Temporal Network Representation Learning

John Boaz Lee, Giang Nguyen, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim

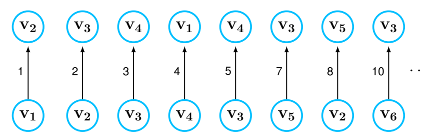
Abstract—Networks evolve continuously over time with the addition, deletion, and changing of links and nodes. Such temporal networks (or edge streams) consist of a sequence of timestamped edges and are seemingly ubiquitous. Despite the importance of accurately modeling the temporal information, most embedding methods ignore it entirely or approximate the temporal network using a sequence of static snapshot graphs. In this work, we introduce the notion of *temporal walks* for learning dynamic embeddings from temporal networks. Temporal walks capture the temporally valid interactions (*e.g.*, flow of information, spread of disease) in the dynamic network in a lossless fashion. Based on the notion of temporal walks, we describe a general class of embeddings called continuous-time dynamic network embeddings (CTDNEs) that completely avoid the issues and problems that arise when approximating the temporal network as a sequence of static snapshot graphs. Unlike previous work, CTDNEs learn dynamic node embeddings directly from the temporal network at the finest temporal granularity and thus use only temporally valid information. As such CTDNEs naturally support online learning of the node embeddings in a streaming real-time fashion. The experiments demonstrate the effectiveness of this class of embedding methods for prediction in temporal networks.

Index Terms—Dynamic node embeddings, temporal node embeddings, temporal walks, stream walks, graph streams, edge streams, temporal networks, dynamic networks, network representation learning, dynamic graph embedding, online learning, incremental learning

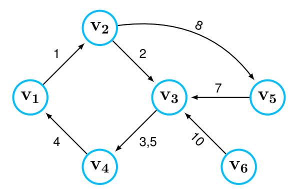
1 Introduction

Dynamic networks are seemingly ubiquitous in the real-world. Such networks evolve over time with the addition, deletion, and changing of nodes and links. The temporal information in these networks is known to be important to accurately model, predict, and understand network data [1], [2]. Despite the importance of these dynamics, most previous work on embedding methods have ignored the temporal information in network data [3], [4], [5], [6], [7], [8], [9], [10], [11], [12].

We address the problem of learning an appropriate network representation from *continuous-time dynamic networks* (Figure 1) for improving the accuracy of predictive models. We propose *continuous-time dynamic network embeddings* (CTDNE) and describe a general framework for learning such embeddings based on the notion of *temporal random walks* (walks that respect time). The framework is general with many interchangeable components and can be used in a straightforward fashion for incorporating temporal dependencies into existing node embedding and deep graph models that use random walks. Most importantly, the CTDNEs are learned from temporal random walks that represent actual *temporally valid sequences* of node interactions and thus avoids the issues and information loss that arises when time is ignored [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] or approximated as a sequence of discrete static snapshot graphs [13], [14], [15], [16], [17] (Figure 2) as done in previous work. The proposed approach (1) obeys the direction of time and (2) biases the random walks towards edges (and nodes) that are more recent and more frequent. The result is a more appropriate time-dependent network representation that captures the important temporal properties of the continuous-time dynamic network at the finest most natural temporal granularity without loss of information while using walks that are temporally valid (as opposed to walks that do not obey time and thus are invalid and noisy as they represent sequences that are impossible with respect to time). Hence, the framework allows existing embedding methods to be easily adapted for learning more appropriate network representations from continuous-time dynamic networks by ensuring time is respected and avoiding impossible sequences of events.



(a) Graph (edge) stream



(b) Continuous-Time Dynamic Network (CTDN)

Fig. 1. Dynamic network. Edges are labeled by time. Observe that existing methods that ignore time would consider $v_4 \longrightarrow v_1 \longrightarrow v_2$ a *valid* walk, however, $v_4 \longrightarrow v_1 \longrightarrow v_2$ is clearly *invalid with respect to time* since $v_1 \longrightarrow v_2$ exists in the past with respect to $v_4 \longrightarrow v_1$. In this work, we propose the notion of *temporal random walks* for embeddings that capture the *true temporally valid* behavior in networks. In addition, our approach naturally supports learning in *graph streams* where edges arrive continuously over time (*e.g.*, every second/millisecond)

The proposed approach learns a more appropriate network representation from continuous-time dynamic networks that captures the important temporal dependencies

- *J. Lee and G. Nguyen is with WPI Email: {jtlee,ghnguyen}@wpi.edu*
- *R. Rossi is with Adobe Research E-mail: rrossi@adobe.com*
- *N. Ahmed is with Intel Labs Email: nesreen.k.ahmed@intel.com*
- *E. Koh and S. Kim is with Adobe Research Email: {eunyee,sukim}@adobe.com*

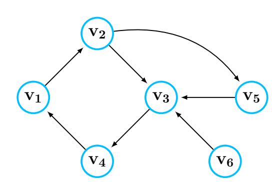
of the network at the finest most natural granularity (*e.g.*, at a time scale of seconds or milliseconds). This is in contrast to approximating the dynamic network as a sequence of static snapshot graphs $G_1, \ldots, G_t$ where each static snapshot graph represents all edges that occur between a user-specified discrete-time interval (*e.g.*, day or week) [18], [19], [20]. Besides the obvious loss of information, there are many other issues such as selecting an appropriate aggregation granularity which is known to be an important and challenging problem in itself that can lead to poor predictive performance or misleading results. In addition, our approach naturally supports learning in *graph streams* where edges arrive continuously over time (*e.g.*, every second/millisecond) [21], [22], [23], [24] and therefore can be used for a variety of applications requring real-time performance [25], [26], [27].

We demonstrate the effectiveness of the proposed framework and generalized dynamic network embedding method for temporal link prediction in several real-world networks from a variety of application domains. Overall, the proposed method achieves an average gain of 11.9% across all methods and graphs. The results indicate that modeling temporal dependencies in graphs is important for learning appropriate and meaningful network representations. In addition, any existing embedding method or deep graph model that uses random walks can benefit from the proposed framework (*e.g.*, [3], [4], [8], [9], [10], [11], [12], [28]) as it serves as a basis for incorporating important temporal dependencies into existing methods. Methods generalized by the framework are able to learn more meaningful and accurate time-dependent network embeddings that capture important properties from continuous-time dynamic networks.

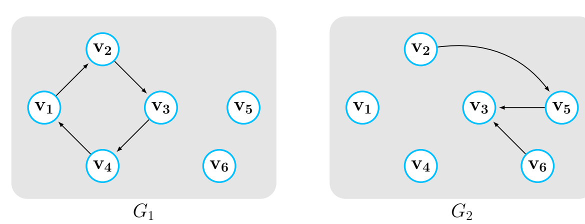
Previous embedding methods and deep graph models that use random walks search over the space of random walks $\mathbb{S}$ on G, whereas the class of models (continuous-time dynamic network embeddings) proposed in this work learn temporal embeddings by searching over the space $\mathbb{S}_T$ of temporal random walks that obey time and thus $\mathbb{S}_T$ includes only *temporally valid walks*. See Figure 3 for intuition. Informally, a *temporal walk* S_t from node v_{i_1} to node $v_{i_{L+1}}$ is defined as a sequence of edges $\{(v_{i_1}, v_{i_2}, t_{i_1}), (v_{i_2}, v_{i_3}, t_{i_2}), \ldots, (v_{i_L}, v_{i_{L+1}}, t_{i_L})\}$ such that $t_{i_1} \leq t_{i_2} \leq \ldots \leq t_{i_L}$. A temporal walk represents a *temporally valid* sequence of edges traversed in increasing order of edge times and therefore respect time. For instance, suppose each edge represents a contact (*e.g.*, email, phone call, proximity) between two entities, then a temporal random walk represents a feasible route for a piece of information through the dynamic network. It is straightforward to see that existing methods that ignore time learn embeddings from a set of random walks that are not actually possible when time is respected and thus represent invalid sequences of events.

The sequence that links (events) occur in a network carries important information, *e.g.*, if the event (link) represents an email communication sent from one user to another, then the state of the user who receives the email message changes in response to the email communication. For instance, suppose we have two emails $e_i = (v_1, v_2)$ from v_1 to v_2 and $e_j = (v_2, v_3)$ from v_2 to v_3; and let $\mathcal{T}(v_1, v_2)$ be the time of an email $e_i = (v_1, v_2)$. If $\mathcal{T}(v_1, v_2) < \mathcal{T}(v_2, v_3)$ then the message $e_j = (v_2, v_3)$ may reflect the information received from the email communication $e_i = (v_1, v_2)$. However, if $\mathcal{T}(v_1, v_2) > \mathcal{T}(v_2, v_3)$ then the message $e_j = (v_2, v_3)$ cannot contain any information communicated in the email $e_i = (v_1, v_2)$. This is just one simple example illustrating the importance of modeling the actual sequence of events (email communications). Embedding methods that ignore time are prone to many issues such as learning inappropriate node embeddings that do not accurately capture the dynamics in the network such as the real-world interactions or flow of information among nodes. An example of information loss that occurs when time is ignored or the actual dynamic network is approximated using a sequence of discrete static snapshot graphs is shown in Figure 1 and 2, respectively. This is true for networks that involve the flow or diffusion of information through a network [29], [30], [31], networks modeling the spread of disease/infection [32], spread of influence in social networks (with applications to product adoption, viral marketing) [33], [34], or more generally any type of dynamical system or diffusion process over a network [29], [30], [31].

The proposed approach naturally supports generating dynamic node embeddings for any pair of nodes at a specific time t. More specifically, given a newly arrived edge between node i and j at time t, we simply add the edge to the graph, perform a number of temporal random walks that contain those nodes, and then update the embedding vectors for those nodes (via a partial fast update) using only those walks. In this case, there is obviously no need to recompute the embedding vectors for all such nodes in the graph as the update is very minor and an online partial update can be performed fast. This obviously includes the case where either node in the new edge has never been seen previously. The above is obviously a special case of our framework and is a trivial modification. Notice that we can also obviously drop-out past edges as they may become stale.



(a) Static graph ignoring time



(b) Discrete-Time Dynamic Network (DTDN)

Fig. 2. Representing the continuous-time dynamic network as a static graph or discrete-time dynamic network (DTDN). Noise and information loss occurs when the true dynamic network (Figure 1) is approximated as a sequence of discrete static snapshot graphs $G_1, \ldots, G_t$ using a user-defined aggregation time-scale s (temporal granularity). Suppose the dynamic network in Figure 1 is used and $s = 5$, then G_1 includes all edges in the time-interval $[1, 5]$ whereas G_2 includes all edges in $[6, 10]$ and so on. Notice that in the static snapshot graph G_1 the walk $v_4 \longrightarrow v_1 \longrightarrow v_2$ is still possible despite it being *invalid* while the perfectly valid temporal walk $v_1 \longrightarrow v_2 \longrightarrow v_5$ is impossible. Both cases are captured correctly without any loss using the proposed notion of *temporal walk* on the actual dynamic network.

