A Survey on Embedding Dynamic Graphs

Claudio D. T. Barros[1], Matheus R. F. Mendonça[1], Alex B. Vieira[2], and Artur Ziviani[1]

[1]National Laboratory for Scientific Computing (LNCC), Petrópolis, RJ, Brazil
[2]Federal University of Juiz de Fora (UFJF), Juiz de Fora, MG, Brazil

Abstract

Embedding static graphs in low-dimensional vector spaces plays a key role in network analytics and inference, supporting applications like node classification, link prediction, and graph visualization. However, many real-world networks present dynamic behavior, including topological evolution, feature evolution, and diffusion. Therefore, several methods for embedding dynamic graphs have been proposed to learn network representations over time, facing novel challenges, such as time-domain modeling, temporal features to be captured, and the temporal granularity to be embedded. In this survey, we overview dynamic graph embedding, discussing its fundamentals and the recent advances developed so far. We introduce the formal definition of dynamic graph embedding, focusing on the problem setting and introducing a novel taxonomy for dynamic graph embedding input and output. We further explore different dynamic behaviors that may be encompassed by embeddings, classifying by topological evolution, feature evolution, and processes on networks. Afterward, we describe existing techniques and propose a taxonomy for dynamic graph embedding techniques based on algorithmic approaches, from matrix and tensor factorization to deep learning, random walks, and temporal point processes. We also elucidate main applications, including dynamic link prediction, anomaly detection, and diffusion prediction, and we further state some promising research directions in the area.

Keywords: dynamic networks, graph embedding, graph representation learning, dynamic graphs, dynamic graph embedding.

1 Introduction

Graph-structured networks arise naturally in several real-world complex systems, including social networks, biological networks, knowledge graphs, and finance, whose interactions between nodes allow the understanding of the structural information of these domains [1]. Therefore, graph-aware learning tasks play a key role in machine learning and network science literature, and scalable approaches to deal with these high-dimensional non-Euclidean data have been explored to address the computational challenges associated with graph data-driven analytics and inference. Embedding graphs in low-dimensional vector spaces have been applied to extract features from networks and encoding topological and semantic information, and many researchers have been using these network representation learning approaches for several applications, including node classification and clustering, link prediction, and network visualization [2, 3, 4].

Previous work on network representation learning has focused on static graphs, either representing existing connections at a fixed moment (i.e., a graph snapshot), or node interactions over time that are aggregated into a single static graph [5]. However, several real networks display dynamic behavior, including nodes and edges being added or removed from the system [6], labels and other properties changing over time [7], and diffusion in the network [8]. The network temporal correlations are lost during the aggregating process and, in this sense, approaches to develop embedding methods for dynamic

networks have been proposed over the past few years. These efforts have improved tasks such as link prediction [9] and node classification [10] over time, while enabling novel applications, including event prediction [8], anomaly detection [10] and diffusion prediction [11].

Several challenges arise when developing an approach to embed dynamic graphs, as (i) how to model the time domain, i.e. discrete-time or continuous-time, (ii) which dynamic behaviors are desired to be captured, and (iii) which temporal granularity will be represented in the vector space, i.e. the same granularity as the dataset, or a coarser granularity summarizing dynamics in a finer timescale. Considering the increasing number of studies proposing dynamic graph embedding techniques, these discussions are becoming more important to advance the comprehension of dynamic network representation learning. Therefore, in this survey, we overview the problem of embedding dynamic graphs, discussing its fundamental aspects and the recent advances that have been made so far. We introduce the formal definition of dynamic graph embedding, discussing different dynamic network models whose representations may be extracted, and introducing a novel taxonomy for the problem settings, i.e. embeddings input and output. Moreover, we explore and classify different dynamic behaviors that may be captured by embeddings, describe existing methods, discuss their similarities and differences, and propose a detailed taxonomy based on algorithmic approaches.

To the best of our knowledge, a few attempts have been made so far to survey dynamic graph embeddings. Kazemi et al. [12] focus on recent representation learning techniques for dynamic graphs by using an encoder-decoder framework. Xie et al. [13] propose a taxonomy based on algorithmic approaches to encode graphs. Additionally, Skarding et al. [14] survey how the dynamic network topology can be modeled using dynamic graph neural networks. Our work is different from the aforementioned surveys since we discuss different dynamic network models that have been or may be used for embedding, in addition to detecting temporal behaviors in networks that can be captured. Moreover, we extend dynamic graph embedding techniques taxonomy, encompassing methods based on graph kernels, temporal point processes, and agnostic methods. In this sense, this survey has the following contributions:

- A taxonomy of dynamic graph embedding based on problem settings, extending graph embedding input and output to handle temporal heterogeneity (i.e. timestamps with labels, classifying network behavior over time) and temporal embeddings (i.e. different temporal granularities to represent in the low-dimensional vector space);
- A discussion about different dynamic behaviors in networks that embedding models may capture, including topological evolution (concerning both node and edge addition or removal), feature evolution (regarding changes of nodes/edges features or labels over time) and processes on networks (diffusion and global role of nodes and its evolution). Furthermore, we also bring some perspectives about temporal point processes on networks.
- A detailed analysis of embedding techniques for dynamic graphs, focused on a classification concerning their algorithmic approaches, comparing different methods proposed in the literature and discussing their similarities, differences and other particularities;
- The categorization, according to the topological structure, of several dynamic graph embedding applications, focused on: node related tasks, edge related tasks, node, and edge related tasks, and graph-related tasks;
- A discussion of future research directions in the area in terms of problem settings, solution techniques, and modeling, in addition to applications and representation learning on generalized graphs (i.e. hypergraphs and higher-order graphs).

The remainder of this survey is organized as follows. In Section 2, we introduce the fundamentals behind the embedding of dynamic graphs, reviewing static graph embedding, defining the problem of dynamic graph embedding, presenting different dynamic graph models explored in the embedding scenario, along with other problem settings, including the dynamic graph embedding input and output as well as

the dynamic behaviors that may be captured. In Section 3, we categorize the literature based on the embedding techniques, unraveling insights behind each paradigm and we provide a detailed comparison of different techniques. After that, we present in Section 4 some concrete examples of applications enabled by dynamic graph embedding methods discussed in Section 3, allowing the reader to better grasp the practical utility of these methods. Finally, our conclusions are presented in Section 5, alongside some discussions on potential future research directions in the field of dynamic graph embedding.

2 Fundamentals Behind the Embedding of Dynamic Graphs

In this section, we first review basic graph concepts and static graph embedding, introducing the definition of dynamic graphs. Then, we formally define the dynamic graph embedding problem. Thereafter, we describe possible problem settings, starting with dynamic graph embedding input and detailing different outputs. We expand the initial graph embedding concepts considering time-varying graph models, time granularity, and temporal aggregation, and discuss dynamic behaviors that may be captured by embeddings.

2.1 Graphs and Static Graph Embedding

A **graph** $G = (V, E)$ is a mathematical structure, where $V = \{v_1, ..., v_N\}$ is a finite set of N nodes (vertices), and $E \subseteq \{(v_i, v_j) \,|\, (v_i, v_j) \in V \times V\}$ is a finite set of unordered pairs of vertices, whose elements $e_{ij} = (v_i, v_j)$ are called edges (links). The **adjacency matrix** A of a graph is an $N \times N$ matrix whose element $A_{ij} = 1$ if edge $e_{ij} \in E$, or $A_{ij} = 0$ otherwise. A **directed graph** is a graph in which an edge $e_{ij} \in E$ is an ordered pair, i.e. the edge e_{ij} is oriented. Otherwise, the graph is **undirected**. A **weighted graph** is a graph in which a weight function $w : E \to \mathbb{R}$ is assigned to it. Each edge has a weight associated with it, and it is possible to define a **weight matrix** W such that $W_{ij} = w(e_{ij})$. Otherwise, the graph is **unweighted**. A **homogeneous graph** is a graph in which the number of node types $\mathcal{L}^n$ and the number of edge types $\mathcal{L}^e$ is 1, i.e. $|\mathcal{L}^v| = |\mathcal{L}^e| = 1$, and every node in G belongs to a single node category and every edge belongs to a single edge category. A **heterogeneous graph** is a graph in which $|\mathcal{L}^v| > 1$ or $|\mathcal{L}^e| > 1$.

Different ways to define **proximity** or **similarity** between nodes in a graph may be conceived [3]. The **first-order proximity** $S_{ij}^{(1)}$ between nodes v_i and v_j is the weight of the edge e_{ij}, i.e., W_{ij}. The **second-order proximity** $S_{ij}^{(2)}$ between nodes v_i and v_j is a similarity between v_i's neighbourhood $S_i^{(1)}$ and v_j's neighbourhood $S_j^{(1)}$ given by some defined metric, where $S_i^{(1)} = [S_{i1}^{(1)}, ..., S_{iN}^{(1)}]$ and $S_j^{(1)} = [S_{j1}^{(1)}, ..., S_{jN}^{(1)}]$. **Higher-order proximities** can be defined as well, including the Katz centrality [15], which is a weighted summation over the paths between two vertices in the graph whose weight is an exponential decay function of its length, and the Adamic/Adar Index [16], which counts the number of vertices connecting two nodes taking into account a weight depending on the reciprocal of the neighbor's degree.

The central idea behind graph embedding lies in learning a mapping that embeds nodes, edges, subgraphs, or even entire graphs, in a low-dimensional vector space, where the embedding dimension is expected to be much lower than the total number of nodes in the network. More specifically, given a graph $G = (V, E)$, and a predefined embedding dimension d, such that $d \ll |V|$, the problem of graph embedding is to map G into a d-dimensional space, in which graph properties are preserved as much as possible, i.e. topology and similarity measures [2, 3, 4]. Based on the output of the graph embedding, four categories may be defined: (i) node embedding, where vector embeddings are learned for each node; (ii) edge embedding, where edges are mapped into the embedding space; (iii) substructure embedding, in which subgraphs (i.e. clusters, communities, graphlets, ...) are represented in the vector space; and (iv) whole-graph embedding, i.e. an entire graph is mapped into a single vector. [3] (see Fig. 1).

(b) Node Embedding

(c) Edge Embedding

(d) Substructure Embedding
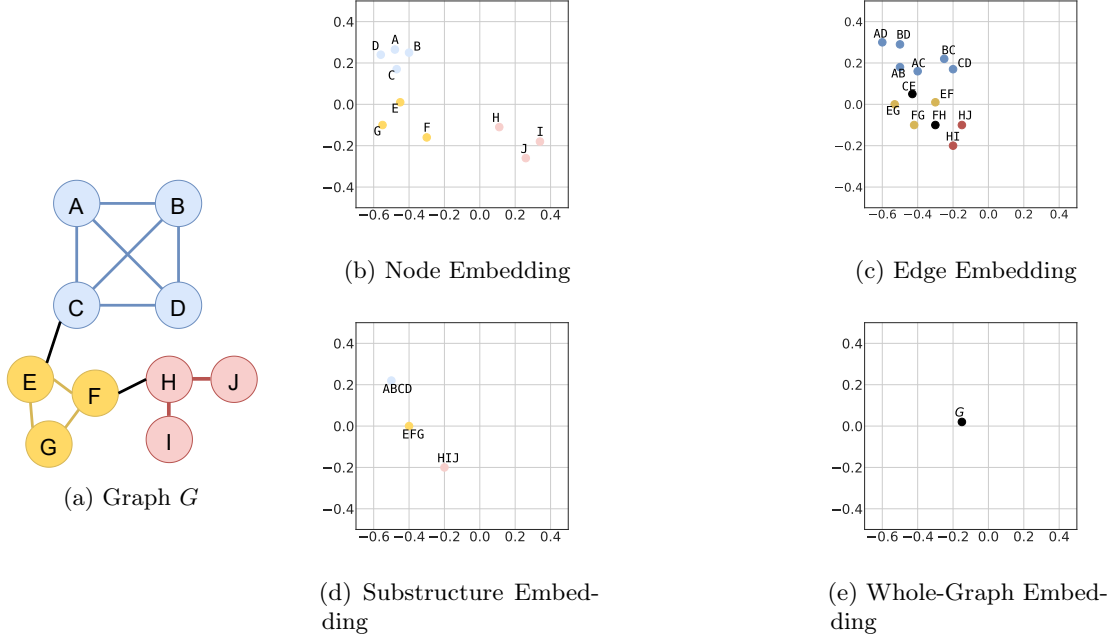
(e) Whole-Graph Embedding

(a) Graph G

Figure 1: A toy example of embedding a graph into 2D space taking into account different granularities. (a) Sample network used as a reference for graph embedding, where the different node colors depict different substructures, and different edge colors depict intra-substructure and inter-substructure connections. (b-e) Different static graph embedding outputs, as described by Cai et al. [3], where $d = 2$. Note that the colors refer to the substructures and connections displayed in (a).

Static graphs embedding taxonomies have been proposed in the past few years [4, 3]. Graph embedding based on **matrix factorization** represents some graph similarity in the form of a matrix and factorizes this matrix to obtain a node embedding. The problem of graph embedding is treated as a structure-preserving dimensionality reduction problem, which assumes the input data lie in a low dimensional manifold. Approaches based on **deep learning** apply deep neural architectures on graphs, including autoencoders (AEs), convolutional neural networks (CNNs), and variational autoencoders (VAEs). **Random walk approaches** generate node sequences from a graph to create contexts for each node, then applying techniques from natural language processing for learning embeddings. They try to preserve higher-order proximity between nodes by maximizing the probability of occurrence of subsequent nodes in fixed-length random walks, using neural language models, such as SkipGram. Cai et al. [3] further suggest other paradigms, such as **edge reconstruction based optimization**, which learns representations that directly optimize either edge reconstruction probability or edge reconstruction loss; **graph kernel-based methods**, which decompose the graph into atomic substructures (as graphlets and subtrees) and build a vector using these features; and **generative models**, which specify the joint distribution of the input features and the class labels conditioned on a set of parameters.

Several further discussions about static graph embeddings were presented in other surveys and reviews [2, 17, 18], along with some works concerning knowledge graph embedding, specifying their analysis to tasks including knowledge graph completion and relation extraction [19, 20]. Moreover, Goyal developed GEM [4], an open-source Python library that provides a framework for graph embedding implementation, and Grattarola and Alippi have presented Spektral [21], another open-source Python library for building graph neural networks with Keras API and TensorFlow 2, handling tasks such as node classification, link prediction, and graph generation.

2.2 Dynamic Graphs

There are different mathematical formulations to describe interactions between nodes over the lifetime of a system [6, 22, 23]. A possible definition is to describe a dynamic graph by a mathematical structure $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, where $\mathcal{V} = \{V(t)\}_{t \in \mathcal{T}}$ is a collection of node sets over time, $\mathcal{E} = \{E(t)\}_{t \in \mathcal{T}}$ is a collection of edge sets over time, and $\mathcal{T}$ is the time span. For each $t \in \mathcal{T}$, it is possible to define a graph snapshot $G(t) = (V(t), E(t))$, i.e. a static graph representing a fixed timestamp t of the dynamic graph. Adjacency matrix $A(t)$, weight matrix $W(t)$ and similarity matrix $S(t)$ are now time-dependent, and can be calculated for each snapshot $G(t)$, as well as node types $\mathcal{L}^n(t)$ and edge types $\mathcal{L}^e(t)$.

