Assignment 2

Deadline: 11:59 PM 22 March 2023

Important Details

- This is a **group** assignment.
- You have to submit your code in a PRIVATE GitHub repository and share the GitHub link in Moodle submission form. You need to add 'sahilm1992' and 'mrdlgpt' as a collaborator to your GitHub repository. If your submission is modified post due-date, the submission will not be evaluated.
- You have to provide a file named Readme in your home folder containing instructions to execute your code/install dependencies.
- **For all your experiments use the same machine.**
- Plagiarism results in F grade.
- The deadline is strict. No extensions (even with penalty) will be given.

1 Assignment

In this assignment, you will benchmark architectures discussed in the class (links to their codes in section 4) against GCN [1], GraphSAGE [2], GIN [3], or GAT [4] (the one you picked in the form, see section 4 for clarity). To benchmark these architectures, you are supposed to perform the following experiments.

Node Classification – given a graph $G = (V, E)$, the task of node classification is the task of assigning labels l_v to each node $v \in V$ from a predefined label set L (that is $l_v \in L$). The function $f : V \rightarrow L$ that maps $v \mapsto l_v$ will be learned using a GNN. Further, v may have features represented by a vector $\mathbf{x}_v$. The GNN component of f will then take these features as input. The output of the GNN for each node can be projected using an MLP followed by a Softmax to get a probability distribution $p_i^v, \forall i \in L$ for each node v. Finally, the Cross Entropy Loss[1] for each node will then be calculated as

$$\mathcal{L}(v) = -\sum_{i=1}^{|L|} y_i^v \log(p_i^v)$$

$$\text{where } y_i^v = \begin{cases} 1, & \text{gold label} = i \\ 0, & \text{otherwise} \end{cases}$$

$$f(v) = \underset{i=\{1,\ldots,|L|\}}{\arg\max}\ p_i^v$$

Graph Similarity Learning – the task is to learn a function h that takes two graphs G_1 and G_2 and gives a score that answers *how similar are the two graphs G_1 and G_2?* The neural parameterization of the function h will take vector representations of graphs G_1 and G_2. These representations (embeddings) can be computed by first having a GNN compute representations of every node

[1] In PyTorch, Softmax is followed by NLLLoss, while CrossEntropyLoss accepts input without Softmax applied on it.

in the graph and then summing the representations of each node (feature-wise) to obtain graph representation.

$$\mathbf{x}_G = \sum_{v \in V} \text{GNN}(\mathbf{x}_v)$$

The function h will then take a dot product of $\mathbf{x}_{G_1}$ and $\mathbf{x}_{G_2}$ (there can be other variations, for example the sum of the two or concatenation of the two). You will be using dot product in the experiments.

$$h(G_1, G_2) = \text{MLP}(\mathbf{x}_{G_1}^T \mathbf{x}_{G_2})$$

Final output will be a scalar and the loss will be MSE loss.

Graph Classification – the task of graph classification is to learn a function ζ that will take a graph G and map it to a label from a predetermined label space L. This is just an extension of the node classification task. The representation of the graph can again be computed as described above in Graph Similarity task i.e summing representation of all nodes (feature-wise). An MLP followed by Softmax can be used to project this representation to label probability space. Training is performed using Cross Entropy Loss for multi-class problems. However, unlike in node classification, here the loss will be with respect to each graph.

$$\mathcal{L}(G) = -\sum_{i=1}^{|L|} y_i^G \log(p_i^G)$$

where p_i^G is the probability predicted by the neural network that graph G has label $i \in L$.

$$\zeta(G) = \underset{i=\{1,\ldots,|L|\}}{\arg\max}\ p_i^G$$

In all the above tasks, training and testing should be performed on disjoint sets and the final evaluation score assigned to the method should be the evaluation metric on the test set. For example, in node classification, train on $V^{tr} \subset V$, use $V^{val} \subset V$ and $V^{val} \cap V^{tr} = \varnothing$ for model selection, while evaluate on $V - (V^{tr} \cup V^{val})$. For graph similarity learning, evaluate on graphs (graph pairs) not seen during training. Likewise, for graph classification, evaluate on graphs that are not part of the training set. During training, the losses should be computed and aggregated only on the training set. For choosing the best model, you will be using early stopping with a patience of 30 epochs, that is you will use the model with the lowest validation loss and stop training if the loss doesn't improve beyond the best validation loss for 30 consecutive epochs.

1.1 Datasets and Splitting

For the given datasets we use the default train-val-test split as provided by PyTorch Geometric loaders. For each of the task described above, you shall use the datasets listed below.