Summary of Main Contributions: This work makes three main contributions. First, we propose the notion of *temporal walks* for embedding methods. This notion can be used in a straightforward fashion to adapt other existing and/or future state-of-the-art methods for learning embeddings from temporal networks (graph streams). Second, unlike previous work that learn embeddings using an approximation of the actual dynamic network (*i.e.*, sequence of static graphs), we describe a new class of embeddings called *continuous-time dynamic network embeddings* (CTDNE) that are learned directly from the graph stream. CTDNEs avoid the issues and information loss that arise when time is ignored or the dynamic network (graph stream) is approximated as a sequence of discrete static snapshot graphs. This new class of embeddings leverage the proposed notion of *temporal walks* that captures the *temporally valid interactions* (*e.g.*, flow of information, spread of diseases) in the dynamic network (graph stream) in a lossless fashion. As an aside, since these embeddings are learned directly from the graph stream at the finest granularity, they can also be learned in an online fashion, *i.e.*, node embeddings are updated in real-time after every new edge (or batch of edges). Finally, we describe a framework for learning them based on the notion of *temporal walks*. The proposed framework provides a basis for generalizing existing (or future state-of-the-art) embedding methods that use the traditional notion of random walks over static or discrete approximation of the actual dynamic network.

2 Continuous-Time Dynamic Embeddings

While previous work uses discrete approximations of the dynamic network (*i.e.*, a sequence of discrete static snapshot graphs), this paper proposes an entirely new direction that instead focuses on learning embeddings directly from the graph stream using only temporally valid information.

In this work, instead of approximating the dynamic network as a sequence of discrete static snapshot graphs defined as $G_1, \ldots, G_T$ where $G_i = (V, E_t)$ and E_t are the edges active between the timespan $[t_{i-1}, t_i]$, we model the *temporal interactions* in a lossless fashion as a *continuous-time dynamic network* (CTDN) defined formally as:

Definition 1 (Continuous-Time Dynamic Network) *Given a graph $G = (V, E_T, \mathcal{T})$, let V be a set of vertices, and $E_T \subseteq V \times V \times \mathbb{R}^+$ be a set of temporal edges between vertices in V, and $\mathcal{T} : E \rightarrow \mathbb{R}^+$ is a function that maps each edge to a corresponding timestamp. At the finest granularity, each edge $e_i = (u, v, t) \in E_T$ may be assigned a unique time $t \in \mathbb{R}^+$.*

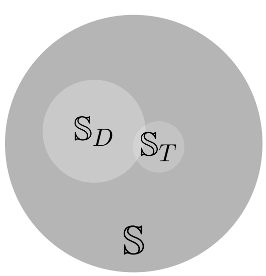


Fig. 3. Illustration depicting the space of all possible random walks $\mathbb{S}$ (up to a fixed length L) including (i) the space of temporal (time-obeying) random walks denoted as $\mathbb{S}_T$ that capture the temporally valid flow of information (or disease, etc.) in the network without any loss and (ii) the space of random walks that are possible when the dynamic network is approximated as a sequence of discrete static snapshot graphs denoted as $\mathbb{S}_D$. Notice there is a very small overlap between $\mathbb{S}_T$ and $\mathbb{S}_D$.

In continuous-time dynamic networks (*i.e.*, temporal networks, graph streams), edges occur over a time span $\mathcal{T} \subseteq \mathbb{T}$ where $\mathbb{T}$ is the temporal domain.[1] For continuous-time systems $\mathbb{T} = \mathbb{R}^+$. In such networks, a *valid* walk is defined as a sequence of nodes connected by edges with non-decreasing timestamps. In other words, if each edge captures the time of contact between two entities, then a (valid temporal) walk may represent a feasible route for a piece of information. More formally,

Definition 2 (Temporal Walk) *A temporal walk from v_1 to v_k in G is a sequence of vertices $\langle v_1, v_2, \cdots, v_k \rangle$ such that $\langle v_i, v_{i+1} \rangle \in E_T$ for $1 \leq i < k$, and $\mathcal{T}(v_i, v_{i+1}) \leq \mathcal{T}(v_{i+1}, v_{i+2})$ for $1 \leq i < (k-1)$. For two arbitrary vertices $u, v \in V$, we say that u is temporally connected to v if there exists a temporal walk from u to v.*

The definition of temporal walk echoes the standard definition of a walk in static graphs but with an additional constraint that requires the walk to respect time, that is, edges must be traversed in increasing order of edge times. As such, temporal walks are naturally asymmetric. Modeling the dynamic network in a continuous fashion makes it completely trivial to add or remove edges and nodes. For instance, suppose we have a new edge (v, u, t) at time t, then we can sample a small number of temporal walks ending in (v, u) and perform a fast partial update to obtain the updated embeddings (See Section 3.3 for more details). This is another advantage to our approach compared to previous work that use discrete static snapshot graphs to approximate the dynamic network. Note that performing a temporal walk forward through time is equivalent to one backward through time. However, for the streaming case (online learning of the embeddings) where we receive an edge (v, u, t) at time t, then we sample a temporal walk backward through time. A *temporally invalid walk* is a walk that does not respect time. Any method that uses temporally invalid walks or approximates the dynamic network as a sequence of static snapshot graphs is said to have *temporal loss*.

We define a new type of embedding for dynamic networks (graph streams) called continuous-time dynamic network embedding (CTDNEs).

Definition 3 (Continuous-Time Dynamic Network Embedding) *Given a dynamic network $G = (V, E_T, \mathcal{T})$ (graph stream), the goal is to learn a function $f : V \rightarrow \mathbb{R}^D$ that maps nodes in the continuous-time dynamic network (graph stream) G to D-dimensional time-dependent embeddings using only data that is temporally valid* (e.g., *temporal walks defined in Definition 2).*

Unlike previous work that ignores time or *approximates* the dynamic network as a sequence of discrete static snapshot graphs $G_1, \ldots, G_t$, CTDNEs proposed in this work are learned from temporal random walks that capture the true temporal interactions (*e.g.*, flow of information, spread of diseases, etc.) in the dynamic network in a lossless fashion. CTDNEs (or simply dynamic node embeddings) can be learned incrementally or in a streaming fashion where embeddings are updated in real-time as new edges arrive.

1. The terms temporal network, graph stream, and continuous-time dynamic network are used interchangeably.

For this new class of dynamic node embeddings, we describe a general framework for learning such temporally valid embeddings from the graph stream in Section 3.

3 Framework

While Section 2 formally introduced the new class of embeddings investigated in this work, this section describes a general framework for deriving them based on the notion of *temporal walks*. The framework has two main interchangeable components that can be used to *temporally bias* the learning of the dynamic node embeddings. We describe each component in Section 3.1 and 3.2. In particular, the CTDNE framework generates *biased (and unbiased) temporal random walks* from CTDNs in Section 3.1-3.2 that are then used in Section 3.3 for deriving time-dependent embeddings that are learned from temporally valid node sequences that capture in a lossless fashion the actual flow of information or spread of disease in a network. It is straightforward to use the CTDNE framework for temporal networks where edges are active only for a specified time-period.

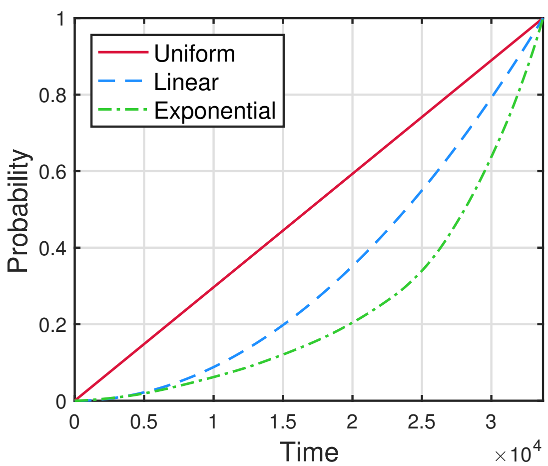


Fig. 4. Example initial edge selection cumulative probability distributions (CPDs) for each of the variants investigated (uniform, linear, and exponential). Observe that exponential biases the selection of the initial edge towards those occurring more recently than in the past, whereas linear lies between exponential and uniform.

3.1 Initial Temporal Edge Selection

This section describes approaches to temporally bias the temporal random walks by selecting the initial temporal edge to begin the temporal random walk. In general, each temporal walk starts from a temporal edge $e_i \in E_T$ at time $t = \mathcal{T}$ selected from a distribution $\mathbb{F}_s$. The distribution used to select the initial temporal edge can either be uniform in which case there is no bias or the selection can be temporally biased using an arbitrary weighted (non-uniform) distribution for $\mathbb{F}_s$. For instance, to learn node embeddings for the temporal link prediction task, we may want to begin more temporal walks from edges closer to the current time point as the events/relationships in the distant past may be less predictive or indicative of the state of the system now. Selecting the initial temporal edge in an unbiased fashion is discussed in Section 3.1.1 whereas strategies that temporally bias the selection of the initial edge are discussed in Section 3.1.2. In the case of learning CTDNEs in an online fashion, we do not need to select the initial edge since we simply sample a number of temporal walks that end at the new edge. See Section 3.3 for more details on learning CTDNEs in an online fashion.

3.1.1 *Unbiased*

In the case of initial edge selection, each edge $e_i = (v, u, t) \in E_T$ has the same probability of being selected:

$$\mathbb{P}(e) = 1/|E_T| \tag{1}$$

This corresponds to selecting the initial temporal edge using a uniform distribution.

3.1.2 *Biased*

We describe two techniques to temporally bias the selection of the initial edge that determines the start of the temporal random walk. In particular, we select the initial temporal edge using a temporally weighted distribution based on exponential and linear functions. However, the proposed continuous-time dynamic network embedding framework is flexible with many interchangeable components and therefore can easily support other temporally weighted distributions for selecting the initial temporal edge.

Exponential: We can also bias initial edge selection using an exponential distribution, in which case each edge $e \in E_T$ is assigned the probability:

$$\mathbb{P}(e) = \frac{\exp\left[\mathcal{T}(e) - t_{\min}\right]}{\sum_{e' \in E_T} \exp\left[\mathcal{T}(e') - t_{\min}\right]} \tag{2}$$

where $t_{\min}$ is the minimum time associated with an edge in the dynamic graph. This defines a distribution that heavily favors edges appearing later in time.

