EvolveGCN:
Evolving Graph Convolutional Networks for Dynamic Graphs

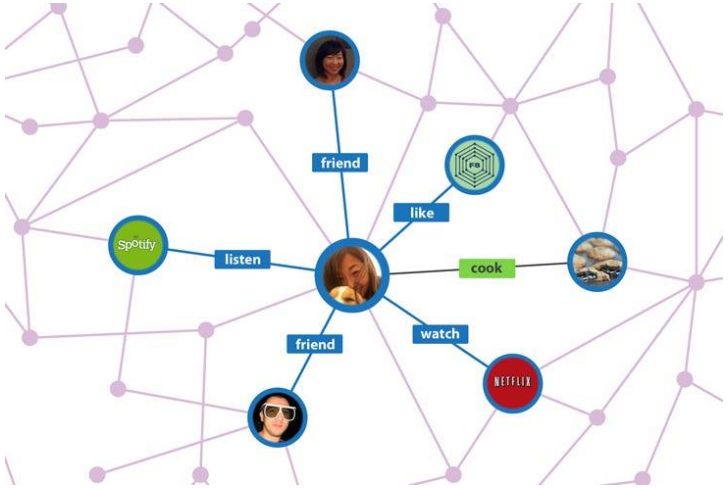
Jae Young Chung
Seoul National University

Contents

Motivation

- What is dynamic graph?
  - An graph which change its node features, # nodes, and edge connectivities, over time

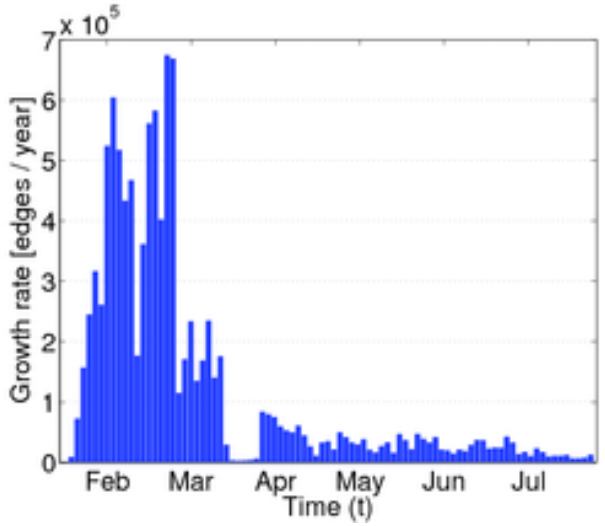
- Why dynamic graph?
  - Static graph is inefficient with large nodes when the graph varying

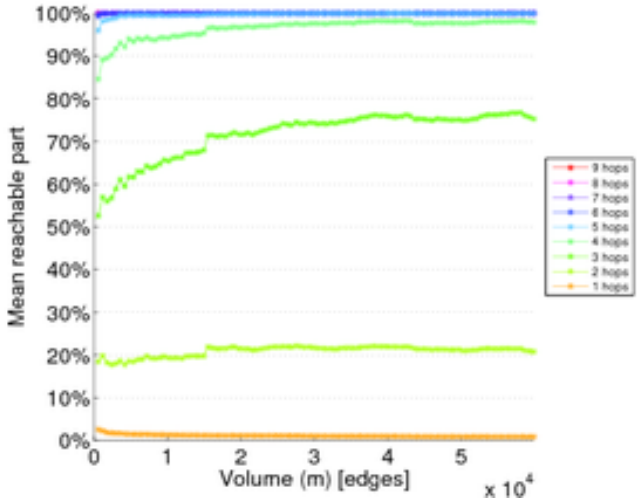


Motivation

- Application examples
  - Example1) Messages between the users of an online community of students from the University of California, Irvine.