Casteigts et al. [6] alternatively defined a dynamic graph as $\mathcal{G} = (V, E, \mathcal{T}, \rho_v, \rho_e)$, where V is a node set containing every node that is present in the network at any given time $t \in \mathcal{T}$, E is an edge set defined similarly, and further defining a node presence function $\rho_v : V \times \mathcal{T} \to \{0, 1\}$, indicating whether a given node $v \in V$ is available at a given time $t \in \mathcal{T}$, and an edge presence function $\rho_e : E \times \mathcal{T} \to \{0, 1\}$, specifying if a given edge $e \in E$ exists at a timestamp $t \in \mathcal{T}$.

Figure 2 depicts a dynamic network as a time-varying graph, containing 9 nodes along lifetime $\mathcal{T} = [0, 7)$. It is noteworthy that, at the beginning of the network lifetime, only nodes A, B, C, and D are present, as well as links (A,B), (A,C), (B,C), and (B,D). As time passes, new nodes and edges arrive, even as nodes and edges are removed from the system. In the end, we have nodes B, E, F, G, H, and I, and links (B,E), (B,F), (E,G), and (H,I).

The formulations we described above are sufficient for understanding the dynamic graph embedding methods we review in this paper. It is important to mention that dynamic graphs can present even more complex temporal patterns, such as latency [6] (i.e. nodes/edges not arising instantaneously in the network, instead of taking a finite time interval to be established) and spatial-temporal edges [22] (i.e. a node at a given timestamp connected to another node at another timestamp). In Sec. 5, we propose future directions for embedding dynamic graphs concerning these properties.
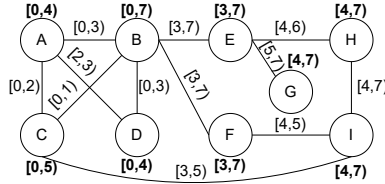

Figure 2: A representation of a small dynamic network, showing edge presence (continuous intervals above edges) and node presence (bold continuous intervals next to nodes) intervals.

2.3 Dynamic Graph Modeling

One of the first aspects to be considered when modeling a dynamic network is to define its life span $\mathcal{T}$. Two different approaches may be adopted to model the system's time domain: **discrete-time approaches**, where $\mathcal{T}$ is a discrete set, hence the evolution of a dynamic graph can be described by a sequence of static graphs, with a fixed timestamp; and **continuous-time approach**, where $\mathcal{T}$ is a continuous set, therefore the evolution is modeled at a finer temporal granularity to encompass different events in real time [8]. Computationally, dynamic graph models assuming discrete-time domain are easier to manipulate. Most of the existing embedding methods are based on this approach [10]. However, some authors have proposed to model more sophisticated phenomena, such as stochastic events, to leverage applications such as event time prediction (i.e. predict when an edge or a node is created or removed from a network) [8, 24]. Therefore, these approaches must rely on continuous-time lifespan to capture temporal evolution at an appropriate granularity. We briefly discuss some dynamic graph models covered

by embedding methods presented in this survey.

2.3.1 Graph Snapshots

This model represents a dynamic graph as a list of static graphs, i.e. $\mathcal{G} = \{G(t_0), ..., G(t_{N_S-1})\}$, where $G(t_k) = (V(t_k), E(t_k))$ is a static graph with timestamp t_k ($k \in \{0, ..., N_S - 1\}$), N_S is the number of snapshots, $V(t_k)$ is the node set at timestamp t_k and $E(t_k)$ is the edge set including all edges within the period $[t_k, t_{k+1})$ [25]. Most of the methods for embedding dynamic graphs manage this model as the input, either by adopting directly a sequence of successive state sub-graphs that represent the network in a discrete way as time passes [26, 10, 27], or splitting the time domain into non-overlapping windows of fixed duration, establishing a static graph for each window [28, 29].

2.3.2 Difference Network Models

In many real problems, the number of edges inserted or removed at any given time is much smaller than the total number of edges. i.e. the topological evolution is sparse [30, 31]. These models representing these network changes take as input an initial graph G_{t_0}, and a list of adjacency matrix changes $\Delta\mathcal{A} = \{\Delta A(t_1), ..., \Delta A(t_{N_R-1})\}$, where $\Delta A(t_k) = A(t_k) - A(t_{k-1})$, and N_R is the total number of recorded timestamps. This definition may be extended for other similarity matrices [32, 7], and difference networks may be divided into a link formation network (concerning positive values of adjacency matrix change) and a link dissolution network (regarding negative values of adjacency matrix change) [33, 29]. Note that these models do not handle nodes being added or removed from the network, as they rely on matrices with a fixed dimensionality.

2.3.3 Continuous-Time Network Models

Continuous-time approaches may include timestamped edges (edges with the information about the time they were created, or the time intervals concerning their existence in the network) [34] and link streams (a list of node interactions over time) [23]. Events in the network, such as the creation and removal of nodes and edges, may occur in any time $t \in \mathcal{T}$, and maybe instantaneous (i.e. much faster than the typical temporal granularity of the system) or may be assigned with a latency [6]. Several dynamic graph embedding methods rely on continuous-time networks, either by modeling timestamped edges as stochastic point processes [24] or leveraging link streams [35]. Furthermore, node arrival and removal may be included by these network models, as proposed in the literature by stream graphs [23] and appeared in some embedding techniques [8, 36].

2.4 Temporal Point Processes on Graphs

As discussed above, nodes and edges network churn may be modeled by stochastic events in a continuous-time domain, normally as stochastic point processes, random processes whose realization is comprised of discrete events in a continuous time. A **temporal point process** is a point process that can be represented as a counting process $N(t)$, recording the number of events up to time t, thus being useful for modeling sequential asynchronous discrete events occurring in continuous time [12]. The **conditional intensity function** $\lambda(t)$ characterizes a temporal point process such that $\lambda(t)\,\Delta t$ is the conditional probability of observing an event in the tiny window $[t, t + \Delta t)$ given the network history, i.e., all events before t, and only one event can happen in this tiny interval Δt. Similarly, a **survival function** $S(t)$ determines the conditional probability that no event happens during a time window $[t, t + \Delta t)$ given the network history, and the **conditional density** $f(t) = \lambda(t)S(t)$ for an event that occurs at time t is further defined as well.

The functional form of the intensity $\lambda(t)$ is often designed to capture the phenomena of interests, some of them include Poisson Process, Hawkes Process, Self-Correcting Process, Power Law, or Rayleigh Process [37]. Many dynamic graph embedding techniques consider that interactions between nodes are stochastic processes whose probabilities depend on the topological structure of the network, and node features (if applicable) at each timestamp [8, 38, 39, 40, 41, 42].

2.5 Dynamic Graph Embedding Input

In addition to the dynamic network modeling, discussed in Sec. 2.3, dynamic graphs can be (i) homogeneous, in which only topological information over time is available, (ii) heterogeneous, in which either nodes, edges (topological heterogeneity) or timestamps (temporal heterogeneity) are assigned with labels, (iii) attributed (or with additional information), where nodes and edges may hold several different features, and (iv) constructed from non-relational data (see Fig. 3). This proposed taxonomy extends Cai et al. [3], who encompassed static graph embedding input considering static networks, without leveraging dynamic aspects neither handling the difference between topological and temporal heterogeneity. In the following, we discuss each dynamic graph embedding input shown in Figure 3.
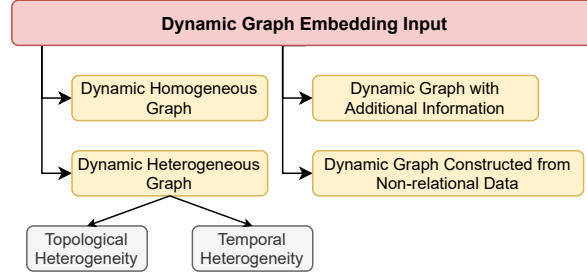

Figure 3: The proposed taxonomy for dynamic graph embedding input, an extension of Cai et al. [3] to encompass dynamic networks and to consider topological heterogeneity (similar to static networks) and temporal heterogeneity (i.e. timestamps having labels).

2.5.1 Dynamic Homogeneous Graph

Undirected and unweighted homogeneous graphs are widely used as dynamic graph embedding inputs due to their simplicity and to handle only basic structural information over time [43]. Several embedding methods, however, are proposed to handle weighted [44, 45] and directed dynamic graphs [10].

2.5.2 Dynamic Heterogeneous Graph

Embedding methods to handle topological heterogeneity, i.e. nodes or edges having labels, are usually concerned with node and edge classification [46, 47]. Nevertheless, graph snapshots may have labels, characterizing different global behavior [48, 49] and describing another type of heterogeneity, which we have named **temporal heterogeneity**.

2.5.3 Dynamic Graph with Additional Information

Additional attributes may be assigned to nodes, such as a set of numerical or categorical features. It is possible to define a time-dependent node feature matrix $F(t) \in \mathbb{R}^{N \times f}$, where f is the number of additional node features, and learn representations leveraging these features in addition to their topological structure [7, 50]. Although an edge feature matrix would be defined as well, its usage is much less common.

2.5.4 Dynamic Graph Constructed from Non-Relational Data

Non-relational time series data can be transformed into a dynamic graph by defining a similarity measure between two data instances, and constructing a similarity matrix $S(t)$ afterward. Several papers use this step as an intermediate to learn vector representations from this constructed graph to support some task-driven application, such as traffic forecasting [51], predicting bike-sharing demand [52], predicting social events [53] and missing label classification on videos [54].

2.6 Problem Formulation and Output for Dynamic Graph Embedding

It is important to mathematically formulate dynamic graph embedding to understand its outputs. Given a dynamic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, where $\mathcal{V} = \{V(t)\}_{t \in \mathcal{T}}$ and $\mathcal{E} = \{E(t)\}_{t \in \mathcal{T}}$, and an embedding dimension d, the problem of embedding dynamic graph is regarded as learning how to map $\mathcal{G}$ into a d-dimensional vector space over time, in which both topological information and temporal dependencies of the network are captured, either by learning representations able to reconstruct the dynamic graph $\mathcal{G}$, to predict the behavior of the network at timestamps outside the lifespan $\mathcal{T}$, or to directly handle a task-driven application such as node classification. When the graph topology evolves, two possible interpretations are possible for the evolution of embeddings: (i) the vector representations move along the embedding space, making it possible to trace the trajectory of each node; or (ii) the embedding space itself evolves in time, thus being possible to learn mappings between embedding spaces in consecutive timestamps [43].

The time-domain of vector representations and the network does not need to be identical, i.e., $\mathbb{T} \neq \mathcal{T}$. For instance, a dynamic graph may have daily information about interactions between users in a social network, but the network analytics and inference are more interested in capturing weekly or even monthly features. Hence, even though the network life span is given by daily timestamps, vector representations are extracted for a coarser temporal granularity.

Therefore, to define different dynamic graph embedding outputs, it is important to separate between (i) **topological embedding**, which is similar to the definitions for static graph embedding [3] and concerns node embedding, edge embedding, substructure embedding, and graph snapshot embedding, all of them over time, and (ii) **temporal embedding**, regarding the relation between network temporal granularity given by $\mathcal{T}$ and embedding temporal granularity given by $\mathbb{T}$. The complete classification we propose is shown in Figure 4 and is further discussed in the following topics.
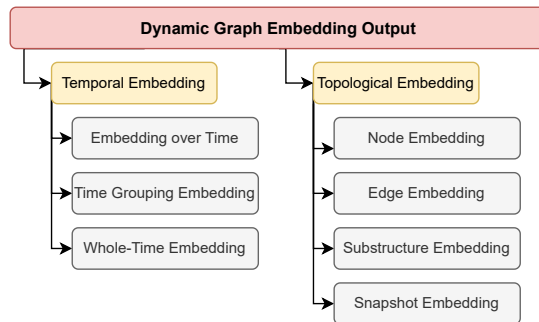

Figure 4: Dynamic graph embedding output taxonomy proposed in this survey.

2.6.1 Temporal Embedding

Temporal embedding concerns the relationship between the **input temporal domain** $\mathcal{T}$ and the **output temporal domain** $\mathbb{T}$. Defining $T(\mathcal{G})$ as a set compressing a topological property of the dynamic

graph (i.e. node set, edge set, induced subgraph set or entire graph, as described later in Section 2.6.2), one may distinguish between three different classifications for temporal embedding, as shown in Figure 4 and further illustrated in Figure 5.

Definition 1. An **embedding over time** is a mapping $\mu_t : T(\mathcal{G}) \times \mathcal{T} \to \mathbb{R}^d \times \mathcal{T}$ (i.e. $\mathbb{T} = \mathcal{T}$). Therefore, each node/edge/substructure/graph at time $t \in \mathcal{T}$ is represented as a vector in a low dimensional space over t.

In this type of temporal embedding, the mapping μ_t is a bijection in time, allowing to distinguish, in the vector space, each time instant for each mapped entity of the network (Figure 5b shows an example).

Definition 2. A **time-grouping embedding** is a mapping $\mu_g : T(\mathcal{G}) \times \mathcal{T} \to \mathbb{R}^d \times \mathbb{T}$, where $\mathbb{T}$ is a discrete set composed of elements that aggregate timestamps or time intervals of $\mathcal{T}$. Therefore, instead of representing each node/edge/substructure/graph at every time $t \in \mathcal{T}$, they are represented at every time aggregate $t' \in \mathbb{T}$.

Time aggregates are useful for representing networks whose desired time information is of a different granularity than that available in the data. As discussed, vector representations may be required each week from data obtained daily. Figure 5c contains another example, where two timestamps have been aggregated into a single one.

Definition 3. A **whole-time embedding** is a mapping $\mu_w : T(\mathcal{G}) \times \mathcal{T} \to \mathbb{R}^d$, i.e. $\mathbb{T}$ is a unitary set indicating that every timestamp is aggregated into a single point. Therefore, each node/edge/substructure/graph at every time $t \in \mathcal{T}$ is represented as a single vector in $\mathbb{R}^d$.

Temporal aggregation can be performed: (i) as a step before embedding, aggregating temporal data before applying an embedding method; or (ii) after representation learning over the dynamic graph, usually by performing operations in the vector space (e.g. weighted averages or non-linear functions) to obtain lower temporal granularity representations (i.e. embedding daily network data to extract weekly or even monthly network representations).

2.6.2 Topological Embedding

Embedding topological properties of a dynamic graph is similar to Fig. 1, and the main difference is the coupling between the temporal embedding discussed above and each graph structure, as presented in the right branch of Fig. 4. WLoG, we are considering $\mathcal{V} = V \times \mathcal{T}$ and $\mathcal{E} = E \times \mathcal{T}$ to simplify notations.

Definition 4. A **dynamic node embedding** is a mapping $\nu_n : V \times \mathcal{T} \to \mathbb{R}^d \times \mathbb{T}$. Therefore, each node at time $t \in \mathbb{T}$ is represented as a vector in a low dimensional space.

