dyngraph2vec: Capturing Network Dynamics using Dynamic Graph Representation Learning

Palash Goyal[*]
USC Information Sciences Institute
palashgo@usc.edu

Sujit Rokka Chhetri[*]
University of California-Irvine
schhetri@uci.edu

Arquimedes Canedo
Siemens Corporate Technology
arquimedes.canedo@siemens.com

Learning graph representations is a fundamental task aimed at capturing various properties of graphs in vector space. The most recent methods learn such representations for static networks. However, real world networks evolve over time and have varying dynamics. Capturing such evolution is key to predicting the properties of unseen networks. To understand how the network dynamics affect the prediction performance, we propose an embedding approach which learns the structure of evolution in dynamic graphs and can predict unseen links with higher precision. Our model, *dyngraph2vec*, learns the temporal transitions in the network using a deep architecture composed of dense and recurrent layers. We motivate the need of capturing dynamics for prediction on a toy data set created using stochastic block models. We then demonstrate the efficacy of dyngraph2vec over existing state-of-the-art methods on two real world data sets. We observe that learning dynamics can improve the quality of embedding and yield better performance in link prediction.

Introduction

Understanding and analyzing graphs is an essential topic that has been widely studied over the past decades. Many real world problems can be formulated as link predictions in graphs (Gehrke, Ginsparg, and Kleinberg 2003; Freeman 2000; Theocharidis et al. 2009; Goyal, Sapienza, and Ferrara 2018). For example, link prediction in an author collaboration network (Gehrke, Ginsparg, and Kleinberg 2003) can be used to predict potential future author collaboration. Similarly, new connections between proteins can be discovered using protein interaction networks (Pavlopoulos, Wegener, and Schneider 2008), and new friendships can be predicted using social networks (Wasserman and Faust 1994). Recent work on obtaining such predictions use graph representation learning. These methods represent each node in the network with a fixed dimensional embedding, and map link prediction in the network space to a nearest neighbor search in the embedding space (Goyal and Ferrara 2018). It has been shown that such techniques can outperform traditional link prediction methods on graphs (Grover and Leskovec 2016; Ou et al. 2016a).

[*]These authors contributed equally to this work.

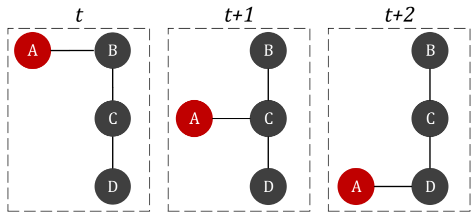


Figure 1: User A breaks ties with his friend at each time step and befriends a friend of a friend. Such temporal patterns require knowledge across multiple time steps for accurate prediction.

Existing works on graph representation learning primarily focus on static graphs of two types: (i) aggregated, consisting of all edges until time T; and (ii) snapshot, which comprise of edges at the current time step t. These models learn latent representations of the static graph and use them to predict missing links (Ahmed et al. 2013; Perozzi, Al-Rfou, and Skiena 2014; Cao, Lu, and Xu 2015; Tang et al. 2015; Grover and Leskovec 2016; Ou et al. 2016a; Goyal et al. 2018a). However, real networks often have complex dynamics which govern their evolution. As an illustration, consider the social network shown in in Figure 1. In this example, user A moves from one friend to another in such a way that only a friend of a friend is followed, and making sure not to befriend an old friend. Methods based on static networks can only observe the network at time $t+1$ and cannot ascertain if A will befriend B or D in the next time step. Instead, observing multiple snapshots can capture the network dynamics and predict A's connection to D with high certainty.

In this work, we aim to capture the underlying network dynamics of evolution. Given temporal snapshots of graphs, our goal is to learn a representation of nodes at each time step while capturing the dynamics such that we can predict their future connections. Learning such representations is a challenging task. Firstly, the temporal patterns may exist over varying period lengths. For example, in Figure 1, user A may hold to each friend for a varying k length. Secondly, different vertices may have different patterns. In Figure 1, user A may break ties with friends whereas other users continue with their ties. Capturing such variations is extremely challenging. Existing research builds upon simplified assumptions to overcome these challenges. Methods including DynamicTriad (Zhou et al. 2018), DynGEM (Goyal et al. 2018b) and TIMERS (Zhang et al. 2017) assume that the patterns are of short duration (length 2) and only consider the previous time step graph to predict new links. Furthermore, DynGEM and TIMERS make the assumption that the changes are smooth and use a regularization to disallow rapid changes.

In this work, we present a model which overcomes the above challenges. *dyngraph2vec* uses multiple non-linear layers to learn structural patterns in each network. Furthermore, it uses recurrent layers to learn the temporal transitions in the network. The look back parameter in the recurrent layers controls the length of temporal patterns learned. We focus our experiments on the task of link prediction. We compare dyngraph2vec with the state-of-the-art algorithms for dynamic graph embedding and show its performance on several real world networks including collaboration networks and social networks. Our experiments show that using a deep model with recurrent layers can capture temporal dynamics of the networks and significantly outperform the state-of-the-art methods on link prediction.

Overall, our paper makes the following contributions:

1. We propose *dyngraph2vec*, a dynamic graph embedding model which captures temporal dynamics.
2. We demonstrate that capturing network dynamics can significantly improve the performance on link prediction.
3. We present variations of our model to show the key advantages and differences.
4. We publish a library, DynamicGEM [1], implementing the variations of our model and state-of-the-art dynamic embedding approaches.

