Department of Computer Science and Technology

Machine Learning

Homework 2

Sahand Sabour

2020280401

1 DeepWalk

1.1 Motivation

1.2 Methodology

1.3 Experiments

1.4 Personal Opinion

2 (GCN)

2.1 Motivation

2.2 Methodology

2.3 Experiments

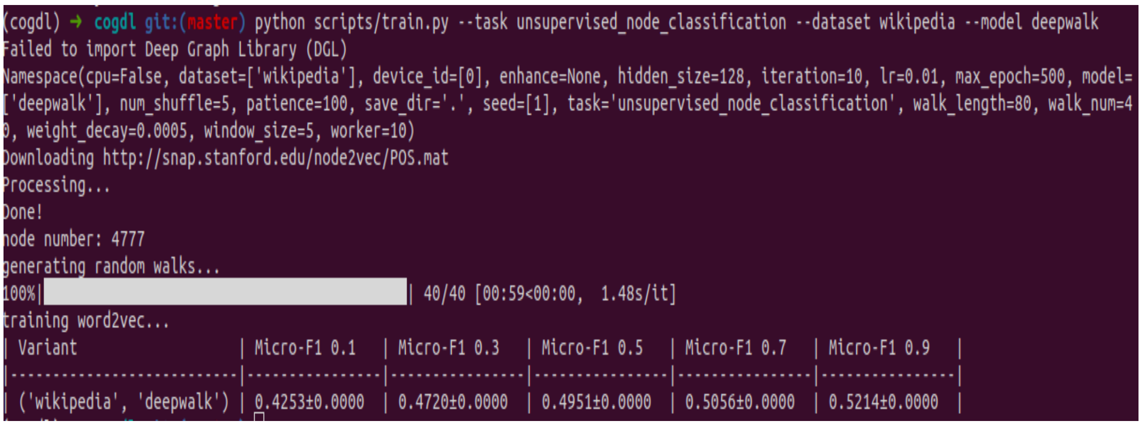
2.4 Personal Opinion

2.5 Comparison with DeepWalk

3 CogDL

3.1 Testing CogDL Models

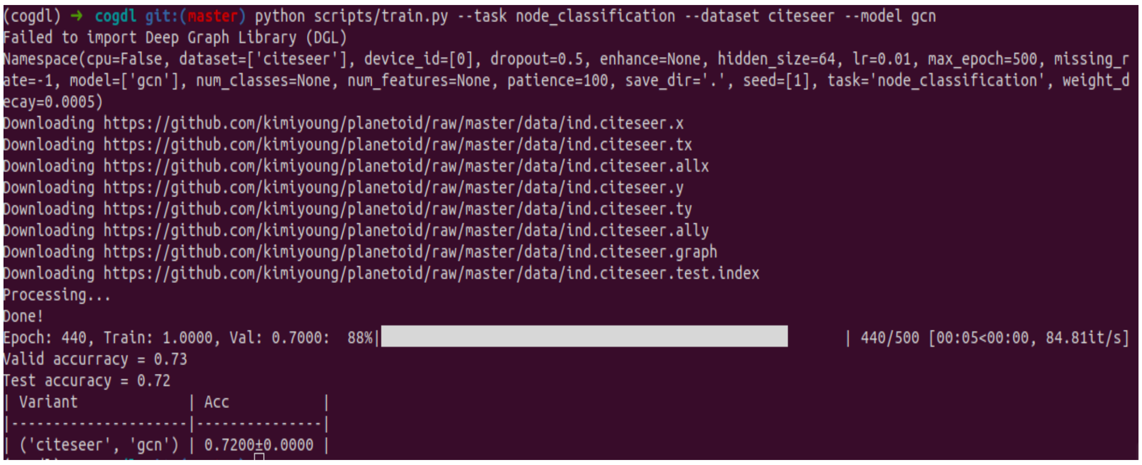
In this assignment, we are asked to run two models on two different datasets by running the provided scripts. The first script is as follows: python scripts/train.py –task unsupervised_node_classification – dataset wikipedia –model deepwalk; by running this script, we would be training and testing the DeepWalk model on the Wikipedia dataset for the task of node classification. The results of this process is provided in the figure below (Figure 1).



```
(cogdl) ➜ cogdl git:(master) python scripts/train.py --task unsupervised_node_classification --dataset wikipedia --model deepwalk
Failed to import Deep Graph Library (DGL)
Namespace(cpu=False, dataset=['wikipedia'], device_id=[0], enhance=None, hidden_size=128, iteration=10, lr=0.01, max_epoch=500, model=
['deepwalk'], num_shuffle=5, patience=100, save_dir='.', seed=[1], task='unsupervised_node_classification', walk_length=80, walk_num=4
0, weight_decay=0.0005, window_size=5, worker=10)
Downloading http://snap.stanford.edu/node2vec/POS.mat
Processing...
Done!
node number: 4777
generating random walks...
100%|████████████████████████████████| 40/40 [00:59<00:00,  1.48s/it]
training word2vec...
| Variant                  | Micro-F1 0.1  | Micro-F1 0.3  | Micro-F1 0.5  | Micro-F1 0.7  | Micro-F1 0.9  |
|--------------------------|---------------|---------------|---------------|---------------|---------------|
| ('wikipedia', 'deepwalk') | 0.4253±0.0000 | 0.4720±0.0000 | 0.4951±0.0000 | 0.5056±0.0000 | 0.5214±0.0000 |
```

