EN.553.744 Data Science Methods for Large-Scale Graphs

Homework 2 (March 24 update)

Luana Ruiz, Caio F. D. Netto, Ruijia Zhang

March 24, 2025

You must use Google Colab as your development environment—your `.ipynb` notebook will be your deliverable.

1 Introduction

The goals of this homework are to implement machine learning models in PyTorch Geometric and compare them in terms of their expressivity and stability to graph perturbations.

The setting is a point cloud classification problem, where the goal is to classify graphs corresponding to point clouds of human subjects into different poses.

2 Data Processing

The dataset used in this homework is FAUST, a collection of 3D point cloud representations of human body scans in various poses. Each scan is represented as a graph, where nodes correspond to points in the 3D space, and edges define local connectivity between neighboring points. The classification task involves assigning each graph to one of several pose categories.

After loading the dataset, we will first analyze its structure (number of graphs, number of classes, etc.). Then, we will create a modified dataset, in which all node features are anonymized and replaced with a scalar value of 1.

Exercise 1. *Load the FAUST dataset from PyTorch Geometric. You will have to download the raw dataset from this link, and then load it into Python with the commands:*

```
TrainDataset1 = FAUST(root=dataset_path,pre_transform=FaceToEdge())
TestDataset1 =
    FAUST(root=dataset_path,pre_transform=FaceToEdge(),train=False)
```

```
Dataset1 = TrainDataset1 + TestDataset1
```

where `dataset_path` *is the dataset location in your computer. Report the total number of graphs and the number of classes in the dataset (including both training and test samples).*

Exercise 2. *Report the minimum, mean and maximum of the following quantities over all graphs in the dataset:*

- *Number of nodes*
- *Number of edges*
- *Average degree*

Exercise 3. *Transform* `Dataset1` *into a dataset with scalar, anonymous node features. I.e., for each graph, X should be a $n \times 1$ tensor with entries 1. We will call this modified dataset* `Dataset2`*.*

Exercise 4. *Pick four random samples from four different classes of* `Dataset1` *and plot them in NetworkX. Use the node features (3D point coordinates) as node labels in NetworkX to make sure you can visualize the object each graph corresponds to.*

3 GNNs and Expressivity

Different GNN architectures have varying levels of expressivity, which influences their ability to distinguish graph structures and generalize across tasks.

We will implement and compare four GNN architectures: graph convolutional network (GCN); ChebNet; GraphSAGE; and graph isomorphism network (GIN). A key question is how well these architectures leverage non-anonymous node features (i.e., original 3D coordinates of point clouds, `Dataset1`) versus using only structural information (`Dataset2`).

Exercise 5. *Define and instantiate four types of 3-layer PyTorch Geometric GNNs, each with the following convolution/message-passing scheme:*

- *GCN*
- *ChebNet with $K = 3$*
- *GraphSAGE with max aggregation*
- *GIN with sum aggregation*

All GNNs should have $d_1 = d_2 = 32$, ReLU nonlinearities in the first two layers, sum readout, and softmax or log softmax nonlinearity in the final layer.

Exercise 6. *Using the* `DataLoader` *class, divide the graphs in the training split of* `Dataset1` *into random batches with 8 graphs each. Set aside 10% of the training samples for validation.*

Exercise 7. *Using the data loader from Exercise 6, train the models from Exercise 5 using Adam with default forgetting factors. Plot the training loss and validation accuracy versus the number of epochs.*

Hint: If you'd like, you can use a `NeighborLoader` within each batch of the data loader to reduce the complexity of the gradient updates[1].

Exercise 8. *Report the classification accuracy of each model trained in Exercise 7 in the test set of* `Dataset1`.

Exercise 9. *Using the* `DataLoader` *class, divide the graphs in the training split of* `Dataset2` *into random batches with* 8 *graphs each. Set aside 10% of the training samples for validation.*

Exercise 10. *Using the data loader from Exercise 9, train the models from Exercise 5 using Adam with default forgetting factors. Plot the training loss and validation accuracy versus the number of epochs.*

Hint: If you'd like, you can use a `NeighborLoader` within each batch of the data loader to reduce the complexity of the gradient updates.

Exercise 11. *Report the classification accuracy of each model trained in Exercise 10 in the test set of* `Dataset2`.

Exercise 12. *Compare the performance of different GNN architectures on the dataset with node features versus the dataset without node features. Which model is the most expressive? How do node features impact classification accuracy? Discuss any unexpected results.*

4 Stability of Convolutional GNNs

Stability refers to the robustness of GNNs against perturbations in graph structure. Given that FAUST consists of 3D point clouds, small variations in node coordinates should ideally not lead to large deviations in classification performance.

To study stability, we will investigate the following factors:

- Effect of increasing layers
- Effect of nonlinearities
- Effect of bounded Lipschitz constant

We will then measure model accuracy under these perturbations to understand which architectures and training strategies improve robustness.

[1] The `DataLoader` class groups multiple graphs into one batch by defining a single `Data` object per batch corresponding to one graph with `batch_size` connected components. Hence, you can apply `NeighborLoader` directly to this "batch graph".

Exercise 13. *Instantiate four ChebNet models with $K = 3$ and the following hyperparameters:*

- $L = 2$, *no nonlinearity,* $d_1 = 32$ *(*`Model1`*)*
- $L = 2$, *ReLU nonlinearity,* $d_1 = 32$ *(*`Model2`*)*
- $L = 3$, *no nonlinearities,* $d_1 = d_2 = 32$ *(*`Model3`*)*
- $L = 3$, *ReLU nonlinearities,* $d_1 = d_2 = 32$ *(*`Model4`*)*

All GNNs should have sum readout and softmax or log softmax nonlinearity in the final layer. Train all GNNs on `Dataset1`*.*

Exercise 14. *Create three perturbations of FAUST by perturbing the coordinates of the point clouds in the test set with a Gaussian with mean ϵ and variance 2ϵ:*

- $\epsilon = 0.002$ *(*`TestData3`*)*
- $\epsilon = 0.004$ *(*`TestData4`*)*
- $\epsilon = 0.006$ *(*`TestData5`*)*

Plot the test accuracy on `TestData3` *through* `TestData5` *versus ϵ for all models in Exercise 13. Use the same color for models with same L, solid lines for models with nonlinearities and dashed lines for models without. Include a legend. What do you observe?*

Exercise 15. *Modify your loss function to penalize high Lipschitz constants, i.e., add a penalty term corresponding to the maximum weight in the model. Using this loss function, instantiate and train two models* `Model2'` *and* `Model4'` *with the same parameters as* `Model2` *and* `Model4` *respectively.*

Exercise 16. *Plot the test accuracy of* `Model2`*,* `Model4`*,* `Model2'` *and* `Model4'` *on* `TestData3` *through* `TestData5` *versus ϵ. Include a legend. What do you observe?*