DynGAN: Generative Adversarial Networks for Dynamic Network Embedding

Ayush Maheshwari[#] **Ayush Goyal**[*] **Manjesh Kumar Hanawal**[†] **Ganesh Ramakrishnan**[#]
Department of CSE[#] Department of IEOR,[†]
Indian Institute of Technology Bombay, India[#†]
Samsung Research, Bengaluru, India[*]
`{ayusham,ganesh}@cse.iitb.ac.in`[#], `ayushgoyaal@gmail.com`[*], `mhanawal@iitb.ac.in`[†]

Abstract

Embedding large graphs in a low-dimensional space has proven useful in various applications. However, there is a limited focus on real-world networks that are dynamic in nature and continuously evolving with time. In this paper, we propose a novel adversarial algorithm to learn representation of dynamic networks. We leverage generative adversarial networks and recurrent networks to capture temporal and structural information. We conduct extensive experiments on the task of graph reconstruction, link prediction and graph prediction. Experimental results demonstrate consistent, stable, and better results against state-of-the-art methods in many cases.

1 Introduction

Network embedding has benefited tremendously in social networks, biological networks, gene-protein networks, etc. The basic idea is to embed a network into low-dimensional vector space such that the structural properties are preserved. Dynamic networks are evolving networks wherein the structure of nodes and edges changes with time. For example, users in a social network add or remove friends such that the user community evolves over time. Dynamic graphs are represented as a sequence of snapshots of graphs at different time steps. Each snapshot represents edges and nodes that occur between a user-specified discrete-time interval (e.g. day or week or month).

Network embedding on a single graph has been researched extensively in the recent years and various techniques have been proposed to solve the problem of link prediction(Wang et al. [2016a], Chen et al. [2016], Maheshwari et al. [2019]), node classification(Yang et al. [2016], Huang and Mamoulis [2017]), and clustering(Wang et al. [2017a], Nie et al. [2017]). However, the time-series component in dynamic graphs produces a new challenge for network embedding. Key factors influencing the performance of embedding are a) growing nature of graphs, b) unstable nature of graphs at consecutive time-steps, and c) scalability of learned embeddings.

Various approaches have been proposed for generating embeddings for dynamic graphs. LIST(Yu et al. [2017]) models first-order proximity and uses a time-dependent matrix to express graph structure as a function of time. Dyngraph2vec(Goyal et al. [2019]) uses dense and recurrent layers to capture temporal transitions in the network. DynGEM(Goyal et al. [2018]) uses deep autoencoder to capture non-linearity in dynamic graphs. DynGraphGAN(Xiong et al. [2019]) uses Generative Adversarial Network(Goodfellow et al. [2014]) (GAN) and evaluates the performance of embeddings on graph reconstruction and link prediction. It uses a graph convolution network as a discriminator to distinguish between fake and real edges. However, dynGraphGAN uses graph datasets having atmost 16 snapshots and shows marginal improvements over baselines.

[*]This work was conducted during his graduation studies at IIT Bombay

In this paper, we develop a graph embedding algorithm, referred to as DynGAN, to generate stable embeddings of dynamic networks. DynGAN employs GAN at its core and leverages recurrent neural networks to capture temporal transitions. DynGAN incrementally learns to embed for each snapshot of the graph. We initialise embedding of next step from the previous time step and learn gradients over it. This ensures stability of the embeddings and requires lesser time to converge after first few iterations. We demonstrate our results on three tasks, namely, a) graph reconstruction, b) link prediction, and c) graph prediction. We perform extensive experiments on benchmark datasets and compare with state-of-the-art baselines. Our experiments achieve improved results on graph reconstruction and link prediction over state-of-the-art results. We also report significant gains on graph prediction for one dataset. Our contributions are summarised as follows:

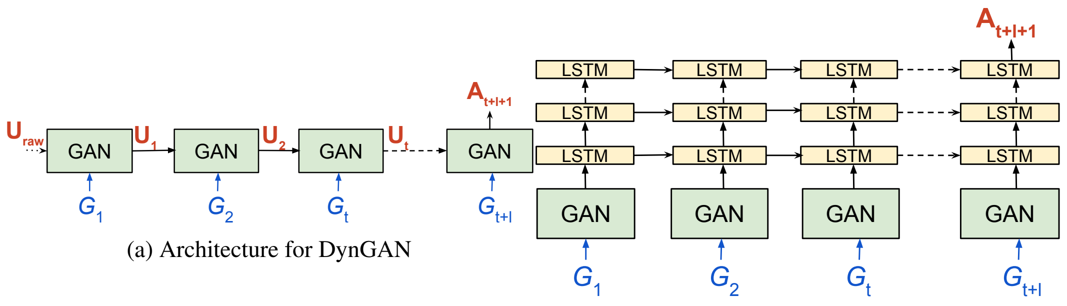
- We propose a novel adversarial network based architecture for generating network embeddings for dynamic graphs.
- We construct experiments over three tasks and two large real-world datasets. Our model is able to show significant improvement over state-of-the-art baselines on all tasks.