Related Work

Graph representation learning techniques can be broadly divided into two categories: (i) static graph embedding, which represents each node in the graph with a single vector; and (ii) dynamic graph embedding, which considers multiple snapshots of a graph and obtains a time series of vectors for each node. Most analysis has been done on static graph embedding. Recently, however, some works have been devoted to studying dynamic graph embedding.

Static Graph Embedding

Methods to represent nodes of a graph typically aim to preserve certain properties of the original graph in the embedding space. Based on this observation, methods can be divided into: (i) distance preserving, and (ii) structure preserving. Distance preserving methods devise objective functions such that the distance between nodes in the original graph and the embedding space have similar rankings. For example, Laplacian Eigenmaps (Belkin and Niyogi 2001) minimizes the sum of the distance between the embeddings of neighboring nodes under the constraints of translational invariance, thus keeping the nodes close in the embedding space. Similarly, Graph Factorization (Ahmed et al. 2013) approximates the edge weight with the dot product of the nodes' embeddings, thus preserving distance in the inner product space. Recent methods have gone further to preserve higher order distances. Higher Order Proximity Embedding (HOPE) (Ou et al. 2016a) uses multiple higher order functions to compute a similarity matrix from a graph's adjacency matrix and uses Singular Value Decomposition (SVD) to learn the representation. GraRep (Cao, Lu, and Xu 2015) considers the node transition matrix and its higher powers to construct a similarity matrix.

On the other hand, structure preserving methods aim to preserve the roles of individual nodes in the graph. *node2vec* (Grover and Leskovec 2016) uses a combination of breadth first search and depth first search to find nodes similar to a node in terms of distance and role. Recently, deep learning methods to learn network representations have been proposed. These methods inherently preserve the higher order graph properties including distance and structure. SDNE (Wang, Cui, and Zhu 2016), DNGR (Cao, Lu, and Xu 2016) and VGAE (Kipf and Welling 2016b) use deep autoencoders for this purpose. Some other recent approaches use graph convolutional networks to learn inherent graph structure (Kipf and Welling 2016a; Bruna et al. 2013; Henaff, Bruna, and LeCun 2015).

Dynamic Graph Embedding

Embedding dynamic graphs is an emerging topic still under investigation. Some methods have been proposed to extend static graph embedding approaches by adding regularization (Zhu et al. 2016; Zhang et al. 2017). DynGEM (Goyal et al. 2017) uses the learned embedding from previous time step graphs to initialize the current time step embedding. Although it does not explicitly use regularization, such initialization implicitly keeps the new embedding close to the previous. DynamicTriad (Zhou et al. 2018) relaxes the temporal smoothness assumption but only considers patterns spanning two time steps. Our model uses recurrent layers to learn temporal patterns over long sequences of graphs and multiple fully connected layer to capture intricate patterns at each time step.

Motivating Example

We consider a toy example to motivate the idea of capturing network dynamics. Consider an evolution of graph G, $\mathcal{G} = \{G_1, .., G_T\}$, where G_t represents the state of graph at time t. The initial graph G_1 is generated using the Stochastic Block Model (Wang and Wong 1987) with 2 communities (represented by colors indigo and yellow in Figure 3), each with 500 nodes. The in-block and cross-block probabilities are set to 0.1 and 0.01 respectively. The evolution pattern can be defined as a three step process. In the first step (shown in Figure 3(a)), we randomly and uniformly select 10 nodes (colored red in Figure 3) from the yellow community. In step two (shown in Figure 3(b)), we randomly add 30 edges between each of the selected nodes in step one and random nodes in indigo community. This is similar to having more than cross-block probability but less than in-block probability. In step three (shown in Figure 3(c)), the community membership of the nodes selected in step 2 is changed

[1] www.anonymous-url.com

from yellow to indigo. Similarly, the edges (colored red in Figure 3) are either removed or added to reflect the cross-block and in-block connection probabilities. Then, for the next time step (shown in Figure 3(d)), the same three steps are repeated to evolve the graph. Informally, this can be interpreted as a two step movement of users from one community to another by initially increasing friends in the other community and subsequently moving to it.

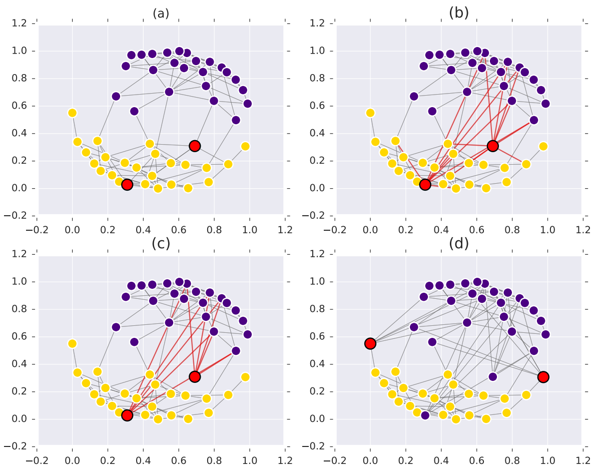


Figure 3: Motivating example of network evolution - community shift (for clarity, only showing 50 of 500 nodes and 2 out 10 migrating nodes).