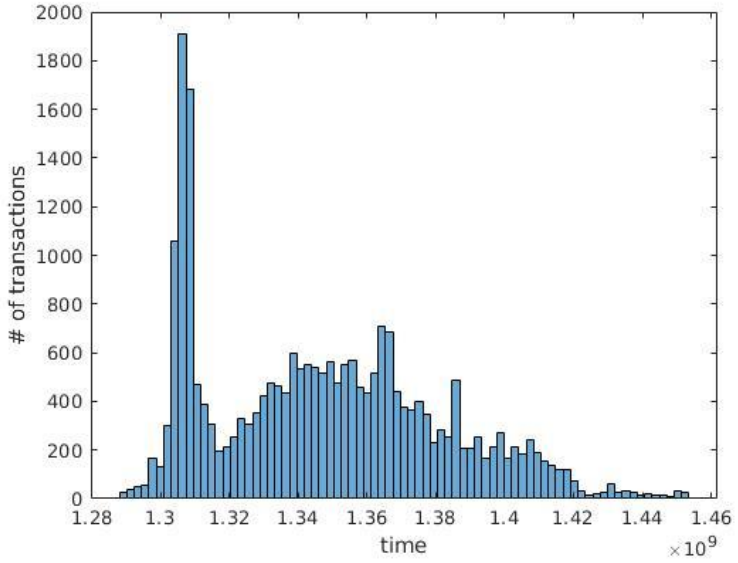
    - Predicting the relations between users from msg passings
    - The number of messages decreases as times goes on.
    - The relativity also decreases.

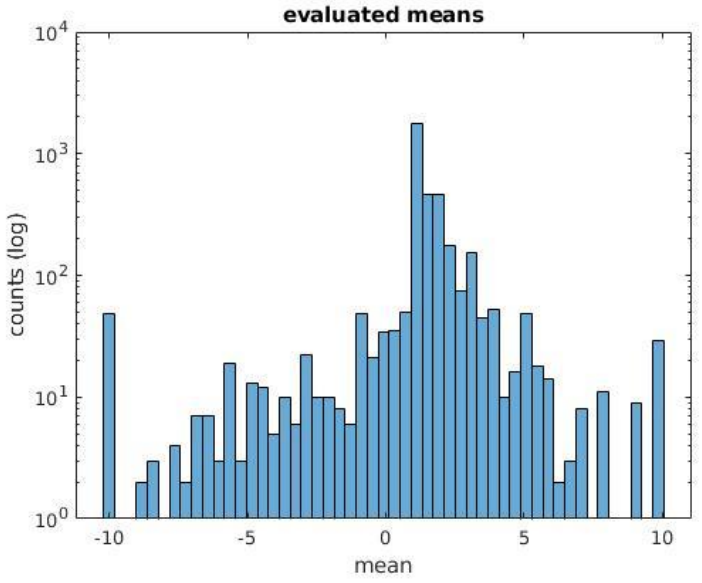


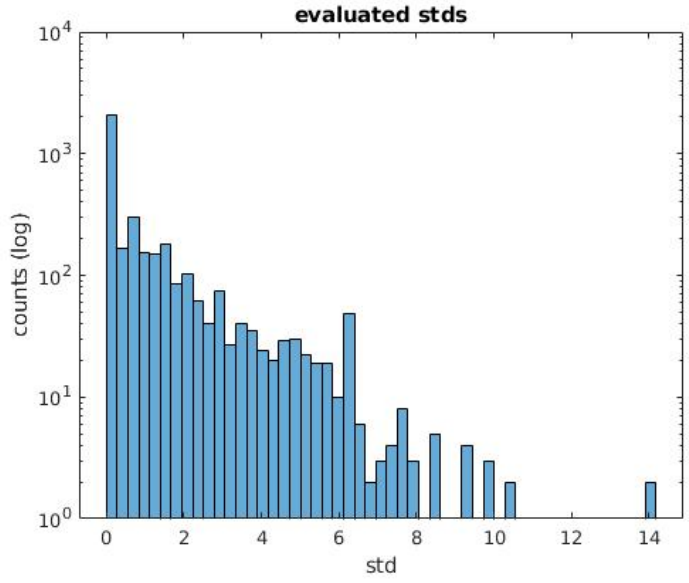


Motivation

- Application examples
  - Example2) Bitcoin-Alpha transactions
    - Who-trusts-whom network of people who trade using Bitcoin-Alpha