Linear: When the time difference between two time-wise consecutive edges is large, it can sometimes be helpful to map the edges to discrete time steps. Let $\eta : E_T \rightarrow \mathbb{Z}^+$ be a function that sorts (in ascending order by time) the edges in the graph. In other words η maps each edge to an index with $\eta(e) = 1$ for the earliest edge e. In this case, each edge $e \in \eta(E_T)$ will be assigned the probability:

$$\mathbb{P}(e) = \frac{\eta(e)}{\sum_{e' \in E_T} \eta(e')} \tag{3}$$

See Figure 4 for examples of the uniform, linear, and exponential variants.

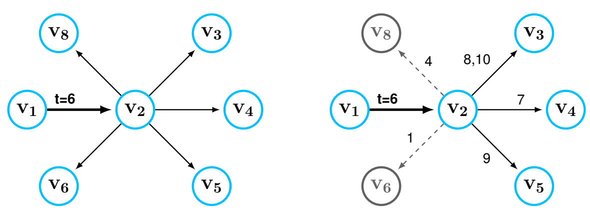
3.2 Temporal Random Walks

After selecting the initial edge $e_i = (u, v, t)$ at time t to begin the temporal random walk (Section 3.1) using $\mathbb{F}_s$, how can we perform a temporal random walk starting from that edge? We define the set of temporal neighbors of a node v at time t as follows:

Definition 4 (Temporal Neighborhood) *The set of temporal neighbors of a node v at time t denoted as $\Gamma_t(v)$ are:*

$$\Gamma_t(v) = \left\{(w, t') \mid e = (v, w, t') \in E_T \wedge \mathcal{T}(e) > t\right\} \tag{4}$$

Observe that the same neighbor w can appear multiple times in $\Gamma_t(v)$ since multiple temporal edges can exist between the same pair of nodes. See Figure 5 for an example. The next node in a temporal random walk can then be chosen from the set $\Gamma_t(v)$. Here we use a second distribution $\mathbb{F}_\Gamma$ to *temporally bias* the neighbor selection. Again, this distribution can either be uniform, in which case no bias is applied, or more intuitively biased to consider time. For instance, we may want to bias the sampling strategy towards walks that exhibit smaller "in-between" time for consecutive edges. That is, for each consecutive pair of edges (u, v, t), and $(v, w, t{+}k)$ in the random walk, we want k to be small. For temporal link prediction on a dynamic social network, restricting the "in-between" time allows us to sample walks that do not group friends from different time periods together. As an example, if k is small we are likely to sample the random walk sequence $(v_1, v_2, t), (v_2, v_3, t + k)$ which makes sense as v_1 and v_3 are more likely to know each other since v_2 has interacted with them both recently. On the other hand, if k is large we are unlikely to sample the sequence. This helps to separate people that v_2 interacted with during very different time periods (*e.g.* high-school and graduate school) as they are less likely to know each other.



(a) Neighborhood $\Gamma(v_2)$ (b) Temporal neigh. $\Gamma_t(v_2)$

Fig. 5. Temporal neighborhood of a node v_2 at time $t = 6$ denoted as $\Gamma_t(v_2)$. Notice that $\Gamma_t(v_2) = \{v_4, v_3, v_5, v_3\}$ is an ordered multiset where the temporal neighbors are sorted in ascending order by time with the nodes more recent appearing first. Moreover, the same node can appear multiple times (*e.g.*, a user sends another user multiple emails, or an association/event occurs multiple times between the same entities). This is in contrast to the definition of neighborhood used by previous work that is not parameterized by time, *e.g.*, $\Gamma(v_2) = \{v_3, v_4, v_5, v_6, v_8\}$ or $\Gamma(v_2) = \{v_3, v_3, v_4, v_5, v_6, v_8\}$ if multigraphs are supported.

3.2.1 Unbiased

For unbiased temporal neighbor selection, given an arbitrary edge $e = (u, v, t)$, each temporal neighbor $w \in \Gamma_t(v)$ of node v at time t has the following probability of being selected:

$$\mathbb{P}(w) = 1/|\Gamma_t(v)| \tag{5}$$

3.2.2 Biased

We describe two techniques to bias the temporal random walks by sampling the next node in a temporal walk via temporally weighted distributions based on exponential and linear functions. However, the continuous-time dynamic network embedding framework is flexible and can easily be used with other application or domain-dependent *temporal bias functions*.

Exponential: When exponential decay is used, we formulate the probability as follows. Given an arbitrary edge $e = (u, v, t)$, each temporal neighbor $w \in \Gamma_t(v)$ has the following probability of being selected:

$$\mathbb{P}(w) = \frac{\exp\big[\tau(w) - \tau(v)\big]}{\sum_{w' \in \Gamma_t(v)} \exp\big[\tau(w') - \tau(v)\big]} \tag{6}$$

Note that we abuse the notation slightly here and use τ to mean the mapping to the corresponding time. This is similar to the exponentially decaying probability of consecutive contacts observed in the spread of computer viruses and worms [35].

Linear: Here, we define $\delta : V \times \mathbb{R}^+ \to \mathbb{Z}^+$ as a function which sorts temporal neighbors in descending order time-wise. The probability of each temporal neighbor $w \in \Gamma_t(v)$ of node v at time t is then defined as:

$$\mathbb{P}(w) = \frac{\delta(w)}{\sum_{w' \in \Gamma_t(v)} \delta(w')} \tag{7}$$

This distribution biases the selection towards edges that are closer in time to the current node.

3.2.3 Temporal Context Windows

Since temporal walks preserve time, it is possible for a walk to run out of *temporally valid* edges to traverse. Therefore, we do not impose a strict length on the temporal random walks. Instead, we simply require each temporal walk to have a minimum length ω (in this work, ω is equivalent to the context window size for skip-gram [36]). A maximum length L can be provided to accommodate longer walks. A temporal walk $\mathcal{S}_{t_i}$ with length $|\mathcal{S}_{t_i}|$ is considered valid iff

$$\omega \leq |\mathcal{S}_{t_i}| \leq L$$

Given a set of temporal random walks $\{\mathcal{S}_{t_1}, \mathcal{S}_{t_2}, \cdots, \mathcal{S}_{t_k}\}$, we define the temporal context window count β as the total number of context windows of size ω that can be derived from the set of temporal random walks. Formally, this can be written as:

$$\beta = \sum_{i=1}^{k} \big(|\mathcal{S}_{t_i}| - \omega + 1\big) \tag{8}$$

When deriving a set of temporal walks, we typically set β to be a multiple of $N = |V|$. Note that this is only an implementation detail and is not important for Online CTDNEs.

3.3 Learning Dynamic Node Embeddings

Given a temporal walk $\mathcal{S}_t$, we can now formulate the task of learning time-preserving node embeddings in a CTDN as the optimization problem:

$$\max_f \; \log \mathbb{P}\big(\, W_T = \{v_{i-\omega}, \cdots, v_{i+\omega}\} \setminus v_i \mid f(v_i)\big) \tag{9}$$

where $f : V \to \mathbb{R}^D$ is the node embedding function, ω is the context window size for optimization, and

$$W_T = \{v_{i-\omega}, \cdots, v_{i+\omega}\}$$

such that

$$\mathcal{T}(v_{i-\omega}, v_{i-\omega+1}) < \cdots < \mathcal{T}(v_{i+\omega-1}, v_{i+\omega})$$

is an arbitrary temporal context window $W_T \subseteq S_t$. For tractability, we assume conditional independence between the nodes of a temporal context window when observed with respect to the source node v_i. That is:

$$\mathbb{P}\big(\, W_T | f(v_i)\big) = \prod_{v_{i+k} \in W_T} \mathbb{P}\big(v_{i+k} | f(v_i)\big) \tag{10}$$

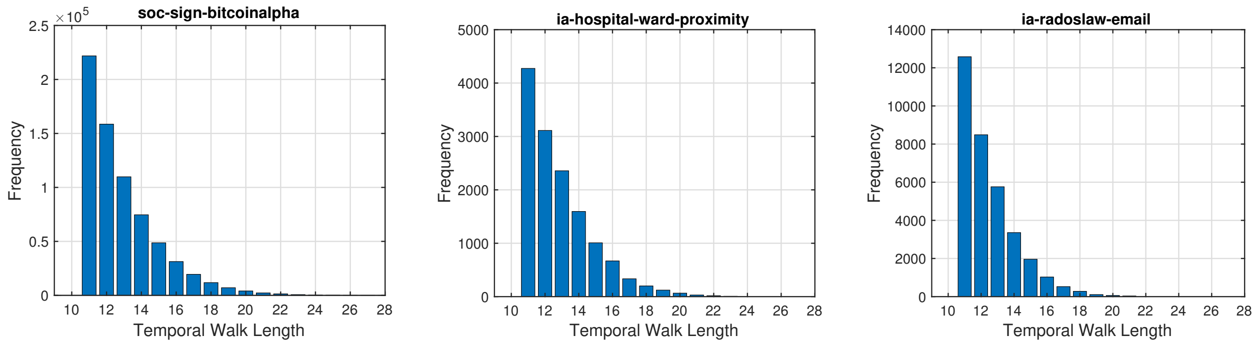


Fig. 6. Frequency of *temporal random walks* by length

We can model the conditional likelihood of every source-nearby node pair (v_i, v_j) as a softmax unit parameterized by a dot product of their feature vectors:

$$\mathbb{P}\big(v_j|f(v_i)\big) = \frac{\exp\big[f(v_j)\cdot f(v_i)\big]}{\sum_{v_k\in V}\exp\big[f(v_k)\cdot f(v_i)\big]} \tag{11}$$

Using Eq. 10- 11, the optimization problem in Eq. 9 reduces to:

$$\max_{f} \sum_{v_i\in V}\Bigg(-\log Z_i + \sum_{v_j\in W_T} f(v_j)\cdot f(v_i)\Bigg) \tag{12}$$

where the term $Z_i = \sum_{v_j\in V}\exp\big[f(v_i)\cdot f(v_j)\big]$ can be approximated by negative sampling. Given a graph G, let $\mathbb{S}$ be the space of all possible random walks on G and let $\mathbb{S}_T$ be the space of all temporal random walks on G. It is straightforward to see that the space of temporal random walks $\mathbb{S}_T$ is contained within $\mathbb{S}$, and $\mathbb{S}_T$ represents only a tiny fraction of possible random walks in $\mathbb{S}$. Existing methods sample a set of random walks $\mathcal{S}$ from $\mathbb{S}$ whereas this work focuses on sampling a set of *temporal random walks* $\mathcal{S}_t$ from $\mathbb{S}_T \subseteq \mathbb{S}$. In general, the probability of an existing method sampling a temporal random walk from $\mathbb{S}$ by chance is extremely small and therefore the vast majority of random walks sampled by these methods represent sequences of events between nodes that are invalid (not possible) when time is respected. For instance, suppose each edge represents an interaction/event (*e.g.*, email, phone call, spatial proximity) between two people, then a temporal random walk may represent a feasible route for a piece of information through the dynamic network or a temporally valid pathway for the spread of an infectious disease.

