	5	20	

5 rows × 3193 columns

```
In [49]: adj_mx=pd.read_csv('adj_mx_proper.csv',index_col=0)
         adj_mx.head()
```

Out[49]:

	288416374	288416379	288416380	288416386	288416399	314622896	314622918	314
288416374	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
288416379	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
288416380	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
288416386	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
288416399	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 3193 columns

```
In [50]: print(f"{np.count_nonzero(adj_mx.values) = }\nNonzero values in adj_mx = {ad

         np.count_nonzero(adj_mx.values) = 7269
         Nonzero values in adj_mx = [173.86  20.52 143.32 ...   49.63   79.14   99.37]

In [51]: print(f"Density of adjacency matrix = {7269/(3193 * 3193) * 100 : .4f}%")

         Density of adjacency matrix =  0.0713%
```

All the best