Learning Dynamic Context Graph Embedding

Chuanchang Chen CHENCHUANCHANG@ZJU.EDU.CN
Yubo Tao * TAOYUBO@CAD.ZJU.EDU.CN
Hai Lin LIN@CAD.ZJU.EDU.CN
State Key Lab of CAD&CG, Zhejiang University

Editors: Sinno Jialin Pan and Masashi Sugiyama

Abstract

Graph embeddings represent nodes as low-dimensional vectors to preserve the proximity between nodes and communities of graphs for network analysis. The temporal edges (e.g., relationships, contacts, and emails) in dynamic graphs are important for graph evolution analysis, but few existing methods in graph embeddings can capture the dynamic information from temporal edges. In this study, we propose a dynamic graph embedding method to analyze the evolution patterns of dynamic graphs effectively. Our method uses diffuse context sampling to preserve the proximity between nodes, and applies dynamic context graph embeddings to train discrete-time graph embeddings in the same vector space without alignments to preserve the temporal continuity of stable nodes. We compare our method with several state-of-the-art methods for link prediction, and the experiments demonstrate that our method generally performs better at the task. Our method is further verified using a real-world dynamic graph by visualizing the evolution of its community structure at different timesteps.

Keywords: Dynamic graphs, dynamic graph embeddings, context embedding

1. Introduction

Graphs describe the relationships between entities, such as social relationships within a population, interactions between biological proteins, and co-occurrence relationships between words. Dynamic graphs are very common in many application domains in which entities and connections change over time, such as email graphs and instant messaging graphs. Analyzing these graphs can provide insight into evolution patterns in these domains. Recently, dynamic graph analysis has received considerable attention, and many embedding-based approaches have been proposed for temporal link prediction Nguyen et al. (2018), node prediction Zhou et al. (2018), and multi-label node classification Ma et al. (2018). Thus, utilizing embeddings to investigate graph evolution is a promising research direction.

Previous methods for graph embeddings, such as matrix factorization Cao et al. (2015), random walk Perozzi et al. (2014); Grover and Leskovec (2016); Ribeiro et al. (2017), deep neural network Cao et al. (2016); Wang et al. (2016); Ni et al. (2018); Dai et al. (2018), and many others Tang et al. (2015); Gu et al. (2018), generally focus on static graphs to preserve the proximity between nodes. These methods fail to capture the temporal information in dynamic graphs. Recently, continuous-time dynamic graph embeddings Nguyen et al. (2018)

* Corresponding author: taoyubo@cad.zju.edu.cn

have been used to learn continuous changes in temporal graphs via temporal walk, which is a walking strategy with a time-ascending order. However, this method represents all graph information in one embedding and cannot effectively capture the temporal changes of nodes over time. DynGEM Kamra et al. (2017) uses a deep autoencoder model to learn discrete-time dynamic graph embeddings incrementally. DynTriad Zhou et al. (2018) seeks to train adjacent graphs in a dynamic graph jointly into a sequence of embeddings by imposing a *triad*. However, these discrete-time dynamic graph embedding methods focus only on learning representations from adjacent snapshots and ignore the latent context information behind all snapshots.

In this study, we attempt to learn discrete-time graph embeddings for dynamic graphs. Each node has the same number of low-dimensional vectors as the number of timesteps, and both the proximity and temporal continuity of each node are preserved in the dynamic graph embedding. A direct method to solve this problem is to learn each graph separately via static graph embedding methods, and then align all graph embeddings into the same vector space by alignment methods Hamilton et al. (2016). However, it is challenging to align these embeddings, because of nonlinear movements of evolving nodes, and such alignment error would reduce the performance of downstream tasks. Therefore, we propose a **D**ynamic **C**ontext Graph **E**mbedding (DCGE) method, based on Bayesian network theory and Bernoulli prior probability, to learn a sequence of snapshots in a dynamic graph and generate continuous embedded vectors for nodes in the same vector space, without additional embedding alignments. DCGE can capture how each node changes from one snapshot to the next by jointly learning context embeddings, which embed the context information of each node through all snapshots as the sharing parameters. Generally, this study makes the following contributions:

- We propose a diffuse context sampling algorithm based on Bayesian network theory, and a novel dynamic graph embedding method that incorporates latent context information into graph embedding learning. The context embeddings can be learned through multiple snapshots jointly, to retain temporal context information.
- Our method is more effective for link prediction than other state-of-the-art techniques, on both continuous-time and discrete-time graphs.
- Our method can be used to analyze and visualize the evolution of a graph while preserving the temporal continuity of stable nodes and community structure over time.

2. Related Work

With the rise of social networks (e.g., Facebook and Twitter) and big data (millions or billions of interaction records), graph embedding methods have attracted considerable attention from both industrial and academic researchers. The key point of graph embeddings is to learn a low-dimensional vector representation for each node and to preserve the proximity between nodes, which can be used for many downstream tasks, including node classification Perozzi et al. (2014), link prediction Ou et al. (2016), node clustering Wang

et al. (2017), anomaly detection Hu et al. (2016), and collaboration prediction Chen and Sun (2017).