Node embedding over time is useful for several applications such as time-dependent node classification, network clustering evolution, and link prediction. It also allows one to track node trajectories in the embedding space and extract information about node behavior and roles in the network. [55, 28].

Definition 5. A **dynamic edge embedding** is a mapping $\nu_e : E \times \mathcal{T} \to \mathbb{R}^d \times \mathbb{T}$. Therefore, each edge at time $t \in \mathbb{T}$ is represented as a vector in a low dimensional space.

In addition to edge embedding, a few methods map both edges and nodes, in particular for embedding dynamic knowledge graphs or user-item interaction graphs [41, 56].

Definition 6. A **dynamic substructure embedding** is a mapping $\nu_h : S(\mathcal{G}) \times \mathcal{T} \to \mathbb{R}^d \times \mathbb{T}$, where $S(\mathcal{G})$ is a set of induced subgraphs in G at each time $t \in \mathcal{T}$. Therefore, each substructure at time $t \in \mathcal{T}$ is represented as a vector in a low dimensional space.

(a) Dynamic graph $\mathcal{G}$

(b) Embedding dynamic graph (c) Time-grouping embedding (d) Whole-time embedding
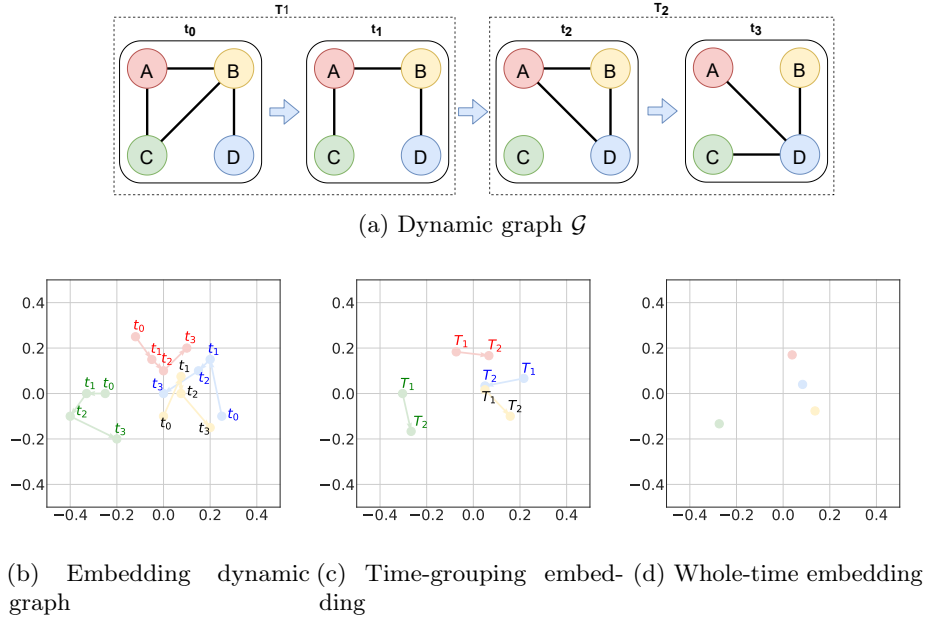
Figure 5: A toy example of embedding each node of a dynamic graph into 2D space taking into account different temporal granularities. (a) Sample network used as a reference for dynamic node embedding, where the snapshot model is considered, different node colors are used to distinguish each of them, and the four timestamps t_0 to t_3 are grouped into two time aggregates T_1 (holding t_0 and t_1) and T_2 (which holds t_2 and t_3). (b) In embedding over time, each node is mapped into the low-dimensional space for each timestamp, thus describing trajectories in embedding space. These trajectories are illustrated by arrows, indicating the time flow. In this case, $\mathcal{T} = \mathbb{T}_a = \{t_0, t_1, t_2, t_3\}$. (c) Although time granularity of the network is described by $\mathcal{T}$, the graph is mapped into a different time granularity $\mathbb{T}_b = \{T_1, T_2\}$, where each new timestamp aggregates temporal information about the original timestamps. The representations were defined from embedding over time handcrafted design, and (d) In the whole-time aggregation embedding, $\mathbb{T}_c$ is a unitary set, and every timestamp is aggregated into a single representation for each node.

Several dynamic graph embedding methods generalize traditional node or link prediction tasks to consider joint prediction over larger k-node induced subgraphs [57] and graphlets [58, 59].

Definition 7. A **snapshot embedding** is a mapping $\nu_{\mathcal{G}} : \mathcal{G} \times \mathcal{T} \to \mathbb{R}^d \times \mathbb{T}$, where $\mathcal{G} = \{G_t\}_{t \in \mathcal{T}}$. Hence, each graph snapshot at time $t \in \mathcal{T}$ is represented as a vector in low dimensional space.

Snapshot embeddings are useful for tracking network behavior over time, when the topological structure of the network is related to some emerging property or global interpretation of node interactions [49, 48].

2.7   Dynamic Behaviors

Embedding methods can also be identified according to the type of time dynamics they capture. Most methods can capture the evolution of connectivity between network nodes, i.e. addition and removal of edges. However, several works generalize the method to include adding and removing nodes in the network. In addition, there are methods capable of capturing other temporal properties of networks, including varying edge weights, changing node and edge classification, evolving node, and edge attributes, and dynamic processes in the network (such as diffusion cascades).

We mapped three groups of dynamical behaviors on networks: **topological evolution** (nodes and edges varying over time), **feature evolution** (node and edge features changing over time), and **processes on networks** (a time-dependent process taking place on the network), as shown in Figure 6.
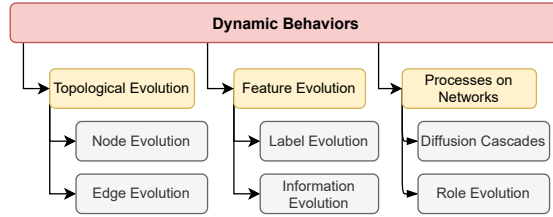

Figure 6: Several dynamical properties that may be captured by embedding methods, divided by topological evolution (concerning changes in the node and edge set), feature evolution (related to node or edge label or other additional information changes over time) and processes on the network (regarding diffusion and role evolution of nodes in a network).

2.7.1   Topological Evolution

Topological evolution means that nodes and edges may vary over time, being added/removed from the network. **Node evolution** is characterized by changes on the node set V, whereas the **Edge evolution** concerns connections between nodes that may be continuously formed or broken. Several dynamic graph embedding methods require that the number of nodes in a network does not change over time, since they handle adjacency or similarity matrices with fixed dimensionality [43]. Moreover, approaches considering edge creation may be captured concerning a trend through historical records of interaction between a pair of nodes (i.e. if two nodes have recently made multiple connections, they are more likely to make future connections), and ternary closure (if a third node is a common neighbor of two unconnected nodes, there is a greater tendency to close the triangle and form a clique) [57].

2.7.2   Feature Evolution

In many problems involving heterogeneous graphs, it is assumed nodes have fixed labels. However, it is also possible to observe a **label evolution**: In a citation network, an author may have as its main research area a different topic compared to previous years. Such changes are linked in some way to the topological evolution of the network. In node classification tasks, each node in a graph has a class

label, hence it is possible to predict the class label for the nodes in a graph $G(t_k)$ using previous graphs $G(t_0), ..., G(t_{k-1})$ [31].

Even more, in several real-world networks, nodes and edges may have rich attributes (i.e. additional information) that are changing over time in addition to the network structure and the topology may influence attribute modification. Therefore, embedding methods may also capture **information evolution** [50, 56]. Furthermore, edge weights may also change over time in weighted networks, and their changes may be handled by embedding methods [60].

2.7.3 Processes on Networks

It is possible to analyze dynamic processes on the network, such as information diffusion or disease spreading, and other emergent properties, such as node roles changing over time. Two of the most basic and widely-studied diffusion models include [61]: (i) linear threshold model, where each node v_i is influenced by each neighbor $v_j \in \mathcal{N}_{v_i}$ according to a sum of weights b_{ij}, and if this sum is higher than a random choice of a threshold θ_{v_i} at time t_k, the node v_j is activated at time t_{k+1}; and (ii) independent cascade model, where an active node v_i influences its neighborhood $\mathcal{N}_{v_i}$ with a probability parameter p_{ij} (for each node $v_j \in \mathcal{N}_{v_i}$) independently of the network history. In this context, the problem of modeling diffusion by independent cascades comes down to learning probability distributions characterizing the hidden influence between users, to discover the main communication channels of the network. A group of nodes sharing similar roles in the network can be regarded as a set of nodes that are more structurally similar to nodes inside the set than outside, whereas communities are sets of nodes with more connections inside the set than outside. Hence the dynamic role evolution aims to automatically discover groups of nodes (representing common patterns of behavior) based on their latent features given by its representations [55].

2.8 Stability and Temporal Smoothness

A successful dynamic graph embedding algorithm should create stable embeddings over time. In other words, an embedding should be able to learn similar representations at consecutive timestamps if the underlying graph changes only slightly. More specifically, given the dynamic graph $\mathcal{G}$, if the graph snapshot $G(t_{k+1})$ is similar to $G(t_k)$ (for instance, adjacency matrices $A(t_{k+1})$ and $A(t_k)$), the embedding matrix $Z(t_{k+1})$ is expected to be similar to $Z(t_k)$. Goyal et al. [10] propose the stability constant $K_A(\nu)$ as a metric to evaluate the stability of a dynamic graph embedding function ν in terms of the adjacency matrix A over time. More specifically, the authors consider the stability of any embedding at a given timestamp as the ratio of the Frobenius norm of the difference between embedding matrices and the Frobenius norm of the difference between adjacency matrices, at consecutive timestamps. Then, the stability constant is the maximum difference between stabilities calculated along the entire life span $\mathcal{T}$ of the network, and the authors claim that a dynamic embedding ν is stable as long as the stability constant is small. It is possible to further extend these proposals to include any similarity matrix S, and to take the limit of two consecutive timestamps $t_{k+1} - t_k = \Delta t \to 0$ in order to analyze continuity aspects of embeddings (see future directions in Sec. 5).

Although the above discussion is related to the global behavior of embedding, most of the proposed embedding methods seek local stability, i.e., for each node in the network. If the local topological structure around a node v has undergone few changes, it is expected that the representations $z_v(t_{k+1})$ and $z_v(t_k)$ are similar, assuming a **temporal smoothness** in the embedding space [31, 62, 63].

3 Techniques for the Embedding of Dynamic Graphs

In this section, we propose a taxonomy of dynamic graph embedding techniques, summarized in Figure 7. This taxonomy has been inspired by previous static graph embedding classifications [4, 3]. Classifications for dynamic graphs can be seen as extensions of static graph methods: (i) **matrix factorization approaches**; (ii) **deep learning approaches**; (iii) **random walk-based methods** (which may be understood as node sequence sampling methods); (iv) **optimization based on edge reconstruction**, which also leverages **temporal smoothness**, and (v) **graph kernel methods**. Here we include **tensor factorization approaches**, fusing them with matrix factorization approaches and defining general **factorization based approaches**, and introduce two novel paradigms: (i) **temporal point process based** methods, which handles similarity matrix changes as stochastic processes; and (ii) **agnostic models**, which learn embeddings over time independent of the approach used for each graph snapshot in dynamic networks.

Earlier approaches to map dynamic networks into vector spaces proposed learning independent vector representations of each snapshot employing static graph embedding methods. However, since representation learning assumes that the probability mass of the data concentrates in manifolds that have much smaller dimensionality than the original space where the data lives, the evolution of the network may cause two relevant impacts on the embedding space: (i) the embedding vectors move on the manifold; and (ii) the manifold itself evolves in time [43]. Therefore, integrating spatial topology of network and the temporal network evolution into feature vectors encompassing these temporal correlations enhances the performance of prediction, classification and many other temporal network analysis problems [64].
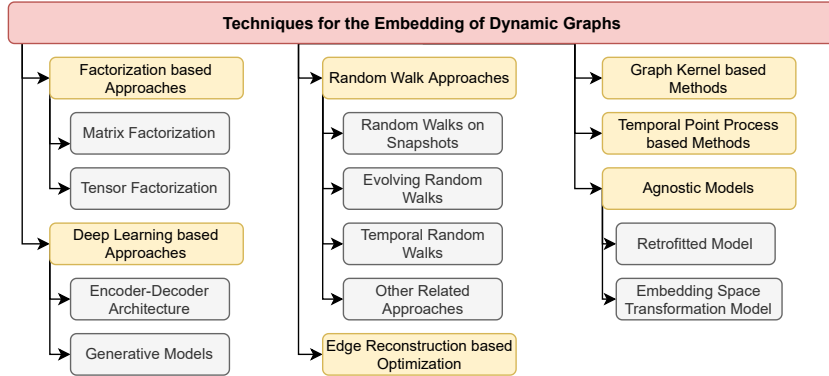

Figure 7: Proposed taxonomy for dynamic graph embedding techniques, organized by algorithmic approaches.

Next, we present each classification proposed by our taxonomy, discussing its insights, how it leverages temporal dependence in addition to topological structure, and describing several methods following each approach.Concrete examples of applications using the methods covered here are presented only in the following section.

3.1 Factorization-based methods

Factorization-based approaches generate node embeddings over time by finding low-rank decompositions of time-dependent similarity measures. These similarity measures can be represented by: (i) sequence of matrices over time; or (ii) three-way tensors, where the first two dimensions are related to the similarity between nodes, and the third dimension is the temporal slice. The matrix representation relies on capturing the temporal evolution between representations in adjacent timestamps, and the embedding learns how the network changes over time, separating the topological contribution from the temporal

contribution. The tensor representation, on the other hand, couples topology and temporal evolution in a structure to be factorized in a unified way. Consequently, it is valid to separate the factorization approaches into **matrix factorization approaches** and **tensor factorization approaches**.

3.1.1 Matrix factorization approaches

Matrix factorization approaches express the evolutionary structure of networks in the form of matrices, therefore leveraging the time-dependent structural correlation among pairs of nodes. Dynamic graph embedding based on matrix factorization may be classified, as in static graphs, according to which type of loss function the approach minimizes. While most approaches factorize the similarity matrix and define an inner-product function between node embeddings to approximate the proximity measure [55, 60, 28, 32, 9], graph Laplacian Eigenmaps can be used to reconstruct time-dependent adjacency matrix from an eigendecomposition of Laplacian matrix [7]. The novelty in the embedding of dynamic networks consists in the way of propagating the factorization over time, maintaining the stability of the representations while inserting temporal dependence into matrix decomposition. It is possible to distinguish three paradigms based on the modeling for the connections between different timestamps: (i) adding temporal smoothing directly to the loss function, thus ensuring the stability of embeddings over time and focusing on node trajectories; (ii) updating the similarity matrix by using matrix perturbation theory, assuming that temporal evolution changes network topology slightly; and (iii) defining a temporal matrix factorization, decomposing the similarity matrix into a constant term and a time-dependent term. Figure 8 shows these two classifications.
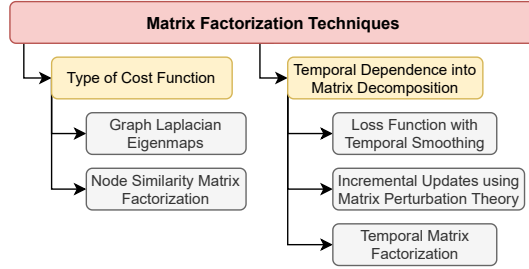

Figure 8: Classification of matrix factorization techniques for dynamic graph embedding, taking into account the type of cost function (similar to Cai et al. [3]) and how temporal dependence is leveraged into matrix decomposition.