We summarize the procedure to learn time-preserving embeddings for CTDNs in Algorithm 1. Our procedure in Algorithm 1 generalizes the Skip-Gram architecture to learn continuous-time dynamic network embeddings (CTDNEs). However, the framework can easily be used for other deep graph models that leverage random walks (*e.g.*, [12]) as the temporal walks can serve as input vectors for neural networks. There are many methods that can be adapted to learn CTDN embeddings using *temporal random walks* (*e.g.*, node2vec [4], struc2vec [8], role2vec [37]) and the proposed framework is not tied to any particular approach.

While Algorithm 1 gives an overview of CTDNE, Algorithm 3 provides an online implementation of CTDNE. We point out that Algorithm 1 is useful for prediction tasks

Algorithm 1 Continuous-Time Dynamic Network Embeddings

Input: a dynamic network (graph stream) $G = (V, E_T, \mathcal{T})$, temporal context window count β, context window size ω, embedding dimensions D

```
Initialize number of temporal context windows C = 0
while β − C > 0 do
    Sample an edge e_t = (v, u) via F_s (or use new edge at time t)
    t ← T(e_t)
    S_t = TemporalWalk(G, e_t, t, L, ω + β − C − 1)
    if |S_t| > ω then
        Add the temporal walk S_t to S_T
        C ← C + (|S_t| − ω + 1)
end while
Z = StochasticGradientDescent(ω, D, S_T)   ▷ update embeddings
return dynamic node embeddings Z
```

Algorithm 2 Temporal Random Walk

```
procedure TemporalWalk(G′, e = (s, r), t, L, C)
    Set i ← s and initialize temporal walk S_t = [ s, r ]
    for p = 1 to min(L, C) − 1 do
        Γ_t(i) = {(w, t′) | e = (i, w, t′) ∈ E_T ∧ T(i) > t}
        if |Γ_t(i)| > 0 then
            Select node j from distribution F_Γ(Γ_t(i))
            Append j to S_t
            Set t ← T(i, j) and set i ← j
        else terminate temporal walk
    return temporal walk S_t of length |S_t| rooted at node s
```

where the goal is to learn a model using all data up to time t for prediction of a future discrete or real-valued attribute or state (*e.g.*, if a link exists or not). Since this work evaluates CTDNEs for link prediction, we include it mainly for the reader to understand one evaluation strategy using CTDNE. However, there are some other applications that may require online or incremental updates every time a new edge arrives. For such applications, Algorithm 3 gives an overview of CTDNE in this setting. Recall that CTDNE naturally supports such streaming settings where edges (or new nodes) arrive continuously over time [22] and the goal is to update the embeddings in real-time via fast efficient updates. Many approaches have been proposed to incrementally update the embeddings using the skip-gram model [38], [39], [40], [41] and any such approach can be used. Online optimization techniques have been investigated for decades, see [42], [43], [44], [45], [46], [47]. While Algorithm 3 assumes the graph stream is infinite, we can always obtain the updated dynamic node embeddings **Z** at time t. As an aside, given an edge (v, u) at time t, we obtain temporal walks that end in

Algorithm 3 Online CTDNE

Input: a dynamic network (graph stream) G, embedding dimensions D
Output: dynamic node embeddings $\mathbf{Z}$ at time t

```
while new edge (v, u, t) arrives at time t do
    Add edge (v, u, t) to E ← E ∪ {(v, u, t)} and V ← V ∪ {v, u}
    Sample temporal walks S_t ending in edge (v, u)
    Update embeddings via online word2vec/SGD using only S_t
end while
```

(v, u) by essentially reversing the temporal walk and going backwards through time (see Figure 9). As discussed, both are obviously equivalent and this is an implementation detail that allows us to easily obtain a set of temporal walks that include the new edge. However, it is important for learning online CTDNEs since temporal walks are obtained for every new edge that arrives. Since the edge arrives at time t, we know that no other edge exists at a future time. Furthermore, since the goal is to obtain temporal walks that include the new edge, then we know (v, u) will be at the end of the temporal walk, and we simply obtain the temporal walk by going backwards through time. Recall that previously we discussed approaches for sampling the start edge of a temporal walk. However, in the case of online CTDNE, we are simply given the new edge (v, u) at time t and perform a few temporal walks and then learn updated embeddings. Furthermore, we can relax the requirement of updating the embeddings after every new edge, and instead, we can wait until a fixed number of edges arrive before updating the embeddings or wait until a fixed amount of time elapses. We call such an approach batched CTDNE updating. The only difference in Algorithm 3 is that instead of performing an update immediately, we would wait until one of the above conditions become true and then perform a batch update. We can also drop edges that occur in the distant past or that have a very small weight.

3.4 Hyperparameters

While other methods have a lot of hyperparameters that require tuning such as node2vec [4], the proposed framework has a single hyperparameter that requires tuning. Note that since the framework is general and flexible with many interchangeable components, there is of course the possibility of introducing additional hyperparameters depending on the approaches used to bias the temporal walks.

Arbitrary walk length: In our work, we allow temporal walks to have arbitrary lengths which we simply restrict to be between the range $[\omega, L]$. We argue that arbitrary-sized walks between ω and L allow more accurate representations of node behaviors. For instance, a walk starting at u can return to u after traversing L edges, showing a closed community. On the other hand, another walk starting from v can end immediately at minimum length ω without ever going back to v. These are two distant cases that would be misrepresented if a fixed random walk length is imposed. Observe that L can be set to any reasonably large value such as $L = 80$ as it serves as an upper bound on the temporal random walk length, which is of course bounded by the dynamic network (unlike random walks which can be infinite). However, we set $\omega = 10$ and found other reasonable values to not impact performance on most graphs. Figure 7 investigates the number of times each node appears in the sampled temporal walks. We also study the frequency of starting a temporal random walk from each node in Figure 8.

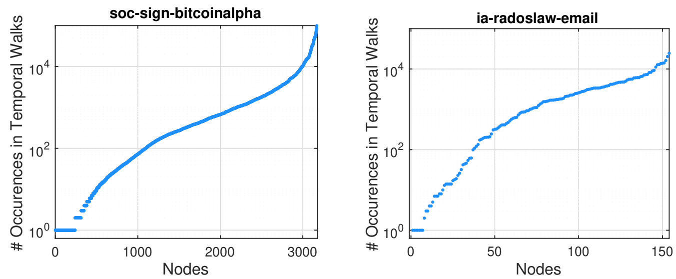


Fig. 7. Number of occurrences of each node in the set of sampled temporal walks.

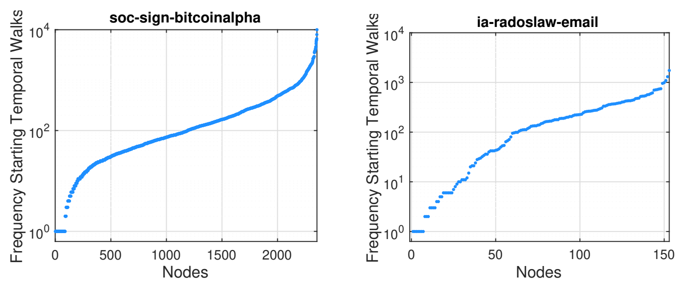


Fig. 8. Frequency of starting a temporal random walk from each node. Unlike previous approaches that sample a fixed number of random walks for each node, the proposed framework samples an edge between two nodes to obtain a timestamp to begin the temporal random walk.

Exponential base Suppose the exponential function is used to bias the temporal random walk (Eq. 6) or bias the selection of the initial edge to begin the temporal walk (Eq. 2), then we allow the user to choose the base b of the exponential function for the exponential distribution. In the case of initial temporal edge selection (Eq. 6), a large base b would cause the function to grow rapidly. Notice that if the observed temporal interactions (*e.g.* edges) in the dynamic network span a large time period, the probability of choosing one of the recent edges may be much larger than the probability to choose all other edges resulting in sampled walks that are skewed too much towards recent edges.

4 Theoretical Analysis

Let $N = |V|$ denote the number of nodes, $M = |E_T|$ be the number of edges, D = dimensionality of the embedding, R = the number of temporal walks per node, L = the maximum length of a temporal random walk, and Δ = the maximum degree of a node. Recall that while R is not required, we use it here since the number of temporal random walks $|\mathcal{S}_T|$ is a multiple of the number of nodes $N = |V|$ and thus can be written as RN similar to previous work.

4.1 Time Complexity

LEMMA 1 *The time complexity for learning CTDNEs using the generalized Skip-gram architecture in Section 3.3 is*

$$\mathcal{O}(M + N(R \log M + RL\Delta + D)) \tag{13}$$

and the time complexity for learning CTDNEs with unbiased *temporal random walks (uniform) is:*

$$\mathcal{O}(N(R \log M + RL \log \Delta + D)) \tag{14}$$

PROOF. The time complexity of each of the three steps is provided below. We assume the exponential variant is used for both $\mathbb{F}_s$ and $\mathbb{F}_\Gamma$ since this CTDNE variant is the most computationally expensive among the nine CTDNE variants expressed from using uniform, linear, or exponential for $\mathbb{F}_s$ and $\mathbb{F}_\Gamma$. Edges are assumed to be ordered by time such that $\mathcal{T}(e_1) \leq \mathcal{T}(e_2) \leq \cdots \leq \mathcal{T}(e_M)$. Similarly, the neighbors of each node are also ordered by time.