2 Problem Setup

Let $G = (V, E)$ be a given undirected graph where $V = v_1, v_2, \ldots, v_n$ is the vertex set and $E = (v_i, v_j)$ is the associated edge set such that $v_i, v_j \in V$.

Dynamic network. A series of undirected graphs $G_1, G_2 \ldots G_T$ where $G_t = (V_t, E_t)$ represents a graph at time t. Our goal is to learn low-dimensional stable representation of vertices v_i over time such that temporal and structural properties of the series of graph are effectively captured. In essence, consecutive embeddings should differ little if graph structure does not change much.

2.1 Algorithms for Dynamic Graphs



(a) Architecture for DynGAN

(b) Architecture for DynGAN-LSTM

Figure 1

We develop two algorithms that are adversarial network based deep learning models. Both employ a generator-discriminator model to learn embeddings for dynamic graphs. We train both generator G and discriminator D simultaneously and expect G to learn the data distribution. Previous works have demonstrated the effectiveness of GAN in graph networks(Wang et al. [2017b], Maheshwari et al. [2019]). **Generator** captures the data distribution and learns a parameter θ_g such that $G(v|v_c; \theta_g)$ can approximate the true distribution. v and v_c are the sampled vertices in the generator[2]. **Discriminator** estimates a probability to differentiate the samples arriving from the generator and true distribution. It learns a parameter θ_d such that $D(v_i, v_j; \theta_d)$ can discriminate between the presence or absence of an edge between v_i and v_j. The minimax game with objective function $V(G, D)$ can be formalised as $\min\limits_{\theta_G} \max\limits_{\theta_D} V(G, D)$ where $V(G, D)$ is given as

$$\sum_{c=1}^{V} \left(\mathbb{E}_{v \sim p_{true}(\cdot|v_c)} \left[\log D(v, v_c; \theta_D, w_D) \right] + \mathbb{E}_{v \sim G(\cdot|v_c; \theta_G, w_G)} \left[\log \left(1 - D(v, v_c; \theta_D, w_D) \right) \right] \right)$$

[2]Refer to supplementary material for sampling strategy

Parameters of generator and discriminator are learnt alternately by minimizing and maximizing the objective function. Discriminator D is a sigmoid function over a pair of vertices and generator G is a softmax function over all the vertices in V.[3]

2.2 DynGAN

Architecture of our DynGAN model is shown in Figure 1a. GAN is initialized with random embedding U_{raw}. For each time step t, our model generates embedding U_t that is fed as an input to the next GAN component. Such an architecture is capable of handling evolving graphs due to preservation of weights from previous time step. Additionally, it ensures the stability of embeddings due to initialization of current time step embeddings with the previous time step output embeddings. Hence, training converges very fast after few iterations of initial time steps. We perform experiments using this architecture on all three tasks. However, DynGAN fails to capture temporal sequence in the network due to its limited capability to capture previous time-step information. Hence, dynGAN does not perform equally well on graph prediction. To overcome this limitation, we propose a new architecture based on the combination of GAN and Long Short Term Memory (LSTM), named as DynGAN-LSTM. We leverage this architecture to consider previous time stamps for the task of graph prediction.

2.3 DynGAN-LSTM

Our problem is to learn an embedding u_{v_i} in a low-dimensional representation for each node v_i such that embeddings can capture temporal and structural patterns necessary to predict v_{i+1} for the task of graph prediction. We use the DynGAN model(Figure 1b to learn embeddings at each time step and pass them through a sequence of LSTM networks to capture sequential information. LSTMs are chosen because they are expected to capture evolution in the graph that allows us to predict the next graph. The input to the model is an undirected graph that is transformed into a node adjacency matrix. Given an adjacency matrix A_t at time t our model optimizes the following loss function,

$L_{t+1} = ||\hat{A}_{t+l+1} - A_{t+1+1} \odot \beta||_F^2 = ||(f(y_t, ..., y_{t+l}) - A_{t+l+1})||||(f(A_t, ..., A_{t+1}) - A'_{t+l+1}) \odot \beta||_F^2$. Here, β is a hyperparameter penalizing observed edges that gives higher weights to observed edges than unobserved edges(Belkin and Niyogi [2002]), l is the temporal look back factor that controls the range of sequential dependency in our model and $\odot$ represents elementwise product. Parameters are tuned by this loss function to penalize incorrect reconstruction of edges at time $t+l+1$ by using previous time step embeddings.