Some graph embedding methods are based on matrix factorization Cao et al. (2015); these construct a k-step transition probability matrix to measure node similarity at different scales. Inspired by the good performance of word2vec in natural language processing, researchers have incorporated random walk into the skip-gram model Mikolov et al. (2013a) to learn graph embeddings; such methods include DeepWalk Perozzi et al. (2014) and node2vec Grover and Leskovec (2016). These methods use random walk to produce a series of node sequences as the text and apply the skip-gram model to learn graph representations. Struc2vec Ribeiro et al. (2017) focuses on the structural identities of nodes and constructs a weighted multilayer graph for random walk to capture the hierarchical structural similarity. Recently, considerable attention has been attracted by some embedding methods based on deep neural networks Wang et al. (2016); Cao et al. (2016); Ni et al. (2018); Hamilton et al. (2017); Veličković et al. (2018) for learning nonlinear mapping functions. To enhance the robustness of representations, Dai et al. employed generative adversarial graphs to capture latent features in graph embeddings Dai et al. (2018).

Most existing graph embedding methods focus only on static graphs, but learning dynamic graph embedding is an active research topic. CTDNE Nguyen et al. (2018) incorporates temporal information into existing graph embedding methods based on random walk by introducing a time-series order. DynGEM Kamra et al. (2017) combines deep autoencoders with a layer expansion to generate embeddings, and can handle dynamic graphs with a growing size. DynTriad Zhou et al. (2018) focuses on a local structure, called a triad, to learn the proximity information and evolution patterns. DepthLGP Ma et al. (2018) tackles the issue of updating out-of-sample nodes into graph embeddings by combining a probabilistic model with deep learning.

Existing methods focus only on a static graph to consider the influence of neighbors Tang et al. (2015); Hamilton et al. (2017); Veličković et al. (2018), or process predefined motifs in temporal graphs Zhou et al. (2018). In contrast, DCGE learns the global context embeddings by combining them with local context information through the dynamic network with multiple snapshots. In this manner, the context characteristics of nodes can be used to generate reasonable graph embeddings.

3. Problem Definition

In this study, we seek to solve the problems of the proximity and temporal representation of dynamic graphs. A dynamic graph is defined as follows:

Definition 1 (Dynamic Graph) *A dynamic graph is a series of graphs* $\Gamma(V, E) = \{G_1, ..., G_T\}$, *where* $G_t = (V_t, E_t)$, V *and* E *include all nodes and edges in* Γ, T *is the number of graphs,* $V_t \subseteq V$ *is a node set, and* $E_t \subseteq E$ *includes all temporal edges within the timespan* $[P_t, P_{t+1}]$. *Each* $e_i = (u, v, p_i) \in E_t$ *is a temporal edge between the node* $u \in V_t$ *and the node* $v \in V_t$ *at time* $p_i \in [P_t, P_{t+1}]$.

Our goal is to learn dynamic graph embeddings $Y_1, ..., Y_T$ and $Y_t \in R^{|V_t| \times D}$, where $|V_t|$ is the number of nodes at timestep t and D is the dimension of the embeddings. The concept of dynamic graph embeddings is defined formally as follows:

Definition 2 (Dynamic Graph Embedding) *Given a dynamic graph* $\Gamma(V, E) = \{G_1, ..., G_T\}$, *a dynamic graph embedding projects each node* $u \in V_t$ *into a low-dimensional vector space by a mapping function* $f_g : u \mapsto y_u^{(t)} \in R^D, D \ll |V_t|, t \in [1, T]$.

Each embedding vector $y_v^{(t)}$ is a representation of node v in graph G_t, and different representations of node v are distinct, because of the various graph structures in each snapshot. Even though a dynamic graph changes over time, the properties of each node that interacts with the other nodes are mostly stable. To use this context information in different snapshots sufficiently, we introduce the definition of context graph embedding as follows:

Definition 3 (Context Graph Embedding) *Given a dynamic graph* $\Gamma(V, E) = \{G_1, ..., G_T\}$, *a context graph embedding encodes all of the local context information of a node* $u \in V$ *from each temporal graph* G_t *to a single low-dimensional vector by a mapping function* $f_c : u \mapsto c_u \in R^D, D \ll |V|$. *Each node has a unique context graph embedding.*

A dynamic embedding matrix $Y_t \in R^{|V_t| \times D}$ is used to represent the proximity and temporal properties for each graph G_t, and a context graph embedding matrix $C \in R^{|V| \times D}$ represents the temporal context information of each node. The dynamic graph embedding and context graph embedding generally satisfy the following properties:

- **Proximity Preservation**. The embeddings should preserve the proximity between nodes, i.e., if nodes u and v are connected, or have similar neighbor nodes at timestep t, $y_u^{(t)}$ and $y_v^{(t)}$ should be located nearby in the vector space.
- **Context Consistency**. Although dynamic graphs include multiple snapshots, c_v should retain the stable context properties of node v as the neighbor of other nodes. For example, if $(v, u) \in E_t$ and $(v, w) \notin E_t$, $\langle c_v, y_u^t \rangle > \langle c_v, y_w^t \rangle$, where $\langle a, b \rangle$ is the inner product of a and b. Moreover, if $(v, u) \in E_t$ and $(v, u) \notin E_{t+1}$, $\langle c_v, y_u^t \rangle > \langle c_v, y_u^{t+1} \rangle$.
- **Temporal Continuity**. The embeddings should preserve the temporal similarity of stable nodes, i.e., if a node u has similar neighbors at timesteps t and $t + 1$, $y_u^{(t)}$ and $y_u^{(t+1)}$ should be located nearby in the vector space.