- **Jointly optimizing loss function and temporal smoothing:** The methods based on the above insight perform the matrix factorization of each snapshot, in addition to bound the representations through a loss function term that is minimized when, for each node, the embeddings at two consecutive timestamps are similar. In other words, these approaches assume temporal smoothness in the embedding space, and the optimal embedding should jointly reconstruct the similarity matrix over time while being continuous. Given the loss function $\mathcal{L}_0(t)$ minimized by a matrix factorization approach on static graphs at each timestamp $t \in \mathcal{T}$, the jointly similarity reconstruction and temporal smoothing loss function for a dynamic graph can be defined as:

$$\mathcal{L} = \sum_{t \in \mathcal{T}} \mathcal{L}_0(t) + \tau \sum_{t \in \mathcal{T}} \sum_{t' \in \mathcal{T}|(t'-t) \leq \Delta t} \mathcal{L}_{\Delta t}(t', t), \tag{1}$$

where Δt is the time interval between consecutive timestamps in $\mathcal{T}$, $\mathcal{L}_{\Delta t}(t', t)$ is a loss function for each pair of consecutive timestamps (t and t') implying temporal smoothness. $\tau > 0$ regulates the temporal smoothness term contribution. Some approaches following this paradigm include Ferreira et al. [28] and Zhu et al. [9].

One can implement two generalizations of the loss function: (i) the contribution of each timestamp may not be homogeneous, since the future in general depends more directly on the recent past than on the distant past, weighing the loss function $\mathcal{L}_0$ with a function $f(t)$ that is close to 1 when $t \approx \tau$ (where τ is the last timestamp, i.e. t_{N_S-1} for snapshot model), and close to 0 if $t \ll \tau$ (as an example, $f(t) = e^{\tau-t}$); and (ii) the dataset may contain non-homogeneous time intervals, i.e., a function $g(\Delta t)$ may be defined in order to leverage different values of Δt, and the smoothness of embeddings at timestamps t and $t + \Delta t$ may be less important when Δt is large.

- **Incremental updates on embeddings:** In these methods, the initial graph snapshot G_0 gives the initial similarity matrix $S(0)$ using a standard matrix decomposition. Afterwards, the embeddings for subsequent timestamp are updated by assuming that $||S(\Delta t) - S(0)|| < \epsilon$ for some small ϵ, i.e. the similarity matrix $S(\Delta t)$ is a perturbation of the initial matrix $S(0)$. Therefore, it is possible to update a low dimensional representation of the nodes using first-order matrix perturbation theory in symmetric matrices iteratively [65]. Li et al. [7] apply matrix perturbation theory for Laplacian Eigenmaps, whereas Zhang et al. [32] propose TIMERS to update SVD decompositions incrementally.

  It is noteworthy that these approaches accumulate errors due to the perturbative approach made, and it may not be very effective if the network evolves intensely over time. Nevertheless, since for many real networks the temporal evolution is quite sparse (i.e. the number of added or removed edges is much lower than the total number of edges), these methods present promising results in dynamic link prediction, node clustering and node classification. Moreover, these strategies may reduce error accumulation over time, setting a restart time, i.e. a time interval at which the algorithm recalculates the factorization instead of the incremental update [32].

- **Temporal Matrix Factorization**: In these methods, the time-dependent similarity matrix S_t is decomposed by a temporal rank-k matrix factorization model as follows [60]:

$$S(t) = h(U \times V(t)^T), \tag{2}$$

where both U and $V(t)$ are $|V| \times k$ matrices, U is a constant matrix, $V(t)$ is a time-dependent matrix and $h(\cdot)$ is an element-wise function. For undirected networks, $S(t)$ is symmetric and it is possible to (i) average $UV(t)^T$ and its transpose as the prediction of $S(t)$; or (ii) factorize $S(t)$ as the product of a time-dependent matrix $V(t)$ and its transpose [60]. Therefore, this method learns two types of embedding: (i) a constant term embedding, given by the rows of U and represents persistent properties between pairs of nodes; and (ii) a time-varying embedding, given by the rows of $V(t)$ and represents changes in topology over time. A non-linearity may be inserted by the function h, e.g. a logistic function to interpret the reconstruction $U \times V(t)^T$ as a probability measure for the similarity.

The main challenge to handle this approach is to describe the time-dependent matrices $V(t)$ for each timestamp. The dynamic behavioral mixed-membership model (DBMM) proposed by Rossi et al. [55] was the first to employ this factorization, and proposed: (i) a transition matrix T in order to bound $V(t + \Delta t) \approx V(t)T$, (ii) a stacked transition model, which bounds training examples from l previous timestamps, and (iii) a summary transition model, defining $V(t)$ at a specific timestamp as a linear combination of time-dependent matrices at previous timestamps. Yu et al. [60], who developed the formal description of the temporal matrix factorization approach described above, represented $V(t)$ as a polynomial function over time of order p, i.e. $V(t) = \sum_{i=0}^{p} W^{(i)} t^i$, where $\{W^{(i)}\}_{i=0}^{p}$ are $|V| \times k$ matrices which need to be learnt from the model along with U. The LIST model [64] leverages a temporal matrix factorization to learn the feature vector of each node by simultaneously optimizing the temporal smoothness constraint and network propagation constraint, ensuring that two vertices

that are connected are likely to share similar features.

Table 1 compares the approaches based on matrix factorization discussed in this section. Some combinations of the matrix factorization approaches were not yet explored, such as graph Laplacian eigenmaps including a temporal smoothing term and a Laplacian temporal matrix factorization.

Table 1: Matrix factorization based Dynamic Graph Embedding.

		Matrix Factorization based Dynamic Graph Embedding						
		BCGD [9]	[28]	DANE [7]	TIMERS [32]	DBMM [55]	TMF [60]	LIST [64]
Cost Function Type	Laplacian Eigenmaps			✓				
	Similarity Matrix Factorization	✓	✓		✓	✓	✓	✓
Temporal Dependence	Temporal Smoothing	✓	✓					
	Incremental Updates			✓	✓			
	Temporal Matrix Factorization					✓	✓	✓

3.1.2 Tensor Factorization

Tensors are higher-order generalizations of vectors and matrices, represented as $X \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, where the order of X is $N > 2$ [66]. Dynamic networks are usually expressed as three-way tensors, i.e. $N = 3$. Several tensor factorization methods that can extract latent structure in the data have been proposed, including CANDECOMP/PARAFAC (CP) family, Tucker family, and alternative models [66].

For dynamic networks, CP decomposition is the most popular approach, since it learns both node embeddings and temporal embeddings with computational efficiency. Dunlavy et al. [5] used the temporal profiles computed by CP as a basis for predicting the scores in future timestamps, using a forecasting method for time-series data with periodic patterns. Rafailidis and Nanopoulos [67] represent continuous user-item interactions over time using a three-way tensor by proposing a measure of user-preference dynamics (UPD) that captures the rate at which the current preferences of each user have been shifted, and generate recommendations based on CP tensor factorization.

Even though models based on Tucker decomposition achieve good performance, they require a lot of computational power, which may explain the reason every tensor factorization approach for dynamic graphs relies on CP decomposition [68]. The alternative models discussed by Acar et al. [66], including Multilinear Engine (ME), STATIS, and multiblock multiway models are also yet to be explored by dynamic graph embedding methods.

3.2 Approaches based on Deep Learning

Deep Learning has shown a remarkable performance in a wide variety of research fields, including computer vision and language modeling, and it also benefits several applications related to representation learning and other tasks over graphs. Many successful static graph embedding methods have been proposed in the last few years, from Graph Neural Networks [69] and Convolutional Neural Networks on Graphs [70], to autoencoders, such as Structural Deep Network Embedding (SDNE) [71].

Different architectures based on neural networks are used both to extract the topological properties of a network and to capture temporal dependencies therein. Some preliminary works developed a fully-connected neural networks, either using them as an autoencoder or as a part of the decoding process [10, 27, 72, 50]. Architectures based on recurrent neural networks (RNNs), including long short-term memory units (LSTMs) [73] and gated recurrent units (GRUs) [74], leverage a sequence of graphs, or their representations, in order to learn or enhance embeddings taking into consideration temporal correlation, and store information over time to handle more complex correlations beyond consecutive timestamps [27, 75, 76]. Attention mechanisms [77] are used to further improve understanding of the most relevant time points for each representation [63, 78]. Convolutional neural networks (CNNs) [79] for graphs have been widely adopted in order to handle topological properties, including Graph Convolutional

Networks (GCNs) [80]. Iterative propagation procedures have been also employed to learn the graph topology, as in Graph Neural Networks (GNNs) [69], GraphSAGE [81], and Gated Graph Neural Networks (GGNNs), the former also being able to learn the reachability across the nodes in a graph using GRUs [76]. These approaches have been also explored by several dynamic graph embedding methods [72, 82]. Even further, instead of using RNNs to explore the characteristics of the network over time, many succeeding models also employ convolutional networks (for instance, 1D CNNs) to leverage temporal dependencies [83, 53, 84, 85].

Neural networks may also learn an approximation of the network distribution by using generative models, such as the variational autoencoders (VAEs) [86] and generative adversarial networks (GANs) [87]. These approaches learn low-dimensional latent representations of the training data that store information about the type of output the model needs to generate, using a generative network to capture the data distribution and a recognition network (or a discriminator network) to estimate the probability that a sample came from the data distribution. Variational graph autoencoder (VGAE) [88] and Graph-GAN [89] employ these approaches to static graphs. These generative models are used in dynamic graphs to learn these data distributions over time [83, 31, 90, 91, 92, 85, 44].

In order to define a taxonomy that categorizes every method with minimal overlap between different categories, we classify the approaches according to the general architecture of the neural network. We have identified two main general architectures: (i) encoder-decoder perspective, which holds the majority of works concerning learning representations and decoding them for an application (i.e. network reconstruction, link prediction, or node/graph classification); and (ii) encoder, sampling, and decoder perspective, which contains the generative models and handles representations as probability distributions (see Figure 9 for the complete proposed taxonomy for deep learning based approaches). In the following, we further detail each of these methods and also present the most important existing works for each of them.
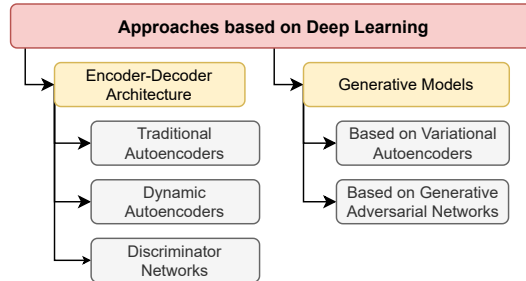

Figure 9: Classification of deep learning techniques for dynamic graph embedding, dividing into encoder-decoder perspective and generative models.

3.2.1 Encoder-Decoder Architecture

The encoder-decoder architecture consists of an encoder, which maps the data into a low-dimensional representation, and a decoder, that aims to either (i) reconstruct the original data; or (ii) solve an application-driven problem, such as a binary or multi-class classification problem. In the first case, the network is called an autoencoder, since the encoding seeks to be as lossless as possible. In the second case, the encoding is lossy and the original data cannot be fully recovered.

Concerning dynamic graphs, an encoder receives graph snapshots as input and the decoder may exhibit three distinct outputs: (i) traditional autoencoders, whose representations reconstruct each graph snapshot, hence following the lossless encoding paradigm; (ii) dynamic autoencoders, whose representations do not reconstruct each graph snapshot whereas instead they reconstruct a snapshot in a future timestamp, therefore predicting the network structure; and (iii) discriminator networks, whose embeddings do not

reconstruct the network topology at all, but are intended to learn node labels, node clusters or a global network property, and the loss functions are usually application-driven.

- **Traditional Autoencoders**: These architectures are applied for each graph snapshot, similar to the SDNE in static graphs [71]. DynGEM [10] builds a fully-connected autoencoder for each graph snapshot, using a transfer learning paradigm to share parameters between two consecutive autoencoders and a strategy that allows the autoencoder network to widen its layers and inserts new layers in order to handle a growing number of nodes in the graph. LDANE [50] follows a similar strategy, and handles node attributes by adding a margin-based ranking loss term in the loss function that ensures the embeddings of two similar nodes are closer than the embedding of two non-similar nodes.

  Models employing recurrent neural networks and their hidden states to encode dynamic graph structure have also been proposed. For instance, Taheri et al. [48] developed DyGGNN, which leverages Gated Graph Neural Networks (GGNNs) to capture graph topology and couples it with an LSTM encoder to handle graph dynamics, and with an LSTM decoder to reconstruct the structure of the dynamic graph at each timestamp. Other approaches include DySAT [63] and DGNN [93]. Table 2 summarizes the deep learning approach employed by each of these methods.

Table 2: Traditional Autoencoders for Dynamic Graph Embedding.

Algorithm	Deep Learning Model
DynGEM [10]	Fully-Connected Autoencoder (based on SDNE [71])
LDANE [50]	Similar to DynGEM, also handles node attributes (margin-based ranking loss term)
DyGGNN [48]	Gated Graph Neural Networks and LSTMs
DySat [63]	Structural and Temporal Self-Attention
DGNN [93]	Attentive LSTMs

- **Dynamic Autoencoders**: The input of these methods based on dynamic autoencoders is the historical records of the network. The output is the reconstructed graph at a future time. Bonner et al. [83] regarded this approach as the temporal graph offset reconstruction problem, i.e. creating temporal graph embeddings that recreate a future timestamp of the graph.

  Several embedding methods developed a recurrent architecture to capture the dynamics of the network in order to predict its future state, such as Goyal et al. [27], which propose three different strategies for taking a set of graph snapshots, from autoencoders (dyngraph2vecAE) to LSTM networks (dyngraph2vecRNN), and the combination of both of them (dyngraph2vecAERNN). Chen et al. [75] propose an encoder-LSTM-decoder (E-LSTM-D), which resembles dyngraph2vecAERNN, but uses rectified linear units (ReLUs) as the activation function for each encoder/decoder layer, and adds a regularization term to prevent overfitting. AdaNN [94] employs a triple attention module, leveraging topology, node attributes and temporal attention to further feed them into two connected GRUs and concatenating them into a joint state vector. TRRN [95] adopts memories to enhance temporal capacity, applying multi-head self-attention and learning contextualized representations feeding different factors (including node features and topological features) and updated memories into LSTMs.