3 Experiments and Results

We evaluate the performance of DynGAN model on the task of link prediction , graph reconstruction and graph prediction. We test DynGAN-LSTM on the task graph prediction. We use real-world dynamic graphs summarized in Table 1.

3.1 Datasets

HEP-TH(Gehrke et al. [2003]) is a collaboration network containing abstracts of paper in High Energy Physics Theory. It contains 136 time steps and the number of nodes range from 150 to 14446. **Autonomous Systems(AS)**(Leskovec and Krevl [2016]) is a communication network containing logs of Border Gateway Protocol. The dataset contains 733 time steps with a fixed number of nodes but number of edges ranges from 487 to 26467.

Dataset	#Nodes	#Edges	#Time steps
HEP-TH	150-14446	268-48274	136
AS	7716	487-26467	733

Table 1: Summary of datasets used for the experiments

[3]The detailed treatment of the graph GAN model is given in supplementary.

3.2 Baseline and Evaluation Metrics

For graph reconstruction and link prediction, we compare our model against SDNE(Wang et al. [2016b]) and dynGEM(Goyal et al. [2018]). For the task of graph prediction, we compare against dynGEM and three variants of dyngraph2vec(Goyal et al. [2019]) namely, dyngraph2vecAE, dyngraph2vecRNN, and dyngraph2vecAERNN. In our experiments, we evaluate the performance of our model using Mean Average Precision (MAP) and Precision@k (P@k) as a evaluation metric for all three tasks.

3.3 Results

Graph Reconstruction In this task, we attempt to accurately reconstruct the graph from the learned embeddings of nodes. We reconstruct the edges between pair of nodes using DynGAN model. Our model beats state-of-the-art methods by huge margin.

Link Prediction We test the applicability of our model on link prediction for unobserved edges. We train our model on $\{G_1, G_2..., G_{t-1}\}$ and 85 percent of edges of G_t and test on remaining edges of G_t. The results for graph reconstruction and link prediction are shown in Table 2. The results show that DynGAN model is able to outperform all deep neural networks and autoencoder-based models by a huge margin.

Task	Graph Reconstruction		Link Prediction	
Algorithm	AS	HEP-TH	AS	HEP-TH
SDNE	0.214	0.51	0.09	0.1
dynGEM	0.216	0.491	0.21	0.26
DynGAN	**0.465**	**0.65**	**0.464**	**0.636**

Table 2: Average MAP for the task of graph reconstruction and link prediction

Graph Prediction - In this task, we train the model with $\{G_1, G_2..., G_{t-1}\}$ snapshots of graphs to predict G_t. Instead of predicting over all time-steps, we consider last 50 snapshots of the datasets. We observe that change in the number of nodes and edges are more frequent in last snapshots of the graph (refer supplementary material). We remark that the efficiency of the models must be tested when a sudden change in the nodes and edge occurs. We train our model with a lookback factor of 2 and embedding dimension of 32.

	MAP Estimate		Precision@alledges	
Algorithm	AS	HEP-TH	AS	HEP-TH
dynGEM	0.097	0.258	0.0613	0.073
dyngraph2vecAE	0.182	0.395	0.018	0.003
dyngraph2vecRNN	0.235	0.545	0.438	**0.402**
dyngraph2vecAERNN	**0.275**	**0.595**	0.002	0.005
DynGAN	0.26	0.376	0.152	0.1811
DynGANLSTM	0.232	0.45	**0.637**	0.262

Table 3: Average MAP & Precision@all edges for last 50 snapshots of AS & HEP-TH.

In the last 50 snapshots, when nodes and edges are changing by a large number in consecutive snapshots, our model performs marginally lower than dyngraph2vec. However, our model performs better on AS dataset when precision metric is considered and predicts consistently across various scenarios, unlike other models.