Our DCGE model is designed to preserve the characteristics mentioned above.

4. Proposed Method

Our proposed method, DCGE, includes two main components: diffuse context sampling and dynamic graph embedding learning. Diffuse context sampling, based on Bayesian network theory, aims to generate the context for each node, to capture the desirable context for each node in every snapshot. We use the Bernoulli prior probability model to learn the adjacent similarity (direct connection between two nodes) and context similarity (internal connection between a node and its context) jointly, to embed the dynamic graph.

The remainder of this section is organized as follows. First, we introduce a diffuse context sampling algorithm based on Bayesian network theory. Second, we discuss how to take account of both adjacent similarity and context similarity with the Bernoulli prior probability model, to ensure that our dynamic graph embeddings are aligned naturally.

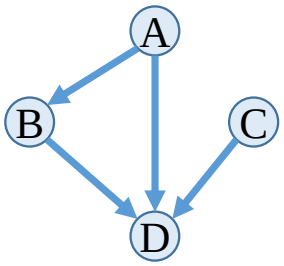


Figure 1: An example of a Bayesian network, which is a DAG.

4.1. Diffuse Context Sampling

Context is a common concept in natural language processing (NLP) that refers to the adjacent words of a word in a sentence. Deepwalk Perozzi et al. (2014) introduces the context of nodes into graph analysis by using random walk combined with the skip-gram model, but assumes that the context is independent. In a graph, the context is an h-hop neighbor set of the target node v (for example, like a subgraph without v). Intuitively, the neighbors of node v are strongly interrelated, and influence node v by a complex diffusion process. Generally, we can compute the multivariate dependent joint conditional probability distribution as

$$P(x_1, x_2, ..., x_n) = P(x_1)P(x_2|x_1)...P(x_n|x_1, x_2, ..., x_{n-1}). \tag{1}$$

Because the joint distribution between different variables is complex, it is difficult to compute $P(x_1, x_2, ..., x_n)$ directly. In a *Bayesian network*, each node stores the joint conditional probabilities of all direct parent nodes and is independent of the other (indirect) parent nodes in a directed acyclic graph (DAG). As shown in Figure 1, the other nodes influence the target node via cascading transmission. A Bayesian network assumes that the conditional probability of the target node depends only on its parents. As a consequence of this property, Equation 1 can be simplified to

$$P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i|Parents(x_i)). \tag{2}$$

Intuitively, the transfer probabilities of a node from its parents are truly different; that is, the influence of celebrities is more powerful than the influence of ordinary people. Most graphs express this information by the weights on the edges. Previous work, such as PageRank Page (1997), use the node degree to estimate the importance of each node (page). We use the weight proportion as the empirical distribution to compute the transfer probability of edge (v, u) from node v to node u, and the conditional probability is

$$P(x|v) = \frac{w_{v,x}}{\sum_{(v,u)\in E} w_{u,v}}, \tag{3}$$

where $w_{u,v}$ is the weight of edge (u, v). We use the above formula to measure the transfer probability in the Bayesian network. We define $P(x_i|Parents(x_i))$ as

$$P(x_i|Parents(x_i)) = \prod_{v\in Parents(x_i)} P(x_i|v). \tag{4}$$

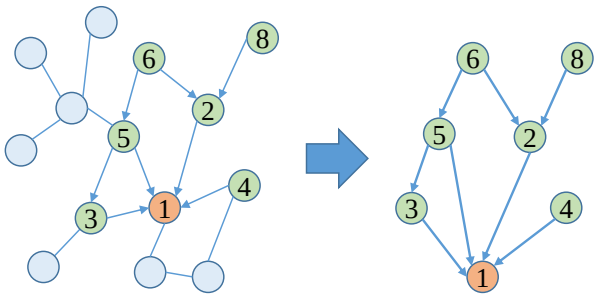


Figure 2: An example of reconstructing a context as a Bayesian network. The conditional probability of node 1 can be computed as $P(x = 1|Parents(1)) = P(x = 1|x = 3)P(x = 1|x = 5)P(x = 1|x = 2)P(x = 1|x = 4)$.

Therefore, we can easily combine Equation 1 with the equations above:

$$P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} \prod_{v \in Parents(x_i)} \frac{w_{v,x_i}}{\sum_{(v,u)\in E} w_{u,v}}. \tag{5}$$

For example, as shown in Figure 3, an impact diffusion process can be reconstructed as a DAG. Through such a transfer relationship, the relation between a node and its context is more interpretable. Therefore, we use the proposed diffusion context sampling method, based on Bayesian networks, to generate high-quality contexts for each node.