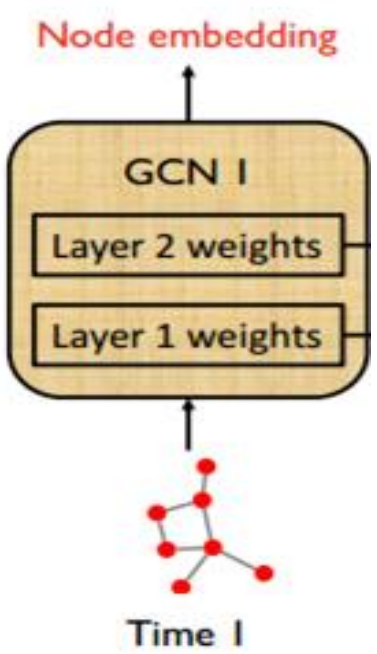


Previous methods

- Laplacian eigenvectors based
  - Matrix factorization (Roweis and Saul, 2000)
  - DANE (Li et al., 2017)
- Random walk based
  - Transition probabilities conditioned on history (Grover and Leskovec, 2016)
  - CTDANE (Nguyen et al., 2018)
  - NetWalk (Yu et al., 2018)
- Deep Learning based
  - DynGEM (Goyal et al., 2017)
  - Seo et al. 2016, Manessia et al. 2017, Narayan et al. 2018

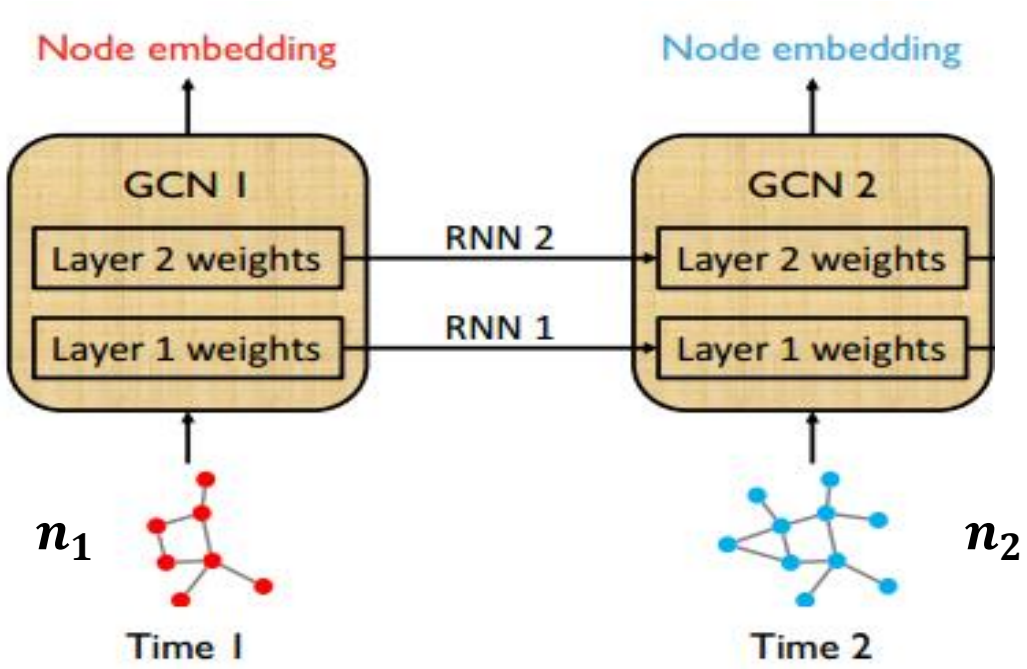
Proposed Method

- GCN (Kipf and Welling, 2017)
  - At time t, l-th layer
    - Adjacency matrix A_t / Node embedding matrix $H_t^{(l)}$ / Weight matrix $W_t^{(l)}$
    - $H_t^{(l+1)} = \sigma\left(\hat{A}_t H_t^{(l)} W_t^{(l)}\right)$ $where$ $\hat{A}_t = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}, \tilde{A} = A + I, \tilde{D} = diag\left(\sum_j \tilde{A}_{ij}\right)$
    - $H_t^0 = X_t$: node feature