4 Conclusion

We introduced DynGAN and DynGAN-LSTM, a model for capturing temporal and structural information in the dynamic networks. It learns the evolution pattern in an adversarial manner and predicts node embeddings. We conduct extensive experiments on benchmark datasets containing large timesteps and high variations. Our model demonstrates superiority and consistency of results in graph reconstruction, link prediction, and graph prediction and outperforms state-of-the-art methods.

References

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016a.

Jifan Chen, Qi Zhang, and Xuanjing Huang. Incorporate group information to enhance network embedding. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1901–1904. ACM, 2016.

Ayush Maheshwari, Ayush Goyal, Amit Kumar, Manjesh Kumar Hanawal, and Ganesh Ramakrishnan. Representation learning on graphs by integrating content and structure information. In *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*, pages 88–94. IEEE, 2019.

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

Zhipeng Huang and Nikos Mamoulis. Heterogeneous information network embedding for meta path based proximity. *arXiv preprint arXiv:1701.05291*, 2017.

Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017a.

Feiping Nie, Wei Zhu, and Xuelong Li. Unsupervised large graph embedding. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

Wenchao Yu, Wei Cheng, Charu C Aggarwal, Haifeng Chen, and Wei Wang. Link prediction with spatial and temporal consistency in dynamic networks. In *IJCAI*, pages 3343–3349, 2017.

Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 2019.

Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.

Yun Xiong, Yao Zhang, Hanjie Fu, Wei Wang, Yangyong Zhu, and S Yu Philip. Dyngraphgan: Dynamic graph embedding via generative adversarial networks. In *International Conference on Database Systems for Advanced Applications*, pages 536–552. Springer, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. GraphGAN: Graph representation learning with generative adversarial nets. *arXiv preprint arXiv:1711.08267*, 2017b.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

Johannes Gehrke, Paul Ginsparg, and Jon Kleinberg. Overview of the 2003 kdd cup. *Acm SIGKDD Explorations Newsletter*, 5(2):149–151, 2003.

Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection (2014). *URL http://snap. stanford. edu/data*, page 49, 2016.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016b.

(Supplementary Material) DynGAN: Generative Adversarial Networks for Dynamic Network Embedding

5 Variation of edges and nodes across time steps

Figure 2 shows change in number of edges and nodes for each snapshot for both AS and Hep-th dataset. We choose last 50 time steps for training and testing the proposed models. It is evident from the figure that there are drastic change at some time steps in the graph. The problem of capturing embedding become more difficult due to this very nature of dynamic graphs.

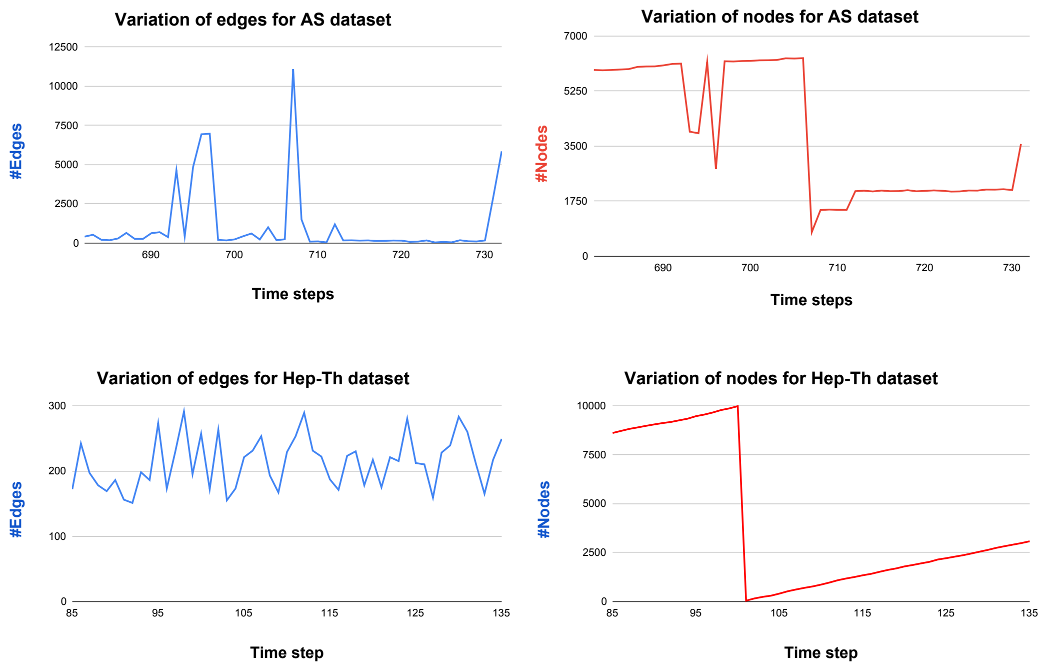


Figure 2: Change in number of nodes for last 50 time steps for both datasets.