First, we initialize the context set $S(v) = v$. Second, we randomly select a node u from $S(v)$ and compute the transition probability of each edge $(u, x) \in E$ as Equation 3. The next node is chosen by Equation 3 and we add node x to $S(v)$. Again, we randomly select another node from $S(v)$ and repeat this process until the size of $S(v)$ is $k + 1$. Finally, we remove node v from $S(v)$ and receive the context. Notice that our method can also be applied to an undirected graph by converting each undirected edge to two directed edges. The detailed pseudocode for context sampling is presented in Algorithm 1.

Algorithm 1 Diffuse Context Sampling

Input: k the maximum number of nodes
$G_1, ..., G_T$ the snapshots to sample
Output: Context Set S
$S = \{\}$
for $t \in (1, T)$ **do**
  for each $v \in V_t$ **do**
    $S(v) = \{v\}$
    while size($S(v)$) $\leq$ k+1 **do**
      Randomly select node $u \in S(v)$
      Select edge $(x, u) \in E_t$ as Eq.4
      Add x to $S(v)$
    end while
    Add $S(v) - v$ to S
  end for
end for

4.2. Dynamic Context Graph Embedding

In this section, we introduce how to learn dynamic graph embeddings with the characteristics mentioned in Section 3. To preserve the proximity between nodes, we introduce the concepts of neighbor similarity loss and context similarity loss. First, neighbor similarity suggests that connected nodes should be embedded closely in the latent representation space. Therefore, we compute each pair of connected nodes to estimate the neighbor similarity. Second, the context similarity loss solves two requirements in learning dynamic graph embeddings. One is the alignment requirement, which forces graph embeddings in different snapshots to be aligned into the same vector space. The other requirement is to learn the context property of nodes through all snapshots, to use the context information adequately. Moreover, we assume that a dynamic graph will evolve smoothly over time, so graph embeddings in adjacent snapshots should be smoothed by our temporal smoothness loss. We discuss each type of loss mentioned above in the following.

Adjacent proximity, or first-order proximity, expresses the direct relationship between two nodes connected by an edge. A connection between two entities could represent an interaction, a phone call, or a friendship.

Adjacent similarity loss. Because nodes coexist with each other via edges, we measure the joint probability of an existing edge $e(i, j)$ as

$$P^{+}_{i,j} = \frac{1}{1 + e^{-\langle y_i, y_j \rangle}}, \tag{6}$$

where $y_i \in R^D$ is the low-dimensional vector representation of node i, and $\langle a, b \rangle$ is the inner product of a and b. We use the Bernoulli distribution as the empirical distribution of existing edges. The prior probability can be defined as $\hat{P^{+}_{i,j}} = \frac{w_{i,j}}{W}$, where $W = \sum_{(i,j)\in E} w_{i,j}$ and $w_{i,j}$ is the weight of edge $e(i, j)$, which represents the strength of the relationship. For example, by default, in an unweighted graph, $w_{i,j} = 1$ for existing edges and $w_{i,j} = 0$ for nonexistent edges. To preserve first-order proximity, we use KL-divergence to measure the distance between two distributions; we define the positive adjacent similarity loss, with some simplification, as

$$L_{e^+} = -\sum_{(i,j)\in E_t} w_{i,j} \log P^{+}_{i,j}. \tag{7}$$

The edges of the graph are positive samples, and we consider the nonexistent edges as negative samples with the probability $P^{-}_{i,j}$. The negative adjacent similarity loss is

$$L_{e^-} = -\sum_{(i,j)\notin E_t} w_{i,j} \log P^{-}_{i,j}, \tag{8}$$

where we notice that $w_{i,j}$ is naturally set to zero; therefore, we cannot optimize the negative adjacent similarity loss directly. Because the value of the sigmoid function is in the range (0,1), we redefine Equation 8 as

$$L_{e^-} = -\sum_{(i,j)\notin E_t} \log(1 - P^{-}_{i,j}). \tag{9}$$

The total adjacent similarity loss is

$$L_e = L_{e^+} + L_{e^-}. \tag{10}$$

Although an edge is the most direct relationship in a graph, first-order proximity alone is insufficient to describe the structure of a graph. Single-neighbor similarity can estimate the direct coexistence relationship between nodes, but ignores the context information that implies potential connections. Second-order proximity is an important property for exploring the indirect relationship between nodes, and has been applied well by previous work Tang et al. (2015). We use context embeddings to preserve the k-order ($k \geq 2$) proximity of nodes in dynamic graphs.