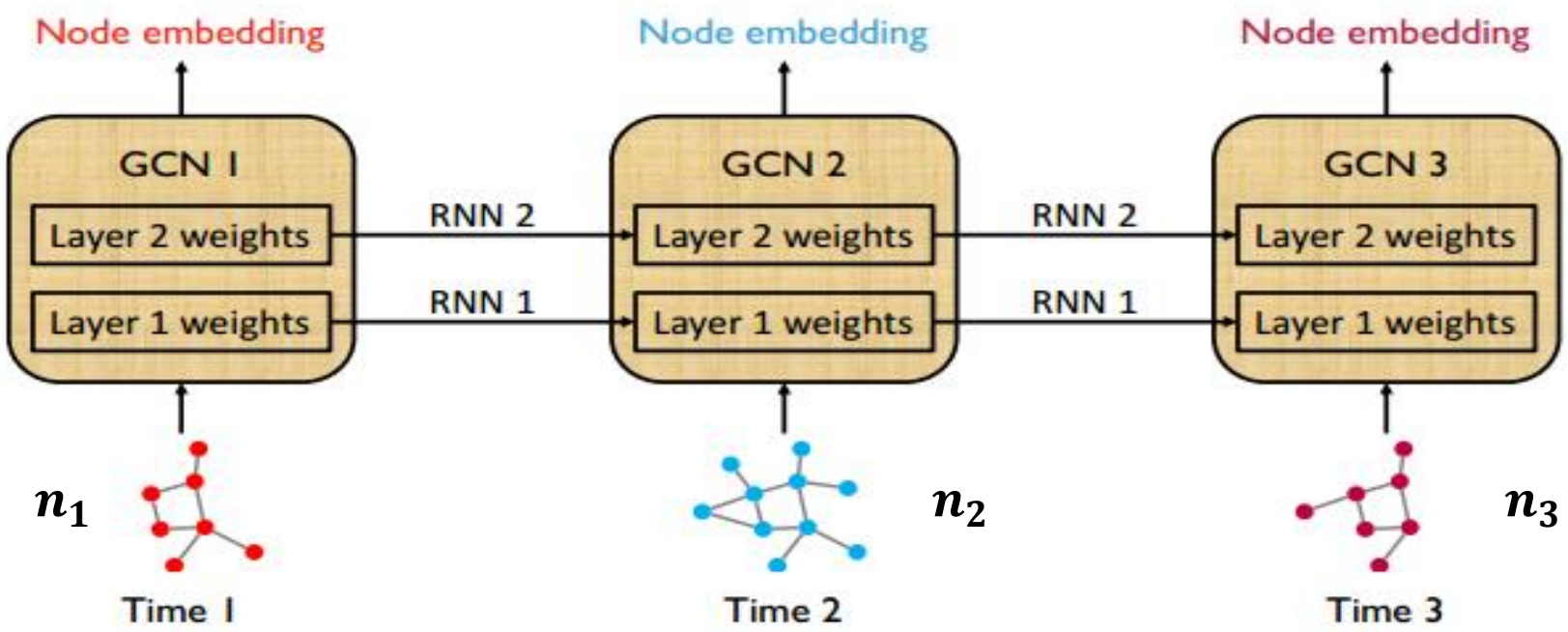
Proposed Method

- EvolveGCN
  - At time $t = 1$
    - $A_1 : n_1 \times n_1$ / $H_1^{(l)} : n_1 \times p^{(l)}$ / $W_1^{(l)} : p^{(l)} \times p^{(l+1)}$
  - At time $t = 2$
    - $A_1 : n_2 \times n_2$ / $H_2^{(l)} : n_2 \times p^{(l)}$ / $W_2^{(l)} : p^{(l)} \times p^{(l+1)}$

