GDM: Graph Dismantling with Machine learning

This repository contains the scripts and data from the "Machine learning dismantling and early-warning signals of disintegration in complex systems" paper by M. Grassia, M. De Domenico and G. Mangioni.

TODO: ADD PAPER LINK AND BIBTEX CITATION

Setup

Expected time for this step: 30m

Our test environment is hosted on a machine running Ubuntu 16.04 and equipped with a nVidia K80 (two cores with 12GB VRAM each). We use nVidia driver version 410.104 and CUDA version 10.0.

While suggested to improve prediction performance, a GPU is not needed as predicting values on a CPU is still feasible in very reasonable time (e.g., in our environment *hyves* network takes just 3m on CPU vs 1m on the GPU).

After driver and CUDA installation, run the following to check wether they are installed correctly.

```
# GPU drivers and CUDA version:
nvidia-smi
```

The python dependencies for this package can be installed with *conda* using:

```
cd GDM
conda env create -f environment.yml
```

Unfortunately, conda wouldn't let us recreate our exact environment due to some weird behavior with "conflicting packages". If you wish to check our full environment, check the environment_full.yml file.

The command above should create a new conda environment ("ndml") that can be activated with

```
conda activate ndml
```

After activating the environment, check whether everything is installed correctly as follows:

```
# Check if pytorch is installed
python -c "import torch; print(torch.__version__)"

# Check if cuda is recognized by pytorch
python -c "import torch; print(torch.cuda.is_available())"

# Check which CUDA version pytorch is using
python -c "import torch; print(torch.version.cuda)"
```

We refer to pytorch geometric's full documentation for any issue during the installation phase or for OS specific steps.

Please note that newer versions of pytorch_geometric introduce changes to the GAT layers that break compatibility with our models, even though no error is shown.

Before installing our package, ensure that you have g++ installed in order to compile our external dismantler. In our environment, we tested using g++-9.

Lastly, install our package with pip:

```
cd GDM
pip install -e . -vv
```

The dismantler build command is run during this step, so no further action should be required. If you get any compilation error message, you can manually check if the dependencies for the Tessil's robin-map are met by running the official test scripts as follows.

```
git clone https://github.com/Tessil/robin-map.git
cd robin-map
git reset --hard 5c6b2e5c55239999f989e996bcede0e1f46056f7
cd tests
mkdir build
cd build
cmake ..
cmake --build .
./tsl_robin_map_tests
```

The robin-map is a data-structure our dismantler depends on and we redistribute it with our code. If you wish to check them out, keep in mind that in our tests we use revision *5c6b2e5c55239999f989e996bcede0e1f46056f7*. For any issue with the robin-map we refer to the official GitHub page linked above.

To manually compile the dismantler (e.g., build failed during the pip install)

```
cd network_dismantling/common/external_dismantlers/
make clean
make
```

Uncompress the dataset

Due to GitHub file size limitations, we had to compress the dataset and split the files into smaller ones. You should be able to extract the files opening the

dataset.tar.gz.aa

file with your compression tool GUI (like Ark).

If that doesn't work, the following command should do the trick:

```
cd dataset
cat dataset.tar.gz.* | tar xzvf -
```

Using the scripts

Every runnable script in this package comes with an *ArgParse* interface that provides an help command (–help flag) For instance:

```
python
    network_dismantling/machine_learning/pytorch/reproduce_results.py
    --help
```

Reproducing our results

After the setup detailed above, you can reproduce the results in the paper using the scripts detailed in this section. The base script (reproduce_results.py) and its variations will run the experiments and store the run info (i.e., the resulting AUC, the removals list that include the removed node, the resulting LCC and SLCC size for each removal, etc.) into the a new file that can be used for plotting.

Dismantling results

```
# If you wish to reproduce the results on medium/small networks
python -u
    network_dismantling/machine_learning/pytorch/reproduce_results.py
    -lt dataset/test/dataset/

# If you wish to reproduce the results on large networks
python -u
    network_dismantling/machine_learning/pytorch/reproduce_results.py
    -lt dataset/test_large/dataset/

# If you wish to reproduce the results on synthetic networks
python -u
    network_dismantling/machine_learning/pytorch/reproduce_results.py
    -lt dataset/test_synth/dataset/ --file
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model.SYNTH.csv
```

If the run file is not found, you can specify its location with the –file flag. Keep in mind that the script requires the same folder structure that comes with the data to get some information about the models to use.

You can specify how many jobs to run in parallel with the –jobs (-j) flag and how many processes should access each of your GPUs simultaneously with the –simultaneous_access (-sa) flag. The GPU is used only to predict the dismantling order and will be freed as soon as the predictions are computed, so it makes perfect sense to limit the simultaneous access to avoid Out of Memory (OOM) issues while pushing on parallelism.

The expected run-time really depends on your hardware configuration and on the number of jobs you use. For all the small/medium size networks, however, it should be quite fast (a few minutes at most). The large networks can take more time (check the table in the paper to get an idea) and also keep in mind that, due to their size, parallelism and simultaneous access should be reduced to avoid OOM issues.