Initial Temporal Edge Selection: To derive $\mathbb{F}_s$ for any of the variants used in this work (uniform, linear, exponential) it takes $\mathcal{O}(M)$ time since each variant can be computed with a single or at most two passes over the edges. Selecting an initial edge via $\mathbb{F}_s$ takes $\mathcal{O}(\log M)$ time. Now $\mathbb{F}_s$ is used to select the initial edge for each temporal random walk $S_t \in \mathcal{S}_T$ and thus an initial edge is selected $RN = |\mathcal{S}_T|$ times. This gives a total time complexity of $\mathcal{O}(M + RN \log M)$.[2]

Temporal Random Walks: After the initial edge is selected, the next step is to select the next temporally valid neighbor from the set of temporal neighbors $\Gamma_t(v)$ of a given node v at time t using a (weighted) distribution $\mathbb{F}_\Gamma$ (*e.g.*, uniform, linear, exponential). Note $\mathbb{F}_\Gamma$ must be computed and maintained for each node. Given a node v and a time t_* associated with the previous edge traversal in the temporal random walk, the first step in any variant (uniform, linear, exponential; Section 3.2) is to obtain the ordered set of temporal neighbors $\Gamma_t(v) \subseteq \Gamma(v)$ of node v that occur after t_*. Since the set of all temporal neighbors is already stored and ordered by time, we only need to find the index of the neighbor $w \in \Gamma(v)$ with time $t > t_*$ as this gives us $\Gamma_t(v)$. Therefore, $\Gamma_t(v)$ is derived in $\log|\Gamma(v)|$ via a binary search over the ordered set $\Gamma(v)$. In the worst case, $\mathcal{O}(\log \Delta)$ where $\Delta = \max_{v \in V} |\Gamma(v)|$ is the maximum degree. After obtaining $\Gamma_t(v) \subseteq \Gamma(v)$, we derive $\mathbb{F}_\Gamma$ in $\mathcal{O}(\Delta)$ time when $d_v = \Delta$. Now, selecting the next temporally valid neighbor according to $\mathbb{F}_\Gamma$ takes $\mathcal{O}(\log \Delta)$ for exponential and linear and $o(1)$ for uniform. For the uniform variant, we select the next temporally valid neighbor in $o(1)$ constant time by $j \sim \text{UniformDiscrete}\{1, 2, \ldots, |\Gamma_t(v)|\}$ and then obtain the selected temporal neighbor by directly indexing into $\Gamma_t(v)$. Therefore, the time complexity to select the next node in a biased temporal random walk is $\mathcal{O}(\log \Delta + \Delta) = \mathcal{O}(\Delta)$ in the worst case and $\mathcal{O}(\log \Delta)$ for unbiased (uniform).

For a temporal random walk of length L, the time complexity is $\mathcal{O}(L\Delta)$ for a biased walk with linear/exponential and $\mathcal{O}(L \log \Delta)$ for an unbiased walk. Therefore, the time complexity for RN biased temporal random walks of length L is $\mathcal{O}(RNL\Delta)$ in the worst case and $\mathcal{O}(RNL \log \Delta)$ for unbiased.

Learning time-dependent embeddings: For the Skip-Gram-based generalization given in Section 3.3, the time complexity per iteration of Stochastic Gradient Descent (SGD) is $\mathcal{O}(ND)$ where $D \ll N$. While the time complexity of a single iteration of SGD is less than a single iteration of Alternating Least Squares (ALS) [48], SGD requires more iterations to obtain a good enough model and is sensitive to the choice of learning rate [49], [50]. Moreover, SGD is more challenging to parallelize compared to ALS [48] or Cyclic Coordinate Descent (CCD) [51], [52]. Nevertheless, the choice of optimization scheme depends on the objective function of the embedding method generalized via the CTDNE framework.

2. Note for uniform initial edge selection, the time complexity is linear in the number of temporal random walks $\mathcal{O}(RN)$.

4.2 Space Complexity

Storing the F_s distribution takes $\mathcal{O}(M)$ space. The temporal neighborhoods do not require any additional space (as we simply store an index). Storing $\mathbb{F}_\Gamma$ takes $\mathcal{O}(\Delta)$ (which can be reused for each node in the temporal random walk). The embedding matrix $\mathbf{Z}$ takes $\mathcal{O}(ND)$ space. Therefore, the space complexity of CTDNEs is $\mathcal{O}(M + ND + \Delta) = \mathcal{O}(M + ND)$. This obviously holds in the online stream setting where edges arrive continuously over time and updates are made in an online fashion since this is a special case of the more general CTDNE setting.

5 Experiments

The experiments are designed to investigate the effectiveness of the proposed *continuous-time dynamic network embeddings* (CTDNE) framework for prediction. To ensure the results and findings of this work are significant and meaningful, we investigate a wide range of temporal networks from a variety of application domains with fundamentally different structural and temporal characteristics. A summary of the dynamic networks used for evaluation and their statistics are provided in Table 1. All networks investigated are continuous-time dynamic networks with $\mathbb{T} = \mathbb{R}^+$. For these dynamic networks, the time scale of the edges is at the level of seconds or milliseconds, *i.e.*, the edge timestamps record the time an edge occurred at the level of seconds or milliseconds (finest granularity given as input). Our approach uses the finest time scale available in the graph data as input. All data is from NetworkRepository [53] and is easily accessible for reproducibility.

We designed the experiments to answer three important questions. First, are continuous-time dynamic network embeddings more useful than embeddings derived from methods that ignore time? Second, does the choice of distribution $\mathbb{F}_s$ and $\mathbb{F}_\Gamma$ (used to derive unbiased/biased temporal random walks for learning CTDNE's) depend on the underlying graph structure and category/domain it arises from (*e.g.*,

TABLE 1
Dynamic Network Data and Statistics.

Let $|E_T|$ = number of *temporal edges*; $\bar{d}$ = average temporal node degree; and $d_{\max}$ = max temporal node degree.

Dynamic Network	$\lvert E_T \rvert$	$\bar{d}$	$d_{\max}$	**Timespan (days)**
ia-contact	28.2K	206.2	2092	3.97
ia-hypertext	20.8K	368.5	1483	2.46
ia-enron-employees	50.5K	669.8	5177	1137.55
ia-radoslaw-email	82.9K	993.1	9053	271.19
ia-email-EU	332.3K	674.1	10571	803.93
fb-forum	33.7K	75.0	1841	164.49
soc-bitcoinA	24.1K	12.8	888	1901.00
soc-wiki-elec	107K	30.1	1346	1378.34

social networks, information networks); or is there particular distributions (uniform, linear, exponential) that perform best irregardless? Third, are CTDNEs better than embeddings learned from a sequence of discrete snapshot graphs (DTNE methods)?

5.1 Experimental setup

Since this work is the first to learn embeddings over a CTDN, there are no methods that are directly comparable. Nevertheless, we evaluate the framework presented in Section 3 for learning continuous-time dynamic network representations by first comparing CTDNE against a number of recent embedding methods including node2vec [4], DeepWalk [3], and LINE [5]. For node2vec, we use the same hyperparameters ($D = 128$, $R = 10$, $L = 80$, $\omega = 10$) and grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as mentioned in [4]. The same hyperparameters are used for DeepWalk (with the exception of p and q). Unless otherwise mentioned, CTDNE methods use $\omega = 10$ and $D = 128$. Recall that L in CTDNE is the maximum length of a temporal random walk and can be set to any reasonable length. In this work, we set $L = 80$; however, as shown in Figure 6 the maximum length of a temporal random walk is often much smaller (and always smaller for the graphs investigated in this work). Hence, L is never actually used to terminate a temporal random walk. For LINE, we also use $D = 128$ with 2nd-order-proximity and number of samples $T = 60$ million.

TABLE 2
AUC Scores for Temporal Link Prediction.

Dynamic Network	DeepWalk	Node2Vec	LINE	CTDNE	(GAIN)
ia-contact	0.845	0.874	0.736	**0.913**	(+10.37%)
ia-hypertext	0.620	0.641	0.621	**0.671**	(+6.51%)
ia-enron-employees	0.719	0.759	0.550	**0.777**	(+13.00%)
ia-radoslaw-email	0.734	0.741	0.615	**0.811**	(+14.83%)
ia-email-EU	0.820	0.860	0.650	**0.890**	(+12.73%)
fb-forum	0.670	0.790	0.640	**0.826**	(+15.25%)
soc-bitcoinA	0.840	0.870	0.670	**0.891**	(+10.96%)
soc-wiki-elec	0.820	0.840	0.620	**0.857**	(+11.32%)

5.2 Comparison

We evaluate the performance of the proposed framework on the temporal link prediction task. To generate a set of labeled examples for link prediction, we first sort the edges in each graph by time (ascending) and use the first 75% for representation learning. The remaining 25% are considered as positive links and we sample an equal number of negative edges randomly. Since the temporal network is a multi-graph where an edge between two nodes can appear multiple times with different timestamps, we take care to ensure edges that appear in the training set do not appear in the test set. We perform link prediction on this labeled data $\mathcal{X}$ of positive and negative edges. After the embeddings are learned for each node, we derive edge embeddings by combining the learned embedding vectors of the corresponding nodes. More formally, given embedding vectors $\mathbf{z}_i$ and $\mathbf{z}_j$ for node i and j, we derive an edge embedding vector $\mathbf{z}_{ij} \in \mathbb{R}^D$ as:

$$\mathbf{z}_{ij} = \Phi(\mathbf{z}_i, \mathbf{z}_j) \tag{15}$$

where

$$\Phi \in \big\{(\mathbf{z}_i + \mathbf{z}_j)/2,\;\; \mathbf{z}_i \odot \mathbf{z}_j,\;\; |\mathbf{z}_i - \mathbf{z}_j|\,,\;\; (\mathbf{z}_i - \mathbf{z}_j)^{\circ 2}\big\}$$

and $\mathbf{z}_i \odot \mathbf{z}_j$ is the element-wise (Hadamard) product and $\mathbf{z}^{\circ 2}$ is the Hadamard power. We use logistic regression (LR) with hold-out validation of 25%. Experiments are repeated for 10 random seed initializations and the average performance is reported. Unless otherwise mentioned, we use ROC AUC (denoted as AUC for short) to evaluate the models and use the same number of dimensions D for all models.

The same amount of information is used for learning by all baseline methods. In particular, the number of *temporal context windows* is $\beta = R \times N \times (L - \omega + 1)$ where R denotes the number of walks for each node and L is the length of a random walk required by the baseline methods. Recall that R and L are *not* required by CTDNE and are only used above to ensure that all methods use exactly the same amount of information for evaluation purposes. Note since CTDNE does not collect a fixed amount of random walks (of a fixed length) for each node as done by many other embedding methods [3], [4], instead the user simply specifies the # of temporal context windows (expected) per node and the total number of temporal context windows β is derived as a multiple of the number of nodes $N = |V|$. Hence, CTDNE is also easier to use as it requires a lot less hyperparameters that must be carefully tuned by the user. Observe that it is possible (though unlikely) that a node $u \in V$ is not in a valid temporal walk, *i.e.*, the node does not appear in any temporal walk S_t with length at least $|S_t| > \omega$. If such a case occurs, we simply relax the notion of temporal random walk for that node by ensuring the node appears in at least one random walk of sufficient length, even if part of the random walk does not obey time. As an aside, relaxing the notion of temporal random walks by allowing the walk to sometimes violate the time-constraint can be viewed as a form of regularization.