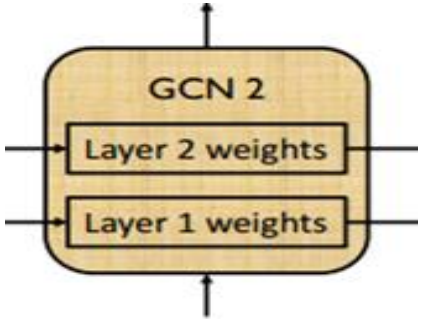


Proposed Method

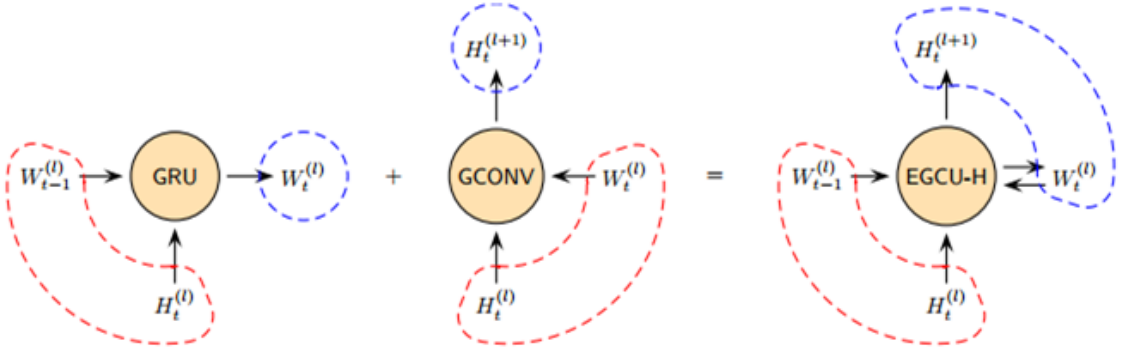
- Use the RNN to regulate the GCN model at every step
  - RNN updates the weights $W_t^{(l)}$ of GCN based on the current information
  - Only RNN parameters are trained, not GCN
  - Note that the dimension of adjacency matrix does not decrease
    - $n_1 < n_2 = n_3$



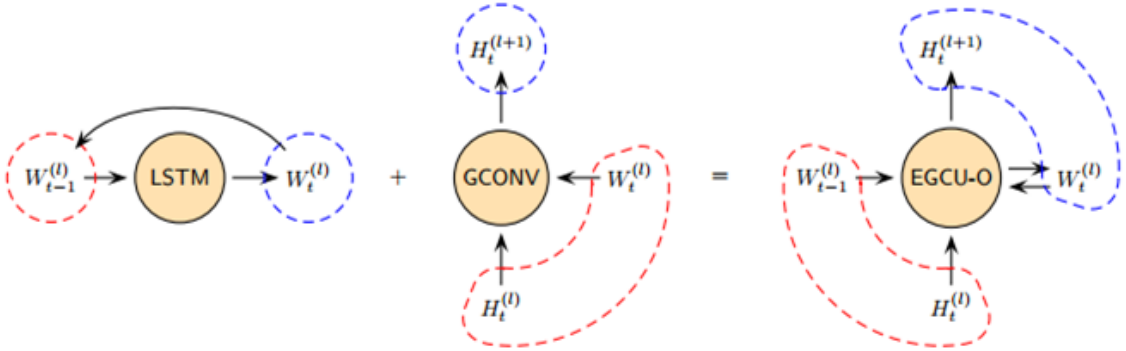
Proposed Method



- Two types of EvolveGCN
  - EvolveGCN-H
    - Treating as a hidden state
    - $GRU\left(H_t^{(l)}, W_t^{(l)}\right)$



  - EvolveGCN-O
    - Treating as an output
    - $LSTM\left(W_t^{(l)}\right)$



Proposed Method

- Two types of EvolveGCN
  - EvolveGCN-H

function $[H_t^{(l+1)}, W_t^{(l)}] = \text{EGCU-H}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
$W_t^{(l)} = \text{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$
$H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
end function