  Graph convolution combined with recurrent units have been exploited for building dynamic autoencoders. GC-LSTM [72] uses convolutions to extract topological features while coupling with an LSTM in order to learn temporal features of the dynamic network. EvolveGCN [47] proposes two versions that follow a similar approach: (i) EvolveGCN-H, where the GCN parameters are hidden states of GRUs that take node embeddings as input; and (ii) EvolveGCN-O, where the GCN parameters are input/output of an LSTM unit. T-GCN [84] uses GCNs to learn topological structures, then passing these features to GRUs in order to extract temporal dependencies.

  Table 3 summarizes the deep learning approach employed by each of the methods based on dynamic autoencoders.

Table 3: Dynamic Autoencoders for Dynamic Graph Embedding.

Algorithm	Deep Learning Model
TO-GAE [83]	GCNs over time
dyngraph2vecAE [27]	Fully-Connected (FC) Autoencoders
dyngraph2vecRNN [27]	Sparsely Connected LSTMs
dyngraph2vecAERNN [27]	FC Encoder, LSTMs and FC Decoder
E-LSTM-D [75]	FC Encoder, LSTMs and FC Decoder
AdaNN [94]	Spatial, Attribute-Topology and Temporal Attention, and GRUs
TRRN [95]	FC Encoder, Transformer-Style Self-Attention and LSTMs
GC-LSTM [72]	Graph Convolutions and LSTMs
EvolveGCN-H [47]	GCNs and GRUs
EvolveGCN-O [47]	GCNs and LSTMs
T-GCN [84]	GCNs and GRUs

- **Discriminator Networks:** This approach considers that a neural network must learn application-driven representations, such as properly classifying nodes or graphs/subgraphs over time [57], extracting network properties [82], or predicting a specific global feature [51]. Several approaches combine GCNs and recurrent networks, such as DynGraph2Seq [82], TSGNet [96] and NAAM [97]. Topological features may also be extracted by other techniques as an alternative to GCNs, as Xu et al. showed by implementing STAR [78]. Other approaches use convolutions for both spatial and temporal feature extraction, including Spatio-Temporal Graph Convolutional Network (STGCN) [51].

These discriminator networks are commonly employed to embed graphs constructed from non-relational data. For instance, DynamicGCN [53] extracts and learns graph representations from historical event documents, encoding the input data into a sequence of graphs with node embeddings, and developing a graph convolutional network model to predict the occurrence of certain type of events. TD-Graph LSTM [54], in the other hand, is applied to action-driven video object detection, passing each frame through a spatial convolutional network in order to detect similar regions in consecutive frames, and construct a temporal graph structure by connecting semantically similar regions. LSTM units take the spatial visual features as the input states, incorporating temporal motion patterns for participating objects in the action while minimizing an action-driven object categorization loss. Li et al. [52] propose a spatial-temporal graph embedding model called STG2Vec, which includes a temporal attention and incorporates multi-source information to fed a collaborative temporal modeling based on LSTMs.

Table 4 lists the approaches described above and summarizes the deep learning approach employed by each of them, along with the specific task each method attempts to solve.

Table 4: Discriminator Networks for Dynamic Graph Embedding.

Algorithm	Deep Learning Model	Task
DynGraph2Seq [82]	GCNs, LSTMs and Hierarchical Attention	Sequence of Target Health Stages
TSGNet [96]	GCNs and LSTMs	Node Classification
NAAM [97]	GCNs, LSTMs (or BiLSTMs) and Temporal Attention	Forecasting User Interactions
STAR [78]	Spatio-Temporal Attention and GRUs	Node Classification
STGCN [51]	GCNs and Gated Convolutional Neural Networks	Traffic Forecasting
DynamicGCN [53]	GCNs with Updates from Previous Timestamps	Event Prediction
TD-Graph LSTM [54]	CNNs and LSTMs	Missing Label Classification
STG2Vec [52]	Temporal Attention and LSTMs	Bike-sharing Demand

3.2.2 Generative Models

Generative algorithms attempt to predict features given a certain label, i.e. to learn the data distribution patterns, in contrast to discriminative models, which attempt to learn representations and classify each input data. Therefore, these approaches have the power to synthesize data in addition to compress and to learn embeddings. Regarding generative neural networks for dynamic graph embedding, two groups of approaches arise: (i) methods based on variational autoencoders (VAEs), which encodes an input as a distribution over the latent space; and (ii) methods based on generative adversarial

networks (GANs), which train both a generator network to synthesize graphs and a discriminator network to distinguish between true graphs and generated ones.

- **Based on Variational Autoencoders:** The encoder for a variational autoencoder takes a data point and produces a distribution, usually parameterized as a multivariate Gaussian. In this case, the encoder predicts the mean and standard deviation of the Gaussian distribution, and the lower-dimensional embedding is sampled from this distribution. The decoder is a variational approximation, which takes an embedding and produces an output [86].

  Every variational autoencoder for dynamic graphs is inspired by VGAE [88] to address the encoding of each snapshot, differing on how to handle the graph evolution. These methods include (i) Dyn-VGAE [31], which addresses a temporal smoothness loss term, (ii) TO-GVAE [83], which applies VGAE over time to reconstruct subsequent snapshots, and (iii) VGRNN [92], which adopts VGAE whose prior distribution parameters are based on the hidden states in previous timestamps. There are several proposals to enhance the encoder model. For instance, Bonner et al. [90] propose a Temporal Neighbourhood Aggregation (TNA) block to comprise a GCN with a GRU in the encoder, controlling the combination of topological and temporal learning via a final linear layer. Zhao et al. [91] have developed a framework called BurstGraph, which splits the adjacency matrix of a graph into a standard adjacency matrix and a burst adjacency matrix to pick up unexpected behavior within a time duration. A variational autoencoder is employed using GraphSAGE [81] as the encoder, and two decoders: a standard decoder, which learns representations Z^v, and a bursty decoder, which learns sparse embeddings Z^b.

- **Based on Generative Adversarial Networks:** Generative adversarial networks (GANs) are algorithmic architectures that use two neural networks, pitting one against the other by designing a game-theoretical minimax game to combine generative and discriminative models. This approach has been applied for graph representation learning by a framework called GraphGAN [89]. While a generator network attemps to approximate the true graph connectivity distribution, the discriminator network aims to discriminate the connectivity of each node pair. Therefore, the generator network tries to deceive the discriminator network, whereas the discriminator network improves itself to distinguish better and better between true edges and generated edges.

  Inspired by these recent approach for graph representation learning, some methods arise for dynamic graph embedding. DynGraphGAN [85] designs the discriminator network with the following components: (i) a GCN to encode neighborhood features of nodes; and (ii) CNNs to learn temporal graph evolution along the time dimension. The generator network implements a sigmoid function of the inner product of two node's embeddings at a timestamp t to estimate the probability distribution of an edge connecting these nodes at time t. GCN-GAN [44] displays an architecture whose generative network consists of a GCN layer, an LSTM layer and a fully-connected output layer, and the discriminator network is a fully-connected feedforward neural network.

  Each of the GAN-based methods previously mentioned are trained with a single graph. Therefore, the trained model is capable of generating artificial snapshots following a similar structure and dynamic of the original graph used during training. Also, note that the resulting model is limited to creating snapshots with the same number of nodes as the graph used in training, given that number of parameters of the model is proportional to the number of nodes in the TVG. This dependency is observed, for example, for the last layer of the generative model of the GCN-GAN model, which builds an adjacency matrix with dimensions $N \times N$ from an embedding vector (outputted by the LSTM layer).

Table 5 summarizes the generative model and architectures employed by the methods discussed in this section.

Table 5: Generative Models for Dynamic Graph Embedding.

Algorithm	VAE	GAN	Deep Learning Models
TO-GVAE [83]	✓		GCNs
Dyn-VGAE [31]	✓		Original VAE with Temporal Smoothness
TNA [90]	✓		GCNs and GRUs
[91]	✓		GraphSAGE [81] and RNNs
VGRNN [92]	✓		GCNs and LSTMs
DynGraphGAN [85]		✓	GCNs and CNNs
GCN-GAN [44]		✓	GCNs and LSTMs

3.3 Random Walk Approaches

Another class of methods for graph embedding functions relies on random walks. Multiple random walks of fixed length L are considered sentences, generating a context for each node and trying to extract higher-order dependencies without adjacency matrices.

The node sequence matrix is therefore generated and factorized, usually, by applying a neural network architecture, the most popular being the Skip-Gram [98, 99], to produce low dimensional vector representations for each node while maintaining their proximity in the new embedded space.

Random walks applied to dynamic graphs must generate time-dependent contexts $C(t)$ in addition to sequences that capture topological dependencies. Then, the methods based on random walks are separated according to how they include the temporal aspect into the calculation: (i) random walk on snapshots, where a time-dependent node sequence matrix is generated by applying random walks starting on each node at each snapshot, and further optimizing a joint problem that takes into account the temporal dependency; (ii) evolving random walks, where the node sequences are generated for the initial time (first snapshot), then the method incrementally updates node representation by updating random walks starting on nodes affected by topological evolution; and (iii) temporal random walks, which define time-dependent context matrices by allowing random walks across consecutive timestamps and considering time ordering restriction. Also, other node sequence sampling methods besides random walks may be applied to generate contexts, including neighborhood aggregation. In the following, we further detail each of these methods.

3.3.1 Random Walks on Snapshots

This approach performs random walks on each snapshot of a dynamic graph, obtaining vector representations by optimizing a joint problem taking into account temporal dependencies. It is important to note that methods following this approach generate contexts whose temporal connection between two matrices at consecutive time points is not modeled by random walks. Instead, the temporal dependency is defined later in the generation of the embeddings, taking into account the temporal smoothness. For instance, embeddings may be learned for each graph snapshot independently using methods including node2vec and DeepWalk, and afterward, the representations may be combined using operations, from simple vector concatenation to dynamic embeddings and orthogonal transformations to align embedding vectors at consecutive timestamps.

De Winter et al. [100] and Dyn2Vec [29] apply vector concatenation to node representations over time. The former applies node2vec for each snapshot, whereas the latter employs a DeepWalk variant, whose probability of choosing a certain edge depends on the normalized edge weight. Chen et al. [25] initialize node embeddings by using a Gaussian prior with a diagonal covariance, and learn representations over time using dynamic Bernoulli embeddings, considering the rows of the node sequence matrix as the context for each node. tNodeEmbed [101] preserves static network neighborhoods of nodes in a d-dimensional feature space by using Orthogonal Procrustes, and optimizes a LSTM for specific tasks (i.e., link prediction and multi-label node classification). DynSEM [62] train node embeddings for each timestamp using node2vec, align node embeddings into a common space using Orthogonal Procrustes, and optimize a joint loss

function taking into account temporal smoothness.

Table 6 lists the methods described above and points out both static embedding method applied for each snapshot and how they handle temporal dependencies.

Table 6: Random Walks on Snapshots.

Algorithm	Static Embedding Method	Temporal Dependence Handling
(Static) [100]	node2vec	Vector Concatenation
Dyn2Vec [29]	DeepWalk variant	Vector Concatenation
[25]	Gaussian Initialization	Dynamic Bernoulli Embeddings
tNodeEmbed [101]	node2vec	Orthogonal Procrustes and LSTM
DynSEM [62]	node2vec	Orthogonal Procrustes and Temporal Smoothing Loss Function

3.3.2 Evolving Random Walks

Generating random walks for every timestamp is an expensive time-consuming process. Several approaches first generate embeddings for the initial timestamp by using a static random walk approach, then incrementally update node representations taking into account that, in general, only a few nodes are influenced by topological evolution. Dynnode2vec [102] follows this approach by sampling node sequences for only evolving nodes instead of generating random walks for all nodes in a given timestamp and afterward feeding these sequences as an input to a dynamic Skip-Gram model [103], which is initialized at the first snapshot and used for weight initialization of the Skip-Gram of subsequent timestamps.

Other approaches include EvoNRL [104], which initially employs node2vec, stores the random walks in memory, and updates the set of random walks when a single new edge arrives in the network. EvoNRL uses a similar dynamic Skip-Gram model previously mentioned [103]. Sajjad et al. [105] follow the same Skip-Gram implementation over time and proposes random walk update algorithms aiming to be statistically indistinguishable from a set of random walks generated from scratch on the new graph. NetWalk [106] also proposes a network embedding algorithm inspired by the Skip-Gram architecture (which the authors call Clique Embedding). It uses a deep autoencoder neural network to learn vector representations through a stream of random walks while minimizes the pairwise distance among all nodes in each walk. Evolving random walks are used in order to mitigate the computational cost of performing a full random walk in every snapshot. The former is not expected to be as precise as the latter, as shown in some approaches [105]. But the small loss in accuracy is compensated by the huge computational efficiency shown by these methods.

Table 7 lists the methods based on evolving random walks and points out both static embedding methods applied for each snapshot and how they update random walks and vector representations.

Table 7: Methods based on Evolving Random Walks.

Algorithm	Static Embedding Method	Update Method
dynnode2vec [102]	node2vec	Dynamic Skip-Gram Model
EvoNRL [104]	node2vec	Skip-Gram Model over Time
[105]	DeepWalk with Unbiased Random Walk Updates	Skip-Gram Model over Time
NetWalk [106]	Clique Embedding (AutoEncoder)	Vertex Reservoir and Walk Updating

3.3.3 Temporal Random Walk Methods

The methods described in Sections 3.3.1 and 3.3.2 consider that random walks and their updates are made to each snapshot separately. However, one way to include the time dependency directly in a sequence of nodes generated by random walks is to build a method to create a corpus of walks over time, respecting the temporal flux. In the literature, these walks are regarded as temporal walks [34, 6].

It is possible to generalize the Skip-Gram architecture to handle continuous-time dynamic networks, as described by Nguyen et al. [34]. In particular, the authors propose a general framework called CTDNE for learning time-preserving embeddings and propose several methods to select the subsequent nodes

Table 8: Temporal Random Walk Methods.

Algorithm	Neural Network Model	Comments
CTDNE [34]	Skip-Gram model	Defined temporal random walk embedding methods
T-EDGE [45]	Skip-Gram model	Encompasses weighted dynamic networks
[100] (Dynamic)	node2vec	Continuous version of the first random walks on snapshots
STWalk2 [107]	Skip-Gram model	Separates temporal random walks and spatial random Walks
LSTM-node2vec [108]	node2vec and LSTMs	Handles both temporal sequences and static sequences
DeepCas [11]	DeepWalk, GRUs and Attention Mechanism	Diffusion cascades
DAN [109]	Feed-forward neural network and attention mechanism	Diffusion cascades
[111]	GCNs and GraphSAGE	Diffusion cascades
Topo-LSTM [110]	LSTMs	Diffusion cascades

from a starting node, thus performing a temporal random walk: (i) an unbiased temporal neighbor selection; (ii) a biased selection, which may be based on temporal exponentially-weighted decay (i.e., older timestamps have an exponentially lower contribution to the selection) or on temporal linearly-weighted decay.

Several approaches follow CTDNE's paradigm, including Wu et al. [45], which developed T-EDGE to encompass weighted networks, and De Winter et al. [100], proposing a continuous-time version of node2vec. STWalk2 proposed by Pandhre et al. [107], on the other hand, generates a spatial walk for each snapshot and a temporal walk, employing a Skip-Gram network to combine the two learned embeddings to get node representations. LSTM-node2vec [108] trains an LSTM autoencoder with node sequences generated by temporal random walks, and afterward initialize node2vec with the input layer weights of the trained LSTM encoder for each snapshot at time t.