Our task is to learn the embeddings predictive of the change in community of the 10 nodes. Figure 2 shows the results of the state-of-the-art dynamic graph embedding techniques (*DynGEM*, *optimalSVD*, and *DynamicTriad*) and the three variations of our model: *dyngraph2vecAE*, *dyngraph2vecRNN* and *dyngraph2vecAERNN* (see Methodology Section for the description of the methods). Figure 2 shows the embeddings of nodes after the first step of evolution. The nodes selected for community shift are colored in red. We show the results for 4 runs of the model to ensure robustness. Figure 2(a) shows that DynGEM brings the red nodes closer to the edge of yellow community but does not move any of the nodes to the other community. Similarly, DynamicTriad results in Figure 2(c) show that it only shifts 1 to 4 nodes to its actual community in the next step. The optimalSVD method in Figure 2(b) is not able to shift any nodes. However, our *dyngraph2vecAE* and *dyngraph2vecRNN*, and *dyngraph2vecAERNN* (shown in Figure 2(d-f)) successfully capture the dynamics and move the embedding of most of the 10 selected nodes to the indigo community, keeping the rest of the nodes intact. This shows that capturing dynamics is critical in understanding the evolution of networks.

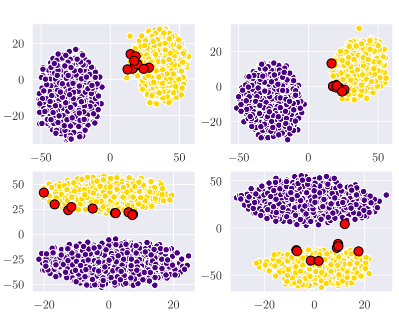
Methodology

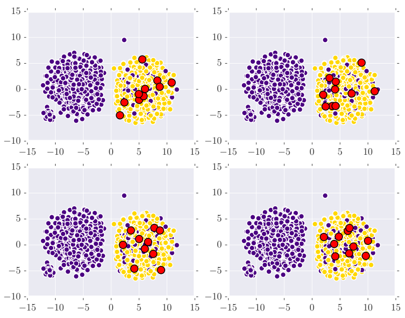
In this section, we define the problem statement. We then explain multiple variations of deep learning models capable of capturing temporal patterns in dynamic graphs. Finally, we design the loss functions and optimization approach.

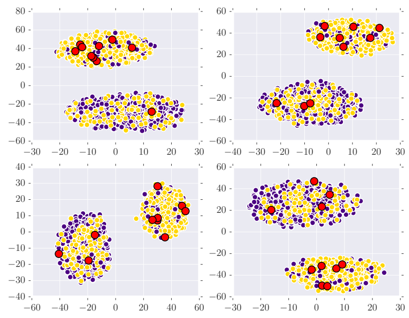
Problem Statement

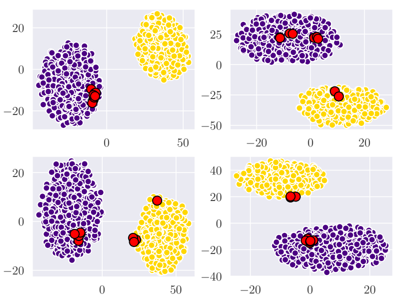
Consider a weighted graph $G(V, E)$, with V and E as the set of vertices and edges respectively. We denote the adjacency matrix of G by A, i.e. for an edge $(i, j) \in E$, A_{ij} denotes its weight, else $A_{ij} = 0$. An evolution of graph G is denoted as $\mathcal{G} = \{G_1, .., G_T\}$, where G_t represents the state of graph at time t.

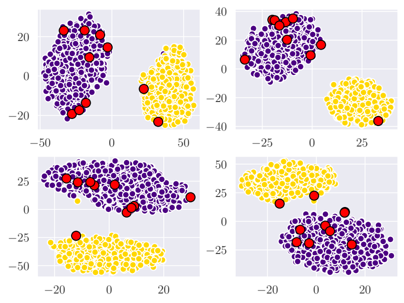
We define our problem as follows: *Given an evolution of graph G, $\mathcal{G}$, we aim to represent each node v in a series of low-dimensional vector space $y_{v_1}, \ldots y_{v_t}$ by learn-*


(a) DynGEM


(b) *optimalSVD*


(c) DynamicTriad


(d) *dyngraph2vecAE*


(e) dyngraph2vecRNN

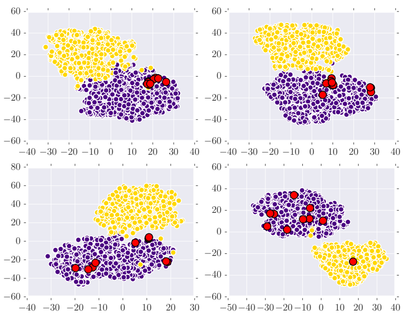
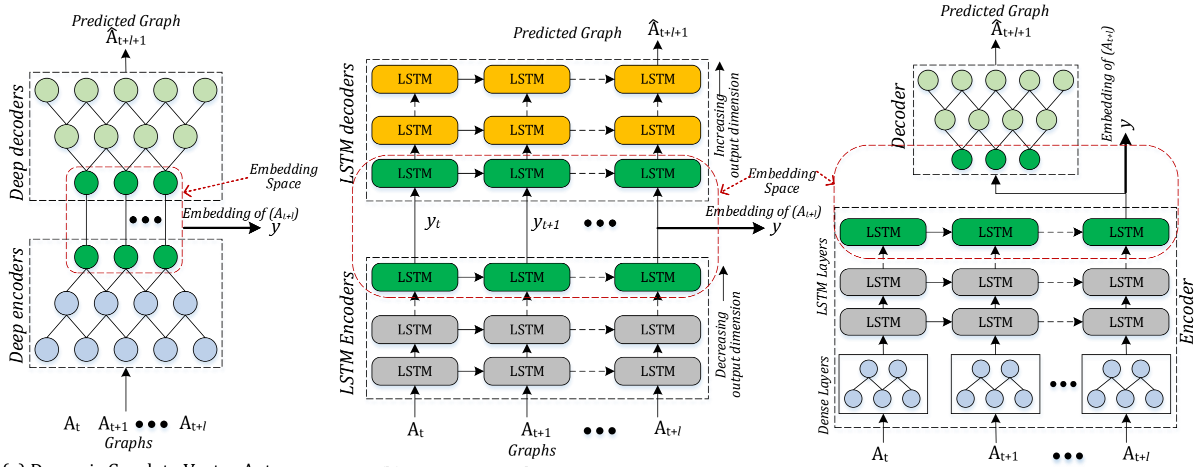

(f) dyngraph2vecAERNN

Figure 2: Motivating example of network evolution - community shift.