Figure 1: Result of running the first script

Accordingly, the second script (python scripts/train.py –task node_classification –dataset citeseer – model gcn) analyzes the same task but for training and testing the GCN model on the citeseer dataset. The obtained results are demonstrated in the below figure (Figure 2).



```
(cogdl) ➜ cogdl git:(master) python scripts/train.py --task node_classification --dataset citeseer --model gcn
Failed to import Deep Graph Library (DGL)
Namespace(cpu=False, dataset=['citeseer'], device_id=[0], dropout=0.5, enhance=None, hidden_size=64, lr=0.01, max_epoch=500, missing_r
ate=-1, model=['gcn'], num_classes=None, num_features=None, patience=100, save_dir='.', seed=[1], task='node_classification', weight_d
ecay=0.0005)
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.x
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.tx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.allx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.y
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.ty
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.ally
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.graph
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.citeseer.test.index
Processing...
Done!
Epoch: 440, Train: 1.0000, Val: 0.7000:  88%|█████████████████████████      | 440/500 [00:05<00:00, 84.81it/s]
Valid accurracy = 0.73
Test accuracy = 0.72
| Variant             | Acc           |
|---------------------|---------------|
| ('citeseer', 'gcn') | 0.7200±0.0000 |
```

Figure 2: Result of running the second script

3.2 API Design

CogDL is a graph representation learning toolkit, based on PyTorch. Similar to what is demonstrated in the two scripts of the previous section, the main entry point of the model training is train.py located in the scripts folder. Initially, the arguments provided by the user are evaluated: first, it checks whether the provided arguments are valid; second, it checks whether the provided model matches the input task and dataset (based on match.yml file); third, the model implementation itself checks whether user provides all the required arguments.After the validity of the provided arguments is evaluated, CogDL checks whether the requested dataset is available offline. If this dataset has yet to be downloaded, the download script is called to download the corresponding datasets.

CogDL's API is fairly easy to use. First, setting the hyper-parameters for a specific model can be achieved by modifying the keys of 'args' variable, which is globally used in machine learning frameworks. For instance, for the task of graph classification, on the proteins dataset with the hgpsl model, we would have

```
args = default_args()
args.task = 'graph_classification'
args.dataset = 'proteins'
args.model = 'hgpsl'
```

In addition, the model and its dataset can be built via the following lines:

```
dataset = build_dataset(args)
model = build_model(args)
```

Consequently, we can create the task using the model and dataset and train it via

```
task = build_task(args, dataset=dataset, model=model)
ret = task.train()
```

3.3 Implementation

In this assignment, I implemented the Hierarchical Graph Pooling with Structure Learning model [1]. The pull request id for this implementation is #80 (submitted by Sahandfer).