Context similarity loss. Assuming that the context of each node in the graph is stable, we aim to find a representation for each context in the latent space that preserves the property of second-order, or higher-order, proximity of nodes. Random walk-based models Perozzi et al. (2014); Grover and Leskovec (2016); Ribeiro et al. (2017) split a fixed window in walks as the context of nodes. Instead of random walk using such a linear transfer probability, we use a diffuse context sampling method to generate the context of each node, as we discuss in Section 4.1; this method is a nonlinear transfer probability process. To learn context information from the latent space, we introduce context embedding $C \in R^{|V| \times D}$. Context embedding has been used to align discrete-time embeddings in the dynamic word embedding method Rudolph and Blei (2018). The context of an entity in a graph can be the neighbors around the corresponding node. Even though context changes continually in a dynamic graph, most nodes have stable context properties. Overall, the context vector can reflect the stable context property of each node and group all embeddings into the same vector space. Context proximity should be transductive; for example, if $\langle y_i, y_k \rangle > \langle y_j, y_k \rangle$, then $\langle c_i, c_k \rangle > \langle c_j, c_k \rangle$, where $\langle a, b \rangle$ is the inner product of a and b. Therefore, the empirical probability of a node v that has a k-node context $S(v)$ is $\hat{P}(v|S(v))$. As we discuss in Section 4.1, computing $\hat{P}(v|S(v))$ directly is extremely difficult because of the complex interaction in contexts, and we can assume that the context is a Bayesian network. According to Bayesian theory, $\hat{P}(v|S(v))$ can be changed to

$$\hat{P}(v|S(v)) = \frac{P(S(v)|v)P(v)}{P(S(v))}. \tag{11}$$

To simplify our model, we assume $P(S(v)) = \prod_{u \in S(v)} P(u)$ and $P(v) = \frac{d(v)}{\sum_{u \in V} d(u)}$, where $d(v) = \sum_{(u,v) \in E} w_{u,v}$. We then combine Equations 1, 2, and 11 as

$$\hat{P}(v|S(v)) = \frac{P(v)}{P(S(v))} P(u_1|v)P(u_2|u_1, v)...P(u_k|u_1, u_2, ..., v), \tag{12}$$

where $u_1, ..., u_k \in S(v)$ and $P(u_i|u_1, ..., v)$ is the transfer probability of node u_i in the Bayesian network that is generated by the diffuse context sampling method described in Section 4.1. Notice that we do not need to compute the conditional probability $P(u_i|u_1, ..., v)$ again, because it is computed in the diffuse context sampling process, according to Equations 4 and 5, and we define

$$P(u_i|u_1, ..., u_{i-1}) = \prod_{j=1}^{i-1} P(u_i|u_j), \tag{13}$$

$$= \prod_{j=1}^{i-1} \frac{w_{u_j,u_i}}{\sum_{(u_j,r) \in E} w_{u_j,r}}. \tag{14}$$

Thus, the conditional probability between node v and its context $S(v)$ is

$$P(v|S(v)) = \sigma(\frac{\langle y_v, \sum_{j \in S(v)} c_j \rangle}{k}), \tag{15}$$

where $\sigma(\cdot)$ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. In addition, we use KL-divergence to measure the distance between $P(v|S(v))$ and its empirical distribution $\hat{P}(v|S(v))$ as the context similarity loss function:

$$L_{cs} = -\sum_{v \in V} \hat{P}(v|S(v)) \log \sigma(\frac{\langle y_v, \sum_{u \in S(v)} c_u \rangle}{k}). \tag{16}$$

Temporal smoothness loss. To penalize the adjacent embedding vectors y_v^{t-1} and y_v^t for drifting too far from each other, the temporal smoothness loss is

$$L_{smooth} = \begin{cases} \sum_{v \in V} \|y_v^t - y_v^{t-1}\|_2 & t \geq 2 \\ 0 & t = 1 \end{cases} \tag{17}$$

Finally, we group all likelihoods as the optimization objective:

$$\arg\min_{\{Y_1, ..., Y_T\}, C} \sum_{t=1}^{T} (L_e^t + L_{cs}^t + L_{smooth}). \tag{18}$$

4.3. Optimization

Because negative edges are far more numerous than positive edges, computing all of the negative edges is extremely time-consuming. Therefore, we use negative sampling, to randomly select negative edges, to reduce the computation time of L_{e^-}. In this paper, we use the unigram distribution Mikolov et al. (2013b) raised to the power of 0.75. We denote the set of negative sampling edges by E_t^-, and reformulate Equation 9 as

$$L_{e^-} = -\sum_{(i,j) \in E_t^-} \log(1 - P_{i,j}^-). \tag{19}$$

Normalization. Intuitively, the embeddings should be normalized, to improve the training and encourage convergence. Therefore, we force a normalization operation on every embedding vector at each epoch. This operation can help to train our model faster. The general formula for normalization is

$$y_v = (y_{v_1}, y_{v_2}, ..., y_{v_D}), \tag{20}$$

$$\hat{y}_v = (\hat{y}_{v_1}, \hat{y}_{v_2}, ..., \hat{y}_{v_D}), \tag{21}$$

$$\hat{y}_{v_i} = \frac{y_{v_i} - \mu}{\Sigma}, \tag{22}$$

$$\mu = \frac{1}{D} \sum_{i=1}^{D} y_{v_i}, \Sigma = \frac{1}{D} \sum_{i=1}^{D} (y_{v_i} - \mu)^2. \tag{23}$$

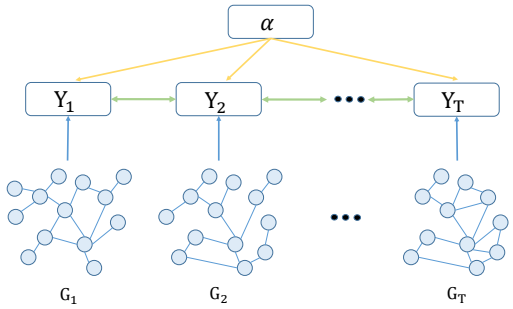


Figure 3: The main process describes how DCGE works. We split a dynamic graph into T snapshots, and train graph embeddings $Y_1, ..., Y_T$ for each snapshot. The context embedding C is shared across all snapshots and is trained with the graph embeddings jointly.