Diffusion prediction problems are related to temporal random walks, but the exact timestamp of diffusion is not necessarily known. instead only the temporal ordering is defined (i.e., typically one does not know exactly when some information have passed from a node to another, but the source and the target of the diffusion process is known). Models following this objective include (i) DeepCas [11], which uses GRUs and attention mechanisms to predict the future size of the cascade, (ii) DAN [109], which outputs the probability distribution of the next infected node leveraging feed-forward neural networks and attention mechanism, and (iii) Topo-LSTM [110], employing LSTMs to handle temporal dependence over diffusion. Moreover, Yang et al. [111] implement GRUs, GCNs and GraphSAGE to predict next affected nodes, and a reinforcement learning framework to predict cascade size.

Temporal Random Walks represents a more natural way to deal with dynamic continuous graphs [34, 100], since it does not require any time discretization of the graph into snapshots. It is also the ideal approach for diffusion problems, as we have shown. Table 8 lists the temporal random walk methods and points out both static embedding method applied for each snapshot and how they update random walks and vector representations.

3.3.4 Other Node Sequence Sampling Methods

Some techniques exhibit steps or insights given by two different random walk based approaches. For instance, several models create a graph containing the nodes at a given time t and their neighbors at the same timestamp t and the previous timestamps in a defined time window, and employ random walk procedures leveraging temporal ordering [107, 112]. In particular, DHNE [46] gives exponential-decaying weights for edges connecting nodes and past neighbors. These approaches share elements from random walks over snapshots and temporal random walks. Furthermore, StreamWalk algorithm introduced by Beres et al. [113] compresses both evolving random walks and temporal random walks by updating the weight for walks to handle more recent edges.

Although most node sequence sampling methods are based on random walks, some techniques rely on other ways to aggregate the neighborhood. Liu et al. [114] develop a spatial-temporal neural attention mechanism to estimate the co-occurrence matrix and guide the embedding algorithm to focus on the

Table 9: Other Node Sequence Sampling Methods.

Algorithm	Comments
DHNE [46]	Historical-current graphs
STWalk1 [107]	Similar to historical-current graphs
DyAne [112]	Supra-adjacency representation
StreamWalk [113]	Temporal random walks updated for affected nodes
DKGE [115]	Attentive GCNs over subgraphs, and online learning
[114]	Co-ocurrence matrix using spatial-temporal neural attention model
weg2vec [35]	Weighted event graphs

context information with higher importance. Dynamic Knowledge Graph Embedding (DKGE) [115], on the other hand, applies an attentive GCN (AGCN) to learn contextual subgraph embeddings over knowledge graphs, integrating them with knowledge embedding of entities and relations to build the joint representation of each object in the graph. The temporal evolution is leveraged by an online learning strategy that learns knowledge embeddings and contextual element embeddings of emerging entities and relations, as well as knowledge embeddings of existing entities and relations with changed contexts (i.e. whose induced subgraphs are changed). Torricelli et al. [35] introduce weg2vec, which takes a dynamic network and project it into a weighted link stream (the authors called weighted event graph), sampling neighborhoods for events (i.e. the edges of the original dynamic graph) from the link stream (i.e. to create a graph that connects edges concerning involved nodes, co-occurrence, and event time difference), and inputting sequences of connected events to a Skip-Gram model.

Table 9 lists the methods described in this section and shows comments about their peculiarities, i.e., how they leverage temporal dependence or how they choose node context to apply the Skip-Gram model.

3.4 Edge Reconstruction based Optimization with Temporal Smoothing

Following the taxonomic approach proposed by Cai et al. [3] for static graphs, some methods for dynamic graphs have been identified whose approach is similar to techniques that directly optimize an objective function based on edge reconstruction. In addition to either maximizing edge reconstruction probability or minimizing edge reconstruction loss, these approaches also preserve temporal smoothness. It is noteworthy that these methods may be understood as reconstructing temporal edges between a node v at a given time t_i and the same node at the subsequent timestamp, therefore justifying this category even more adequately for embedding methods.

DynamicTriad [26] is a representative method of this category, aiming to preserve both structural information and evolution patterns of a network by modeling how a closed triad (i.e. three vertices connected) develops from an open triad (i.e. three vertices where two of them are not connected). The authors define the probability that an open triad (v, u, w) (where v and u are not connected) evolves into a closed triad, and the probability of the edge (v, u) will not be created, joining these probabilities into a distance-based loss function. Moreover, the model supposes that highly connected nodes should be embedded closely in the low-dimensional vector space and imposes this condition by a margin-based rank loss function, and finally considers temporal smoothness at consecutive time stamps.

Other approaches, solely based on a distance loss function include DNE [30], and Liu et al. [36]. Time-Aware KB Embedding [116] learns node embeddings by modeling relationships as translation operators in the low-dimensional vector space [117] and optimizes a joint margin-based ranking loss function concerning both temporal order score function (the temporal encoding) and translation embeddings (the topological encoding). Table 10 lists the methods described above, and the loss function each technique aims to minimize, pointing how they handle temporal dependence.

Note that the temporal smoothing given by a distance-based loss is similar to both matrix factorization problems and skip-gram based models. Indeed, there is a general view that demonstrates the relationship between network embedding approaches, matrix factorization, and Skip-Gram models [120]. Even

Table 10: Edge Reconstruction based Optimization.

Algorithm	Temporal Dependence Handling	Loss Function
DNE [30]	Delta of Theoretical Optimal Solution [118] and Temporal Smoothness	Based on LINE [119]
DynamicTriad [26]	Temporal Smoothness	Triadic Closure and Social Homophily
[36]	Temporal Smoothness	Based on Laplacian Eigenmaps
Time-Aware KB Embedding [116]	Temporal Order Score Function	Joint Margin-Based Rank

further, Liu et al. [120] provide a fundamental connection from an optimization perspective, which is the fundamental idea of edge reconstruction based methods. In this survey, these approaches are separated in the taxonomy to follow more strictly algorithmic properties rather than theoretical aspects of loss functions.

3.5 Methods based on Graph Kernel

As presented by Cai et al. [3] for static graphs, a few methods handle elementary substructures that are decomposed from a whole graph structure. They incorporate topological attributes built in the network processing step, including graphlet transitions count [58], graphlet frequencies over time [59] and adjacency matrix summation [121], to learn representations capable of reconstructing such elaborate attributes using a shallow approach of an autoencoder. Hence, since these substructures are used as a topological building block of a static network, dynamic graph embedding takes into account the transitions between different elementary structures. In addition, Béres et al. [113] developed an online second-order similarity (SecondOrder) that learns neighborhood similarity by Min-Hash fingerprinting, modifying the embedding vector whenever a neighbor of v_i gets more similar to v_j after adding the edge (v_i, v_j) into the network, which may be regarded as a graph kernel based approach.

3.6 Methods based on Temporal Point Process

Several dynamic graph embedding techniques, consider that interactions between nodes are stochastic processes whose probabilities depend on the topological structure of the network, on node features, and the network history. For these methods, it is assumed that an event influences a given node and, consequently, it can interact with other nodes in the network, if they are susceptible to the influence of the current node. Therefore, there is a probability that the event will propagate based on the mathematical definitions presented in Section 2.4, such as the conditional intensity function.

Main methods in this category include DyRep [8], handling a continuous-time deep model of a temporal point process using a conditional intensity function modeling the occurrence of an event p with time scale k between nodes v_i and v_j, and DeepCoEvolve [24], modeling the user-item interaction as a multidimensional temporal point process. Other approaches include KnowEvolve [38], modeling a fact in a knowledge graph as a temporal point process, M^{2}DNE [122], capturing edge evolution by a temporal point process with an attentive mechanism, in addition to a general dynamics equation concerning the linking rate, and HTNE [39] with an attention mechanism for neighborhood formation sequence of a node as a counting process. Furthermore, Knyazev et al. [40] extend DyRep [8], replacing the original encoder with a procedure that, given an event between nodes v_i and v_j: (i) calculates representations of all nodes at time t_{k-1}; (ii) returns an edge embedding for all pair of nodes; (iii) updates the embedding of node v_j based on all edges connected to it; and (iv) updates the edge embedding between nodes v_i and v_j.

MHDNE [123] models the edge formation process as two temporal sequences with historical edge information and network evolution information therein, respectively. In particular, the network evolution is based on open triangles and triadic closure problem [26], and the intensity function for a new edge creation at time t is given by a Hawkes process, leveraging a term dependent on node embeddings, a time decay function on an exponential form, and the distance between node's neighborhood. Wu et al. [42]

propose a Graph Biased Temporal Point Process (GBTPP), which aims to compute the probability of an event propagating to nodes $v_j \in \mathcal{N}(v_i)$ in the future timestamp t_{k+1} given event propagation history and node v_i, which is influenced by the event at time t_k.

3.7 Agnostic Models

The models described so far use some algorithmic paradigm as a basis for their development. A few approaches in the literature are independent of how the vector representations in each timestamp are obtained. They commit to learning the connections between representations at consecutive time points, or even within a time window. Because of this property of independence of the algorithmic procedure, we classify them as Agnostic Models. Two different paradigms may be followed: (i) the retrofitted model, where vector representations are learned for the initial graph by using any of the state-of-the-art static network embeddings, then the dynamics are captured by retrofitting the initial embedding with the subsequent graph snapshots; and (ii) the embedding space transformation method, where the representations of each graph snapshot are calculated by any static method separately, then a transformation function that connects an embedding at time t to the embedding at the next timestamp is learned. In the following, we detail these two paradigms.

3.7.1 Retrofitted Model

The retrofitted model [43] is based on the local temporal smoothness assumption, considering the vertex-centric evolution of the network. Therefore, for the first timestamp, the model employs an existing static embedding method to learn vector representations, but for subsequent timestamps, the vectors are updated by a local update method. In retrofitting, embeddings at previous timestamp $z_i(t-1)$ are revised by using the embedding of its neighborhood available from the graph snapshot at time t, hence the resulting vector $z_i(t)$ is similar to both the prior vector $z_i(t-1)$ and the vectors of its adjacent nodes in timestamp t. This update rule is carried out until convergence for each time step. It is noteworthy that, except for the first timestamp, the retrofitted model does not learn embeddings directly from data, instead it presumes a temporal smoothing criteria to guarantee node representations over time.

3.7.2 Embedding Space Transformation Model

The second agnostic approach assumes that the network time evolution is a global process attempting to fulfill the global temporal smoothness objective by considering the temporal evolution of a network as a transformation over the node embedding vectors of successive timestamps. Once the transformation operator is learned, it can map the latent representation from a known snapshot to the next unobserved snapshot.

Saha et al. [43] introduce two paradigms for embedding space evolution: (i) **homogeneous transformation**, where the transformation is assumed to be the same across any two successive timestamps; and (ii) **heterogeneous transformation**, which refrains from the uniformity assumption i.e. every pair of timestamps has a different transformation procedure. These two paradigms are further discussed in the following:

- **Homogeneous Transformation:** This transformation is shared across timestamps, and Saha et al. [43] have proposed a linear transformation in order to learn these mappings between embedding spaces.
- **Heterogeneous Transformation:** Methods based on heterogeneous transformation learn a different projection matrix for each pair of network snapshots. Saha et al. [43] employ a linear

heterogeneous transformation model, learning $N_S - 1$ different transformation matrices and obtaining a final transformation matrix by combining these different matrices.

3.8 Other Dynamic Graph Embedding Approaches

At last, some methods do not fit into any of the discussed categories, either by presenting some specific methodology and different from all approaches, or by combining different techniques without having one in particular as the main one. It is also interesting to note that some methods aggregate temporal information of the dynamic network into a static graph (i.e. a network containing all interactions and vertices that were present from the beginning of a network until the final timestamp of analysis) [5, 33]. Therefore, these works use static embedding methods over a network that encompasses temporal information stored at its edges.

4 Dynamic Graph Embedding Applications

In this section, we provide an overview of different network applications that are typically improved by the embedding methods for dynamic graphs presented so far. Applications of dynamic graph embeddings for network mining can be divided into which elements of the network they are oriented to or focused on: (i) **node related**, including node classification, recommendation systems, and trajectory analysis; (ii) **edge related**, including link prediction and event time prediction; and (iii) **graph related**, including graph classification over time, network visualization and diffusion prediction. The complete list of tasks discussed in this survey is shown in Figure 10.
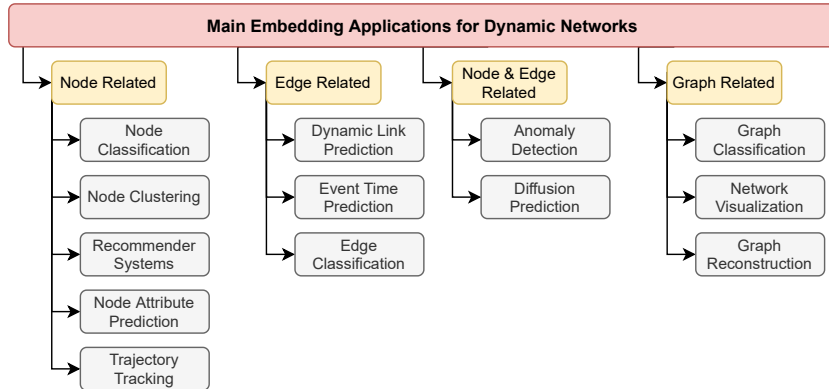

Figure 10: Every embedding application for dynamic networks covered in this survey, organized by which elements of the network they are oriented to or focused on.

4.1 Node-Related Applications

Node embeddings are used for various purposes on network analysis, with applications already known in static graphs, but handled over time, including **node classification**, **node clustering**, and **recommendation systems**, up to novel applications, specific within the dynamic scenario, such as the **node feature prediction** and **trajectory tracking**.

- **Node Classification:** The node classification problem focuses on the assignment of a class label to each node in a graph based on the rules learned from the labelled nodes. In a dynamic network, it is possible to (i) classify nodes whose labels are unknown in a given timestamp t, considering behavior and labels of the other nodes in the network; or (ii) to predict the classification of a node in the future, given that node labels can vary over time [124]. Approaches concerning node

classification apply a classifier on labeled node embeddings for training, including: (i) linear layer [54]; (ii) SVM [50, 48, 46, 125, 123, 58, 121]; (iii) logistic regression [7, 31, 39, 122, 100, 31, 108, 114, 105, 112, 30, 36, 125]; (iv) softmax [47, 96, 93, 101, 78, 10]; and (v) random forests and gradient boosting techniques [100, 58, 121].

- **Node Clustering:** While the classification task is supervised, clustering similar nodes is an unsupervised task, aiming to group similar vertices when information about labels is unavailable. An important challenge in this task is to ensure that the embeddings of similar nodes are close to each other in the vector space while being able to capture possible node transitions between different clusters over time, as in DynGem [10] and dyngraph2vec variations [27]. Clusters in the embedding space can also represent different behavior patterns of nodes over time, which is defined by Rossi et al. [55] as an analysis of the role evolution of each node in the network.