  - EvolveGCN-O

function $[H_t^{(l+1)}, W_t^{(l)}] = \text{EGCU-O}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
$W_t^{(l)} = \text{LSTM}(W_{t-1}^{(l)})$
$H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
end function

function $H_t = g(X_t, H_{t-1})$
$Z_t = \text{sigmoid}(W_Z X_t + U_Z H_{t-1} + B_Z)$
$R_t = \text{sigmoid}(W_R X_t + U_R H_{t-1} + B_R)$
$\widetilde{H}_t = \tanh(W_H X_t + U_H (R_t \circ H_{t-1}) + B_H)$
$H_t = (1 - Z_t) \circ H_{t-1} + Z_t \circ \widetilde{H}_t$
end function

function $H_t = f(X_t)$
Current input X_t is the same as the past output H_{t-1}
$F_t = \text{sigmoid}(W_F X_t + U_F H_{t-1} + B_F)$
$I_t = \text{sigmoid}(W_I X_t + U_I H_{t-1} + B_I)$
$O_t = \text{sigmoid}(W_O X_t + U_O H_{t-1} + B_O)$
$\widetilde{C}_t = \tanh(W_C X_t + U_C H_{t-1} + B_C)$
$C_t = F_t \circ C_{t-1} + I_t \circ \widetilde{C}_t$
$H_t = O_t \circ \tanh(C_t)$
end function

Experiments

- Datasets
  - Bitcoin Alpha
    - Who-trusts-whom network of people who trade using Bitcoin-Alpha
    - Members of Bitcoin Alpha rate other members in a scale of -10 to +10 in steps of 1
  - UC Irvine messages
    - Messages between the users of an online community of students from the University of California, Irvine.
    - A node represents a user. A directed edge represents a sent message. Multiple edges denote multiple messages.
  - Social Network: Reddit Hyperlink Network
    - The hyperlink network represents the directed connections between two subreddits
    - Providing subreddit embeddings
    - Being extracted from Reddit data of 2.5 years from Jan 2014 to April 2017.

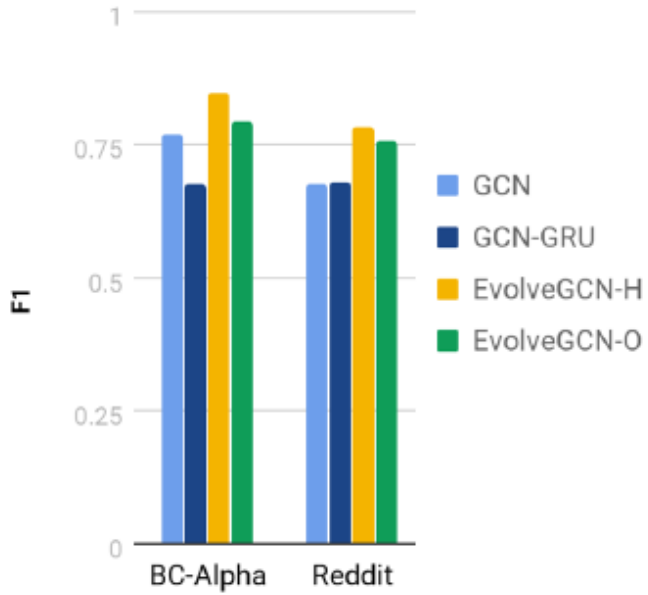
Experiments

	# Nodes	# Edges	# Time Steps (Train/Val/Test)
Bitcoin-Alpha	5,881	35,588	95/14/28
UC Irvine msgs	1,899	59,835	62/9/17
Reddit	55,863	858,490	122/18/34

- Results
  - Link prediction

mAP	GCN	GCN-GRU	DynGEM	dyngraph2vecAE	dyngraph2vecAERNN	EvolvGCN-H	EvolveGCN-O
BC-Alpha	0.0003	0.0001	0.0525	0.0507	**0.1100**	0.0049	0.0036
UCI	0.0251	0.0114	0.0209	0.0044	0.0205	0.0126	**0.0270**

  - Edge classification



Conclusion

- Learning RNN instead of GCN for dynamic graph
- Dataset may not be appropriate
- Not impressive results

코드 주소

- 코드 github 주소: https://github.com/IBM/EvolveGCN
  - 실행파일: run_exp.py
  - 학습을 위한 파일: Cross_Entropy.py, logger.py, log_analyzer.py, models.py, splitter.py, taskers_utils.py, trainer.py, utils.py
  - 모델 파일: egcn_h.py, egcn_o.py
  - 작업 정의 파일: edge_clls_tasker.py, link_pred_tasker.py, node_cls_tasker.py
  - 데이터셋 파일: auto_syst_dl.py, bitcoin_dl.py, elliptic_temporal_dl.py, reddit_dl.py, sbm_dl.py, uc_irv_mess_dl.py