(a) Dynamic Graph to Vector Auto Encoder (**dyngraph2vecAE**) (b) Dynamic Graph to Vector Recurrent Neural Network (**dyngraph2vecRNN**) (c) Dynamic Graph to Vector Autoenncoder Recurrent Neural Network (**dyngraph2vecAERNN**)

Figure 4: dyngraph2vec architecture variations for dynamic graph embedding.

ing mappings $f_t : \{V_1, \ldots, V_t, E_1, \ldots E_t\} \rightarrow \mathbb{R}^d$ *and* $y_{v_i} = f_i(v_1, \ldots, v_i, E_1, \ldots E_i)$ *such that* y_{v_i} *can capture temporal patterns required to predict* $y_{v_{i+1}}$. In other words, the embedding function at each time step uses information from graph evolution to capture network dynamics and can thus predict links with higher precision.

dyngraph2vec

Our *dyngraph2vec* is a deep learning model that takes as input a set of previous graphs and generates as output the graph at the next time step, thus capturing highly non-linear interactions between vertices at each time step and across multiple time steps. The embedding thus learned is predictive of new links. The model learns the network embedding at time step t by optimizing the following loss function:

$$\begin{aligned} L_{t+l} &= \|(\hat{A}_{t+l+1} - A_{t+l+1}) \odot \mathcal{B}\|_F^2, \\ &= \|(f(A_t, \ldots, A_{t+l}) - A_{t+l+1}) \odot \mathcal{B}\|_F^2. \end{aligned} \quad (1)$$

Here we penalize the incorrect reconstruction of edges at time $t+l+1$ by using the embedding at time step $t+l$. The embedding at time step $t+d$ is a function of the graphs at time steps $t, t+1, \ldots, t+l$ where l is the temporal look back. We use a weighting matrix $\mathcal{B}$ to weight the reconstruction of observed edges higher than unobserved links as traditionally used in the literature (Wang, Cui, and Zhu 2016). Here, $\mathcal{B}_{ij} = \beta$ for $(i,j) \in E_{t+l+1}$, else 1.

We propose three variations of our model based on the architecture of deep learning models as shown in Figure 4: (i) *dyngraph2vecAE*, (ii) *dyngraph2vecRNN*, and (iii) *dyngraph2vecAERNN*. Our three methods differ in the formulation of the function $f(.)$.

To model the interconnection of nodes within and across time, our model *dyngraph2vecAE* uses multiple fully connected layers. Thus, for a node u with neighborhood vector set $u_{1..t} = [a_{u_t}, \ldots, a_{u_{t+l}}]$, the hidden representation of the first layer is learned as:

$$y_{u_t}^{(1)} = f_a(W_{AE}^{(1)} u_{1..t} + b^{(1)}), \quad (2)$$

where f_a is the activation function, $W_{AE}^{(1)} \in \mathbb{R}^{d^{(1)} \times nl}$ and $d^{(1)}$, n and l are the dimensions of representation learned by the first layer, number of nodes in the graph, and look back, respectively. The representation of the k^{th} layer is defined as:

$$y_{u_t}^{(k)} = f_a(W_{AE}^{(k)} y_{u_t}^{(k-1)} + b^{(k)}). \quad (3)$$

Note that *dyngraph2vecAE* has $O(nld^{(1)})$ parameters. As most real world graphs are sparse, learning the parameters can be challenging.

To reduce the number of model parameters and achieve a more efficient temporal learning, we propose *dyngraph2vecRNN* and *dyngraph2vecAERNN*. In *dyngraph2vecRNN* we use sparsely connected Long Short Term Memory (LSTM) networks to learn the embedding. LSTM is a type of Recurrent Neural Network (RNN) capable of handling long-term dependency problems. In dynamic graphs, there can be long-term dependencies which may not be captured by fully connected auto-encoders. The hidden state representation of a single LSTM network is defined as:

$$y_{u_t}^{(1)} = o_{u_t}^{(1)} * \tanh(C_{u_t}^{(1)}) \quad (4a)$$

$$o_{u_t}^{(1)} = \sigma_{u_t}(W_{RNN}^{(1)}[y_{u_{t-1}}^{(1)}, u_{1..t}] + b_o^{(1)}) \quad (4b)$$

$$C_{u_t}^{(1)} = f_{u_t}^{(1)} * C_{u_{t-1}}^{(1)} + i_{u_t}^{(1)} * \tilde{C}_{u_t}^{(1)} \quad (4c)$$

$$\tilde{C}_{u_t}^{(1)} = \tanh(W_C^{(1)}.[y_{u_{t-1}}^{(1)}, u_{1..t} + b_c^{(1)}]) \quad (4d)$$

$$i_{u_t}^{(1)} = \sigma(W_i^{(1)}.[y_{u_{t-1}}^{(1)}, u_{1..t}] + b_i^{(1)}) \quad (4e)$$

$$f_{u_t}^{(1)} = \sigma(W_f^{(1)}.[y_{u_{t-1}}^{(1)}, u_{1..t} + b_f^{(1)}]) \quad (4f)$$