6 Sampling strategy

We describe sampling process for choosing pair of vertices by generator motivated by Wang et al. [2017b]. Firstly, we create Breadth First Search (BFS) tree B_v for the vertex set V. Secondly, vertex v_c is selected and we parse the adjacent vertex with the probability estimated from Eq (1). This is repeated until current vertex selects the previously chosen vertex. At the end, initial vertex v_c and the current vertex is chosen as pairs. Due to the tree structure of vertices, the algorithm stops in $O(logV)$ steps .

$$p(v_i|v) = \frac{exp\Big((g_{v_i}^T g_v)\Big)}{\sum_{v_j \in N_c(v)} exp\Big((g_{v_j}^T g_v)\Big)} \tag{1}$$

Algorithm 1 Sampling strategy for generator

```
Require: BFS tree $B_v$ representation of $v \in V$
Ensure: Generated samples $v_{gen}$
1: Select a starting vertex $v_c := v_{cur}$ and $v_c := v_{prev}$;
2: while true do
3:     Select neighboring vertex $v_i$ proportional to $p(v_i|v_c)$ using Eq. (1);
4:     if $v_i = v_{prev}$ then
5:         RETURN $v_{gen}$;
6:     else
7:         $v_{prev} := v_{cur}, v_{cur} := v_i$
8:     end if
9: end while
```

7 Approach for generating embeddings

7.1 Generator

Eq (1) is a softmax function over all neighboring vertices of v. The gradient of $V(G, D)$ can be computed as :

$$\nabla_{\theta_G} V(G, D) = \sum_{c=1}^{V} \mathbb{E}_{v \sim G(\cdot|v_c)} \Big[\nabla_{\theta_G} logG(v|v_c)\ log\Big(1 - D(v, v_c)\Big)\Big] \tag{2}$$

Similarly, gradient with respect to w can be computed as:

$$\nabla_{w_G} V(G, D) = \sum_{c=1}^{V} \mathbb{E}_{v \sim G(\cdot|v_c)} \Big[\nabla_{w_G} logG(v|v_c)\ log\Big(1 - D(v, v_c)\Big)\Big] \tag{3}$$

Generator penalises vertices having larger probability of negative samples.

7.2 Discriminator

$$D(v, v_c) = \sigma(d_v^T, d_{v_c}, w) = \frac{1}{1 + exp(-d_v^T d_{v_c})} \tag{4}$$

Let d_v and d_{v_c}be the discriminator embedding for vertices v and v_c respectively. Then, gradient ascent equation can be computed as :

$$\nabla_{\theta_D} V(G, D) = \begin{cases} \nabla_{\theta_D}\ log\ D(v, v_c),\ if\ v \sim p_{true}; \\ \nabla_{\theta_D} \Big(1 - log\ D(v, v_c)\Big),\ if\ v \sim G \end{cases} \tag{5}$$

Gradient ascent update equation wrt w is computed as :

$$\nabla_{w_D} V(G, D) = \begin{cases} \nabla_{w_D}\ logD(v, v_c), ifv \sim p_{true}; \\ \nabla_{w_D}\ \Big(1 - log\ D(v, v_c)\Big),\ if\ v \sim G \end{cases} \tag{6}$$

The detailed treatment can be referred in Wang et al. [2017b].

Algorithm 2 Approach for generating embeddings

```
Require: Number of vertices to sample from $G$ and $D$
Ensure: Embedding matrix for $G$ and $D$
 1: Initialize $G$ and $D$ with pre-trained or random initializations;
 2: Construct BFS tree $B_v$ for all $v_c \in V$;
 3: while $G$ does not converge do
 4:     for $G$-steps do
 5:         Generate $g$ vertices for each vertex $v_c$ using Algorithm 1;
 6:         Update $\theta_G$ and $w_G$ according to Eq. (1), (2) and (3);
 7:     end for
 8:     for $D$-steps do
 9:         Sample $t$ positive vertices from ground truth and $t$ negative vertices from $G$ for each vertex
            $v_c$;
10:         Update $\theta_D$ and $w_D$ according to Eq. (4), (5) and (6);
11:     end for
12: end while
```