실행 방법

- Requirements
  - Python 3.6 or higher
  - Pytorch 1.0 or higher: https://pytorch.org/get-started/locally/
  - Docker: https://docs.docker.com/get-docker/
  - Nvidia drivers: https://www.nvidia.com/Download/index.aspx?lang=en-us
- Installation
  - Git repository
    - git clone (주소)
  - Build Docker
    - sudo docker build -t gcn_env:latest docker-set-up/
  - Start container
    - sudo docker run -ti --shm-size=256m --gpus all -v $(pwd):/evolveGCN gcn_env:latest
      - 공유메모리가 부족하다는 말이 나오면 위의 옵션을 크게 조절한다. (현재 256MB)

실행 방법

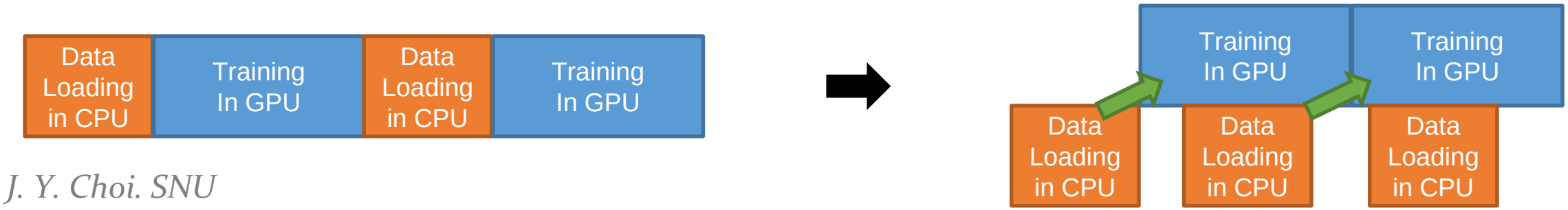
- Usage
  - python run_exp.py --config_file ./path/to/configuration.yaml
  - E.g.) python run_exp.py --config_file ./experiments/parameters_example.yaml

- Dataset
  - README.md에 있는 주소에서 데이터를 받아서 data/ 폴더에 위치 시킴

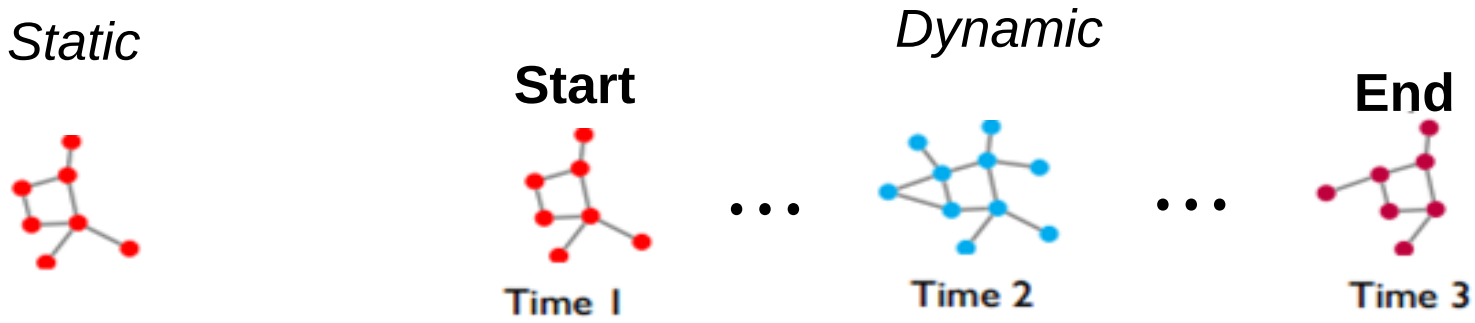
코드 설명

- Run_exp.py
  - Dataset 종류, 실행할 task, 데이터 나누는 방식(splitter), classifier, gcn 종류에서 원하는 옵션을 선택하고 실행. 이에 대한 옵션은 실행시 지정한 .yaml파일에 저장되어 있다.