where C_{u_t} represents the cell states of LSTM, f_{u_t} is the value to trigger the forget gate, o_{u_t} is the value to trigger the output gate, i_{u_t} represents the value to trigger the update gate of the LSTM, $\tilde{C}_{u_t}$ represents the new estimated candidate state, and b represents the biases. There can be l LSTM networks connected in the first layer, where the cell states and hidden representation are passed in a chain from $t-l$ to t LSTM networks. the representation of the k^{th} layer is then given as follows:

$$y_{u_t}^{(k)} = o_{u_t}^{(k)} * \tanh(C_{u_t}^{(k)}) \tag{5a}$$

$$o_{u_t}^{(k)} = \sigma_{u_t}(W_{RNN}^{(k)}[y_{u_{t-1}}^{(k)}, y_{u_t}^{(k-1)}] + b_o^{(k)}) \tag{5b}$$

The problem with passing the sparse neighbourhood vector $u_{1..t} = [a_{u_t}, \ldots, a_{u_{t+l}}]$ of node u to the LSTM network is that the LSTM model parameters (such as the number of memory cells, number of input units, output units, etc.) needed to learn a low dimension representation become large. Rather, the LSTM network may be able to better learn the temporal representation if the sparse neighbourhood vector is reduced to a low dimension representation. To achieve this, we propose a variation of *dyngraph2vec* model called *dyngraph2vecAERNN*. In *dyngraph2vecAERNN* instead of passing the sparse neighbourhood vector, we use a fully connected encoder to initially acquire low dimensional hidden representation given as follows:

$$y_{u_t}^{(p)} = f_a(W_{AERNN}^{(p)} y_{u_t}^{(p-1)} + b^{(p)}). \tag{6}$$

where p represents the output layer of the fully connected encoder. This representation is then passed to the LSTM networks.

$$y_{u_t}^{(p+1)} = o_{u_t}^{(p+1)} * \tanh(C_{u_t}^{(p+1)}) \tag{7a}$$

$$o_{u_t}^{(p+1)} = \sigma_{u_t}(W_{AERNN}^{(p+1)}[y_{u_{t-1}}^{(p+1)}, y_{u_t}^{(p)}] + b_o^{(p+1)}) \tag{7b}$$

Then the hidden representation generated by the LSTM network is passed to a fully connected decoder.

Optimization

We optimize the loss function defined above to get the optimal model parameters. By applying the gradient with respect to the decoder weights on equation 1, we get:

$$\frac{\partial L_t}{\partial W_*^{(K)}} = [2(\hat{A}_{t+1} - A_{t+1}) \odot \mathcal{B}][\frac{\partial f_a(Y^{(K-1)} W_*^{(K)} + b^{(K)})}{\partial W_*^{(K)}}],$$

where $W_*^{(K)}$ is the weight matrix of the penultimate layer for all the three models. For each individual model, we back propagate the gradients based on the neural units to get the derivatives for all previous layers. For the LSTM based *dyngraph2vec* models, back propagation through time is performed to update the weights of the LSTM networks.

After obtaining the derivatives, we optimize the model using stochastic gradient descent (SGD) (Rumelhart, Hinton, and Williams 1988) with Adaptive Moment Estimation (Adam)(Kingma and Ba 2014).

Experiments

In this section, we describe the data sets used and establish the baselines for comparison. Furthermore, we define the evaluation metrics for our experiments and parameter settings. All the experiments were performed on a 64 bit Ubuntu 16.04.1 LTS system with Intel (R) Core (TM) i9-7900X CPU with 19 processors, 10 CPU cores, 3.30 GHz CPU clock frequency, 64 GB RAM, and two Nvidia Titan X, each with 12 GB memory.

Table 1: Dataset Statistics

Name	SBM	Hep-th	AS
Nodes n	1000	150-14446	7716
Edges m	56016	268-48274	487-26467
Time steps T	10	136	733

Datasets

We conduct experiments on two real-world datasets and a synthetic dataset to evaluate our proposed algorithm. The datasets are summarized in Table 1.

Stochastic Block Model (SBM) - community diminishing: In order to test the performance of various static and dynamic graph embedding algorithms, we generated synthetic SBM data with two communities and total of 1000 nodes. The cross-block connectivity probability is 0.01 and in-block connectivity probability is set to 0.1. One of the communities is continuously diminished by migrating the 10-20 nodes to the other community. A total of 10 dynamic graphs are generated for the evaluation.

Hep-th (Gehrke, Ginsparg, and Kleinberg 2003): The first real world data set used to test the dynamic graph embedding algorithms is the collaboration graph of authors in High Energy Physics Theory conference. The original data set contains abstracts of papers in High Energy Physics Theory conference in the period from January 1993 to April 2003. For our evaluation, we consider the last 50 snapshots of this dataset.

Autonomous Systems (AS) (Leskovec, Kleinberg, and Faloutsos 2005): The second real world dataset utilized is a communication network of who-talks-to-whom from the BGP (Border Gateway Protocol) logs. The dataset contains 733 instances spanning from November 8, 1997 to January 2, 2000. For our evaluation, we consider a subset of this dataset which contains the last 50 snapshots.