Results are shown in Table 2. For this experiment, we use the simplest CTDNE variant from the proposed framework and did not apply any *additional bias* to the selection strategy. In other words, both $\mathbb{F}_s$ and $\mathbb{F}_\Gamma$ are set to the uniform distribution. We note, however, that since temporal walks are time-obeying (by Definition 2), the selection is already biased towards edges that appear later in time as the random walk traversal does not go back in time. In Table 2, the proposed approach is shown to perform consistently better than DeepWalk, node2vec, and LINE. This is an indication that important information is lost when temporal information is ignored. Strikingly, the CTDNE model does not leverage the bias introduced by node2vec [4], and yet still outperforms this model by a significant margin. We could have generalized node2vec in a similar manner using the proposed framework from Section 3. Obviously, we can expect to achieve even better predictive performance by using the CTDNE framework to derive a continuous-time node2vec generalization by replacing the notion of random walks in node2vec with the notion of *temporal random walks* biased by the (weighted) distributions $\mathbb{F}_s$ (Section 3.1) and $\mathbb{F}_\Gamma$ (Section 3.2).

In all cases, the proposed approach significantly outperforms the other embedding methods across all dynamic networks (Table 2). The mean gain in AUC averaged over all embedding methods for each dynamic network is shown in Table 2. Notably, CTDNE achieves an overall gain in AUC of 11.9% across all embedding methods and graphs. These results indicate that modeling and incorporating the temporal dependencies in graphs is important for learning appropriate and meaningful network representations. It is also worth noting that many other approaches that leverage random walks can also be generalized using the proposed framework [8], [9], [11], [12], [28], as well as any future state-of-the-art embedding method.

We also find that using a biased distribution (*e.g.*, linear or exponential) improves predictive performance in terms of AUC when compared to the uniform distribution on many graphs. For others however, there is no noticeable gain in performance. This can likely be attributed to the fact that most of the dynamic networks investigated have a relatively short time span (more than 3 years at most). Table 3 provides results for a few other variants from the framework. In particular, Table 3 shows the difference in AUC when applying a biased distribution to the initial edge selection strategy $\mathbb{F}_s$ as well as the temporal neighbor selection $\mathbb{F}_\Gamma$ for the temporal random walk. Interestingly, using a biased distribution for $\mathbb{F}_s$ seems to improve more on the tested datasets. However, for ia-enron-employees, the best result can be observed when both distributions are biased.

We also investigate the difference between discrete-time models that learn embeddings from a sequence of discrete snapshot graphs, and the class of continuous-time embeddings proposed in this paper.

DEFINITION 5 (DTDN EMBEDDING) *A discrete-time dynamic network embedding (DTDNE) is defined as any embedding derived from a sequence of discrete static snapshot graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_t\}$. This includes any embedding learned from temporally smoothed static graphs or any representation derived from the initial sequence of discrete static snapshot graphs.*

Previous work for temporal networks have focused on DTDNE methods as opposed to the class of CTDNE methods proposed in this work. Notice that DTDNE methods use *approximations* of the actual dynamic network whereas the CTDN embeddings do not and leverage the actual valid temporal information without any temporal loss. In this experiment, we create discrete snapshot graphs and learn embeddings for each one using the previous approaches. As an example, suppose we have a sequence of $T = 4$ snapshot graphs where each graph represents a day of activity and further suppose $D = 128$. For each snapshot graph, we learn a (D/T)-dimensional embedding and concatenate them all to obtain a D-dimensional embedding and then evaluate the embedding for link prediction as described previously.

TABLE 3
Results for Different CTDNE Variants

$\mathbb{F}_s$ is the distribution for initial edge sampling and $\mathbb{F}_\Gamma$ is the distribution for temporal neighbor sampling.

Variant $\mathbb{F}_s$	Variant $\mathbb{F}_\Gamma$	contact	hyper	enron	rado
Unif (Eq. 1)	**Unif** (Eq. 5)	0.913	0.671	0.777	0.811
Unif (Eq. 1)	**Lin** (Eq. 7)	0.903	0.665	0.769	0.797
Lin (Eq. 3)	**Unif** (Eq. 5)	0.915	0.675	0.773	0.818
Lin (Eq. 3)	**Lin** (Eq. 7)	0.903	0.667	0.782	0.806
Exp (Eq. 2)	**Exp** (Eq. 6)	**0.921**	0.681	**0.800**	0.820
Unif (Eq. 1)	**Exp** (Eq. 6)	0.913	0.670	0.759	0.803
Exp (Eq. 2)	**Unif** (Eq. 5)	0.920	**0.718**	0.786	**0.827**
Lin (Eq. 3)	**Exp** (Eq. 6)	0.916	0.681	0.782	0.823
Exp (Eq. 2)	**Lin** (Eq. 7)	0.914	0.675	0.747	0.817

TABLE 4
Results Comparing DTDNEs to CTDNEs (AUC)

CTDNE-Unif uses uniform for both $\mathbb{F}_s$ and $\mathbb{F}_\Gamma$ whereas CTDNE-Opt selects the distributions via model learning (and hence corresponds to the best model).

Dynamic Network	DTDNE	CTDNE-Unif	CTDNE-Opt	(GAIN)
ia-contact	0.843	0.913	**0.921**	(+8.30%)
ia-hypertext	0.612	0.671	**0.718**	(+9.64%)
ia-enron-employees	0.721	0.777	**0.800**	(+7.76%)
ia-radoslaw-email	0.785	0.811	**0.827**	(+3.31%)

A challenging problem common with DTDNE methods is how to handle nodes that are not active in a given static snapshot graph G_i (*i.e.*, the node has no edges that occur in G_i). In such situations, we set the node embedding for that static snapshot graph to all zeros. However, we also investigated using the node embedding from the last active snapshot graph as well as setting the embedding of an inactive node to be the mean embedding of the active nodes in the given snapshot graph and observed similar results.

More importantly, unlike DTDNE methods that have many issues and heuristics required to handle them (*e.g.*, the time-scale, how to handle inactive nodes, etc), CTDNEs do not. CTDNEs also avoid many other issues [54] discussed previously that arise from DTDN embedding methods that use a sequence of discrete static snapshot graphs to approximate the actual dynamic network. For instance, it is challenging and unclear how to select the "best" most appropriate time-scale used to create the sequence of static snapshot graphs; and the actual time-scale is highly dependent on the temporal characteristics of the network and the underlying application. More importantly, all DTDNs (irregardless of the time-scale) are *approximations* of the actual dynamic network. Thus, any DTDN embedding method is inherently lossy and is only as good as the discrete approximation of the CTDN (graph stream). Results are provided in Table 4. Since node2vec always performs the best among the baseline methods (Table 2), we use it as a basis for the DTDN embeddings. Overall, the proposed CTDNEs perform better than DTDNEs as shown in Table 4. Note that CTDNE in Table 4 corresponds to using uniform for both $\mathbb{F}_s$ and $\mathbb{F}_\Gamma$. Obviously, better results can be achieved by learning $\mathbb{F}_s$ and $\mathbb{F}_\Gamma$ automatically as shown in Table 3. The gain in AUC for each graph is shown in the rightmost column in Table 4. The mean gain in AUC of CTDNE compared to DTDNE over all graphs is 7.25%.

Online Streaming Results: For some applications, it may be important to handle edges as soon as they arrive in a streaming fashion. In such a streaming setting, we perform fast partial updates to obtain updated embeddings in real-time. Given an edge (i, j, t) at time t, we simply obtain a few temporal walks ending at (i, j) and use these to obtain the updated embeddings. An example is shown in Figure 9. In these experiments, we use online SGD updates (online word2vec) [38], [39], [40], [41] to incrementally learn the embeddings as new edges arrive. However, other incremental optimization schemes can be used as well (*e.g.*, see [42],

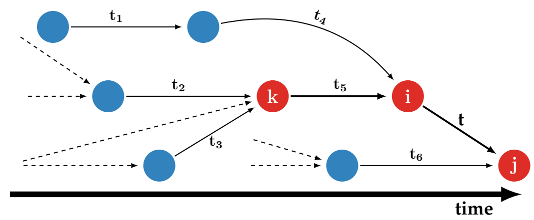


Fig. 9. Temporal Walks for Online CTDNEs. Given a new edge (i, j, t) at time t, we immediately add it to the graph and then sample temporal walks ending at edge (i, j) and use them to update the relevant embeddings. An example of a temporal walk is $k \rightarrow i \rightarrow j$ (red nodes). Note $t > t_6 > t_5 > t_4 > t_3 > t_2 > t_1$. In this example, k and j are the training instances. Hence, $\mathbf{z}_i$ is updated every time i is used in a temporal edge.

[43], [44], [45], [46], [47]). We vary the number of temporal walks sampled for every new edge that arrives. Results are shown in Table 5. Notably, it takes on average only a few milliseconds to update the embeddings across a wide variety of temporal network streams. Note these results are from a python implementation of the approach and thus the runtime to process a single edge in the stream can be significantly reduced even further using a C/C++ implementation of the incremental/online learning approach.

More sophisticated stream network embedding approaches can be developed that instead of using every new edge, we select in real-time whether to update the embeddings using that edge (similar in spirit to other stream graph algorithms [22]). Furthermore, instead of updating the embedding after each new edge in the graph stream, we can update embeddings in a batch fashion (*e.g.*, after some time period or enough edges arrive). Both strategies can be used to improve performance.

TABLE 5
Streaming Online Network Embedding Results

Average runtime (in milliseconds) per edge is reported. We vary the number of walks per new edge from 1 to 10. Recall $|E_T|$ = # of *temporal edges* and $\bar{d}$ = average temporal node degree.

			Time (ms.)		
Dynamic Network	$\lvert E_T \rvert$	$\bar{d}$	**1**	**5**	**10**
ia-hypertext	20.8K	368.5	2.769	3.721	4.927
fb-forum	33.7K	75.0	2.875	3.412	4.230
soc-wiki-elec	107K	30.1	2.788	3.182	3.813
ia-contact	28.2K	206.2	2.968	4.490	6.119
ia-radoslaw-email	82.9K	993.1	3.266	5.797	8.916
soc-bitcoinA	24.1K	12.8	2.679	2.965	3.347

6 Challenges & Future Directions

Attributed Networks & Inductive Learning: The proposed framework for learning *dynamic node embeddings* can be easily generalized to *attributed networks* and for *inductive learning* tasks in temporal networks (graph streams) using the ideas introduced in [37], [55]. More formally, the notion of attributed/feature-based walk (proposed in [37], [55]) can be combined with the notion of temporal random walk (proposed in this paper) as follows [56]:

Definition 6 (Attributed Temporal Walk) *Let $\mathbf{x}_i$ be a d-dimensional feature vector for node v_i. An attributed temporal walk S of length L is defined as a sequence of adjacent node feature-values $\phi(\mathbf{x}_{i_1}), \phi(\mathbf{x}_{i_2}), \ldots, \phi(\mathbf{x}_{i_{L+1}})$ associated with a sequence of indices $i_1, i_2, \ldots, i_{L+1}$ such that*

1) *$(v_{i_t}, v_{i_{t+1}}) \in E_T$ for all $1 \leq t \leq L$*
2) *$\mathcal{T}(v_{i_t}, v_{i_{t+1}}) \leq \mathcal{T}(v_{i_{t+1}}, v_{i_{t+2}})$ for $1 \leq t < L$*
3) *$\phi : \mathbf{x} \rightarrow y$ is a function that maps the input vector $\mathbf{x}$ of a node to a corresponding feature-value $\phi(\mathbf{x})$.*

The feature sequence $\phi(\mathbf{x}_{i_1}), \phi(\mathbf{x}_{i_2}), \ldots, \phi(\mathbf{x}_{i_{L+1}})$ represents the feature-values that occur during a temporally valid walk, i.e., a walk they obeys the direction of time defined in (2).

