Programming Assignment 3

*Tentative Deadline: **25 November**

Part 1:

Link Prediction and Retrieval on Graphs

(Doubts fired to: Pritish)

In this part of the assignment, we will explore link prediction on graphs. During training, we will test out various graph neural network schemes, and during inference, we will attempt fast top-K retrieval using neural LSH (you have seen this in action on images already). See the paper for a refresher.

Datasets: You may experiment on CoraFull and DeezerEurope.

GNN Training: We wish to try out various GNNs for this task, and find the best performing one. Experiment on the following models - **GCN, GAT and GIN** - and record your results. Split the edges of the graph into train, val and test using a split of 60:20:20. During training, you can pass the *subgraph induced* (torch-geometric has functions for these) by all the edges and nodes in the trainset as input to your model. You can use the loss defined in equation (9) of the paper to train your model, where each π_u embedding is replaced by your chosen GNN node embedding, and the scoring function is cosine similarity.
Note that you will require positive (connected by an edge) and negative (not connected) node pairs for the purpose of training - please see this doc. Additionally, in the test set, find and save a set Q of 1000 nodes such that each node is part of atleast one triangle. We will use these during inference.

GNN Inference: If trained properly, your system should score node pairs which are edges higher than those that are non-edges. Given the test portion of the input graph, your GNN model will predict relevant node embeddings. Using these, your job is to retrieve the top-K potential candidate node pairs for each node q in the set Q. So you'd have to pair q with each other node in the test set and rank the results.

Report the Precision@K (K = 1, 5, 10) and MRR (no K required) as metrics on this task for the following models and methods - GCN, GAT, GIN, Adamic Adar and Common Neighbours, where the last two are non-trainable (i.e., just test them using the Q set). Note that cosine sim is the scoring function of choice for the first three, and the last two are scores in themselves.

Faster Inference?: We would like to use both random and neural LSH (as done in the paper) to speed up GNN inference. Use your knowledge from the image retrieval assignment to implement LSH based retrieval for graphs. Take a look at equation (14) in the paper for the purpose of training your neural LSH model, and appendix B for the LSH recipe (also explained in previous tutorial). Note that for a node pair to be in a bucket, it must've had a high cosine similarity score. Experiment for top-K retrieval where K = 1, 5, 10 as given above. Report the time taken by plain GNN inference and LSH based inference, and compare the metrics induced by both.

Part 2:

Zero Shot Retrieval

(Doubts fired to: Prayas)

Important: Read footnote below

Zero-shot retrieval refers to the ability of a model to retrieve relevant information or documents without any task-specific training on labelled examples. Instead, it leverages pre-trained knowledge to interpret and respond to novel queries or domains.
This part of the assignment is related to zero-shot retrieval. Consider BERT, which is trained on the BookCorpus and Wikipedia. Although Wikipedia has substantial scientific information, BERTs performance suffers on scientific retrieval tasks, possibly due to filtering of such data while pre-training, or such data being small as compared to general english text (which is more likely). We shall try a few modifications in an attempt to improve zero-shot performance on BERT. Therefore, you do not need extensive finetuning for this part of the assignment. This part of the assignment is fairly easy and short **if** it's properly understood.

Datasets: **SciDocs**, (If the full dataset is taking too much time, try the reranking dataset Scidocs-Rerank)

Modification 1: The usual inference pipeline goes as follows:

1.) Prepare an input example **x** to put in the bert context window.
2.) Forward pass **x** and take the CLS output **y_cls.**
3.) Project **y_cls** to a real score by some Feed-Forward transformations.

Inspired by Test-time training, consider the following **inference** pipeline:

1.) Prepare an input example **x_i** to put in the bert context window.
2.) Consider the original bert parameters as **M_orig**.
3.) Do one step of masked reconstruction of **x_i** on BERT (reminder, this is inference time)**,** to get the updated model **M_xi.**
4.) Same as 3.) above.
5.) Restore the model back to **M_orig.**

Therefore, during inference, for each example **x_i**, the model first updates its parameters based solely on that example. It then proceeds with the usual inference process. After evaluating **x_i**, the model is reset to its original state.

However, before you do all of this, you need the parameters of the FFL layer after CLS (they do not come with bert). Use HotpotQA. You **do not** need to use the entire dataset, use a fraction or until training converges. Feel free to freeze the rest of the model to speed up training. The whole process should not take more than 2-3 hours if done properly.

Finally, note that hugging-face already provides all the code for masked reconstruction (masked language modelling). You just need to make sure you restore model parameters(as mentioned in 5. above), after every evaluation step.

Modification 2: Note that we restore the model at every inference step. What if the restore granularity is coarser ? Experiment with K=8,16,64 where K is the number of training examples when the model is updated and restored. For the above K is 1 (we restore and

update at every example). Thus for K=8, the model is updated for 8 consecutive examples, and restored to its original state only after the 8th example.
Caution: DO NOT finetune on SciDocs. Remember, this is a zero-shot retrieval task.

FootNote: Does it make sense to reconstruct on query + doc or just doc ? Explain in 1 or 2 lines in your report. Reconstructing on either will be considered for credit, but one is **expected** to perform worse. Which one ?

Extra credit (Optional):
Along these lines, do you have any other suggestions to improve Zero-shot retrieval performance ? You only need to provide a (short) rough sketch for your idea. Preliminary results would be awesome, but not required. Extra points will only be awarded if you can defend your proposal during evaluation, and should be non-trivial. The ideal (short) proposal is backed by citations to other prior art. (Encourage to do this, but not a necessity)
Some helpful thoughts:

1.) The SciDocs dataset can be cast as a Citation Graph. Does this help us ?
2.) Why is zero-shot retrieval so much worse in the first place ?
3.) Why is test-time training **expected** to improve performance ? Any analogies to how we humans might perform the same task
4.) How would we tackle a retrieval task for which we have no knowledge about ? For example a CS student asked to rank webpages in the Medical domain.

Feel free to ponder upon the (ramblings) above.

To submit: Compare performance on SciDocs, without the modified inference vs the new method. Report Precision@10, Recall@10, MRR. Explain your results.

General Hints:
You may use either Google Collab or Kaggle. However Kaggle has a more generous 30 hr per week GPU quota, so please make use of it. Don't forget to terminate GPU usage when you don't need it.

Submission Protocol:
Moodle will be used for the submission of the assignment.

- All the findings need to be put down into a report file.
- Zip all the files used for building/training/testing of the models along with the report into a single file and then submit it on Moodle. Name the compressed file as [Roll1_roll2.(zip/tar.gz)]

Note - Only one member of the team should submit the assignment on Moodle.