Baselines

We compare our model with the following state-of-the-art static and dynamic graph embedding methods:

- *Optimal Singular Value Decomposition* (**OptimalSVD**) (Ou et al. 2016b): It uses the singular value decomposition of the adjacency matrix or its variation (i.e., the transition matrix) to represent the individual nodes in the graph. The low rank SVD decomposition with largest d singular values are then used for graph structure matching, clustering, etc.
- *Incremental Singular Value Decomposition* (**IncSVD**) (Brand 2006): It utilizes a perturbation matrix which captures the changing dynamics of the graphs and performs additive modification on the SVD.
- *Rerun Singular Value Decomposition* (**RerunSVD** or TIMERS) (Zhang et al. 2017): It utilizes the incremental SVD to get the dynamic graph embedding, however, it also uses a tolerance threshold to restart the optimal SVD calculation when the incremental graph embedding starts to deviate.

- *Dynamic Embedding using Dynamic Triad Closure Process* (**dynamicTriad**) (Zhou et al. 2018): It utilizes the triadic closure process to generate a graph embedding that preserves structural and evolution patterns of the graph.
- *Deep Embedding Method for Dynamic Graphs* (**dynGEM**) (Goyal et al. 2018b): It utilizes deep auto-encoders to incrementally generate embedding of a dynamic graph at snapshot t by using only the snapshot at time $t-1$.

Evaluation Metrics

In our experiments, we evaluate our model on link prediction at time step $t+1$ by using all graphs until the time step t . We use $precision@k$ and Mean Average Precision (MAP) as our metrics. $precision@k$ is the fraction of correct predictions in the top k predictions. It is defined as $P@k = \frac{|E_{pred}(k) \cap E_{gt}|}{k}$, where E_{pred} and E_{gt} are the predicted and ground truth edges respectively. MAP averages the precision over all nodes. It can be written as $\frac{\sum_i AP(i)}{|V|}$ where $AP(i) = \frac{\sum_k precision@k(i) \cdot \mathbb{I}\{E_{pred_i}(k) \in E_{gt_i}\}}{|\{k : E_{pred_i}(k) \in E_{gt_i}\}|}$ and $precision@k(i) = \frac{|E_{pred_i}(1:k) \cap E_{gt_i}|}{k}$.

Results And Analysis

In this section we present performance result of various models for link prediction on different datasets.

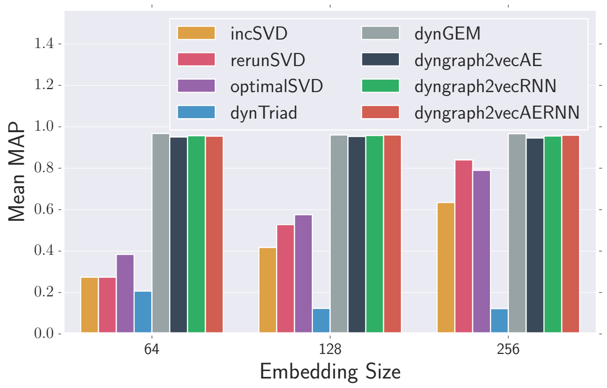


Figure 5: MAP values for the SBM dataset.

SBM Dataset

The MAP values for various algorithms with SBM dataset with diminishing community is shown in Figure 5. The MAP values shown are for link prediction with embedding sizes *64*, *128* and *256*. This figure shows that our methods *dyngraph2vecAE*, *dyngraph2vecRNN* and *dyngraph2vecAERNN* all have higher MAP values compared to the rest of the base-lines except for *dynGEM*. The *dynGEM* algorithm is able to have higher MAP values than all the algorithms. This is due to the fact that *dynGEM* also generates the embedding of graph at snapshot $t+1$ using the graph at snapshot t. Since in our SBM dataset the node-migration criteria are introduced only one time step earlier, the *dynGEM* node embedding technique is able to capture these dynamics. Notice that the MAP values of SVD based methods increase as the embedding size increases. However, this is not the case for *dynTriad*.

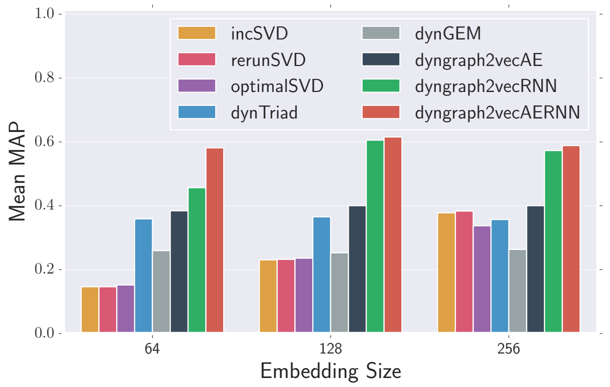


Figure 6: MAP values for the Hep-th dataset.

Hep-th Dataset

The link prediction results for the Hep-th dataset is shown in Figure 6. The proposed *dyngraph2vec* algorithms outperform all the other state-of-the-art static and dynamic algorithms. Among the proposed algorithms, *dyngraph2vecAERNN* has the highest MAP values, followed by *dyngraph2vecRNN* and *dyngraph2vecAE*, respectively. The *dynamicTriad* is able to perform better than the SVD based algorithms. Notice that *dynGEM* is not able to have higher MAP values than the *dyngraph2vec* algorithms in the Hep-th dataset.

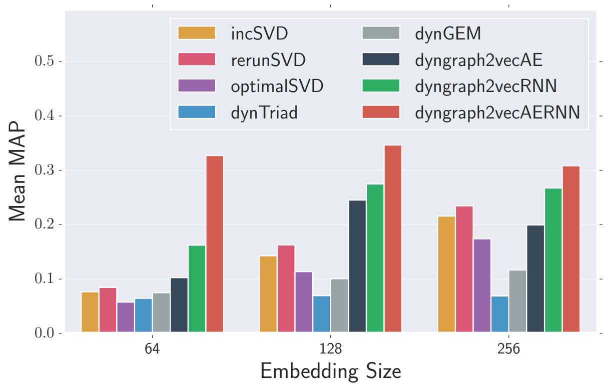


Figure 7: MAP values for the AS dataset.