Attributed temporal random walks can be either uniform (unbiased) or non-uniform (biased). Furthermore, the features used in attributed walks can be (i) intrinsic input attributes (such as profession, political affiliation), (ii) structural features derived from the graph topology (degree, triangles, etc; or even node embeddings from an arbitrary method), or both.

The above formulation is only one such CTDNE approach for handling (i) attributed networks (heterogeneous networks) and (ii) *inductive temporal network representation learning*. Both are challenging problems that remain unsolved. Another interesting research direction is to use *heterogeneous network motifs* [57] to capture the higher-order connectivity and type/attribute structure in heterogeneous networks.

Other Types of Temporal Networks: While this work naturally supports temporal networks and graph streams in general, there are many other networks with more specialized characteristics. For instance, some temporal networks (graph streams) contain edges with start and end times. Developing CTDNE methods for such temporal networks remains a challenge. Furthermore, another open and challenging problem that remains to be addressed is how to develop graph stream embedding techniques that require a fixed amount of space. Other applications may require dynamic node embedding methods that are space-efficient (*e.g.*, by learning a sparse vector representation for each node).

Temporal Weighting and Bias: This paper explored a number of temporal weighting and bias functions for decaying the weights of data that appears further in the past. More research is needed to fully understand the impact and to understand the types of temporal networks and characteristics that each should be used. Some early work has focused on temporally weighting the links, nodes, and attributes prior to learning embeddings [18]. However, this idea has yet to be explored for learning general node embeddings and should be investigated in future work. Other research should investigate new temporal weighting schemes for links, nodes, and attributes [18]. Furthermore, one can also incorporate a decay function for each temporal walk such that more temporal influence is given to recent nodes in the walk than to nodes in the distant past. Hence, each temporal walk is assigned a sequence of weights which can be incorporated into the Skip-Gram approach. For instance, in the case of an exponential decay function $\alpha^{t-1} \cdot \alpha^{t-2} \cdots \alpha^{t-k}$. However, there are many other ways to temporally weight or bias the walk and it is unclear when one approach works better than another. Future work should systematically investigate different approaches.

7 Related Work

The node embedding problem has received considerable attention from the research community in recent years.

See [58] for an early survey on representation learning in relational/graph data. The goal is to learn encodings (embeddings, representations, features) that capture key properties about each node such as their role in the graph based on their structural characteristics (*i.e.*, roles capture distinct structural properties, *e.g.*, hub nodes, bridge nodes, near-cliques) [59] or community (*i.e.*, communities represent groups of nodes that are close together in the graph based on proximity, cohesive/tightly connected nodes) [60], [61].[3] Since nodes that share similar roles (based on structural properties) or communities (based on proximity, cohesiveness) are grouped close to each other in the embedding space, one can easily use the learned embeddings for tasks such as ranking [62], community detection [60], [61], role embeddings [59], [63], link prediction [64], and node classification [18].

Many of the techniques that were initially proposed for solving the node embedding problem were based on graph factorization [6], [65], [66]. More recently, the skip-gram model [36] was introduced in the natural language processing domain to learn vector representations for words. Inspired by skip-gram's success in language modeling, various methods [3], [4], [5] have been proposed to learn node embeddings using skip-gram by treating a graph as a "document." Two of the more notable methods, DeepWalk [3] and node2vec [4], use random walks to sample an ordered sequence of nodes from a graph. The skip-gram model can then be applied to these sequences to learn node embeddings.

Researchers have also tackled the problem of node embedding in more complex graphs, including attributed networks [9], heterogeneous networks [28] and dynamic networks [13], [67], [68]. However, the majority of the work in the area still fail to consider graphs that evolve over time (*i.e.*temporal graphs). A few work have begun to explore the problem of learning node embeddings from temporal networks [13], [14], [15], [16], [17], [69]. All of these approaches *approximate* the dynamic network as a sequence of discrete static snapshot graphs, which are fundamentally different from the class of continuous-time dynamic network embedding methods introduced in this work. Notably, this work is the first to propose *temporal random walks* for embeddings as well as *CTDN embeddings* that use temporal walks to capture the actual temporally valid sequences observed in the CTDN; and thus avoids the issues and information loss that arises when embedding methods simply ignore time or use discrete static snapshot graphs (See Figure 2 for one example). Furthermore, we introduce a unifying framework that can serve as a basis for generalizing other random walk based deep learning (*e.g.*, [12]) and embedding methods (*e.g.*, [4], [8], [9], [11], [28], [70]) for learning more appropriate time-dependent embeddings from temporal networks. Note that we previously studied this idea in [54], [71]. These ideas have been significantly refined for this manuscript. In contrast, previous work has simply introduced new approaches for temporal networks [14] and therefore they focus on an entirely different problem than the one in this work which is a general framework that can be leveraged by other non-temporal approaches. Other work has focused on incremental algorithms (also called dynamic algorithms) for updating spectral clustering as new information arrives [72], which is different from the problem studied in this paper.

3. Notice that while communities are based on proximity (nodes in a community must be close to one another in the graph) and cohesiveness/tightly connected, roles are not based on proximity, and thus nodes with similar roles may be in different communities as roles simply capture nodes that are structurally similar; hence commuities and roles are complimentary concepts; see [59]

TABLE 6
Qualitative Comparison of the Different Classes of Embedding Methods

Qualitative comparison of CTDNE methods to existing methods categorized as either static methods (that ignore time) or DTDNE methods that approximate the actual dynamic network using a sequence of discrete static snapshot graphs. Does the method: use the actual dynamic network at the finest temporal granularity, *e.g.*, seconds or ms (or do they use discrete static approximations of the dynamic network); temporally valid; use temporal bias/smoothing functions to give more importance to recent or temporally recurring information; and does it naturally support graph streams and the streaming/online setting in general where data is continuously arriving over time and embeddings can be incrementally updated in an online fashion.

	Temporally valid	**Finest granularity**	**Temporal bias/smoothing**	**Streaming**
Static	✗	✗	✗	✗
DTDNE	✗	✗	✓	✗
CTDNE	✓	✓	✓	✓

Temporal graph smoothing of a sequence discrete static snapshot graphs was proposed for classification in dynamic networks [18]. The same approach has also been used for deriving role-based embeddings from dynamic networks [13], [73]. More recently, these techniques have been used to derive more meaningful embeddings from a sequence of discrete static snapshot graphs [16], [17]. All of these approaches model the dynamic network as a sequence of discrete static snapshot graphs, which is fundamentally different from the class of continuous-time dynamic network embedding methods introduced in this work. Table 6 provides a qualitative comparison of CTDNE methods to existing static methods or DTDNE methods that approximate the dynamic network as a discrete sequence of static snapshot graphs.

Random walks on graphs have been studied for decades [74]. The theory underlying random walks and their connection to eigenvalues and other fundamental properties of graphs are well-understood [75]. Our work is also related to uniform and non-uniform random walks on graphs [74], [75]. Random walks are at the heart of many important applications such as ranking [62], community detection [60], [61], recommendation [76], link prediction [64], and influence modeling [33]. search engines [77], image segmentation [78], routing in wireless sensor networks [79], and time-series forecasting [31]. These applications and techniques may also benefit from the proposed notion of *temporal random walks*. Recently, Ahmed *et al.* [37] proposed the notion of *attributed random walks* that can be used to generalize existing methods for inductive learning and/or graph-based transfer learning tasks. In future work, we will investigate combining both attributed random walks and temporal random walks to derive even more powerful embeddings from networks.

More recently, there has been significant research in developing network analysis and machine learning methods for modeling temporal networks. Temporal networks have been the focus of recent research including node classification in temporal networks [18], temporal link prediction [80], dynamic community detection [81], dynamic mixed-membership role models [13], [73], [82], anomaly detection in dynamic networks [83], influence modeling and online advertisement [84], finding important entities in dynamic networks [31], [85], and temporal network centrality and measures [35].

8 Conclusion

In this work, we introduced the notion of *temporal walks* for learning embeddings from temporal networks (graph streams). Based on the proposed notion of temporal walks, we described a new class of embeddings that are learned directly from the temporal network (graph stream) without having to approximate the edge stream as a sequence of discrete static snapshot graphs. As such these embeddings can be learned in an online fashion as they are naturally amenable to graph streams and incremental updates. We investigated a framework for learning such dynamic node embeddings using the proposed notion of temporal walks. The embeddings can be learned incrementally or in a streaming fashion where embeddings are updated in real-time as new edges arrive. The proposed approach can be used as a basis for generalizing existing (or future state-of-the-art) random walk-based embedding methods for learning of dynamic node embeddings from dynamic networks (graph streams). The result is a more appropriate dynamic node embedding that captures the important temporal properties of the node in the continuous-time dynamic network. By learning dynamic node embeddings based on temporal walks, we avoid the issues and information loss that arise when time is ignored or approximated using a sequence of discrete static snapshot graphs. In contrast to previous work, the proposed class of embeddings are learned from temporally valid information. The experiments demonstrated the effectiveness of this new class of dynamic embeddings on several real-world networks.

