GraphLog

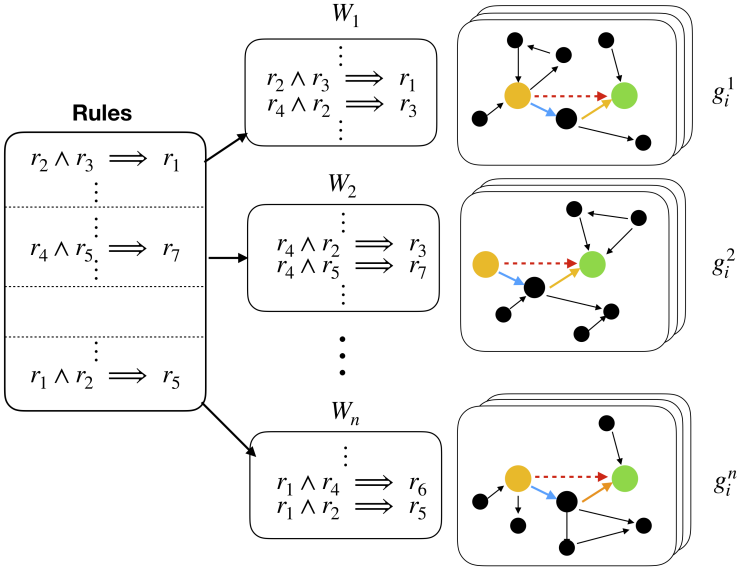
Release 1.0.0

Koustuv Sinha, Shagun Sodhani

May 09, 2020

GETTING STARTED

GraphLog is a multi-purpose, multi-relational graph dataset built using rules grounded in first-order logic. GraphLog can be used to benchmark existing Graph Neural Network (GNN) family of models on relation prediction task.



CHAPTER

ONE

INSTALLATION

Install Pytorch and Pytorch Geometric in your system, according to your system requirements (`cpu` or `cuda`).

1. Install Pytorch
2. Install Pytorch Geometric (and all its dependencies)
3. Install the latest version of GraphLog directly from PyPI:

```
pip install graphlog==1.0.0
```

CHAPTER

TWO

BASIC USAGE

GraphLog can be used as a regular Python module to access the datasets used in the paper *"Evaluating Logical Generalization in Graph Neural Networks"*. Additionally, GraphLog also provides necessary Dataset and DataLoader packages for easy training and evaluation.

2.1 Loading the data

When GraphLog is imported for the first time, it downloads the data and creates a `./data` directory in the current working directory. The downloaded data is then unzipped and placed within this directory.

```
from graphlog import GraphLog
gl = GraphLog()
```

To change the data directory, pass the `data_dir` argument.

```
gl = GraphLog(data_dir='/tmp/data')
```

2.2 Viewing the data

GraphLog consists of multiple `datasets`. Each dataset is built using its own set of **rules**, which themselves are procedurally generated and sampled from a large knowledge-base.

To view all possible datasets in GraphLog:

```
gl.get_dataset_names_by_split()
```

This will provide a list of dataset ids in train, valid and test splits. To load a single dataset, use the `get_dataset_by_name` method:

```
rule_3 = gl.get_dataset_by_name('rule_3')
type(rule_3)

>> graphlog.dataset.GraphLogDataset
```

This will load a `GraphLogDataset` object, which is in-turn a Pytorch `Dataset` instance. Each dataset also has its own training, validation and test splits.

The `GraphLogDataset` object essentially contains Pytorch Geometric graphs, a query tuple of `<source, sink>` nodes for each datapoint, and a label or relation to predict.

You can also view the aggregate statistics of the dataset:

```
gl.compute_stats_by_dataset("rule_3")

>> Data Split : train,
Number of Classes : 16,
Number of Descriptors : 189,
Average Resolution Length : 3.632142857142857,
Average number of nodes : 11.137  and edges : 13.273
```

```
{'num_class': 16,
 'num_des': 189,
 'avg_resolution_length': 3.632142857142857,
 'num_nodes': 11.137,
 'num_edges': 13.273,
 'split': 'train'}
```

You can also convert the dataset into networkx format, in order to perform quick calculations or visualization:

```
import networkx as nx
from graphlog.utils import load_networkx_graphs
nx_graphs, nx_queries = load_networkx_graphs(rule_3.json_graphs["train"])
```

```
nx.info(nx_graphs[0])
```

To view a single graph in the dataset, you can also use the inbuilt `display_single_graph` api.

```
gl.display_single_graph(rule_3, "train",21)
```