Initialization. We initialize each embedding matrix, including C and $Y_1, ..., Y_T$, with Gaussian random initialization, as $C, Y_1, ..., Y_T \sim N(0, 1)$.

We use Adam Kingma and Ba (2015) as an optimizer, to optimize Equation 18 with a proper learning rate $\gamma = 0.001$. In our implementation, we train our model with a mini-batch size of 128, and repeat this process until the early stopping within 20 steps, to avoid overfitting. More details of our model are shown in Algorithm 2.

Algorithm 2 DCGE

Input: A dynamic graphs $\Gamma = \{G_1, ..., G_T\}, G_t = (V_t, E_t)$
  embedding size D
  context size k
  negative samples q
Output: Graph embeddings $Y_1, ..., Y_T$
  Initialize embedding matrix $Y_1, ..., Y_T$ randomly
  Initialize context matrix C randomly
  for $n = 1 \rightarrow N$ **do**
    Sample S from Context Sampling(k,Γ)
    Sample E_t^- from $\{G_1, ..., G_T\}$
    Computing loss L on $S(v)$ and (E_t, E_t^-) according to Eq 18
    Optimize L using Adam then update $Y_1, ..., Y_T$ and C
  end for

5. Experiments

Our method was evaluated by the link prediction task (continuous-time and discrete-time dynamic graphs) and dynamic graph visualization. The former is a classical method to assess the effectiveness for capturing dynamic changes in proximity in adjacent timesteps. The latter focuses on showing the evolution of relationships in the dynamic graph. The experiments were performed on Intel i7 8700K (CPU), RTX2070 (GPU), 32 GB (memory).

For the first task, we used five datasets collected from the Graph Repository Rossi and Ahmed (2015); all datasets are temporal and real. Table 1 shows the statistics of these datasets.

Table 1: The statistics of dynamic graphs. $|V|$ = number of nodes; $|E|$ = number of temporal edges; T = number of timesteps in the training data.

Dataset	$\|V\|$	$\|E\|$	T
mathoverflow	24,818	506,550	2350
ia-facebook	46,952	876,993	1591
soc-epinions	131,828	841,372	944
sx-askubuntu	159,316	964,437	2047
sx-superuser	194,085	1,443,339	2426

For the second task, we implemented t-SNE Maaten and Hinton (2008) on a dynamic graph, to visualize the structure changes of each snapshot and demonstrate the advantages of DCGE.

5.1. Baseline Methods

Our method is a discrete-time graph embedding method; we selected comparable baseline methods from different categories. Node2vec and GAT are two representative static methods: a random walk-based model and a graph neural network-based model. We selected one continuous-time graph embedding method (CTDNE) and two static methods for continuous-time link prediction. For discrete-time link prediction, we compared DCGE with two discrete-time graph embedding methods (GynGEM and DynTriad).

5.2. Quantitative Results

Link prediction is a common application for evaluating the performance of graph embeddings. We set two different prediction tasks, continuous-time link prediction and discrete-time link prediction, to compare DCGE with other methods.

Continuous-time link prediction. A continuous-time graph is a single graph with timestamps on its edges. To generate the training data and testing data, we sorted all temporal edges in time-ascending order, as suggested in Nguyen et al. (2018). We then used the first 75% as the training data (one graph) and the remaining 25% as the testing data (positive links). In the testing, we further randomly sampled an equal number of negative links. For static node embedding methods (Node2vec and GAT) and CTDNE, we learned node embeddings of the current snapshot during training. In the testing, we used the learned model to predict links in the testing data. For node embedding-based methods, we computed the similarity between two nodes by the L2 distance of their node embeddings to predict the probability of existence of an edge between the two nodes. We used a *logistic regression model* as the classifier for edge existence classification, to improve the performance of node embedding-based methods. When DCGE is used for learning static graphs, the context embedding retains only the context information in one graph. Therefore, our model is equivalent to learning two types of embedding for each node, which capture the proximity information and context information separately, to improve the performance. In this task, we removed the temporal smoothness loss in Equation 18. Table 2 shows the AUC scores of the continuous-time link prediction task; DCGE achieves better performance than other state-of-the-art methods.

Table 2: AUC scores of link prediction in continuous-time dynamic graphs.