References

[1] D. Watts and S. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[2] M. Newman, "The structure of scientific collaboration networks," *PNAS*, vol. 98, no. 2, pp. 404–409, 2001.
[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *SIGKDD*, 2014, pp. 701–710.
[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.
[5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale Information Network Embedding," in *WWW*, 2015.
[6] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *CIKM*, 2015, pp. 891–900.
[7] R. A. Rossi, R. Zhou, and N. K. Ahmed, "Deep inductive network representation learning," in *WWW BigNet*, 2018.
[8] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *SIGKDD*, 2017.
[9] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," *arXiv:1705.04969*, 2017.
[10] N. K. Ahmed, R. A. Rossi, R. Zhou, J. B. Lee, X. Kong, T. L. Willke, and H. Eldardiry, "Inductive representation learning in large attributed graphs," in *WiML NIPS*, 2017.
[11] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *CIKM*, 2017, pp. 377–386.
[12] J. B. Lee, R. A. Rossi, and X. Kong, "Graph classification using structural attention," in *SIGKDD*, 2018, pp. 1–9.
[13] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson, "Modeling dynamic behavior in large evolving graphs," in *WSDM*, 2013, pp. 667–676.
[14] R. Hisano, "Semi-supervised graph embedding approach to dynamic link prediction," *arXiv:1610.04351*, 2016.
[15] N. Kamra, U. Gupta, and Y. Liu, "Deep generative dual memory network for continual learning," *arXiv:1710.10368*, 2017.
[16] T. K. Saha, T. Williams, M. A. Hasan, S. Joty, and N. K. Varberg, "Models for capturing temporal smoothness in evolving networks for learning latent representation of nodes," *arXiv:1804.05816*, 2018.
[17] M. Rahman, T. K. Saha, M. A. Hasan, K. S. Xu, and C. K. Reddy, "Dylink2vec: Effective feature representation for link prediction in dynamic networks," *arXiv:1804.05755*, 2018.
[18] R. A. Rossi and J. Neville, "Time-evolving relational classification and ensemble methods," in *PAKDD*, 2012, p. 13.
[19] S. Soundarajan, A. Tamersoy, E. B. Khalil, T. Eliassi-Rad, D. H. Chau, B. Gallagher, and K. Roundy, "Generating graph snapshots from streaming edge data," in *WWW Companion*, 2016, pp. 109–110.
[20] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu, "Graphscope: parameter-free mining of large time-evolving graphs," in *KDD*, 2007.
[21] C. C. Aggarwal, Y. Zhao, and S. Y. Philip, "Outlier detection in graph streams," in *ICDE*, 2011, pp. 399–409.
[22] N. K. Ahmed, N. Duffield, T. L. Willke, and R. A. Rossi, "On sampling from massive graph streams," in *VLDB*, 2017, pp. 1430–1441.
[23] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin, "On dense pattern mining in graph streams," *VLDB*, vol. 3, no. 1-2, pp. 975–984, 2010.
[24] S. Guha and A. McGregor, "Graph synopses, sketches, and streams: A survey," *VLDB*, vol. 5, no. 12, pp. 2030–2031, 2012.
[25] R. Pienta, J. Abello, M. Kahng, and D. H. Chau, "Scalable graph exploration and visualization: Sensemaking challenges and opportunities," in *BigComp*, 2015, pp. 271–278.
[26] Z. Cai, D. Logothetis, and G. Siganos, "Facilitating real-time graph mining," in *International Workshop on Cloud Data Management*, 2012.
[27] N. K. Ahmed and R. A. Rossi, "Interactive visual graph analytics on the web." in *ICWSM*, 2015, pp. 566–569.
[28] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*, 2017.
[29] K. Lerman and R. Ghosh, "Information contagion: An empirical study of the spread of news on digg and twitter social networks." *ICWSM*, vol. 10, pp. 90–97, 2010.
[30] D. Acemoglu, A. Ozdaglar, and A. ParandehGheibi, "Spread of (mis) information in social networks," *Games & Econ. Behavior*, vol. 70, no. 2, 2010.
[31] D. F. Gleich and R. A. Rossi, "A dynamical system for pagerank with time-dependent teleportation," *Internet Mathematics*, pp. 188–217, 2014.
[32] SocioPatterns, "Infectious contact networks," http://www.sociopatterns.org/datasets/. Accessed 09/12/12.
[33] A. Java, P. Kolari, T. Finin, and T. Oates, "Modeling the spread of influence on the blogosphere," in *WWW*, 2006, pp. 22–26.
[34] P. Domingos, "Mining social networks for viral marketing," *IEEE Intelligent Systems*, vol. 20, no. 1, pp. 80–82, 2005.
[35] P. Holme and J. Saramäki, "Temporal networks," *Phy. Rep.*, 2012.
[36] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR Workshop*, 2013.
[37] N. K. Ahmed, R. A. Rossi, R. Zhou, J. B. Lee, X. Kong, T. L. Willke, and H. Eldardiry, "Learning role-based graph embeddings," in *arXiv:1802.02896*, 2018.
[38] N. Kaji and H. Kobayashi, "Incremental skip-gram model with negative sampling," *arXiv:1704.03956*, 2017.
[39] H. Peng, J. Li, Y. Song, and Y. Liu, "Incrementally learning the hierarchical softmax function for neural language models." in *AAAI*, 2017, pp. 3267–3273.
[40] H. Luo, Z. Liu, H. Luan, and M. Sun, "Online learning of interpretable word embeddings," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1687–1692.
[41] S. Li, J. Zhu, and C. Miao, "Psdvec: A toolbox for incremental and scalable word embedding," *Neurocomp.*, vol. 237, pp. 405–409, 2017.
[42] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.
[43] A. D. Flaxman, A. T. Kalai, and H. B. McMahan, "Online convex optimization in the bandit setting: gradient descent without a gradient," in *SODA*. Society for Industrial and Applied Mathematics, 2005, pp. 385–394.
[44] P. Zhao, J. Wang, P. Wu, R. Jin, and S. C. Hoi, "Fast bounded online gradient descent algorithms for scalable kernel-based online learning," *arXiv:1206.4633*, 2012.

[45] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-newton method for online convex optimization," in *AISTATS*, 2007, pp. 436–443.

[46] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle pointsonline stochastic gradient for tensor decomposition," in *Conference on Learning Theory*, 2015, pp. 797–842.

[47] Y. Ying and M. Pontil, "Online gradient descent learning algorithms," *Foundations of Comp. Math.*, vol. 8, no. 5, pp. 561–596, 2008.

[48] I. Pilászy, D. Zibriczky, and D. Tikk, "Fast ALS-based matrix factorization for explicit and implicit feedback datasets," in *RecSys*, 2010.

[49] H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon, "Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion," *VLDB*, vol. 7, no. 11, pp. 975–986, 2014.

[50] J. Oh, W.-S. Han, H. Yu, and X. Jiang, "Fast and robust parallel sgd matrix factorization," in *SIGKDD*. ACM, 2015, pp. 865–874.

[51] J. Kim, Y. He, and H. Park, "Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework," *J. of Glob. Opt.*, vol. 58, no. 2, pp. 285–319, 2014.

[52] R. A. Rossi and R. Zhou, "Scalable relational learning for large heterogeneous networks," in *DSAA*, 2015, pp. 1–10.

[53] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015, pp. 4292–4293. [Online]. Available: http://networkrepository.com

[54] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *WWW BigNet*, 2018.

[55] N. K. Ahmed, R. A. Rossi, R. Zhou, J. B. Lee, X. Kong, T. L. Willke, and H. Eldardiry, "A framework for generalizing graph-based representation learning methods," in *arXiv:1709.04596*, 2017, pp. 1–8.

[56] D. Jin, M. Heimann, R. A. Rossi, and D. Koutra, "Node2BITS: Compact Time- and Attribute-aware Node Representations for User Stitching," in *arXiv preprint*, 2019, p. 22.

[57] R. A. Rossi, N. K. Ahmed, A. Carranza, D. Arbour, A. Rao, S. Kim, and E. Koh, "Heterogeneous network motifs," in *arXiv:1901.10026*, 2019, p. 18.

[58] R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville, "Transforming graph data for statistical relational learning," *Journal of Artificial Intelligence Research (JAIR)*, vol. 45, no. 1, pp. 363–441, 2012.

[59] R. Rossi and N. Ahmed, "Role discovery in networks," *TKDE*, vol. 27, no. 4, pp. 1112–1131, 2015.

[60] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *NIPS*, 2002, pp. 849–856.

[61] P. Pons and M. Latapy, "Computing communities in large networks using random walks." *J. Graph Alg. Appl.*, vol. 10, no. 2, p. 191, 2006.

[62] L. Page, S. Brin, R. Motwani, and T. Winograd, "PageRank citation ranking: Bringing order to the web," *Stanford Tech. Report*, 1998.

[63] N. K. Ahmed, R. A. Rossi, T. L. Willke, and R. Zhou, *Edge Role Discovery via Higher-Order Structures*, 2017, pp. 291–303.

[64] W. Liu and L. Lü, "Link prediction based on local random walk," *Europhysics Letters*, vol. 89, p. 58007, 2010.

[65] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. S. Smola, "Distributed large-scale natural graph factorization," in *WWW*, 2013, pp. 37–48.

[66] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, pp. 1373–1396, 2002.

[67] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," 2018.

[68] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *CIKM*, 2017, pp. 387–396.

[69] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks," *TKDE*, vol. 28, no. 10, pp. 2765–2777, 2016.

[70] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1025–1035.

[71] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Dynamic network embeddings: From random walks to temporal random walks," in *IEEE BigData*, 2018, pp. 1085–1092.

[72] P.-Y. Chen, B. Zhang, M. A. Hasan, and A. O. Hero, "Incremental method for spectral clustering of increasing orders," *arXiv:1512.07349*, 2015.

[73] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson, "Role-Dynamics: Fast mining of large dynamic networks," in *WWW*, 2012, pp. 997–1006.

[74] L. Lovász, "Random walks on graphs," *Combinatorics*, vol. 2, pp. 1–46, 1993.

[75] F. Chung, "Random walks and local cuts in graphs," *Linear Algebra and its applications*, vol. 423, no. 1, pp. 22–32, 2007.

[76] T. Bogers, "Movie recommendation using random walks over the contextual graph," in *Context-Aware Recommender Systems*, 2010.

[77] J.-L. Lassez, R. Rossi, and K. Jeev, "Ranking links on the web: Search and surf engines," in *IEA/AIE*, 2008, pp. 199–208.

[78] L. Grady, "Random walks for image segmentation," *TPAMI*, vol. 28, no. 11, pp. 1768–1783, 2006.

[79] S. D. Servetto and G. Barrenechea, "Constrained random walks on random graphs: routing algorithms for large scale wireless sensor networks," in *Wireless Sensor Networks & App.*, 2002, pp. 12–21.

[80] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *TKDD*, vol. 5, no. 2, 2011.

[81] R. Cazabet and F. Amblard, "Dynamic community detection," in *ESNAM*. Springer, 2014, pp. 404–414.

[82] W. Fu, L. Song, and E. P. Xing, "Dynamic mixed membership blockmodel for evolving networks," in *ICML*, 2009, pp. 329–336.

[83] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova, "Anomaly detection in dynamic networks: a survey," *Wiley Interdisc. Rev.: Comp. Stat.*, vol. 7, no. 3, pp. 223–247, 2015.

[84] A. Goyal, F. Bonchi, and L. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*. ACM, 2010, pp. 241–250.

[85] J. O'Madadhain, J. Hutchins, and P. Smyth, "Prediction and ranking algorithms for event-based network data," *SIGKDD Explorations*, vol. 7, no. 2, p. 30, 2005.