3.4 Suggestions

There are a number of suggestions that I believe would improve the CogDL experience, both as a user and as a contributor:

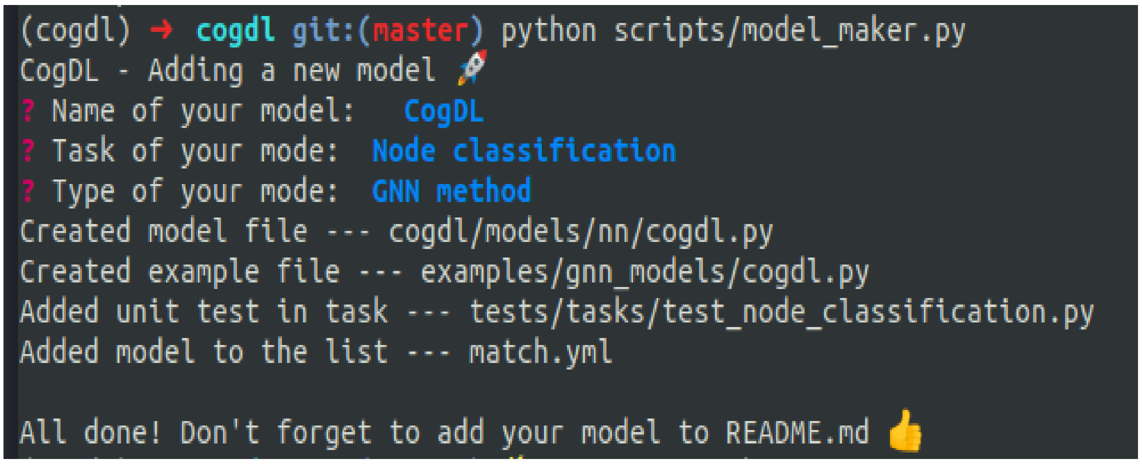
1. Create a unified template for all the models for a certain tasks; this allows users to clearly differentiate between different models and provides contributors with a clear path to implement their models.

2. Include more information on how the evaluation results in the README.md file have been obtained. For instance, for tasks such as graph classification, there aren't any mentions of using k-folds or other arguments to make the models work on certain dataset; yet, these arguments are highly necessary.

3. Make a list of content for the README. The current version of the README file looks rather complex, long, and hard to look through. Providing an interactive list of content would allow the users to focus on the topics of each section and find the section they are looking for much better and faster.

3.5 Contributions

As my contribution to this toolkit, I have made an interactive CLI to make the necessary files for a new model (check figure below).



```
(cogdl) ➜ cogdl git:(master) python scripts/model_maker.py
CogDL - Adding a new model 🚀
? Name of your model:   CogDL
? Task of your mode:  Node classification
? Type of your mode:  GNN method
Created model file --- cogdl/models/nn/cogdl.py
Created example file --- examples/gnn_models/cogdl.py
Added unit test in task --- tests/tasks/test_node_classification.py
Added model to the list --- match.yml

All done! Don't forget to add your model to README.md 👍
```

Figure 3: Model maker CLI

As shown in the figure, the script can be accessed by running 'python scripts/model_maker.py'. The script then asks the user several questions to have the necessary information for creating a new file. Accordingly, the script does the following tasks based on user inputs:

1. Create a model file in the nn folder based on a predefined template.

2. Create an example file in the corresponding folder of the model's type, according to a predefined template.

3. Add the unit test for this model to the corresponding task.

4. Add the model to the list of available models in math.yml.

5. Remind the user to add their own model to README.md since the CLI is yet unavailable to detect the correct location to add the model. In addition, since the model is yet to be defined, its evaluation is not possible.

The pull request id for this contribution is #86 (submitted by Sahandfer).

References

[1] Zhen Zhang et al. "Hierarchical graph pooling with structure learning". In:arXiv(2019) arXiv:1911.05954