Dataset	**Node2vec**	**CTDNE**	**GAT**	**DCGE**
mathoverflow	0.754	0.774	0.801	**0.813**
ia-facebook	0.789	0.811	0.822	**0.849**
soc-epinions	0.713	0.758	0.816	**0.823**
sx-askubuntu	0.701	0.772	0.797	**0.834**
sx-superuser	0.705	0.783	0.811	**0.815**

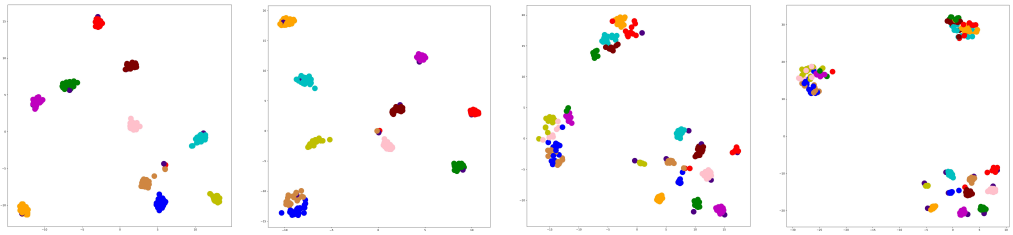
Table 3: AUC scores of link prediction in discrete-time dynamic graphs.

Dataset	**DynGEM**	**DynTriad**	**DCGE**
mathoverflow	0.765	0.792	**0.828**
ia-facebook	0.794	0.816	**0.884**
soc-epinions	0.765	0.821	**0.837**
sx-askubuntu	0.758	0.805	**0.817**
sx-superuser	0.756	0.796	**0.809**

Discrete-time link prediction. This task is to evaluate the ability to capture the evolution of links in discrete-time dynamic networks with multiple snapshots. We split the events into 10 parts equally and constructed 10 snapshots for discrete-time node embeddings methods (DynGEM and DynTriad). The first nine snapshots were the training dataset and the last nine snapshots were used for testing. Similarly to continuous-time link prediction, discrete-time node embedding methods use the node embedding Y_t to predict links in G_{t+1}, except the last snapshot; we report the average performance of different snapshots. The key of discrete-time link prediction is to preserve the temporal information and relative community structure. DCGE uses the context information that other methods ignore. Table 3 demonstrates that our method outperforms the other methods in all cases. In contrast to DynTriad Zhou et al. (2018)—which learns the process of a closed triad, focusing on a 3-node subgraph—DCGE uses the k-node context information to preserve the higher-order proximity between nodes.

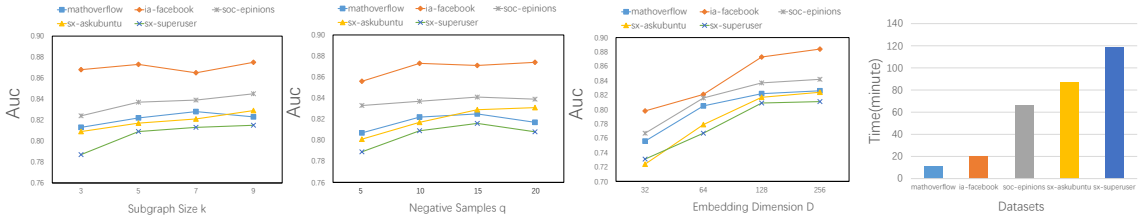
5.3. Qualitative Results

To analyze evolution patterns of real-world dynamic graphs, we visualized the dynamic graph in 2D space using t-SNE Maaten and Hinton (2008). We conducted a qualitative analysis to understand the distinctive power of the dynamic embeddings learned by DCGE and compared our embeddings with those of DynTriad, the state-of-the-art discrete-time embedding method. The primary school data were collected from face-to-face contacts between students and teachers in a school in France during two school days in October 2009 Stehlé et al. (2011). The data describe 242 students from 10 classes, composed of five grades, with each of the grades divided into two classes. There are 10 assigned teachers for 10 classes. We chose to analyze the evolution patterns on the first day, because of the similarity between the two days. We divided the dynamic graph into 10 snapshots, including the interactions of students from 8:45 to 17:05. We encoded each class and teacher with 11 different colors. As shown in Figure 4 (a–b), the projection of students and teachers from the first graph, both methods can distinguish the different classes well. Figure 4 (c–d) illustrates the school lunch break. Half of the students leave school, and the others



(a) DCGE (t=1) (b) Triad (t=1) (c) DCGE (t=5) (d) Triad (t=5)

Figure 4: The visualization of DCGE and DynTriad in two different time slices, via t-SNE.



(a) $k = \{3, 5, 7, 9\}$ (b) $q = \{5, 10, 15, 20\}$ (c) $D = \{32, 64, 128, 256\}$ (d) *runtime*

Figure 5: The AUC scores of discrete-time link prediction for different settings of the hyperparameters k, D, and q. (d) is the runtime of DCGE with default settings ($k = 5$, $D = 128$, and $q = 10$) for different datasets.

have lunch in the dining hall and then play in the playground, as represented in the school timetable Stehlé et al. (2011). We note that DCGE can always keep the graph embeddings of the same class nearby, and distinguish different classes in different snapshots, to facilitate the analysis of the graph evolution.