- **Node Classification** – Wisconsin (link) and Cora (link)
- **Graph Similarity Learning** – AIDS (link) and Linux (link). Note that these datasets do not have a defined validation set, so for AIDS dataset you will be using last 140 (of 560) training graphs as validation and for Linux dataset use last 200 (of 800) training graphs.
- **Graph Classification** – OGBG-PPA (primary link (use `name='ogbg-ppa'`), support link1, support link2)

1.2 Metrics

- **Node classification** and **graph classification** – Macro averaged F1 (use this link).
- **Graph Similarity Learning** – RMSE + F1-score (wiki) on top-K.
  What this means is for any given graph G in the test set, you have to list out K most similar graphs in the training set (K will be given to the code as input, write the code so that evaluation code accepts K as a command line argument) based on the similarity predicted by the model. Call these $G_1^p, \ldots, G_K^p$. Also, there will be top-K graphs from gold values, $G_1^t, \ldots, G_K^t$. Then, the F1-score will be calculated based on precision and recall on the graphs retrieved. RMSE will be calculated as

$$\text{RMSE} = \frac{1}{K}\sqrt{\sum_{i=1}^{K}\left(\widehat{S(G, G_i^p)} - S(G, G_i^p)\right)^2}$$

  where $\widehat{S(G, G_i^p)}$ is similarity between G and G_i^p predicted by model, while $S(G, G_i^p)$ is the true similarity between G and G_i^p. Use $K = 5$ to make plots in your report.

1.3 Experiments

Perform experiments to get the following and accompany them with suitable write-up. Write-up should be informative, clear and concise.

1. Performance based on metrics defined above for each task on its corresponding datasets. Create a results table.
2. Generate plots for (a) Performance vs epochs, (b) Performance vs training time, (c) Training Loss and Validation Loss vs epoch, (d) Training loss and Validation Loss vs training time for each task and corresponding datasets. For the performance/loss vs epoch graph, you may plot at every 5^{th} epoch.
3. Plots for (a) Performance vs volume of training data, (b) Training and validation Loss vs volume of training data for each task for its corresponding datasets. The bigger dataset should be a superset of the smaller dataset. For node and graph classification task, each training data size should maintain the original class distribution. **We will share a script for this.** For the performance/loss vs epoch graph, you may plot at every 5^{th} epoch.
4. Design your own experiments that you think is relevant for your architecture. Think about hyper-parameters, design assumptions, etc.

2 Evaluation

Evaluation will be based upon experiments conducted by you and the write up. Primarily

1. Whether you were able to properly adapt the code for various tasks with proper loss function and with correct datasets.
2. On the presentation of experiments and conclusions: clarity, how easy is it to understand, does it make sense, do conclusions follow logically from experiments, are proper tables/figures used to support the conclusions.

Rough Breakup of marks:

1. 21% for results obtained for the 3 different tasks ($7 + 7 + 7$).
2. 27% for designing own experiments relevant for your chosen architectures.

3. 12% for the varying training data experiment.
4. 25% for the plots of performance/loss vs epochs, training time etc.
5. 15% for correctly integrating code. Note that, if plots/results are correct but if code to generate the result is incorrect, it would lead to penalty in the results component too. Specifically, results which are not reproducible will not be given marks.

Note that write ups and conclusion are part of the experiments and will be weighted accordingly. Further, your code should generate plots/results when executed.

3 Submission instructions

Example directory structure of your submission

```
.
|-- run.sh
|-- EntryNumber-writeup.pdf
+-- other_parts_of_code
```

The run.sh files should contain commands to execute your code. Depending upon each experiment your submission needs to provide the following functionality.

`bash run.sh <task> dataset-name`

where `<task>` may be one of `NC` (Node Classification), `GSL` (Graph Similarity Learning), `GC` (Graph Classification). You can generate multiple `.sh` files based upon the experiments.

Kindly write the commands in Readme file with what the command is supposed to do. No marks will be awarded for an experiment for which its corresponding command with arguments is not mentioned in the Readme. Note that significant deviation between results in report and results generated from code might face penalty.

You need to provide the GitHub url to your code in a form we will share with you.Apart from code files, you need to submit a file named **EntryNumber-writeup.pdf** in the home folder of your submission. The file should contain the relevant writeup, figures/plots and tables as described in section 2.

4 GNN Architecture Codebases

Here, we list the links to the codebases of GNN architectures to be benchmarked. Feel free to take inspiration or adapt segments from codes that aren't your chosen architecture.

- Graphormer [5]
- GraphGPS [6]
- SpeqNets [7]
- ESAN (DSS-GNN) [8]
- NAGphormer [9]

You'll choose one from each column in the table below (already specified in the form) and compare them against each other on above tasks. For the Old architectures, you can use code from PyTorch geometric. Kindly use reasonable set parameters and hyperparameters. Using unreasonable set hyper-parameters and getting very different performance/incorrect conclusions cannot be justified.

New	Old
Graphormer	GCN
GraphGPS	GraphSAGE
SpeqNets	GIN
ESAN (DSS-GNN)	GAT
NAGphormer	

Note: You are free to add as many files as you want in the submission. However, kindly limit your submissions to $\leq$ 50MB.

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2019.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*.

[5] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28877–28888, 2021.

[6] L. Rampasek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," in *Advances in Neural Information Processing Systems*.

[7] C. Morris, G. Rattan, S. Kiefer, and S. Ravanbakhsh, "Speqnets: Sparsity-aware permutation-equivariant graph networks," in *International Conference on Machine Learning*, pp. 16017–16042, PMLR, 2022.

[8] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, "Equivariant subgraph aggregation networks," *arXiv preprint arXiv:2110.02910*, 2021.

[9] J. Chen, K. Gao, G. Li, and K. He, "Nagphormer: A tokenized graph transformer for node classification in large graphs," in *International Conference on Learning Representations*.