

# SGDNet

---

This is a PyTorch implementation of **Signed Graph Diffusion Network** (submitted to ICLR 2021). This paper proposes a novel graph neural network that achieves end-to-end node representation learning for link sign prediction in signed social graphs.

## Prerequisites

---

- python 3.6+
- torch 1.5.0
- numpy 1.18.1
- scipy 1.4.1
- scikit\_learn 0.23.1
- tqdm 4.46.1
- fire 0.3.1
- pytorch 1.5.0
- dotmap 1.3.17
- loguru 0.5.0

## Datasets and Pre-trained SGDNet

---

We provide the datasets used in the paper for reproducibility. You can find the datasets at `./data/${DATASET}` folder where the file's name is `data.tsv`.

- `${DATASET}` is one of `BITCOIN_ALPHA`, `BITCOIN_OTC`, `SLASHDOT` and `EPINIONS`.

The file contains the list of signed edges where each line represents a tuple of (src, dst, sign) which is tab-separated. There are four real-world signed social networks:

- `BITCOIN_ALPHA` : signed social network from the Bitcoin Alpha platform [\[link\]](#)
- `BITCOIN_OTC` : signed social network from the Bitcoin OTC platform [\[link\]](#)
- `SLASHDOT` : signed social network from the Slashdot online review site [\[link\]](#)
- `EPINIONS` : signed social network from the Epinions online review site [\[link\]](#)

This repository also contains pre-trained SGDNet models; you can find them in `./pretrained/${DATASET}` folder where the file's name is `model.pt`. The hyperparameters used for training an SGDNet model are saved at `param.json`.

## Simple Demo

---

You can run the demo script by `bash demo.sh`. It trains SGDNet on `BITCOIN_ALPHA` dataset with hyperparameters stored at `./pretrained/Bitcoin_ALPHA/param.json`. This demo saves the trained model at `./output/Bitcoin_ALPHA/model.pt`. Then, it evaluates the trained model in terms of AUC and F1-macro scores.

## Results of Pre-trained SGDNet

We provide pre-trained SGDNet models which are stored at `./pretrained/${DATASET}/model.pt`, respectively. The experimental results with the pre-trained models are as follows:

Dataset	AUC	F1-macro
Bitcoin-Alpha	0.9177	0.7456
Bitcoin-OTC	0.9227	0.8060
Slashdot	0.8944	0.7792
Epinions	0.9397	0.8521

Note that we conducted the experiments on GTX 1080 Ti (CUDA version 10.1), and the above results were produced with `random-seed=1`.

## Used Hyperparameters

We briefly summarize the hyperparameters used in the above results. The hyperparameters are stored at `./pretrained/${DATASET}/param.json`.

- Hyperparameters of SGDNet
  - `num-layers` (L): number of SGD layers
  - `c`: ratio of local feature injection
  - `num-diff-layers` (K): number of diffusion steps
  - `hid-dim` (d): hidden feature dimension

Hyperparameter	Bitcoin-Alpha	Bitcoin-OTC	Slashdot	Epinions
<code>num-layers</code> (L)	1	2	2	2
<code>c</code>	0.35	0.25	0.55	0.55
<code>num-diff-layers</code> (K)	10	10	10	10
<code>hid-dim</code> (d)	32	32	32	32

- Hyperparameters of optimizer
  - optimizer: Adam

- L2 regularizer ( `weight-decay` ,  $\lambda$ ): 1e-3
- `learning-rate` : 0.01
- `epochs` : 100
- Input feature dimension ( `reduction-dimension` ): 128

## How to Reproduce the Above Results with the Pre-trained Models

You can reproduce the results with the following command which evaluates a test dataset using a pre-trained model.

```
python3 -m run_eval --input-home ../pretrained --dataset ${DATASET} --gpu-id ${GPU_ID}
```

- `${DATASET}` is one of `BITCOIN_ALPHA` , `BITCOIN_OTC` , `SLASHDOT` and `EPINIONS` .
- `${GPU_ID}` is `-1` or your gpu id (maybe, non-negative integer) where `-1` indicates it runs on CPU.

The pre-trained models were generated by the following command:

```
python3 -m run_train --output-home ../output --dataset ${DATASET} --gpu-id ${GPU_ID}
```

Note that those scripts automatically find the file `param.json` for the above hyperparameters. To tune the hyperparameters, modify the json file, or use the below commands.

## Detailed Usage

You can train and evaluate your own datasets using `trainer.py` and `evaluator.py` , respectively. To use those scripts properly, move your working directory to `./src` .

## Training

The following command performs the training process of SGDNet on a given dataset. This automatically splits the dataset by `heldout-ratio` , e.g., if it is 0.2, training:test=0.8:0.2. Note that the split data are guaranteed to be the same if the same `random-seed` is given. After the training is completed, it generates two files called `model.pt` and `param.json` at the `${output-home}/${dataset}` folder where `model.pt` contains parameters of the trained model, and `param.json` has hyperparameters used for the model.

```
python3 -m trainer \
  --data-home ../data \
  --output-home ../output \
  --dataset BITCOIN_ALPHA \
  --heldout-ratio 0.2 \
  --random-seed 1 \
```

```

--reduction-dimension 128 \
--reduction-iterations 30 \
--gpu-id 0 \
--c 0.15 \
--weight-decay 1e-3 \
--learning-rate 0.01 \
--num-layers 1 \
--hid-dim 32 \
--num-diff-layers 10 \
--epochs 100

```

Option	Description	Default
data-home	data directory path	../data
output-home	output directory path	../output
dataset	dataset name	BITCOIN_ALPHA
heldout-ratio	heldout ratio between training and test	0.2
radnom-seed	random seed used for dataset split and torch	1
use-torch-random-seed	whether torch uses the above random seed	True
reduction-dimension	input feature dimension (SVD)	128
reduction-iterations	number of iterations required by SVD computation	30
gpu-id	gpu id	0
c	ratio of local feature injection	0.15
weight-decay	weight decay (L2 regularizer) for optimizer	1e-3
learning-rate	learning rate for optimizer	0.01
num-layers	number of SGD layers (L)	1
hid-dim	hidden feature dimension (d)	32
num-diff-layers	number of diffusion steps (K)	10
epochs	target number of epochs	100

## Evaluation

This performs the evaluation process of SGDNet, and reports AUC and F1-macro scores on the test dataset. This uses `model.pt` and `param.json`; thus, you need to check if they are properly

generated by `trainer.py` before this evaluation. Note that it uses the same random seed used by `trainer.py` so that the test dataset is valid for this evaluation.

```
python3 -m evaluator \
  --input-home ../output \
  --dataset BITCOIN_ALPHA \
  --gpu-id 0
```

Option	Description	Default
<code>input-home</code>	directory where a trained model is stored	<code>../output</code>
<code>dataset</code>	dataset name	<code>BITCOIN_ALPHA</code>
<code>gpu-id</code>	gpu id	<code>0</code>