The reinsertion phase (i.e., GDM+R) can be performed as described below in Reinsert nodes

Reinsert nodes

Given a .csv run file, you can reintroduce the nodes (i.e., get the GDM+R results) with the following:

```
python -u
    network_dismantling/machine_learning/pytorch/reinsert.py -lt
    <NETWORKS_FOLDER> [-f <CSV_RUN_FILE>]

# Examples:
# The default -f (--file) value is
    "out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv",
    i.e., the reproduced results obtained with the steps above.

# If you wish to reintroduce nodes on medium/small networks
python -u
    network_dismantling/machine_learning/pytorch/reinsert.py -lt
    dataset/test/dataset/

# If you wish to reintroduce nodes on large networks
python -u
    network_dismantling/machine_learning/pytorch/reinsert.py -lt
    dataset/test_large/dataset/

# If you wish to reintroduce nodes on synthetic networks
python -u
    network_dismantling/machine_learning/pytorch/reinsert.py -lt
    dataset/test_synth/dataset/ --file
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.SYNTH.csv
```

This step can take quite some time for large networks. We have not optimized or changed the original code from Braunstein et al. to avoid any bias.

EarlyWarning

You can prepare the EarlyWarning signal for plotting with the following command:

```
python -u
    network_dismantling/machine_learning/pytorch/reproduce_results_ew.py
    -f out/df/synth_train_NEW/t_0.18/EW/GAT_Model.csv -lt
    ../dataset/test_grid_ew/dataset/
```

This script will run the dismantling from scratch and store the predictions (p__n) for every node in the output file.

Running a grid search

If you prefer running a grid search (and/or training the models), you can use the *grid.py* script. This script will perform a grid search on all the combinations of input parameters.

If the model is found (i.e., it is already trained and stored in the folders), it won't be trained again. Similarly, if a dismantling is found (same network and parameters), the dismantling won't be run again. On our (shared) hardware, training a model takes about ~5m.

Usage:

```
cd GDM
python -u network_dismantling/machine_learning/pytorch/grid.py
    -lm <TRAIN_NETWORKS_LOCATION> -lt <TEST_NETWORKS_LOCATION>
    [-Ft "<GLOB_STYLE_FILTER>"] -e <NUM_EPOCHS> -wd
    <WEIGHT_DECAY> -r <LEARNING_RATE> -CL
    <CONVOLUTIONAL_LAYER_SIZES> -H <HEADS> -FCL <MPL_LAYERS>
    --features <NODE_FEATURES> -mf <MIN_NODE_FEATURES_NUM> -MF
    <MAX_NUM_FEATURES_NUM> -t <LEARNING_TARGET> -T
    <DISMANTLING_TARGET_SIZE> -j <MAX_JOBS>
    --simultaneous_access <MAX_JOBS_SIMOULTANEOUSLY_ON_GPU>
    [--force_cpu]

# Example:
python -u network_dismantling/machine_learning/pytorch/grid.py
    -lm dataset/synth_train_NEW/dataset/ -lt
    dataset/test/dataset/ -Ft "*" -e 50 -wd 1e-5 -r 0.003 -CL 40
    30 20 10 -CL 30 20 -CL 40 30 -H 1 1 1 1 -H 5 5 -H 10 10 -FCL
    100 100 --features degree clustering_coefficient kcore
    chi_degree -mf 4 -MF 4 -t "t_0.18" -T 0.10 -j 8
    --simultaneous_access 4
```

The dismantling runs will be stored in "out/df/synth__train__NEW/t__0.18/T__0.1/GAT__Model__GRID.csv" while the prediction and dismantling time will be saved in "out/df/synth__train__NEW/t__0.18/T__0.1/GAT__Model__GRID.time.csv".

Dataset generation

We provide the networks we use in our experiments in the repository under the dataset/ folder[1]. However, if you wish to test with your own networks, you can convert them from the edgelist (.el) to the GraphML (.graphml) fileformat via the command:

```
python -u network_dismantling/common/any2graphml.py -i
    <INPUT_FOLDER> -o <OUTPUT_FOLDER> -e <INPUT_FORMAT>

# Example
python -u network_dismantling/common/any2graphml.py -i
    dataset/MY_FOLDER/ -o dataset/MY_FOLDER/dataset/ -e el
```

After converting (or, if you already have networks in such format) you can add the node features and other info required via:

```
python -u
    network_dismantling/machine_learning/pytorch/dataset_generator.py
    -d <GRAPHML_FOLDER> -uf -j <NUM_JOBS> [-F
    <GLOB_STYLE_FILTER>]

# Example
python -u
    network_dismantling/machine_learning/pytorch/dataset_generator.py
    -d dataset/MY_FOLDER/dataset/ -uf -j 4 -F "*"
```

[1]: The networks come from the Konect and NetworkRepository repositories. We refer the reader to the paper for the proper references and citations of each network.