  - Pytorch에서 Dataloader를 이용할 경우, 멀티프로세싱을 지원해서 model이 gpu에서 학습하는 동안 데이터를 로딩한다. (데이터 로딩 시간이 NN forwarding 시간을 넘어서 gpu usage가 출렁거릴 경우 batch size를 줄이면 된다.)
    - https://pytorch.org/docs/1.1.0/_modules/torch/utils/data/dataloader.html



코드 설명



- Splitter.py
  - 데이터(로더)를 세팅할 때, static인 경우와 dynamic인 경우를 나눠 놓았는데, static의 경우 하나의 graph에 대해서만 처리하고 dynamic인 경우 구간(start~end) 안에 있는 graph 시퀀스를 모두 가져온다.
  - Classifier (build_classifier)
    - GCN의 weight들을 가지고 정한 task의 결과를 뽑아주는 모듈. 옵션에 따라 weight의 사이즈가 조절된다.
  - GCN (build_gcn)
    - GCN 모델을 선택하고 dynamic의 경우 GCN weight를 관리할 RNN을 선택하는 모듈

코드 설명

- Trainer.py
  - Trainer class

  -> train( ): train/valid 경우를 포함하며, early stop 구현(early_stop_patience)

  -> run_epoch(…): 네트워크의 forward, loss, optimizer 실행

  -> predict(…): gcn에서 가져온 node embedding을 classifier에 넣어서 prediction 값을 뽑아낸다.

결과 비교

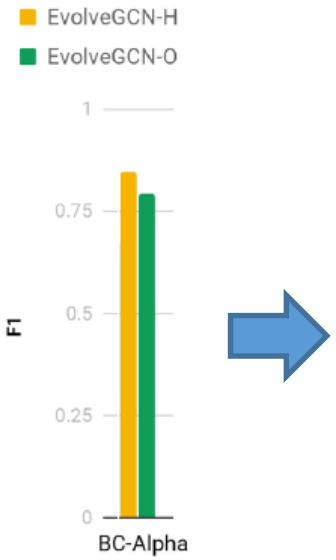
- Link Prediction
  - Bitcoin alpha
  - SBM50

Table 2: Performance of link prediction. Each column is one data set.

	mean average precision					mean reciprocal rank				
	SBM	BC-OTC	BC-Alpha	UCI	AS	SBM	BC-OTC	BC-Alpha	UCI	AS
GCN	0.1987	0.0003	0.0003	0.0251	0.0003	0.0138	0.0025	0.0031	0.1141	0.0555
GCN-GRU	0.1898	0.0001	0.0001	0.0114	0.0713	0.0119	0.0003	0.0004	0.0985	0.3388
DynGEM	0.1680	0.0134	0.0525	0.0209	0.0529	0.0139	0.0921	0.1287	0.1055	0.1028
dyngraph2vecAE	0.0983	0.0090	0.0507	0.0044	0.0331	0.0079	0.0916	0.1478	0.0540	0.0698
dyngraph2vecAERNN	0.1593	**0.0220**	**0.1100**	0.0205	0.0711	0.0120	**0.1268**	**0.1945**	0.0713	0.0493
EvolveGCN-H	0.1947	0.0026	0.0049	0.0126	**0.1534**	**0.0141**	0.0690	0.1104	0.0899	**0.3632**
EvolveGCN-O	**0.1989**	0.0028	0.0036	**0.0270**	0.1139	0.0138	0.0968	0.1185	**0.1379**	0.2746

	Mean Average Precision		Mean Reciprocal Rank	
	SBM	Bitcoin-Alpha	SBM	Bitcoin-Alpha
EvolveGCN-H	0.18029	0.00453	0.01358	0.1133
EvolveGCN-O	0.1932	0.00229	0.01398	0.1064

- Edge Classification
  - Bitcoin alpha



	Mean Average Precision
	Bitcoin-Alpha
EvolveGCN-H	0.91103
EvolveGCN-O	0.91561