AS Dataset

The MAP value for link prediction with various algorithms for the AS dataset is shown in Figure 7. *dyngraph2vecAERNN* outperforms all the state-of-the-art algorithms. The algorithm with second highest MAP score is *dyngraph2vecRNN*. However, *dyngraph2vecAE* has a higher MAP only with a lower embedding of size 64. SVD methods are able to improve their MAP values by increasing the embedding size. However, they are not able to outperform the *dyngraph2vec* algorithms.

Precision@k and MAP exploration

The average $precision@k$ values over all embedding sizes for various datasets are shown in Table 2. The proposed *dyngraph2vec* algorithms generally have higher $precision@k$

values at lower k while having overall higher MAP values. On the other hand, other algorithms have higher $precision@k$ values at lower k, but have lower MAP values calculated over the entire graph nodes.

The summary of MAP values for different embedding sizes (64, 128 and 256) for different datasets is presented in Table 3. The top three highest MAP values are highlighted in bold. For the synthetic SBM dataset, the top three algorithms with highest MAP values are *dynGEM*, *dyngraph2VecAERNN*, and *dyngraph2vecRNN*, respectively. For the Hep-th dataset, the top three algorithm with highest MAP values are *dyngraph2VecAERNN*, *dyngraph2VecRNN*, and *dyngraph2VecAE*, respectively. For the AS dataset, the top three algorithm with highest MAP values are *dyngraph2VecAERNN*, *dyngraph2VecRNN*, and *dyngraph2VecAE*, respectively. These results show that the dyngraph2vec variants are able to capture the graph dynamics much better than the most of the state-of-the-art algorithms in general.

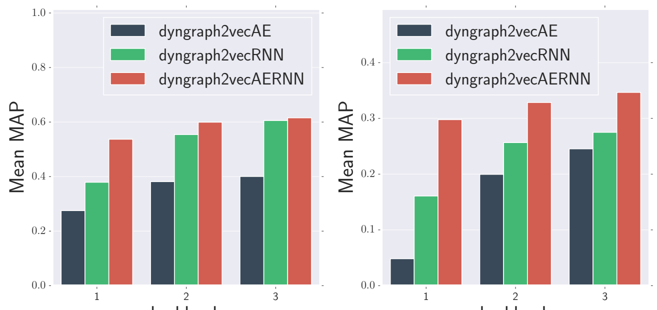
Table 3: Average MAP values over different embedding sizes.

	Average MAP		
Method	SBM	Hep-th	AS
IncrementalSVD	0.4421	0.2518	0.1452
rerunSVD	0.5474	0.2541	0.1607
optimalSVD	0.5831	0.2419	0.1152
dynamicTriad	0.1509	0.3606	0.0677
dynGEM	**0.9648**	0.2587	0.0975
dyngraph2vecAE	0.9500	**0.3951**	**0.1825**
dyngraph2vecRNN	**0.9567**	**0.5451**	**0.2350**
dyngraph2vecAERNN	**0.9581**	**0.5952**	**0.3274**

Hyper-parameter Sensitivity: Look back

One of the important parameters for time-series analysis is how much in the past the method looks to predict the future. To analyze the affect of look back on the MAP score we have trained the *dyngraph2Vec* algorithms with various look back values. The embedding dimension is fixed to 128. The look back size is varied from 1 to 3 with a step size of 1. We then tested the change in MAP values with the real word datasets.

Figure 8 presents the results of look back variation. These results show that MAP scores increase as the look back parameter is increased. The highest MAP value of **0.6155** is achieved for the Hep-th dataset by *dyngraph2VecAERNN* with the look back of 3. Similarly, highest MAP value of **0.3464** is achieved for Hep-th dataset by *dyngraph2VecAERNN* with the look back of 3.



(a) Hep Dataset (b) AS Dataset

Figure 8: Mean MAP values for various algorithms with different look backs.

Discussion

Other Datasets: We have validated our algorithms with a synthetic dynamic SBM and two real world datasets including Hep-th and AS. We leave the test on further datasets as future work.

Hyper-parameters: Currently, we provided the evaluation of the proposed algorithm with embedding size of 64, 128 and 256. We leave the exhaustive evaluation of the proposed algorithms for broader ranges of embedding size and look back size for future work.

Evaluation: We have demonstrated effectiveness of the proposed algorithms for predicting the links of the next time step. However, in dynamic graph networks there are various evaluations such as node classification that can be performed. We leave them as our future work.

Conclusion

This paper introduced dyngraph2vec, a model for capturing temporal patterns in dynamic networks. It learns the evolution patterns of individual nodes and provides an embedding capable of predicting future links with higher precision. We propose three variations of our model based on the architecture with varying capabilities. The experiments show that our model can capture temporal patterns on synthetic and real datasets and outperform state-of-the-art methods in link prediction. There are several directions for future work: (1) interpretability by extending the model to provide more insight into network dynamics and better understand tempo-

Table 2: Average Precision@k over different embedding sizes.