Plotting

Barplots

The barplot script will produce the .pdf output along with a table in .csv and .tex format.

It is worth mentioning that the -P flag (–pivot) will pivots the table.

The following examples will point to your locally reproduced results (i.e., the "out-/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv" file). However, if you don't wish to use that but the data in the repository, you can replace the –file flag with >–file out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model.csv

Real-world networks

For small and medium size networks:

```
# No reinsertion
python -u
    network_dismantling/machine_learning/pytorch/barplot_output.py
    -f
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv
    -qp 'heuristic in ["EGND", "GND", "MS", "EI_s1"] or
    (heuristic=="degree" and static==False)' -po
    out/plot/synth_train_NEW/t_0.18/T_0.1/barplots/no_reinsertion/
    -P

# With reinsertion
python -u
    network_dismantling/machine_learning/pytorch/barplot_output.py
    -f
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv
    -qp 'heuristic in ["CoreHD", "GNDR", "MSR"] or
    (heuristic=="collective_influence_ell2" and static==False)'
    -q 'network in @rw_test_networks' -po
    out/plot/synth_train_NEW/t_0.18/T_0.1/barplots/reinsertion/
    -fr -P

# Full table
python -u
    network_dismantling/machine_learning/pytorch/barplot_output.py
    -f
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv
    -qp '(static==True and heuristic!="random" and
    ~heuristic.str.contains("_ell")) or (static!=True and
    static!=False) or ((heuristic=="degree" or
    heuristic.str.contains("_ell2")) and static!=True)' -po
    out/plot/synth_train_NEW/t_0.18/T_0.1/barplots/full_table/
    -fr
```

For large networks:

```
# Full table
python -u
    network_dismantling/machine_learning/pytorch/barplot_output.py
    -f
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv
    -qp 'heuristic in ["GND", "MS", "GNDR", "MSR", "CoreHD"]'
    -po out/plot/synth_train_NEW/t_0.18/T_0.1/barplots/large/
    -fr -L -P
```

Again, if you wish to un-pivot the table, you can remove the -P flag.

Synthetic networks

We have a script to generate the synthetic network barplot and table. The usage is nearly identical as the barplot_output.py described above. To reproduce the plots in the paper:

```
# Barplot and table
python -u
    network_dismantling/machine_learning/pytorch/barplot_output_synth.py
    -f
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.SYNTH.csv
    -s "r_auc" -po
    out/plot/synth_train_NEW/t_0.18/T_0.1/barplots/synth/  -qp
    'heuristic!="random"' -fr -P

## Once again, if you don't wish to use your locally reproduced
    results, you can point to the run file that comes with the
    data. E.g.:
python -u
    network_dismantling/machine_learning/pytorch/barplot_output_synth.py
    -f out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model.SYNTH.csv
    -s "r_auc" -po
    out/plot/synth_train_NEW/t_0.18/T_0.1/barplots/synth/  -qp
    'heuristic!="random"' -fr -P
```

If you wish to un-pivot the table, you can remove the -P flag.

Dismantling curves

You can plot the dismantling curves using the grid_output.py script.

The following examples will point to your locally reproduced results (i.e., the "out-/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv" file). However, if you don't wish to use that but the data in the repository, you can replace the –file flag with >–file out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model.csv

```
# Plot small/medium size networks
python
    network_dismantling/machine_learning/pytorch/grid_output.py
    -f
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv
    -sf 1 -p -po
    out/plot/synth_train_NEW/t_0.18/T_0.1/dismantling/ -q
    'network in @rw_test_networks' -qp '(static!=True and
    static!=False and ~heuristic.str.contains("_s2")) or
    (static==False and heuristic in ["degree",
    "collective_influence_ell2"])' -fr

```

```
# Plot large networks
python
    network_dismantling/machine_learning/pytorch/grid_output.py
    -f
    out/df/synth_train_NEW/t_0.18/T_0.1/GAT_Model_REPRODUCE.csv
    -sf 1 -p -po
    out/plot/synth_train_NEW/t_0.18/T_0.1/dismantling/ -q
    'network in @rw_large_test_networks' -qp '(heuristic in
    ["GND", "GNDR", "MS", "MSR", "CoreHD"])' -fr
```

EarlyWarning

Plotting the Early Warning signal curve requires another script. After reproducing the results as detailed in the previous section, you can run:

```
python
    network_dismantling/machine_learning/pytorch/grid_output_pi.py
    -f out/df/synth_train_NEW/t_0.18/EW/GAT_Model_REPRODUCE.csv
    -p -po out/plot/synth_train_NEW/t_0.18/T_0.1/EarlyWarning/
    -qp 'heuristic in ["GNDR", "MSR", "degree", "CoreHD"]'
```

This will create a sub-folder for each network in the output folder (out/plot/synth_train_NEW/t_0.18/T_0.1/EarlyWarning/). The sub-folders contain, for each heuristic attack simulation, a plot (PDF) file, plus a .csv file containing the values of LCC, SLCC and EW (“ew” column) after each removal.