Advanced Machine Learning for Graphs and Text (ALTEGRAD) Data Challenge 2018-19

Data Science and Mining Team (DaSciM)
Ecole Polytechnique

February 1, 2019

1 Introduction

A network (or graph) can be represented as a document whose words are the nodes of the network and whose sentences are random walks sampled from the network. Thus, Deep Learning architectures developed in the NLP domain can be applied to graph problems.

In this competition, you will have to solve a multi-target *graph regression* problem with a Deep Learning architecture for NLP, the Hierarchical Attention Network (HAN). This architecture was covered in depth during the fifth lab of the course (18/12/2018). Refer to this lab's material for more details and relevant references.

2 Kaggle

This competition is hosted by the Kaggle in-class platform: https://www.kaggle.com/c/altegrad-19/overview. All general questions **must be asked on the Kaggle forum of the competition**: https://www.kaggle.com/c/altegrad-19/discussion.

3 Data

The dataset is made of 93,719 undirected, unweighted graphs whose edgelists can be found inside the `/data/edge_lists/` subdirectory. Each node of each graph is associated with a unique ID which corresponds to the row index of its 13-dimensional attribute vector in the `/data/embedding.npy` file. These vectors are equivalent to the input word vectors in Deep Learning for NLP, except that fine-tuning them is not necessary. Also, each node is unique, whereas in NLP, words are shared among documents. Note, however, that some nodes may have very similar vectors.

Each graph is associated with a continuous value for each of 4 target variables. For the graphs in the training set, these values are stored in 4 files (one file for each target variable) that can be found inside the `/data/target/train/` subdirectory. Your task is to predict the 4 values associated with each graph in the test set.

There are 74,975 graphs in the training set and 18,744 graphs in the test set. The indexes of the graphs in each set can be found in the `/data/train_idxs.txt` and `/data/test_idxs.txt` files.

4 Rules

Please read carefully and follow the rules below. Teams breaching the rules will get penalized proportionally.

- **Architecture**. You may **only use the HAN architecture** for this competition. Of course, your are allowed to modify/improve the different components of this architecture, or to add new components to it (see Section 7). Using any other standalone Deep Learning architecture (e.g., CNN) or predictive model (XGBoost, Random Forest, etc.), either on its own, or as part of an ensemble with HAN, is **forbidden**.
- **Team size** limit is 3 students. This limit is strict, no exceptions will be made. Make sure you have at least one GPU per team. If you don't have access to a GPU, consider for instance using Google colab, which gives you access to a high-end GPU for up to 12 hours at a time.
- **Final submissions** must be uploaded here: `https://goo.gl/forms/uVdiRKgwQEVFhTsp2` as a single archive containing your code and a PDF report. Upload only **one submission per team**. Do not upload the original data. Make sure the name of your team **as it appears on the Kaggle leaderboard** and the **names of all team members** appear on the first page of the report.
- **Reproducibility**. Your code must allow the jury to reproduce your Kaggle submission.
- **Report format**. PDF reports **must follow the IJCAI format**: `https://www.ijcai.org/authors_kit`. In particular, they must have a maximum of six pages, plus at most one for references.
- **Deadline.** All submissions must be uploaded by **Sunday, March 31st, at midnight Paris time**. *No extension will be granted.*
- **Questions**. All general questions about the competition **must be asked on the Kaggle forum of the competition**: `https://www.kaggle.com/c/altegrad-19/discussion`

5 Baselines

The `networkX`, `Keras` (with `tensorflow` backend) and `scikit-learn` libraries are required. You are provided with two baseline scripts: one **that should be run first**, for creating the document representations of the networks (`preprocessing_baseline.py`) and one for training the basic HAN architecture (`main_baseline.py`). `main_baseline.py` can be run on CPU by setting `is_GPU` to `False` at the top of the script, but we strongly encourage you to use a GPU. Finally, use the `read_results_predict.py` script to generate predictions in a format suitable to be uploaded to Kaggle. The metric that is used to evaluate submissions is the Mean Squared Error (MSE) as described here: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html`.

Note: the same overall architecture should be used for all 4 targets. However, better results are to be expected when hyper-parameters are optimized separately for each target.

6 Grading scheme

Grading will be on 100 points total:

- **30 points** will be allocated based on the raw Kaggle performance, provided that the submission is reproducible. That is, using only your code and the data provided on the competition page, the jury should be able to train your final model and use it to generate the predictions you submitted for scoring.
- The content of the report will be worth **60 points**. Regardless of the performance achieved on Kaggle, the jury intends here to reward the research efforts, creativity and experimental rigor put into your work. Best submissions will propose innovative improvements to the HAN architecture, clearly explain them, and report about their impact on performance (e.g., through ablation studies).
- Finally, **10 points** will be awarded to the organization and readability of the code and report. Best submissions will (1) clearly deliver the solution, providing detailed explanations of each step, (2) provide clear, well organized and commented code, (3) refer to relevant research papers.

7 Research directions

The list below is just to get you started, and definitely not exhaustive. In order to find good ideas, we strongly encourage you to make use of the concepts learned during the course (and especially, during the lab sessions on text and graph mining), and to review the literature.

Sampling

- the number of walks per node and the length of each walk could be tuned; also, using random numbers (within reasonable bounds) rather than fixed numbers could be useful,
- the walks could be biased like in `node2vec`.

Enriching node attribute vectors

- relabeling procedures such as the Weisfeiler-Lehman could be used,
- `DeepWalk` or `node2vec` could be used.

Architecture

- it could improve performance to have multiple context vectors both at the sentence and document level,
- **context-aware self-attention**: currently, each sentence is encoded in isolation, which is suboptimal. Encoding the sentences while taking into account the information that has been captured from the other sentences would be better.
- experimenting with skip-connections could be useful.

Training

- use a different optimizer than adam (e.g., SGD),
- sanity-check the learning process, checking for underfitting/overfitting[1],
- carefully optimize the regularization strategy for each target (initial learning rate, learning rate schedule, batch size, dropout, etc.)

[1]see, e.g., http://cs231n.github.io/neural-networks-3/#baby