	SBM			**Hep-th**			**AS**		
Method	P@100	P@500	P@1000	P@100	P@500	P@1000	P@100	P@500	P@1000
IncrementalSVD	0.9881	0.9832	0.9152	0.9835	0.9578	0.8919	0.9524	0.9468	0.9433
rerunSVD	0.9967	0.9897	0.9248	0.9842	0.9589	0.8932	0.9602	0.9596	0.9578
optimalSVD	0.9996	0.9879	0.9176	1.0000	0.9856	0.9140	0.8290	0.7397	0.6988
dynamicTriad	0.1044	0.1096	0.1047	0.6663	0.5340	0.4805	0.8665	0.8543	0.8024
dynGEM	0.9633	0.9656	0.9673	1.0000	0.9990	0.9784	0.9321	0.9448	0.9377
dyngraph2vecAE	0.9800	0.9851	0.9869	0.9755	0.9638	0.92080	0.8007	0.8028	0.7546
dyngraph2vecRNN	0.9927	0.9905	0.9898	0.8741	0.8827	0.8836	0.8514	0.7955	0.7768
dyngraph2vecAERNN	0.9800	0.9887	0.9917	0.9971	0.9917	0.9785	0.8591	0.8620	0.8577

ral dynamics; (2) automatic hyperparameter optimization for higher accuracy; and (3) graph convolutions to learn from node attributes and reduce the number of parameters.

References

[Ahmed et al. 2013] Ahmed, A.; Shervashidze, N.; Narayanamurthy, S.; Josifovski, V.; and Smola, A. J. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, 37–48. ACM.

[Belkin and Niyogi 2001] Belkin, M., and Niyogi, P. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, 585–591.

[Brand 2006] Brand, M. 2006. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications* 415(1):20–30.

[Bruna et al. 2013] Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

[Cao, Lu, and Xu 2015] Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *KDD15*, 891–900.

[Cao, Lu, and Xu 2016] Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 1145–1152. AAAI Press.

[Freeman 2000] Freeman, L. C. 2000. Visualizing social networks. *Journal of social structure* 1(1):4.

[Gehrke, Ginsparg, and Kleinberg 2003] Gehrke, J.; Ginsparg, P.; and Kleinberg, J. 2003. Overview of the 2003 kdd cup. *ACM SIGKDD Explorations* 5(2).

[Goyal and Ferrara 2018] Goyal, P., and Ferrara, E. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*.

[Goyal et al. 2017] Goyal, P.; Kamra, N.; He, X.; and Liu, Y. 2017. Dyngem: Deep embedding method for dynamic graphs. In *IJCAI International Workshop on Representation Learning for Graphs*.

[Goyal et al. 2018a] Goyal, P.; Hosseinmardi, H.; Ferrara, E.; and Galstyan, A. 2018a. Embedding networks with edge attributes. In *Proceedings of the 29th on Hypertext and Social Media*, 38–42. ACM.

[Goyal et al. 2018b] Goyal, P.; Kamra, N.; He, X.; and Liu, Y. 2018b. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*.

[Goyal, Sapienza, and Ferrara 2018] Goyal, P.; Sapienza, A.; and Ferrara, E. 2018. Recommending teammates with deep neural networks. In *Proceedings of the 29th on Hypertext and Social Media*, 57–61. ACM.

[Grover and Leskovec 2016] Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, 855–864. ACM.

[Henaff, Bruna, and LeCun 2015] Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

[Kingma and Ba 2014] Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kipf and Welling 2016a] Kipf, T. N., and Welling, M. 2016a. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[Kipf and Welling 2016b] Kipf, T. N., and Welling, M. 2016b. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

[Leskovec, Kleinberg, and Faloutsos 2005] Leskovec, J.; Kleinberg, J.; and Faloutsos, C. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 177–187. ACM.

[Ou et al. 2016a] Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016a. Asymmetric transitivity preserving graph embedding. In *Proc. of ACM SIGKDD*, 1105–1114.

[Ou et al. 2016b] Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016b. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 1105–1114. ACM.

[Pavlopoulos, Wegener, and Schneider 2008] Pavlopoulos, G. A.; Wegener, A.-L.; and Schneider, R. 2008. A survey of visualization tools for biological network analysis. *Biodata mining* 1(1):12.

[Perozzi, Al-Rfou, and Skiena 2014] Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings 20th international conference on Knowledge discovery and data mining*, 701–710.

[Rumelhart, Hinton, and Williams 1988] Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1988. Neurocomputing: Foundations of research. *JA Anderson and E. Rosenfeld, Eds* 696–699.

[Tang et al. 2015] Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings 24th International Conference on World Wide Web*, 1067–1077.

[Theocharidis et al. 2009] Theocharidis, A.; Van Dongen, S.; Enright, A.; and Freeman, T. 2009. Network visualization and analysis of gene expression data using biolayout express3d. *Nature protocols* 4:1535–1550.

[Wang and Wong 1987] Wang, Y. J., and Wong, G. Y. 1987. Stochastic blockmodels for directed graphs. *Journal of the American Statistical Association* 82(397):8–19.

[Wang, Cui, and Zhu 2016] Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, 1225–1234. ACM.

[Wasserman and Faust 1994] Wasserman, S., and Faust, K. 1994. *Social network analysis: Methods and applications*, volume 8. Cambridge university press.

[Zhang et al. 2017] Zhang, Z.; Cui, P.; Pei, J.; Wang, X.; and Zhu, W. 2017. Timers: Error-bounded svd restart on dynamic networks. *arXiv preprint arXiv:1711.09541*.

[Zhou et al. 2018] Zhou, L.; Yang, Y.; Ren, X.; Wu, F.; and Zhuang, Y. 2018. Dynamic Network Embedding by Modelling Triadic Closure Process. In *AAAI*.

[Zhu et al. 2016] Zhu, L.; Guo, D.; Yin, J.; Ver Steeg, G.; and Galstyan, A. 2016. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering* 28(10):2765–2777.