2.3 Extracting dataloaders

We provide a method to generate dataloaders for each dataset as follows:

```
rule_3_train = gl.get_dataloader_by_mode(rule_3, mode="train")
rule_3_valid = gl.get_dataloader_by_mode(rule_3, mode="valid")
rule_3_test = gl.get_dataloader_by_mode(rule_3, mode="test")
```

2.4 Supervised Training

A very minimal dummy training script is provided below to show how easy it is to train your models.

```
for batch_idx, batch in enumerate(rule_3_train):
    graphs = batch.graphs
    queries = batch.queries
    targets = batch.targets
    world_graphs = batch.world_graphs
    logits = your_model(graphs, queries, world_graphs)
```

CHAPTER

THREE

ADVANCED USAGE

GraphLog provides an array of datasets, thus making it a perfect candidate to test multi-task, continual, and meta-learning in graphs. Each dataset is derived by its own set of **rules**.

3.1 Similarity

Two datasets can have highly overlapping rules to highly non-overlapping rules. This provides GraphLog a unique way to define the notion of task **similarity**. Two datasets are highly similar if the underlying rules are similar.

```
from graphlog import GraphLog
gl = GraphLog()
```

First, let's get the available datasets in GraphLog

```
datasets = gl.get_dataset_names_by_split()
```

```
datasets["train"][0]

>> 'rule_3'
```

To calculate dataset similarity, we compute the overlap between the actual rules used in the datasets. GraphLog provides an easy API to do so.

```
gl.compute_similarity("rule_0","rule_1")

>> 0.95
```

We see that the datasets `rule_0` and `rule_1` are 95% similar. To get top 10 similar datasets as of `rule_0`, we can call the following method:

```
gl.get_most_similar_datasets("rule_0",10)

>> [('rule_0', 1.0),
 ('rule_1', 0.95),
 ('rule_2', 0.9),
 ('rule_3', 0.85),
 ('rule_4', 0.8),
 ('rule_5', 0.75),
 ('rule_6', 0.7),
 ('rule_7', 0.65),
 ('rule_8', 0.6),
 ('rule_9', 0.55)]
```

3.2 MultiTask training

By providing an easy way to extract datasets and also by grouping them in terms of similarity, we can easily train and in a multi-task scenario. Below we provide a dummy snippet to do so.

```
data_ids = gl.get_most_similar_datasets("rule_0",10)
for epoch in range(100):
    dataset = gl.get_dataset_by_name(random.choice(data_ids))
    train_loader = gl.get_dataloader_by_mode(dataset, "train")
    for batch_id, batch in enumerate(train_loader):
        graphs = batch.graphs
        queries = batch.queries
        labels = batch.targets
        logits = your_model(graphs, queries)
```

3.3 Difficulty

GraphLog also provides an additional option of categorizing each dataset on their relative *difficulty*. We compute difficulty by the scores of supervised learning methods as a proxy. For more details how we label each dataset as per their difficulty, please check out our paper!

We provide additional meta-data to categorize the datasets with respect to their difficulty. To access it, call the following API. This will load the datasets directly in memory.

```
easy_datasets = gl.get_easy_datasets()
moderate_datasets = gl.get_moderate_datasets()
hard_datasets = gl.get_hard_datasets()
```

3.4 Continual Learning

Using any of the above categorizations, GraphLog also provides an option of evaluating models in a continual learning scenario. Here, we provide a simple example to evaluate continual learning on a rolling window of similar datasets, based on overlapping rules. `get_sorted_dataset_ids(mode="train")` API will return the datasets in the order they were created in the paper, which follows a rolling similarity.

```
dataset_names = gl.get_sorted_dataset_ids(mode="train")

for data_id in dataset_names:
    dataset = gl.get_dataset_by_name(data_id)
    for epoch in range(100):
        train_loader = gl.get_dataloader_by_mode(dataset, "train")
        for batch_id, batch in enumerate(train_loader):
            graphs = batch.graphs
            queries = batch.queries
            labels = batch.targets
            logits = your_model(graphs, queries)
```

CHAPTER

FOUR

BLOG

You can read more about GraphLog at our blog post.

CHAPTER

FIVE

PAPER

GraphLog is introduced in the paper "Evaluating Logical Generalization in Graph Neural Networks". If you find our dataset useful, consider citing our work.

```
@article{sinha2020graphlog,
    Author = {Koustuv Sinha and Shagun Sodhani and Joelle Pineau and William L.␣
↪Hamilton},
    Title = {Evaluating Logical Generalization in Graph Neural Networks},
    Year = {2020},
    arxiv = {https://arxiv.org/abs/2003.06560}
}
```

CHAPTER

SIX

COMMUNITY

- If you have questions, open an Issue
- Or, join our Slack channel and post your questions / comments!
- To contribute, open a Pull Request (PR)