- **Recommender Systems:** A dynamic network consisting of users, items, and timestamped interactions between users and items may be explored by embedding methods to recommend items to users according to their interests over time. As discussed by Kazemi et al. a[12], recommendations may suggest items not exactly similar to user's interest in order to attract the user to a novel interest, and recommendations may arouse a future desire for items of certain type even if the user does not display any immediate interest.

- **Node Attribute Prediction:** It is important to predict time-varying attributes of network nodes. Formally, given a time-varying graph $\mathcal{G} = \{G(t_0), ..., G(t_{N_S-1})\}$ with additional node attributes $X(t) \in \mathbb{R}^f$ (where f is the number of these attributes) as the training data, this task aims to estimate the real-valued variable $X(t_{N_S}) \in \mathbb{R}^f$ at time t_{N_S} for each node in the graph [124]. This problem is also known as relational time series regression, and node embeddings over time can serve as input to time series prediction models, such as ARIMA or recurrent neural networks, to enhance prediction by leveraging topological information along with attribute evolution.

- **Trajectory Tracking:** From the representations obtained by embedding over time, it is possible to make an analysis of the trajectories of each entity. Ferreira et al. [28] capture ideological changes of two diverse party systems (Brazil and the United States), as expressed by members' voting behavior, by mapping the network into a temporal latent ideological space. Therefore, the authors have tracked individual members overtime in the low-dimensional vector space, analyzing how vector representations of individual members change and then measure ideological shifts over time. Although this task shares similarities with the node clustering problem, the focus of trajectory tracking is not on the groups themselves, but on the transitions between them.

4.2 Edge-Related Applications

Edge-related tasks comprise the most commonly explored problem by the techniques presented in this survey: **dynamic link prediction**. However, a novel application, when compared with static methods, appears in the literature: **event time prediction**, whose focus is on the detection of the time instant when a new edge should appear. Finally, the **edge classification** may also be treated on heterogeneous dynamic graphs where edges have labels.

- **Dynamic Link Prediction:** Link prediction in dynamic networks is more complex than in static networks, since it comprises two different tasks: (i) **temporal link prediction**, (the prediction of new edges) [126], where, given a sequence of snapshots of an evolving network $\mathcal{G} = \{G(t_0), ..., G(t_{N_S-1})\}$, aims to predict the links in $G(t_{N_S})$ (where N_S is the total number of snapshots), i.e. to construct a function $f(v, u)$ that predicts whether an edge (v, u) exists between nodes v and u at time t_{N_S} [43]; and (ii) **link completion** (prediction of previously observed edges) [126], which consists of finding the missing links along the evolving network. Most approaches

consider link prediction as a classification task, where labels are (i) existence; or (ii) non-existence of an edge. Junuthula et al. [126] provides a deeper discussion regarding evaluation of link prediction on dynamic networks, and Yu et al generalizes temporal link prediction to include: (i) prediction of link weights, considering henceforth the aforementioned definition as a particular case when the link weight is restricted to 0 or 1, and (ii) prediction at timestamp $N(S) + \alpha$, where $\alpha \geq 0$, and the classical definition above regards the special case $\alpha = 0$.

- **Event Time Prediction:** In dynamic networks it is valid to question at which timestamp a given interaction can occur, configuring the event time prediction task. Methods based on temporal point process have a mathematical formulation to predict the next time point t_k for an event given a pair of nodes v and u and network history (i.e. a list of interactions over time), including DyRep [38] and DeepCoevolve [24].

- **Edge Classification:** When the edges of a network can belong to certain classes, the problem of classifying the edges shares similarities with the node classification task. For instance, interactions between nodes may be associated with a trustworthy rating (i.e. users who trust others, or not), or sentiment analysis (i.e. a post on a social network). Therefore, predicting the label of an edge (v_i, v_j) over time may be done concatenating node embeddings or operating over them to obtain edge embeddings, and applying a classifier afterward [47].

4.3 Node- and Edge-Related Tasks

Some tasks can be applied to both nodes and edges in a graph, depending on how the problem is formulated. In this context, two tasks have been identified so far: (i) **anomaly detection**, which can be related to a node with anomalous behavior or to an unwanted or unexpected edge; and (ii) **diffusion prediction**, which can either identify which nodes will be affected by a diffusion process or detect edges most likely to diffuse information.

- **Anomaly Detection:** Anomaly detection is an important application for detecting malicious activity in networks. These anomalies can be detected by unexpected changes in the vector representation of nodes and edges, suggesting that some non-expected activity is arising at some timestamp. Goyal et al. [10] propose the definition of $\delta_t = \|Z(t_{k+1}) - Z(t_k)\|_F$ as the change in embedding between time t_k and t_{k+1}, and a threshold to consider the node behavior as anomalous when its embeddings change above this value. This threshold value can be calibrated diferently to each specific problem, and can be tuned to identify certain types of anomalies based on node behavior changes, where each type of behavior is encoded differently. A similar approach is presented by Rossi et al. [55], which defines several groups of behaviors for the nodes of the network based on node embeddings (i.e. similar to node clustering), hence the authors claim to detect anomalous behaviors from abrupt node transitions between different clusters. Khoshraftar et al. [108] formulate the anomaly detection as a classification task, where edges may belong to anomalous or normal class labels, therefore using node embeddings to compute edge representations and then using their method (LSTM-Node2vec) to classify the edges.

- **Diffusion Prediction:** Diffusion problems solved by embeddings methods may be categorized as (i) a sequence prediction problem (i.e. a microscopic diffusion problem), willing to predict the future affected node given the previously affected ones [109, 111, 110, 127, 128]; or (ii) a regression problem, which predicts the future numerical properties of the network (e.g. the total number of infected nodes in a macroscopic diffusion problem) [11, 111].

4.4 Graph-Related Applications

Problems related to the whole graph usually analyze the network globally, therefore dealing with tasks that are not centered on vertices or edges. The main examples verified in the literature are: **graph classification**, **network visualization**, and **graph reconstruction**.

- **Graph Classification:** Classifying the whole graph over time into one class from a set of predefined categories $\mathcal{L}^G$ is a relevant problem when the topological structure and possible attributes of each node in a network configure a global classifiable behavior [49, 48]. By obtaining a whole graph embedding over time (either by aggregating node embeddings, or using graph kernels, as in Section 3.5), it is possible to use the same classifiers presented in the node classification task to either perform classifications at known timestamps or to predict the classification of the graph in the future.

- **Network Visualization:** It is also possible to visualize dynamic graphs in 2D or 3D space by applying dimensionality reduction techniques that preserve the embedding structure, such as t-SNE [129], to node embeddings. Goyal et al. [10] state that, to avoid visualization instability (i.e. embedding instability over time), t-SNE needs to be initialized with an identical random state for all timestamps.

- **Graph Reconstruction:** The learned vector representations may reconstruct the dynamic graph through operations in the vector space that decode similarity information between pairs of nodes, such as a dot product or pairwise distance to estimate the adjacency matrix or the weight matrix. Goyal et al. [10] propose a methodology to rank pair of nodes according to their corresponding reconstructed similarity, then define the reconstruction precision as the ratio of real links in the top k pair of nodes.

5 Conclusion and Future Directions

In this survey, we conducted a comprehensive review of the literature in embedding methods for dynamic graphs. We defined the problem of embeddings for dynamic graphs inspired by previous surveys on static graph embeddings, whereas introducing some important concepts to consider scenarios for dynamic graphs. We proposed a taxonomy to classify the problem settings for dynamic graph embedding problems, broadening the design presented in the static scenario by introducing fundamental time aspects in embeddings, such as the different dynamic graph models that can be embedded, in addition to embedding outputs that aggregate temporal information or track representation trajectories. We also proposed a taxonomy for the different embedding paradigms for dynamic graphs, classifying them according to the methodology they use, topological-temporal properties that preserve, and assumptions made for the method to be valid. After that, we summarized the applications that the embedding of dynamic graphs enables.

- **Development and Expansion of Libraries and Frameworks for Dynamic Graph Embedding:** With the increasing number of dynamic graph embedding techniques, it is interesting to invest in developing and expanding a framework capable of unifying the different algorithms, applications, and standard benchmark datasets. Goyal et al. [130] propose DynamicGEM, an open-source Python library consisting of state-of-the-art algorithms for dynamic graph embeddings, focusing on node embeddings. The library contains the evaluation framework for graph reconstruction, static and temporal link prediction, node classification, and temporal visualization, with various metrics implemented to evaluate the state-of-the-art methods, and examples of evolving networks from various domains. Since DynamicGEM provides a template to add new algorithms, it would be interesting to invest in further developing the package so as to incrementally insert new techniques.

In addition, expanding the framework to include embeddings of other types (such as edges, hybrids, and whole graphs), as well as methods for continuous-time dynamic graphs. Building a reference package of embedding methods for dynamic graphs (be it DynamicGEM or other) would benefit the community interested in this topic.

- **Mathematical Analysis and Different Stability Metrics:** To define useful and accurate stability metrics is a future research field in the area, with a more theoretical focus. Goyal et al. [10] suggest a metric considering the adjacency matrix and snapshot models. Nevertheless, it is necessary to explore these metrics in more detail, by testing them on real-world networks and checking the behavior of embeddings for different methods and problems. In addition, a more sophisticated mathematical analysis may promote a research field direction to improve understanding of the relationship between representations and the evolution of dynamic graphs.

- **Temporal Multiscale Evolution Embedding:** In real networks, the phenomena of temporal evolution may be associated with different scales (e.g. daily, weekly, monthly, and yearly phenomena). It would be interesting to investigate embedding techniques capable of efficiently capturing these peculiarities in a methodology that deals with temporal multiscale evolution. Trivedi et al. [8] make an important step forward in this direction by considering two different timescales: for dynamics on the network and dynamics of the network.

- **Dynamic Hypergraph Embedding:** The existing models for dynamic graphs only consider edges connecting two nodes in the graph, therefore not being able to expand to hypergraphs, where sets of nodes form connections without necessarily having binary ordered relations. Although there are a few works in hypergraph embedding [131] and even fewer that are extended to dynamic hypergraphs [132], a promising area of research is to develop new methods for dynamic hypergraphs, as well as extending some of the existing dynamic graph embedding methods, allowing them to handle hyperedges.

- **Capturing Latencies and Spatial-Temporal Edge Patterns:** Although the methods described in the survey capture dynamic behaviors such as topological evolution, feature evolution and processes on the network, more sophisticated temporal models take into account that nodes and edges are not created or removed instantaneously in the network [6]. Thus, latency is an important feature of dynamic networks, as it carries information about the affinity between nodes or a node within the network, and no method described above has a methodology for dealing with such factors. In addition, spatio-temporal edges make embedding of dynamic graphs more complex than extracting information from snapshots, or from timestamped edges, as temporal correlations are more complex between non-consecutive timestamps [22].

- **Generalization of Graph Embedding to Higher-Order Dimensional Networks:** With the diversity of models for dynamic graphs, it is noted that there is no consensus to define a more general model and, consequently, an embedding that can be generalized to as many dynamic graphs as possible. Keeping this in mind, there are graph generalization models, including MAGs [22] and Stream Graphs [23], and a future direction of research may be the application of embedding methods in such models. Even more, it is interesting to suggest extending embedding for higher-order graphs, allowing not only the capture of temporal and topological properties but also of multilayer structures at a different time and connectivity scales.

- **Generation of Property-Preserving Network Evolution in Embedding Space:** Several complex network properties, such as pathways, degree distribution, and scale invariance, may change over time, and finding out which patterns of temporal evolution conserve or change these properties is challenging. Cheng et al. [133] propose a structure-preserving model reduction procedure developed for linear network systems, whereas Rossi et al. [55] propose a matrix factorization to discover roles of certain vertices in a network and study possible changes in these roles over time. One

possible direction for future research is to use representations in low-dimensional spaces to study and generate evolutions in a network that are capable of preserving certain properties of interest. One possible way to pursue this idea is to explore generative models, such as variational autoencoders and GANs, and use learned distributions of input data to generate new networks that are similar to training, and thereby capture patterns associated with their characteristics.

Acknowledgment

References

[1] Albert-László Barabási et al. *Network Science*. Cambridge University Press, 2016.

[2] William L Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin*, 2017.

[3] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Trans Knowl Data Eng*, 30(9):1616–1637, 2018.

[4] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[5] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal Link Prediction using Matrix and Tensor Factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10, 2011.

[6] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-Varying Graphs and Dynamic Networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[7] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed Network Embedding for Learning in a Dynamic Environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396. ACM, 2017.

[8] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations*, 2019.

[9] 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.

[10] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. DynGEM: Deep Embedding Method for Dynamic Graphs. *arXiv Preprint arXiv:1805.11273*, 2018.

[11] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. DeepCas: An End-to-end Predictor of Information Cascades. In *Proceedings of the 26th International Conference on World Wide Web*, pages 577–586. International World Wide Web Conferences Steering Committee, 2017.

[12] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[13] Yu Xie, Chunyi Li, Bin Yu, Chen Zhang, and Zhouhua Tang. A Survey on Dynamic Network Embedding. *arXiv preprint arXiv:2006.08093*, 2020.

[14] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and Modelling of Dynamic Networks using Dynamic Graph Neural Networks: A Survey. *IEEE Access*, 2021.

[15] Leo Katz. A New Status Index Derived from Sociometric Analysis. *Psychometrika*, 18(1):39–43, 1953.

[16] Lada A Adamic and Eytan Adar. Friends and Neighbors on the Web. *Social networks*, 25(3):211–230, 2003.

[17] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A Survey on Network Embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[18] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network Representation Learning: A Survey. *IEEE Transactions on Big Data*, 2018.

[19] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[20] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[21] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral. *arXiv preprint arXiv:2006.12138*, 2020.

[22] Klaus Wehmuth, Artur Ziviani, and Eric Fleury. A Unifying Model for Representing Time-Varying Graphs. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[23] Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining (SNAM)*, 8(61), December 2018.

[24] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. Recurrent coevolutionary latent feature processes for continuous-time recommendation. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 29–34, 2016.

[25] Chuanchang Chen, Yubo Tao, and Hai Lin. Dynamic Network Embeddings for Network Evolution Analysis. *arXiv preprint arXiv:1906.09860*, 2019.

[26] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic Network Embedding by Modeling Triadic Closure Process. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[27] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing Network Dynamics using Dynamic Graph Representation Learning. *Knowledge-Based Systems*, 2019.

[28] Carlos Henrique Gomes Ferreira, Fabricio Murai Ferreira, Breno de Sousa Matos, and Jussara Marques de Almeida. Modeling Dynamic Ideological Behavior in Political Networks. *The Journal of Web Science*, 7, 2019.

[29] Sandra Mitrovic and Jochen De Weerdt. Dyn2vec: Exploiting Dynamic Behaviour using Difference Networks-Based Node Embeddings for Classification. In *Proceedings of the International Conference on Data Science*, pages 194–200. CSREA Press, 2019.