5.4. Parameter Analysis

We selected three hyperparameters (k, q, and D) for parameter analysis, by evaluating the performance of our method on the discrete-time link prediction task. We set the parameters $k = 5$, $q = 10$, and $D = 128$ as the default values, and then changed each parameter within a suitable range to evaluate the influence of these parameters. The context size decides how wide each node can reach and generate relationship. Figure 5 (a–c) shows that the performance of our method is stable when $k = 5$ and $q = 10$, and the higher value can afford a little gain barely. Thus, the hyperparameters k and q are rarely sensitive for the link prediction task, and we can tune their values to achieve better performance. Figure 5 (c) shows that the performance improves dramatically when D increases from 32 to 128.

5.5. Efficiency

Our proposed model is composed of two steps: context sampling and embedding learning. Many contexts are generated from each node in the graph, and this step can be processed

in parallel. Therefore, the time for sampling contexts is proportional to the number of nodes in the graph. We implemented the main part of the embedding learning on a GPU because most loss function computations can be transformed easily to matrix operations. The complexity of our model is linearly related to the number of entities and the number of edges, so the execution time of DCGE is linear to the dynamic graph size. Additionally, we use mini-batch to train our model in our implementation. Figure 5(d) shows the run time of our model with each dataset. Note that the run time increases linearly with increasing graph size.

6. Conclusion

In this study, we have proposed an approach to capture the changes of dynamic graphs with context proximity and temporal properties preserved. To evaluate our method, we have compared our method with several state-of-the-art methods, including static and dynamic methods, with continuous-time and discrete-time link prediction. The experiments demonstrated that our method achieves substantial gains and performs effectively in the proximity and evolution analysis of dynamic graphs. For future work, it is desirable to update nodes that never appear in the graph incrementally, instead of retraining. Most existing graph embedding methods focus only on one noticeable facet of a graph, whereas, in the real world, a graph includes diverse facets. Therefore, we would like to design a method to incorporate multi-facet properties into graph embeddings.

7. Acknowledgements

This work was supported by the National Key Research & Development Program of China (2017YFB0202203), National Natural Science Foundation of China (61672452, 61890954, and 61972343), and NSFC-Guangdong Joint Fund (U1611263).

References

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th CIKM*, pages 891–900. ACM, 2015.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

Ting Chen and Yizhou Sun. Task-guided and path-augmented heterogeneous network embedding for author identification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 295–304, 2017.

Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. Adversarial network embedding. In *AAAI*, pages 2167–2174, 2018.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd KDD*, pages 855–864, 2016.

Yupeng Gu, Yizhou Sun, Yanen Li, and Yang Yang. Rare: Social rank regulated large-scale network embedding. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 359–368, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 29*, pages 1024–1034, 2017.

William L Hamilton, Jure Leskovec, and Dan Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1489–1501, 2016.

Renjun Hu, Charu C Aggarwal, Shuai Ma, and Jinpeng Huai. An embedding approach to anomaly detection. In *Proceedings of the 32nd ICDE*, pages 385–396, 2016.

Nitin Kamra, Palash Goyal, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. In *IJCAI International Workshop on Representation Learning for Graphs (ReLiG)*, August 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 370–377, 2018.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR Workshop Papers*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013b.

Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *3rd International Workshop on Learning Representations for Big Networks*, pages 969–976, 2018.

Jingchao Ni, Shiyu Chang, Xiao Liu, Wei Cheng, Haifeng Chen, Dongkuan Xu, and Xiang Zhang. Co-regularized deep multi-network embedding. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 469–478, 2018.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd KDD*, pages 1105–1114, 2016.

Lawrence Page. Pagerank citation ranking: bring order to the web. *Stanford Digital Library working paper*, 1997.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th KDD*, pages 701–710, 2014.

Adrian E Raftery, Xiaoyue Niu, Peter D Hoff, and Ka Yee Yeung. Fast inference for the latent space network model using a case-control approximate likelihood. *Journal of Computational and Graphical Statistics*, 21(4):901–919, 2012.

Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd KDD*, pages 385–394, 2017.

Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, volume 15, pages 4292–4293, 2015.

Maja Rudolph and David Blei. Dynamic bernoulli embeddings for language evolution. In *Proceedings of the 2018 World Wide Web Conference*, pages 1003–1011, 2018.

Purnamrita Sarkar and Andrew W Moore. Dynamic social network analysis using latent space models. In *Advances in Neural Information Processing Systems*, pages 1145–1152, 2006.

Juliette Stehlé, Nicolas Voirin, Alain Barrat, Ciro Cattuto, Lorenzo Isella, Jean-François Pinton, Marco Quaggiotto, Wouter Van den Broeck, Corinne Régis, Bruno Lina, et al. High-resolution measurements of face-to-face contact patterns in a primary school. *PloS one*, 6(8):e23176, 2011.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd KDD*, pages 1225–1234, 2016.

Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *AAAI*, pages 203–209, 2017.

Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. Timers: Error-bounded svd restart on dynamic networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 1–8, 2018.

Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, pages 571–578, 2018.

Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(10):2765–2777, 2016.