[30] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding. In *IJCAI*, pages 2086–2092, 2018.

[31] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. Dynamic Joint Variational Graph Autoencoders. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 385–401. Springer, 2019.

[32] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. TIMERS: Error-Bounded SVD Restart on Dynamic Networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[33] Ryohei Hisano. Semi-Supervised Graph Embedding Approach to Dynamic Link Prediction. In *International Workshop on Complex Networks*, pages 109–121. Springer, 2018.

[34] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-Time Dynamic Network Embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 969–976. International World Wide Web Conferences Steering Committee, 2018.

[35] Maddalena Torricelli, Márton Karsai, and Laetitia Gauvin. weg2vec: Event Embedding for Temporal Networks. *Scientific Reports*, 10(1):1–11, 2020.

[36] Xi Liu, Ping-Chun Hsieh, Nick Duffield, Rui Chen, Muhe Xie, and Xidao Wen. Real-Time Streaming Graph Embedding Through Local Actions. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 285–293. ACM, 2019.

[37] Junchi Yan, Hongteng Xu, and Liangda Li. Modeling and Applications for Temporal Point Processes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3227–3228. ACM, 2019.

[38] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3462–3471. JMLR. org, 2017.

[39] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding Temporal Network via Neighborhood Formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2857–2866. ACM, 2018.

[40] Boris Knyazev, Carolyn Augusta, and Graham W Taylor. Learning Temporal Attention in Dynamic Graphs with Bilinear Interactions. *arXiv preprint arXiv:1909.10367*, 2019.

[41] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. Deep Coevolutionary Network: Embedding User and Item Features for Recommendation. *arXiv preprint arXiv:1609.03675*, 2016.

[42] Weichang Wu, Huanxi Liu, Xiaohu Zhang, Yu Liu, and Hongyuan Zha. Modeling Event Propagation via Graph Biased Temporal Point Process. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[43] Tanay Kumar Saha, Thomas Williams, Mohammad Al Hasan, Shafiq Joty, and Nicholas K Varberg. Models for Capturing Temporal Smoothness in Evolving Networks for Learning Latent Representation of Nodes. *arXiv preprint arXiv:1804.05816*, 2018.

[44] Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. GCN-GAN: A Non-Linear Temporal Link Prediction Model for Weighted Dynamic Networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 388–396. IEEE, 2019.

[45] Jiajing Wu, Dan Lin, Zibin Zheng, and Qi Yuan. T-EDGE: Temporal wEighted MultiDiGraph Embedding for Ethereum Transaction Network Analysis. *arXiv preprint arXiv:1905.08038*, 2019.

[46] Ying Yin, Li-Xin Ji, Jian-Peng Zhang, and Yu-Long Pei. DHNE: Network Representation Learning Method for Dynamic Heterogeneous Networks. *IEEE Access*, 7:134782–134792, 2019.

[47] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[48] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. Learning to Represent the Evolution of Dynamic Graphs with Recurrent Models. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 301–307, 2019.

[49] Aynaz Taheri and Tanya Berger-Wolf. Predictive Temporal Embedding of Dynamic Graphs. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 57–64, 2019.

[50] Hao Wei, Guyu Hu, Wei Bai, Shiming Xia, and Zhisong Pan. Lifelong Representation Learning in Dynamic Attributed Networks. *Neurocomputing*, 358:1–9, 2019.

[51] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[52] Youru Li, Zhenfeng Zhu, Deqiang Kong, Meixiang Xu, and Yao Zhao. Learning Heterogeneous Spatial-Temporal Representation for Bike-Sharing Demand Prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1004–1011, 2019.

[53] Songgaojun Deng, Huzefa Rangwala, and Yue Ning. Learning Dynamic Context Graphs for Predicting Social Events. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1007–1016. ACM, 2019.

[54] Yuan Yuan, Xiaodan Liang, Xiaolong Wang, Dit-Yan Yeung, and Abhinav Gupta. Temporal Dynamic Graph LSTM for Action-Driven Video Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1801–1810, 2017.

[55] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling Dynamic Behavior in Large Evolving Graphs. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pages 667–676. ACM, 2013.

[56] Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. Diachronic Embedding for Temporal Knowledge Graph Completion. *arXiv preprint arXiv:1907.03143*, 2019.

[57] Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. Subgraph Pattern Neural Networks for High-Order Graph Evolution Prediction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[58] Mahmudur Rahman and Mohammad Al Hasan. Link Prediction in Dynamic Networks using Graphlet. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 394–409. Springer, 2016.

[59] V Dave and M Hasan. Triangle Completion Time Prediction using Time-Conserving Embedding, 2019.

[60] Wenchao Yu, Charu C Aggarwal, and Wei Wang. Temporally Factorized Network Modeling for Evolutionary Network Analysis. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 455–464. ACM, 2017.

[61] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[62] Yujing Zhou, Weile Liu, Yang Pei, Lei Wang, Daren Zha, and Tianshu Fu. Dynamic Network Embedding by Semantic Evolution. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[63] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic Graph Representation Learning via Self-Attention Networks. *arXiv preprint arXiv:1812.09430*, 2018.

[64] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Haifeng Chen, and Wei Wang. Link Prediction with Spatial and Temporal Consistency in Dynamic Networks. In *IJCAI*, pages 3343–3349, 2017.

[65] G. W. Stewart. Matrix Perturbation Theory, 1990.

[66] Evrim Acar and Bülent Yener. Unsupervised Multiway Data Analysis: A Literature Survey. *IEEE Trans Knowl Data Eng*, 21(1):6–20, 2008.

[67] Dimitrios Rafailidis and Alexandros Nanopoulos. Modeling the Dynamics of User Preferences in Coupled Tensor Factorization. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 321–324. ACM, 2014.

[68] Xiaomin Fang, Rong Pan, Guoxiang Cao, Xiuqiang He, and Wenyuan Dai. Personalized Tag Recommendation through Nonlinear Tensor Factorization using Gaussian Kernel. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[69] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[70] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016.

[71] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234. ACM, 2016.

[72] Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction. *arXiv preprint arXiv:1812.04206*, 2018.

[73] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[74] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*, 2014.

[75] Jinyin Chen, Jian Zhang, Xuanheng Xu, Chenbo Fu, Dan Zhang, Qingpeng Zhang, and Qi Xuan. E-LSTM-D: A Deep Learning Framework for Dynamic Network Link Prediction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[76] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *arXiv preprint arXiv:1511.05493*, 2015.

[77] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*, 2014.

[78] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Liu, and Xiang Zhang. Spatio-Temporal Attentive RNN for Node Classification in Temporal Attributed Graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3947–3953. AAAI Press, 2019.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[80] Thomas N Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*, 2016.

[81] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[82] Yuyang Gao, Lingfei Wu, Houman Homayoun, and Liang Zhao. DynGraph2Seq: Dynamic-Graph-to-Sequence Interpretable Learning for Health Stage Prediction in Online Health Forums. In *IEEE ICDM*, pages 1042–1047, 2019.

[83] Stephen Bonner, John Brennan, Ibad Kureshi, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Temporal Graph Offset Reconstruction: Towards Temporally Robust Graph Representation Learning. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3737–3746. IEEE, 2018.

[84] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[85] Yun Xiong, Yao Zhang, Hanjie Fu, Wei Wang, Yangyong Zhu, and S Yu Philip. DynGraphGAN: Dynamic Graph Embedding via Generative Adversarial Networks. In *International Conference on Database Systems for Advanced Applications*, pages 536–552. Springer, 2019.

[86] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[87] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[88] Thomas N Kipf and Max Welling. Variational Graph Auto-Encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[89] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[90] Stephen Bonner, Amir Atapour-Abarghouei, Philip T Jackson, John Brennan, Ibad Kureshi, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Temporal Neighbourhood Aggregation: Predicting Future Links in Temporal Graphs via Recurrent Variational Graph Convolutions. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 5336–5345. IEEE, 2019.

[91] Yifeng Zhao, Xiangwei Wang, Hongxia Yang, Le Song, and Jie Tang. Large Scale Evolving Graphs with Burst Detection. In *28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[92] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational Graph Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 10700–10710, 2019.

[93] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming Graph Neural Networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–728, 2020.

[94] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Yameng Gu, Xiao Liu, Jingchao Ni, Bo Zong, Haifeng Chen, and Xiang Zhang. Adaptive Neural Network for Node Classification in Dynamic Networks. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1402–1407. IEEE, 2019.

[95] Dongkuan Xu, Junjie Liang, Wei Cheng, Hua Wei, Haifeng Chen, and Xiang Zhang. Transformer-Style Relational Reasoning with Dynamic Memory Updating for Temporal Network Modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4546–4554, 2021.

[96] Hogun Park and Jennifer Neville. Exploiting Interaction Links for Node classification with deep graph neural networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3223–3230. AAAI Press, 2019.

[97] Prasha Shrestha, Suraj Maharjan, Dustin Arendt, and Svitlana Volkova. Learning from Dynamic User Interaction Graphs to Forecast Diverse Social Behavior. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2033–2042. ACM, 2019.

[98] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[99] 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, 2013.

[100] Sam De Winter, Tim Decuypere, Sandra Mitrović, Bart Baesens, and Jochen De Weerdt. Combining Temporal Aspects of Dynamic Networks with Node2Vec for a More Efficient Dynamic Link Prediction. In *2018 IEEE/ACM ASONAM*, pages 1234–1241. IEEE, 2018.

[101] Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4605–4612. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[102] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3762–3765. IEEE, 2018.

[103] Robert Bamler and Stephan Mandt. Dynamic Word Embeddings. *arXiv preprint arXiv:1702.08359*, 2017.

[104] Farzaneh Heidari and Manos Papagelis. EvoNRL: Evolving Network Representation Learning Based on Random Walks. In *International Conference on Complex Networks and their Applications*, pages 457–469. Springer, 2018.

[105] Hooman Peiro Sajjad, Andrew Docherty, and Yuriy Tyshetskiy. Efficient Representation Learning using Random Walks for Dynamic Graphs. *arXiv preprint arXiv:1901.01346*, 2019.

[106] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks. In *Proceedings of the 24th ACM SIGKDD Int. Conference on Knowledge Discovery & Data Mining*, pages 2672–2681, 2018.

[107] Supriya Pandhre, Himangi Mittal, Manish Gupta, and Vineeth N Balasubramanian. STWalk: Learning Trajectory Representations in Temporal Graphs. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 210–219. ACM, 2018.

[108] Shima Khoshraftar, Sedigheh Mahdavi, Aijun An, Yonggang Hu, and Junfeng Liu. Dynamic Graph Embedding via LSTM History Tracking. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 119–127. IEEE, 2019.

[109] Zhitao Wang, Chengyao Chen, and Wenjie Li. Attention network for information diffusion prediction. In *Companion Proceedings of the The Web Conference 2018*, pages 65–66, 2018.

[110] Jia Wang, Vincent W Zheng, Zemin Liu, and Kevin Chen-Chuan Chang. Topological Recurrent Neural Network for Diffusion Prediction. In *IEEE ICDM*, pages 475–484, 2017.

[111] Cheng Yang, Jian Tang, Maosong Sun, Ganqu Cui, and Zhiyuan Liu. Multi-Scale Information Diffusion Prediction with Reinforced Recurrent Networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4033–4039. AAAI Press, 2019.

[112] Koya Sato, Mizuki Oka, Alain Barrat, and Ciro Cattuto. DyANE: Dynamics-Aware Node Embedding for Temporal Networks. *arXiv preprint arXiv:1909.05976*, 2019.

[113] Ferenc Béres, Domokos M Kelen, Róbert Pálovics, and András A Benczúr. Node Embeddings in Dynamic Graphs. *Applied Network Science*, 4(1):64, 2019.

[114] Zhining Liu, Dawei Zhou, and Jingrui He. Towards Explainable Representation of Time-Evolving Graphs via Spatial-Temporal Graph Attention Networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2137–2140. ACM, 2019.

[115] Tianxing Wu, Arijit Khan, Huan Gao, and Cheng Li. Efficiently Embedding Dynamic Knowledge Graphs. *arXiv preprint arXiv:1910.06708*, 2019.

[116] Tingsong Jiang, Tianyu Liu, Tao Ge, Lei Sha, Sujian Li, Baobao Chang, and Zhifang Sui. Encoding Temporal Information for Time-Aware Link Prediction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2350–2354, 2016.

[117] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-Relational Data. In *Advances in Neural Information Processing Systems*, pages 2787–2795, 2013.

[118] Omer Levy and Yoav Goldberg. Neural Word Embedding as Implicit Matrix Factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

[119] 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.

[120] Xin Liu, Tsuyoshi Murata, Kyoung-Sook Kim, Chatchawan Kotarasu, and Chenyi Zhuang. A General View for Network Embedding as Matrix Factorization. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 375–383. ACM, 2019.

[121] Mahmudur Rahman, Tanay Kumar Saha, Mohammad Al Hasan, Kevin S Xu, and Chandan K Reddy. DyLink2Vec: Effective Feature Representation for Link Prediction in Dynamic Networks. *arXiv preprint arXiv:1804.05755*, 2018.

[122] Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S Yu, and Yanfang Ye. Temporal Network Embedding with Micro-and Macro-dynamics. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 469–478. ACM, 2019.

[123] Ying Yin, Jianpeng Zhang, Yulong Pei, Xiaotao Cheng, and Lixin Ji. MHDNE: Network Embedding Based on Multivariate Hawkes Process. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 409–421. Springer, 2019.

[124] Ryan A Rossi. Relational Time Series Forecasting. *The Knowledge Engineering Review*, 33, 2018.

[125] Yulong Pei, Jianpeng Zhang, GH Fletcher, and Mykola Pechenizkiy. Node Classification in Dynamic Social Networks. *Proceedings of AALTD*, page 54, 2016.

[126] Ruthwik R Junuthula, Kevin S Xu, and Vijay K Devabhaktuni. Evaluating link prediction accuracy in dynamic networks with added and removed edges. In *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 377–384. IEEE, 2016.

[127] Sylvain Lamprier. A Variational Topological Neural Model for Cascade-based Diffusion in Networks. *arXiv preprint arXiv:1812.10962*, 2018.

[128] Yuan Zhang, Tianshu Lyu, and Yan Zhang. COSINE: Community-Preserving Social Network Embedding from Information Diffusion Cascades. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[129] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[130] Palash Goyal, Sujit Rokka Chhetri, Ninareh Mehrabi, Emilio Ferrara, and Arquimedes Canedo. DynamicGEM: A Library for Dynamic Graph Embedding Methods. *arXiv preprint arXiv:1811.10734*, 2018.

[131] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3558–3565, 2019.

[132] Zizhao Zhang, Haojie Lin, Yue Gao, and KLISS BNRist. Dynamic Hypergraph Structure Learning. In *IJCAI*, pages 3162–3169, 2018.

[133] Xiaodong Cheng, Yu Kawano, and Jacquelien MA Scherpen. Graph Structure-Preserving Model Reduction of Linear Network Systems. In *2016 European Control Conference (ECC)*, pages 1970–1975. IEEE, 2016.